

Cisco DNA Enterprise Network Programmability Lab v1.4

Last Updated:27-JAN-2018

About This Solution

Digitization is transforming businesses in every industry, opening up a \$2.1 trillion global market opportunity by 2019, according to IDC. The path to digitization requires a digital network that evolves beyond just connectivity. This new network will enable business innovation, generate insights and create customer experiences. It will reduce cost and complexity with new orchestration and automation capabilities, while helping protect the business with an architecture designed for security.

At the same time that enterprises are addressing how to move to digitization, there has been tremendous innovation in networking, including software defined networking (SDN), network function virtualization (NFV), model-driven programming, overlay networks, open API's, cloud management, orchestration, analytics and more. These innovations have great promise to improve operational efficiency and enable digital applications, yet adoption has been slow due to the difficulty in consuming these many new technologies. The market needs a solution that integrates the critical innovations in networking software –virtualization, automation, analytics, cloud service management, and open and extensible programmability– into an architecture that can achieve these promises in an integrated and easy to consume manner.

Cisco [Digital Network Architecture \(DNA\)](#), is an open, extensible and software driven architecture for digital business. Cisco DNA complements Cisco's market leading, data center based Application Centric Infrastructure (ACI) technology by extending the policy driven approach and software strategy throughout the entire network: from campus to branch, wired to wireless, core to edge.

About This Demonstration

This lab includes the following:

- [Scenario 1: DevOps Style Human Interaction](#)
 - [Cisco Spark REST APIs: Find Room and Post](#)
 - [NeXt UI: DevNet Tutorials](#)
 - [NeXt UI: Hello Topology](#)
 - [NeXt UI: Grouping of Nodes](#)
 - [NeXt UI: Visualize Path Info from APIC-EM](#)
 - [Tropo Scripting APIs: Text-to-Speech and IVR](#)
- [Scenario 2: Device-Level Programmability and Virtualization](#)
 - [IOS EEM: Change Port Description Based on CDP Neighbor](#)
 - [IOS EEM: Assign Static IP If DHCP Fails](#)
 - [IOS EEM: Force a Failover Based on IPSLA-Reported Latency](#)
 - [IOS-XE Guestshell: Enable Guestshell](#)
 - [IOS-XE Guestshell: Python interpreter](#)

- [IOS-XE Guestshell: Prepare Python environment](#)
- [IOS-XE Guestshell: Python packaging](#)
- [IOS-XE Guestshell and EEM: Record commands to the database](#)
- [IOS-XE Guestshell: Outputs collector](#)
- [IOS-XE Guestshell and EEM: Config Diff To Spark](#)
- [IOS-XE NETCONF: Configure an ACL](#)
- [IOS-XE RESTCONF: Configure an ACL](#)
- [IOS-XE RESTCONF: Operational data to Spark](#)
- [Scenario 3: Controller-Level Programmability](#)
 - [Device Count in APIC-EM Inventory](#)
 - [Path Trace via APIC-EM REST APIs](#)

Content Resources

For more information:

- View the content overview: <https://dcloud-cms.cisco.com/demo/enterprise-sdn-dna-programmability-lab-v1>.
- Visit the Cisco dCloud Help page: <https://dcloud-cms.cisco.com/help/>.
- Access all available Cisco dCloud content: <https://dcloud.cisco.com>.
- Contact the Technical Lead or the Business Development Manager for your region: <https://dcloud-cms.cisco.com/help/dcloud-enterprise-networking-contacts>.

Requirements

The tables below outlines the requirements for this preconfigured demonstration.

Table 1. Requirements for completing from a virtual machine within a lab

Required	Optional
<ul style="list-style-type: none"> ● Laptop ● Credentials for developer.cisco.com ● Credentials for developer.ciscopark.com ● Cisco Spark client (www.ciscopark.com) 	<ul style="list-style-type: none"> ● Cisco AnyConnect ● Browser for SSL VPN

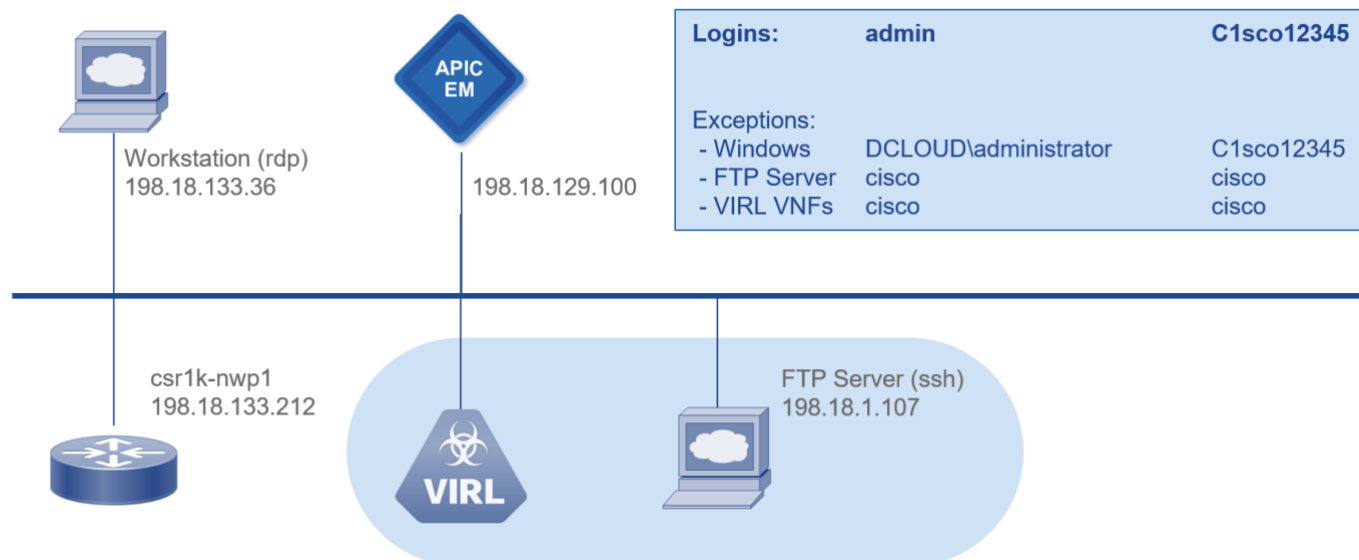
Table 2. Requirements for completing using a laptop

Required	Optional
<ul style="list-style-type: none"> ● ssh client (for example, PuTTY) ● Chrome browser ● Developer's editor with syntax highlighting (such as PSPad, Notepad++, Atom, Xcode or TextWrangler) ● GIT client (such as git-scm or desktop githubclient) ● python 2.7, 3.5 or 3.6 (http://python.org) 	<ul style="list-style-type: none"> ● Cisco AnyConnect

Topology

This content includes preconfigured users and components to illustrate the scripted scenarios and features of the solution. Most components are fully configurable with predefined administrative user accounts. You can see the IP address and user account credentials to use to access a component by clicking the component icon in the **Topology** menu of your active session and in the scenario steps that require their use.

Figure 1. Lab Topology



Get Started

A POD with the Lab Topology has been assigned to you by the instructor. In order to complete the lab, you can either

1. Connect to your Lab POD in dCloud using AnyConnect from the classroom PC
2. Use RDP to access the Workstation within the Lab POD
3. Complete the lab using python, chrome, git and tools already on the Lab POD's workstation

or alternatively if you have an existing development environment on your own PC

1. Connect to your Lab POD in dCloud using AnyConnect from your own PC
2. Complete the lab using python, chrome, git and tools on your own PC

For best performance, connect to the workstation with **Cisco AnyConnect VPN** [\[Show Me How\]](#) and the **local RDP client on your laptop** [\[Show Me How\]](#)

- Workstation 1. **198.18.133.36**, Username. **DCLLOUD\administrator**, Password. **C1sco12345**

NOTE: You can also connect to the workstation using the Cisco dCloud Remote Desktop client [\[Show Me How\]](#). The dCloud Remote Desktop client works best for accessing an active session with minimal interaction. However, many users experience connection and performance issues with this method.

Prepare the Lab Environment

The following steps are required to prepare the lab environment and to familiarize yourself with it.

Steps

1. Get the lab content from the GitHub (<https://github.com/xorrkaz/enterprise-network-programmability>) and ensure it's up to date. On the LAB POD workstation's windows command line, use:

```
C:\Users\administrator> cd C:\Users\administrator\Documents\GitHub\enterprise-network-programmability
C:\Users\administrator\Documents\GitHub\enterprise-network-programmability> git pull
```

NOTE: If you are attending an event or instructor-led lab session, the download link may vary – check with your instructor.

NOTE: If you are taking the lab from your own PC, **git clone** needs to be done first

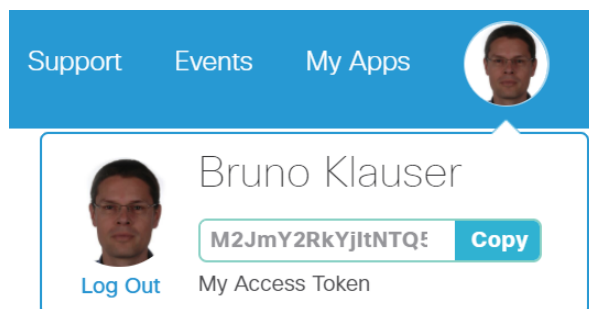
2. Copy **_LabEnv.TEMPLATE.py** to **_LabEnv.py** in **tasks** folder, then edit **_LabEnv.py** to capture your personal information. Update the following variables:

- a. Capture your name in the **LAB_USER** variable – edit the corresponding line to look like the example below.

```
# Replace YOUR-NAME-HERE in the line below
LAB_USER = 'Arthur Dent'
```

- b. Capture your Cisco Spark Access in the **LAB_USER_SPARK_TOKEN** variable. To get your Cisco Spark Access Token, login to <http://developer.ciscospark.com> and copy the token from the avatar icon in the top-right corner of the page.

Figure 2. Spark Access Token from developer.ciscospark.com



Then edit the corresponding line in **_LabEnv.py** and paste the token to look like the example below.

```
LAB_USER_SPARK_TOKEN = 'abcdefghijklmnopqrstuvwxyz'
```

Save the **_LabEnv.py** file.

NOTE: You may find it useful to import environment variables and functions from **_LabEnv.py** using `import _LabEnv` in your scripts, an example for this is `Hello-Lab.py` below.

Run the **Hello-Lab.py** script. Python is an interpreted scripting language – to run the script:

- c. Open a command line prompt within the **tasks** folder
- d. Enter **python Hello-Lab.py**. To validate the environment, the script will find (or create) the spark room and post a welcome message

```
C:\Users\administrator\Documents\GitHub\enterprise-network-programmability\tasks> python Hello-Lab.py
Failed to find room ID for 'dCloud DNA Enterprise Network Programmability Lab v1.3 creating it ...'
Found room ID for 'dCloud DNA Enterprise Network Programmability Lab v1.3' : ...
```

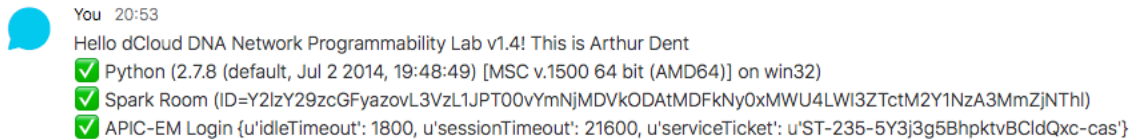
Open **Cisco Spark** client on your desktop (and accept any pending software updates).

Figure 3. Cisco Spark



You should now see a message from yourself – this confirms that your lab environment is ready.

Figure 4. Lab Environment Ready



NOTE: You are now ready to work on any of the Perspectives and Scenarios below

Scenario 1. DevOps Style Human Interaction

While Network Programming enables automated workflows via APIs, those workflows will often interact with human operators and users. As you explore Network Programmability throughout this lab, you will want to print, share, export or visualize your results – a good opportunity to make use of some great and simple human interaction capabilities as well:

Cisco Spark REST APIs: Find Room and Post

Instant messaging is a great way to interact with humans. Cisco Spark offers REST APIs, end-to-end encryption and clients across multiple platforms – making it a very viable choice for operational and DevOps workflows.

Objectives

Use the Spark REST API to find a Spark room by its name and post a message into the room.

Hints

This is the same Spark REST API and which has already been used by the prebuilt `_LabEnv.py` and `Hello-Lab.py` scripts in during the 'Prepare the Lab Environment' task.

There are two ways for you to approach this: either reuse what's already in `_LabEnv.py` (which is simpler) or create your own script (whereby you get to explore the Cisco Spark APIs in more detail). For both approaches there is a sample solution provided:

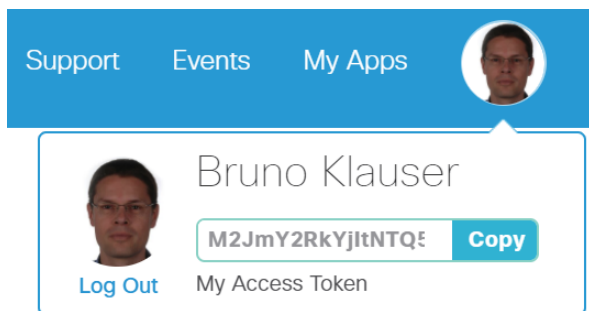
`spark-post-simple.py` and `spark-post-full.py` respectively.

Getting Started

You will need to:

1. Get your Spark Access Token from **developer.ciscospark.com**.

Figure 5. Spark Access Token from developer.ciscospark.com



Login to the Cisco Spark client software on your desktop.

NOTE: Write down the exact name of a Spark Room to use. (You may want to reuse the name defined in the `SPARK_ROOM_NAME` variable within `_LabEnv.py`)

Solution

The following are some key steps when creating a python script:

- First, import some useful modules. The json and requests module are helpful for dealing with http post/get requests and any JSON formatted payload. The _LabEnv module holds default settings specific to this lab environment – such as the name of the common Spark room to be used.

```
import _LabEnv
import json
import requests
import sys
```

- Posting a message into Spark via the API requires the unique room ID. Spark Human Interfaces use the human readable Title of a room to reference it. Hence we need to find the room ID for our room before we can post into it.

To do this, we query the list of all Spark Rooms you're a member of, then try to find the Spark Room ID based on comparing Room Titles with our expected room name.

```
SPARK_ROOM_ID = None

r = requests.get(_LabEnv.SPARK_API_ROOMS, headers=_LabEnv.SPARK_HEADERS, verify=False)
j = json.loads(r.text)

for tmproom in j['items']:
    if tmproom['title'] == _LabEnv.SPARK_ROOM_NAME:
        SPARK_ROOM_ID = tmproom['id']
        print("Found room ID for '" + _LabEnv.SPARK_ROOM_NAME + "' : " + SPARK_ROOM_ID)
        break
```

- Post into the Spark Room using it's Room ID.

```
m = json.dumps({'roomId':SPARK_ROOM_ID,'text':'Hi ' + _LabEnv.LAB_SESSION+ ' this is ' + _LabEnv.LAB_USER})
r = requests.post(_LabEnv.SPARK_API_MESSAGES, data=m, headers=_LabEnv.SPARK_HEADERS, verify=False)
print('Spark Response: ' + r.text)
```

- When you run your script, you should see a command line output similar to:

```
>>python spark-post-full.py
Found room ID for 'dCloud DNA Enterprise Network Programmability Lab v1.3' :
Y2lzY29zcGFyazovL3VzL1JPT00vZjFkMWVjNTAtZWExNS0xMWU2LWFmNjktM2JjOGZhOWEzN2I4
Spark Response:
{"id":"Y2lzY29zcGFyazovL3VzL1JPT00vZjFkMWVjNTAtZWExNS0xMWU2LWFmNjktM2JjOGZhOWEzN2I4","roomId":"Y2lzY29zcGFyazovL3VzL1JPT00vZjFkMWVjNTAtZWExNS0xMWU2LWFmNjktM2JjOGZhOWEzN2I4","roomType":"group","text":"Full Hello dCloud DNA Enterprise Network Programmability Lab v1.3 this is Arthur Dent","personId":"Y2lzY29zcGFyazovL3VzL1BFT1BMRS9jZDVhOTFiNS1jODc5LTQwYzItYjIyYi0yMGQ2MzY5YWM0YjM","personEmail":"bklauser@cisco.com","created":"2017-02-03T14:04:27.782Z"}
```

- When you run your script, your own message posted into the Spark room:

Figure 6. Message Posted in Spark Room



You 15:06

Full Hello dCloud DNA Enterprise Network Programmability Lab v1.3 this is Arthur Dent

NOTE: For further reading, refer to <https://developer.ciscospark.com/getting-started.html>

NeXt UI: DevNet Tutorials

The NeXt UI Toolkit provides a simple framework for visualizing any graph and topology related information. It supports quick ad-hoc interfaces as well as feature rich and visually appealing advanced implementations.

Objectives

Familiarize yourself with the basic anatomy of a user interface built using NeXt UI Toolkit.

Hints

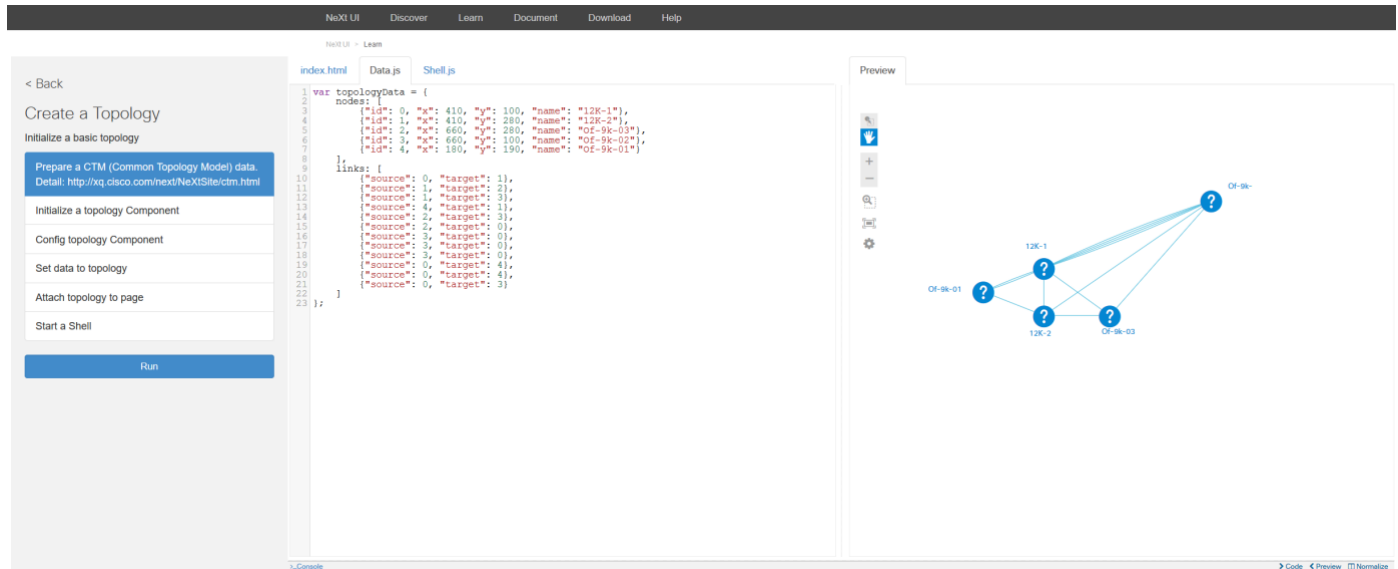
An implementation of the NeXt UI consists of three main elements – which can also be split into three individual files for simplicity:

- index.html: is a static web page linking to
 - the next style sheet next.css
 - the next JavaScript source next.js
 - Your Topology Data Data.js
 - Your customized NeXt user interaction Shell.js
- Data.js: includes your JavaScript variable definition for topologyData with lists of nodes, links and optionally groups of nodes called nodeSets
- Shell.js: creates an instance of a NeXt user interface and links it to your topology data. Optionally, default settings can be changed and customer behavior added

Steps

1. To get started, explore the anatomy of a NeXt UI via the Demo and Tutorials on DevNet. You might want to have a look at the prebuilt demos first (<https://developer.cisco.com/site/neXt/discover/demo/>) Then navigate to the learning tutorial at <https://developer.cisco.com/site/neXt/learn/> . Scroll down to the “Create a Topology” example and modify/interact with it.

Figure 7. NeXt UI Tutorial: “Create a Topology”



Click Run to see the interface in action. Try changing some of the default settings in Shell.js (such as using parallel or curve link types) and add or remove some links and nodes in Data.js

Explore the NeXt default menu in the Preview section.

Click the links and nodes to trigger the tooltip popup windows.

NeXt UI: Hello Topology

Once familiar with the basic structure of a NeXt UI, download the NeXt UI Toolkit and build your own first topology with the three files Data.js, Shell.js and index.html.

Objectives

Build your own NeXt UI Toolkit based Topology interface.

Hints

Scope of the NeXt UI Toolkit is entirely on the client side – ie.: everything that happens in your browser. To get started, you may want to work with a static topology, defined in a file (such as Data.js) and view it from a static html page (such as index.html) in your browser.

For more advanced and dynamic implementations at a later stage, consider integrating the NeXt based interface with your preferred application framework for client-server communications.

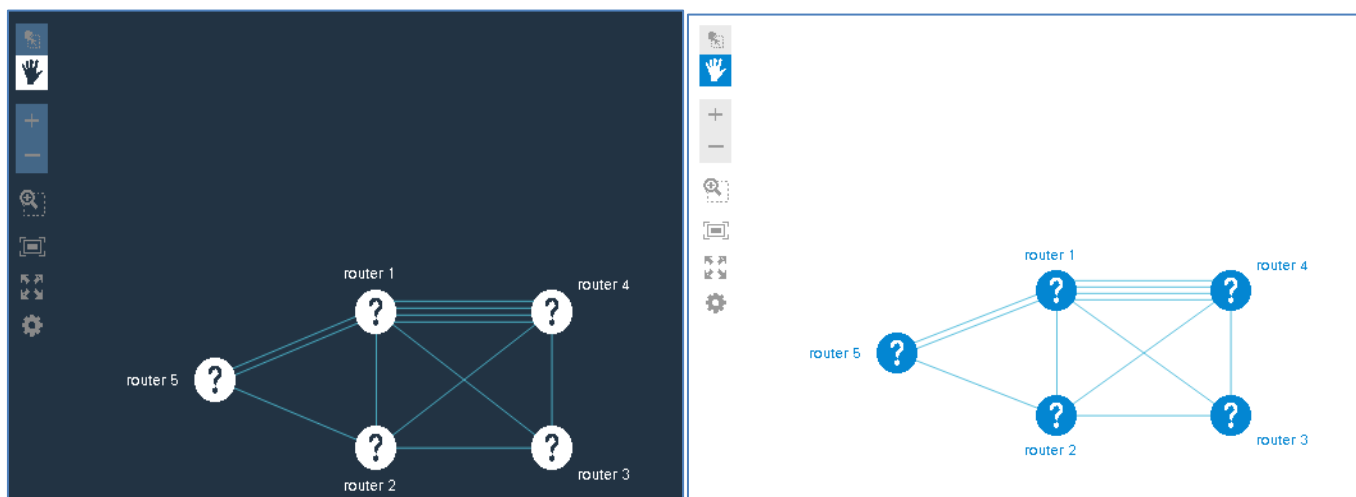
Steps

1. To build your own interface, you need to
 - Download the NeXt UI Toolkit
 - Create your Topology, Shell and HTML files

NeXt UI is available from Cisco DevNet as well as from OpenDaylight.org (<https://www.opendaylight.org/technical-community/getting-started-for-developers/downloads-and-documentation>) under the Eclipse Public License - v 1.0. For convenience, NeXt 0.11.1 Carbon is included with the /tasks/next-samples folder of this repository. A few sample topologies are included as well.

Now point your browser to the index.html file on your disk, modify some of the files and reload your browser page to explore.

Figure 8. Basic Structure of NeXt UI



NeXt UI: Grouping of Nodes

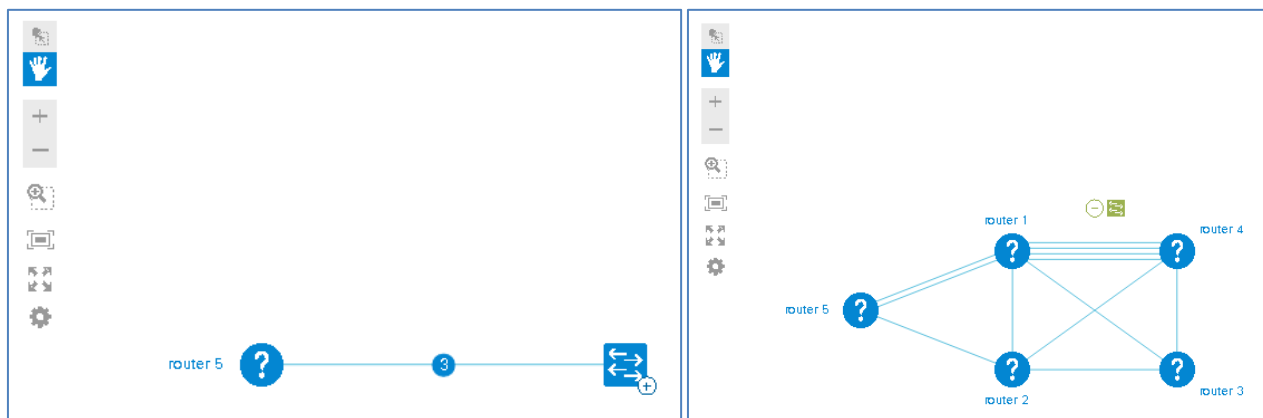
NeXt UI supports hierarchical grouping of Nodes via a concept referred to as nodeSet. A nodeSet includes individual nodes – which can either be nodes or nodeSets themselves.

1. Create a backup copy of your Data.js from the previous NeXt task
2. Then modify the Data.js to add a nodeSet into TopologyData after the list of nodes and links:

```
:
nodeSet: [
  {id: 100, type: "nodeSet", nodes: [0, 1, 2, 3], "x": 550, "y": 190,
    "name": "My Nodes", iconType: "groupM"}
]
:
```

NOTE: Make sure to add a comma after the closing bracket of the links list, before the nodeSet object.

Figure 9. Grouping of Nodes



NOTE: As a bonus task, explore multiple nodeSets and hierarchical nodeSets.

NOTE: Great resources while prototyping and sharing Javascript based user interfaces are JSFiddle.net. The Cisco NeXt UI Toolkit maintain a presence and share code samples

- on JSFiddle at <https://jsfiddle.net/user/nextsupport/fiddles/>
- on Codepen at <http://codepen.io/collection/nrBeEQ/> and <http://codepen.io/collection/nMWevE/>

NeXt UI: Visualize Path Info from APIC-EM

Let's visualize a NeXt topology based on some real data. The result of a Path Trace in APIC-EM represents simple linear topology with a few attributes for both nodes and links – this lends itself nicely for a learning task and has also proven to be a useful scenario in real life proactive troubleshooting implementations.

Objectives

Build a NeXt UI Toolkit based Topology interface from real topology data.

Hints

This is all about creating the Data.js file with NeXt topology information from the APIC-EM path trace JSON data. In a first step, if you do not want to write a file from python directly, you could just print to stdout and pipe or even copy/paste into Data.js from there.

Steps

To build your own interface:

1. If you have already created a script to export Path Trace information from APIC-EM, enhance your script to create a Data.js file. Alternatively, work from the `apic-em-1.1-path-trace-sample.py` and `apic-em-1.1-path-trace-to-spark-sample.py` sample scripts.
2. Several examples of possible Data.js output can be found in the `\tasks\next-samples` folder. A resulting topology could look like the following:

Figure 10. Sample Topology

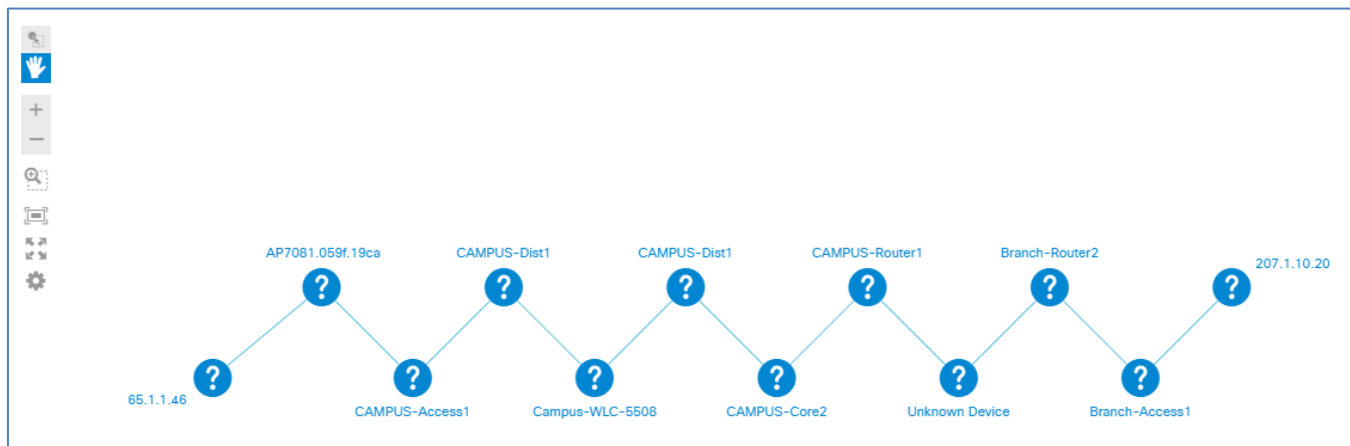
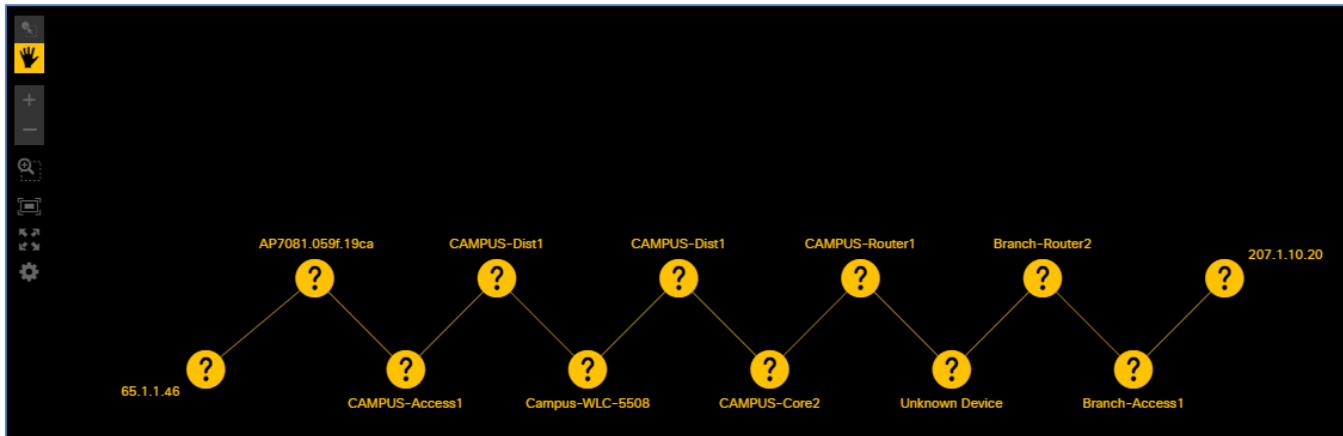


Figure 11. Sample Topology



- Once you have a basic linear topology, try to extend your script to produce a more sophisticated Data.js by selecting more accurate node icons or by grouping some of the nodes.

NOTE: For example, use the hostnames to group all access nodes and all campus nodes into dedicated nodeSets. Try to give your nodes icons representing the device type.

- Your topology should now look similar to another sample provided in `next-path-samples.zip`.

Figure 12. Topology

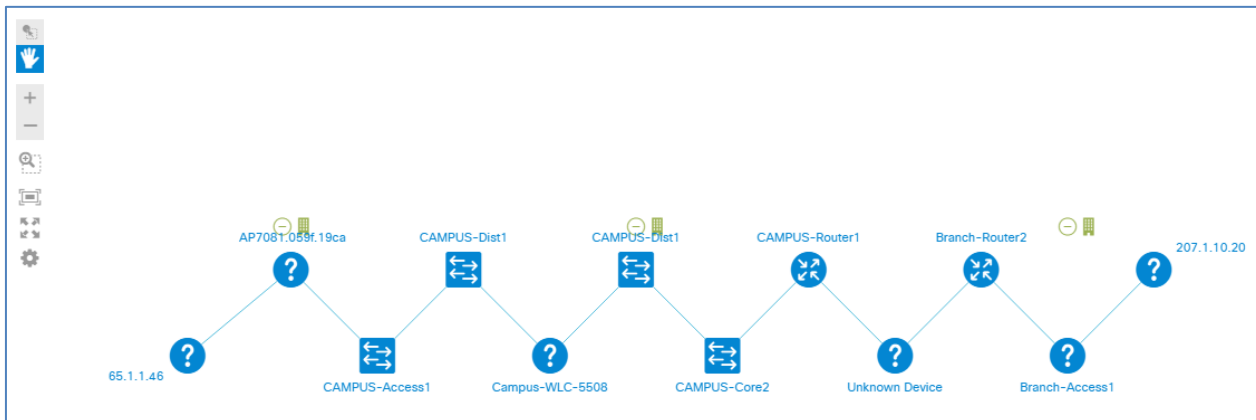
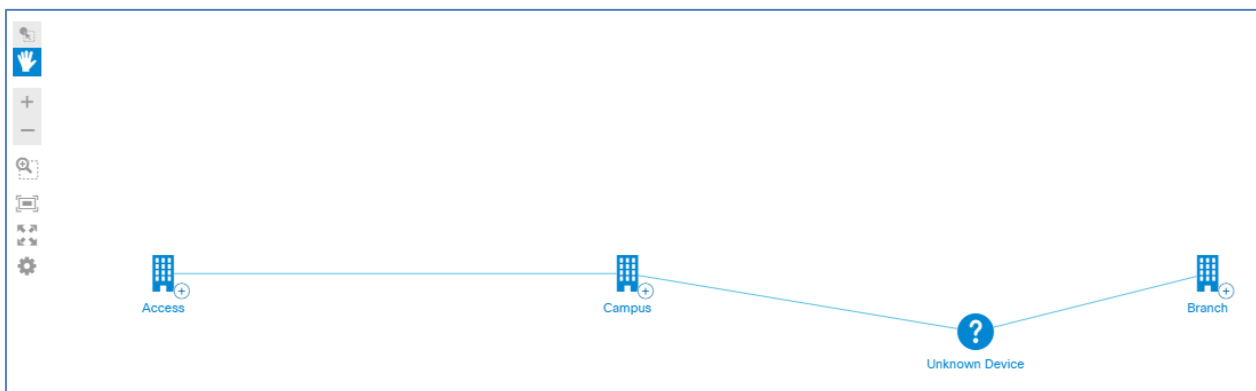


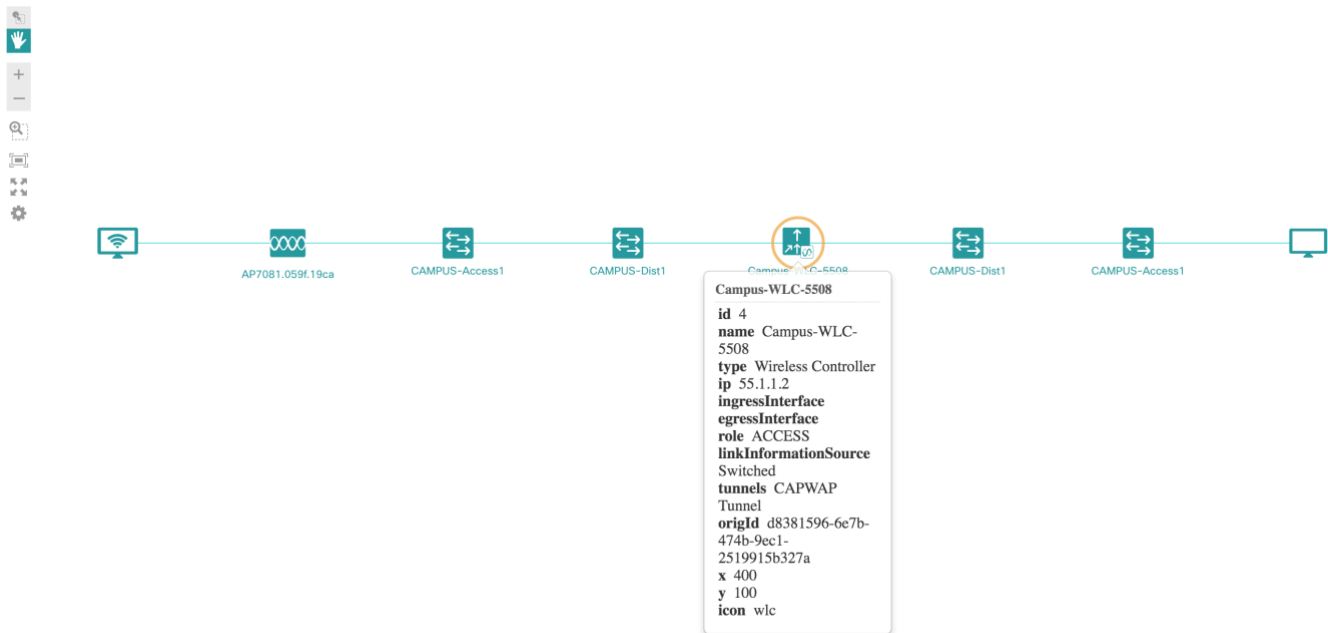
Figure 13. Topology



- Click on a node. You will see a tooltip, which by default is propagated with some generic information. Try enhancing the tooltip by adding node information from the APIC-EM path trace data.

Your tooltip should now look similar to the image below:

Figure 14. Tooltip



Tropo Scripting APIs: Text-to-Speech and IVR

Voice calls and text messaging are effective ways to communicate with field force and remote staff. Tropo makes it simple to automate communications, connecting your code to the phone network with both voice and messaging. You use the web technologies you already know and Tropo's powerful cloud API to bring real-time communications to your apps and DevOps workflows.

Objectives

Here you will build an interactive voice response (IVR) and text-to-speech (TTS) / speech-to-text (STT) script with Tropo.

Hints

IVR scripts dealing with incoming calls and hosted within the Tropo cloud are a great way to get started. They work with ad-hoc phone numbers and using the Tropo Scripting API does not require you to have a web service up and running.

Once in place, it is easy to build and expand from an IVR to outgoing or messaging interactions. When a more advanced implementation needs to interact with your application's web service, the Tropo WebAPI or REST APIs can be used.

Steps

The developer site has prebuilt quickstart code samples to assemble a prototype:

1. Login to **tropo.com**, then navigate to **Docs** and explore the **Tropo Scripting API Quickstarts**. For each type of interaction, these quickstarts offer code snippets in several scripting languages. Have a look at the python samples for **Answering Incoming Calls** and for **Asking a Question**.

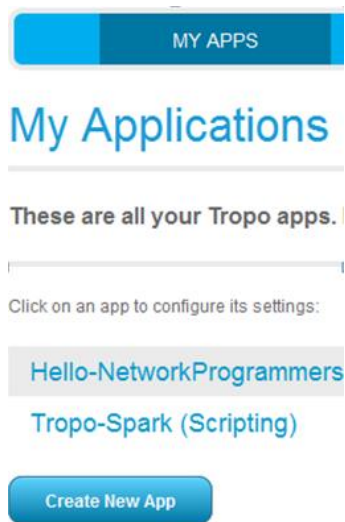
Figure 15. Tropo Scripting API Quickstarts



2. Create your own application and make use of those quickstarts. Open <https://www.tropo.com/applications> in a second browser tab – for example by right clicking on **Applications** in the top menu.
3. This leads you to the area for creating and managing Tropo Applications, Scripting API files, Logs as well as billing and account settings.

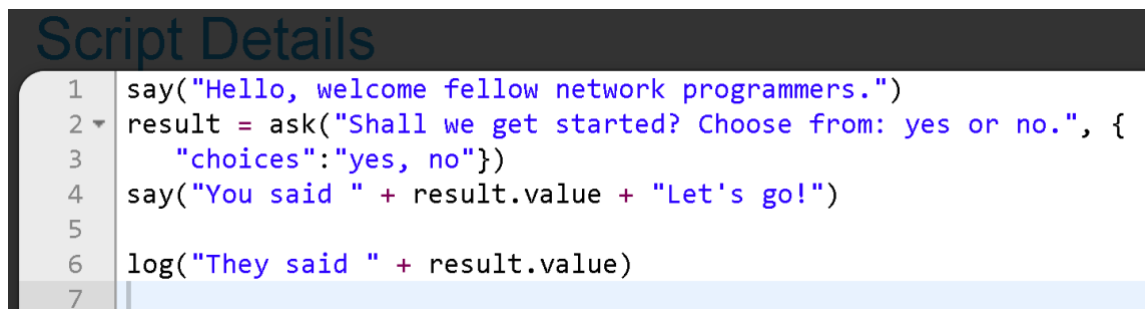
- Click **Create New App** and give it a unique name. Select **Scripting API** as the Type of Application and pick a phone number (availability and regulations of phone numbers varies in different countries – if there is no number available or if it cannot be quickly assigned in your desired location, choose a different one for prototyping).

Figure 16. Create New App



- Finally, add a **Voice Script** to answer an incoming call, ask a question and confirm the response. To do this, refer to the Quickstart examples in your other browser tab. (<https://www.tropo.com/docs/scripting/quickstarts/asking-question>) Your voice script should now look similar to the one below:

Figure 17. Script Details



NOTE: Call in and test your IVR script - your IVR should now behave similar to the one at +41 43 508 13 03.

NOTE: Tropo supports SIP calls – check the bottom of your App details page.

NOTE: After you modify a script, it can take Tropo a few seconds before it becomes available for incoming calls. A nice way to avoid confusion, is to add a version number in the first say statement while prototyping, for example, `say("Version 6. Hello fellow network programmers")`

NOTE: By default, you are interacting with **Vanessa**, a female US English voice persona. Tropo supports speech and speech recognition in many languages – to use your native language if different from English or change the default voice persona to British English check the **International Features**

Scenario 2. Device-Level Programmability and Virtualization

IOS EEM: Change Port Description Based on CDP Neighbor

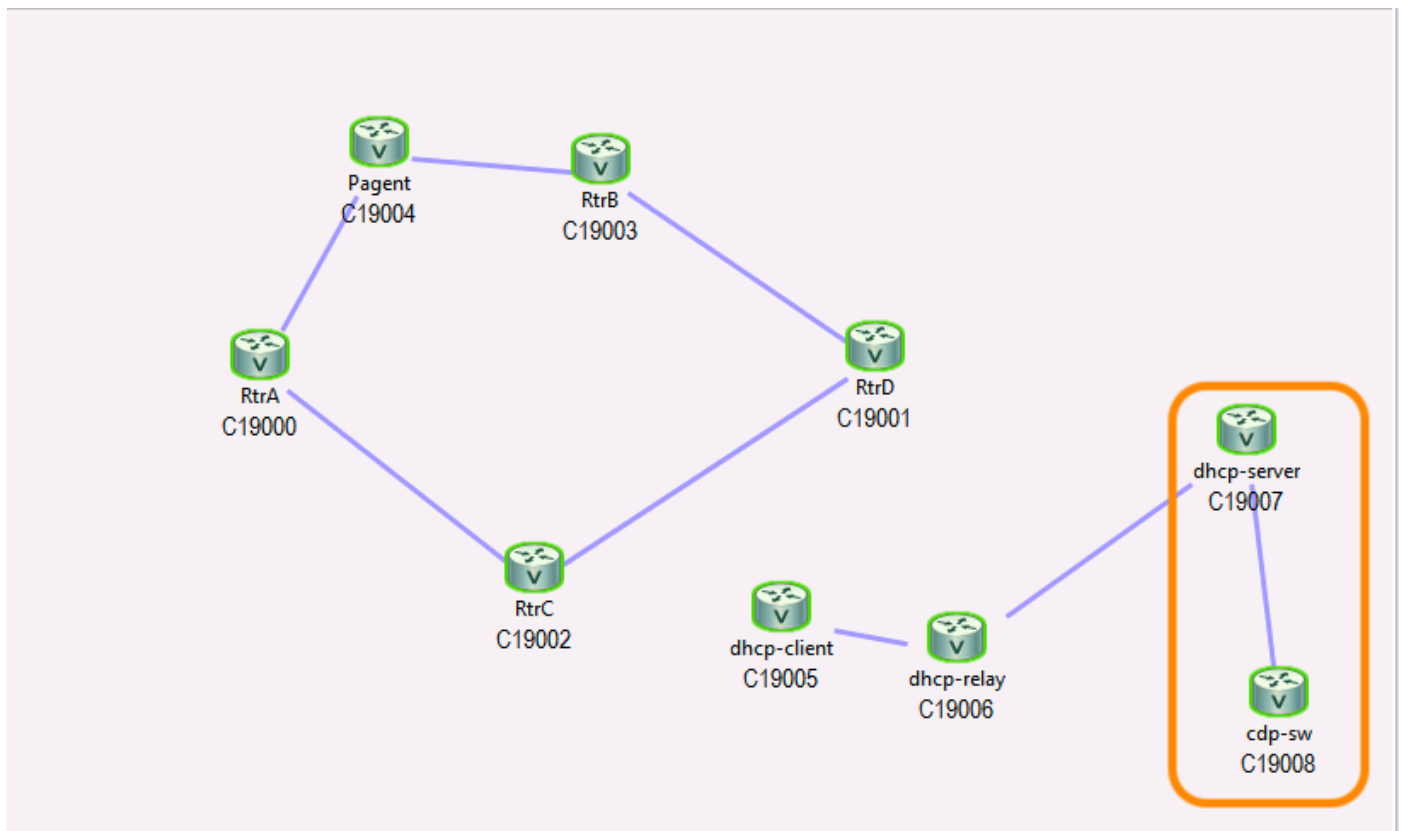
In this task, we will use the Embedded Event Manager within a VIRL topology to change the port description dynamically when a new CDP neighbor is learned. Our EEM applet will run on a VIRL IOSv L2 image, and we will test with different nodes.

Objective

We have to define an EEM applet that will react when a new CDP neighbor is learned on an interface. Since we will be using an L2 device to emulate a switch, we can expect that we will have a high port density. Therefore, we will need to be clever with how we match ports. When a new neighbor is learned, the applet will enter configuration mode for the port in question and add a new description. This description will contain the hostname of the remote device as well as the remote interface. By doing this, port description documentation is up-to-date, and it becomes easier to identify where a node was last connected to the network.

Getting Started

For this task, we will be using a VIRL-based topology. The whole topology is depicted below with the nodes used for this task highlighted.



1. The **cdp-sw** device is the one that we will use for EEM configuration.
2. Open PuTTY on your lab computer and launch the **cdp-sw** session. To do this on another computer telnet to 198.18.1.106 port 19008. This will connect you to the console port of the switch.

3. Enter enable mode using the password "cisco" (without quotes).

Hints

If you prefer not to simply copy and paste the solution, here are some hints to get you started. The Solution section provides one applet that will accomplish this task.

- Configure an EEM applet that uses the Neighbor Discovery event detector. This policy should listen for CDP *add* events.
- In order to make sure that all relevant ports are matched, use a regular expression to match on all GigabitEthernet0/* ports.
- When the policy runs, it should go to the port that triggered the event (you can use "show event manager detector neighbor detail" to show the built-in event detector variables) and configure a port description that includes the hostname of the remote device as well as the remote interface name.
- To test this, do a shut/no shut on interface GigabitEthernet0/1. You should see the description change when the **dhcp-server** router is relearned.

Solution

1. Enter configuration mode on **cdp-sw** and type in the following EEM applet.

```
event manager applet update-port-description
 event neighbor-discovery interface regexp GigabitEthernet0/* cdp add
 action 1.0 cli command "enable"
 action 2.0 cli command "config t"
 action 3.0 cli command "interface $_nd_local_intf_name"
 action 4.0 cli command "description $_nd_cdp_entry_name:$_nd_port_id"
```

2. **NOTE:** After configuring this applet, exit to EXEC mode. If you do not do this, the applet will not register itself!
3. Enter configuration mode for interface GigabitEthernet0/1. Execute shut/no shut under this interface. When the dhcp-server router is relearned, execute "show run int gi0/1" and you should see:

```
interface GigabitEthernet0/1
 description dhcp-server.ciscolive.local:GigabitEthernet0/2
 media-type rj45
 negotiation auto
```

NOTE: For a more interesting solution, check out <https://supportforums.cisco.com/document/100791/automatically-set-port-descriptions>. There are additional solutions there that can apply additional dynamic configuration elements.

IOS EEM: Assign Static IP If DHCP Fails

In this task, use an Embedded Event Manager applet to automatically assign a static IP and a static route to a device if DHCP fails to assign an address within a specific amount of time. Our policy will listen for a syslog message that indicates our DHCP interface has come up, and then wait for a DHCP address to be assigned.

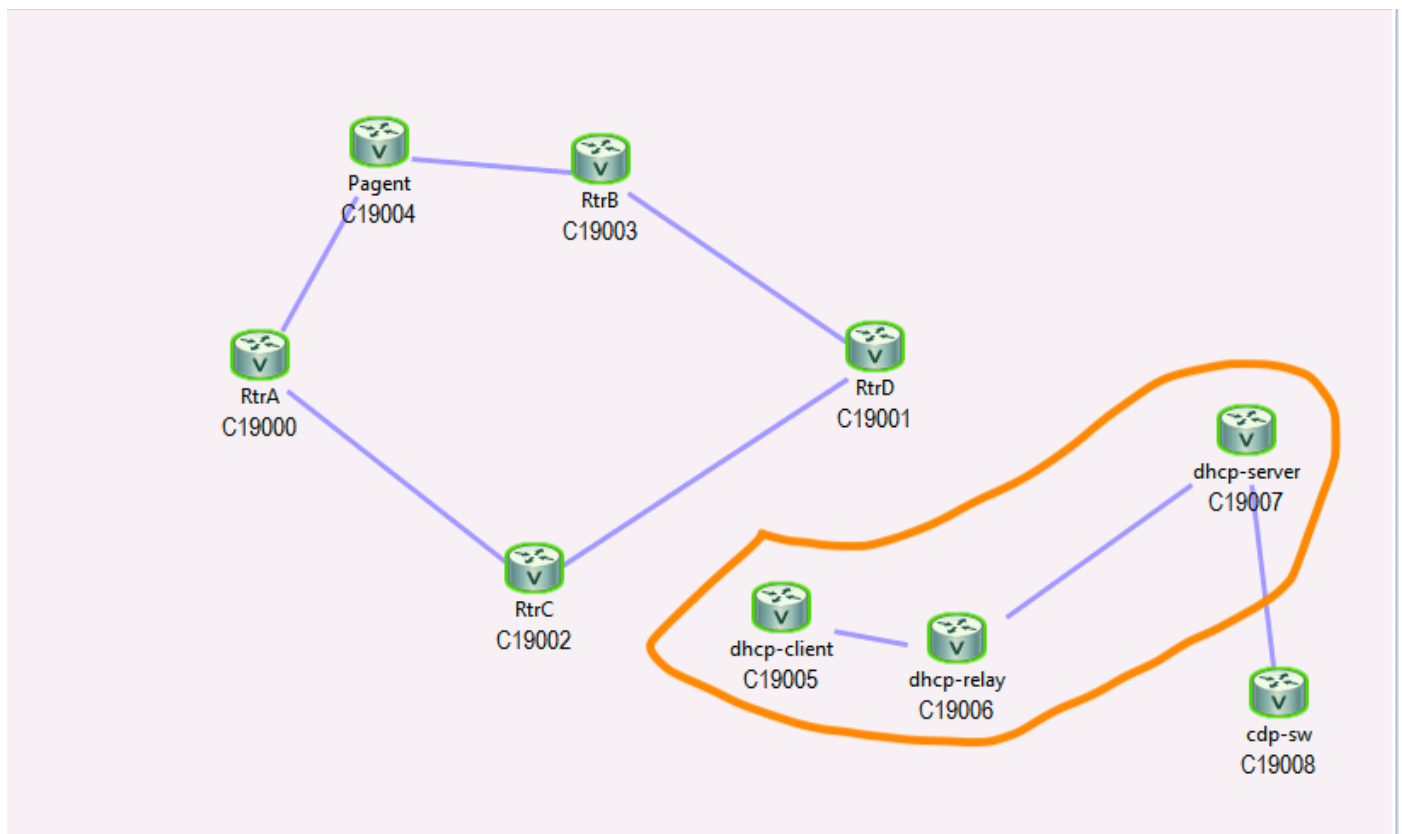
Objective

For this task:

- Detect when the DHCP interfaces has come up.
- Establish a timer to wait for the DHCP server to assign an address.
- If we do not get an address, reconfigure the device.
- If we do get an address within the allotted time, we have to make sure not to reconfigure the device. By using EEM to do this, we can ensure that the device is always reachable even if there is an issue with the DHCP server.

Getting Started

For this task, we will be using a VIRL-based topology. The whole topology is depicted below with the nodes used for this task highlighted.



1. Open PuTTY on your lab computer and open the **dhcp-client** session. To do this on another computer, telnet to 198.18.1.106 port 19005. This will connect you to the console of the client router.

2. Enter enable mode using the password "cisco" (without quotes).

Hints

If you prefer not to simply copy and paste the solution, here are some hints to get you started. The Solution section includes an applet that will accomplish this task.

- Use EEM *nesting* (refer to the slides about IPSLA-based failover). Ultimately, we will need three policies to accomplish the solution. One will listen for the syslog that indicates the DHCP interface has come up, one will be a timer that counts down a specific number of seconds, and one will be a syslog policy that waits for an address to be assigned.
- For purposes of this task, use a **60 second** timer to wait for a DHCP address to be assigned.
- The interface we are using for DHCP is GigabitEthernet0/1. It is already configured for DHCP. By default, when this interface comes up it will get an address from **dhcp-server**. However, if that fails, assign GigabitEthernet0/1 the address **10.0.0.254**. Create a static default route pointing to **10.0.0.1**.
- Once your EEM configuration is done, you can test it by executing shut/no shut on GigabitEthernet0/1. You should see that the interface gets an address via DHCP, and no static IP or static route is configured. Kinda boring.
- Then, connect to the console of **dhcp-relay** by opening the **dhcp-relay** session PuTTY on the lab computer. To do this on another computer, telnet to 198.18.1.106 port 19006. Enter enable mode using the password "cisco" (without quotes). Under interface GigabitEthernet0/1, configure "no ip helper-address 10.0.1.1". Back on **dhcp-client** execute another shut/no shut under interface GigabitEthernet0/1. After 60 seconds, you should see GigabitEthernet0/1 gets a static IP of 10.0.0.254, and a static default route pointing to 10.0.0.1 has been configured.

Solution

1. Enter configuration mode on **dhcp-client** and type in the following EEM applet.

```
event manager environment q "
event manager applet dhcp-intf-up
event syslog pattern "LINEPROTO.*GigabitEthernet0/1,.*changed state to up"
action 001 cli command "enable"
action 002 cli command "config t"
action 003 cli command "event manager applet dhcp-intf-timer"
action 004 cli command "event timer countdown time 60"
action 005 cli command "action 1.0 cli command enable"
action 006 cli command "action 2.0 cli command $q config t$q"
action 007 cli command "action 3.0 cli command $q interface Gi0/1$q"
action 008 cli command "action 4.0 cli command $q ip address 10.0.0.254 255.255.255.0$q"
action 009 cli command "action 5.0 cli command $q ip route 0.0.0.0 0.0.0.0 10.0.0.1$q"
action 010 cli command "action 6.0 cli command $q no event manager applet dhcp-intf-timer-disable$q"
action 011 cli command "action 7.0 cli command $q no event manager applet dhcp-intf-timer$q"
action 012 cli command "event manager applet dhcp-intf-timer-disable"
action 013 cli command "event syslog pattern $q%DHCP-6-ADDRESS_ASSIGN:.*GigabitEthernet0/1 $q"
action 014 cli command "action 1.0 cli command enable"
action 015 cli command "action 2.0 cli command $q config t$q"
action 016 cli command "action 3.0 cli command $q no event manager applet dhcp-intf-timer$q"
action 017 cli command "action 4.0 cli command $q no event manager applet dhcp-intf-timer-disable$q"
action 018 cli command "end"
```

NOTE: After configuring this applet, exit to EXEC mode. If you do not do this, the applet will not register itself!

Under interface GigabitEthernet0/1, execute a shut/no shut. Once the interface comes up, you should see a DHCP-6-ADDRESS_ASSIGN syslog appear within a few seconds. If you execute "show ip interface brief" you should see that GigabitEthernet0/1 has an IP assigned via DHCP.

```
dhcp-client#show ip int brief
```

Interface	IP-Address	OK?	Method	Status	Protocol
GigabitEthernet0/0	10.255.0.46	YES	NVRAM	up	up
GigabitEthernet0/1	10.0.0.2	YES	DHCP	up	up
Loopback0	192.168.0.1	YES	NVRAM	up	up

Connect to the console of **dhcp-relay** by opening the **dhcp-relay** session within PuTTY on the lab computer. To do this on another computer, telnet to 198.18.1.106 port 19006. Enter enable module using the password "cisco" (without quotes). Under interface GigabitEthernet0/1, execute the command "no ip helper-address 10.0.1.1". Back on **dhcp-client**, execute shut/no shut under interface GigabitEthernet0/1 again. Without a working DHCP relay, GigabitEthernet0/1 will not receive an IP address after 60 seconds. At that point, execute the commands "show run int gi0/1" and "show ip route".

```
dhcp-client#show run int gi0/1
interface GigabitEthernet0/1
  description to iosv-2
  ip address 10.0.0.254 255.255.255.0
  duplex auto
  speed auto
  media-type rj45
end
!
dhcp-client#show ip route
Gateway of last resort is 10.0.0.1 to network 0.0.0.0
S* 0.0.0.0/0 [1/0] via 10.0.0.1
    10.0.0.0/8 is variably subnetted, 4 subnets, 3 masks
C    10.0.0.0/24 is directly connected, GigabitEthernet0/1
L    10.0.0.254/32 is directly connected, GigabitEthernet0/1
C    10.255.0.0/16 is directly connected, GigabitEthernet0/0
L    10.255.0.46/32 is directly connected, GigabitEthernet0/0
    192.168.0.0/32 is subnetted, 1 subnets
C    192.168.0.1 is directly connected, Loopback0
```

NOTE: You can see how nesting is extremely useful to have very tight control over device functions so that you can achieve desired results under specific time frames and specific conditions. And in this case, you can maintain connectivity to DHCP-assigned devices when there is a problem with the DHCP server.

IOS EEM: Force a Failover Based on IPSLA-Reported Latency

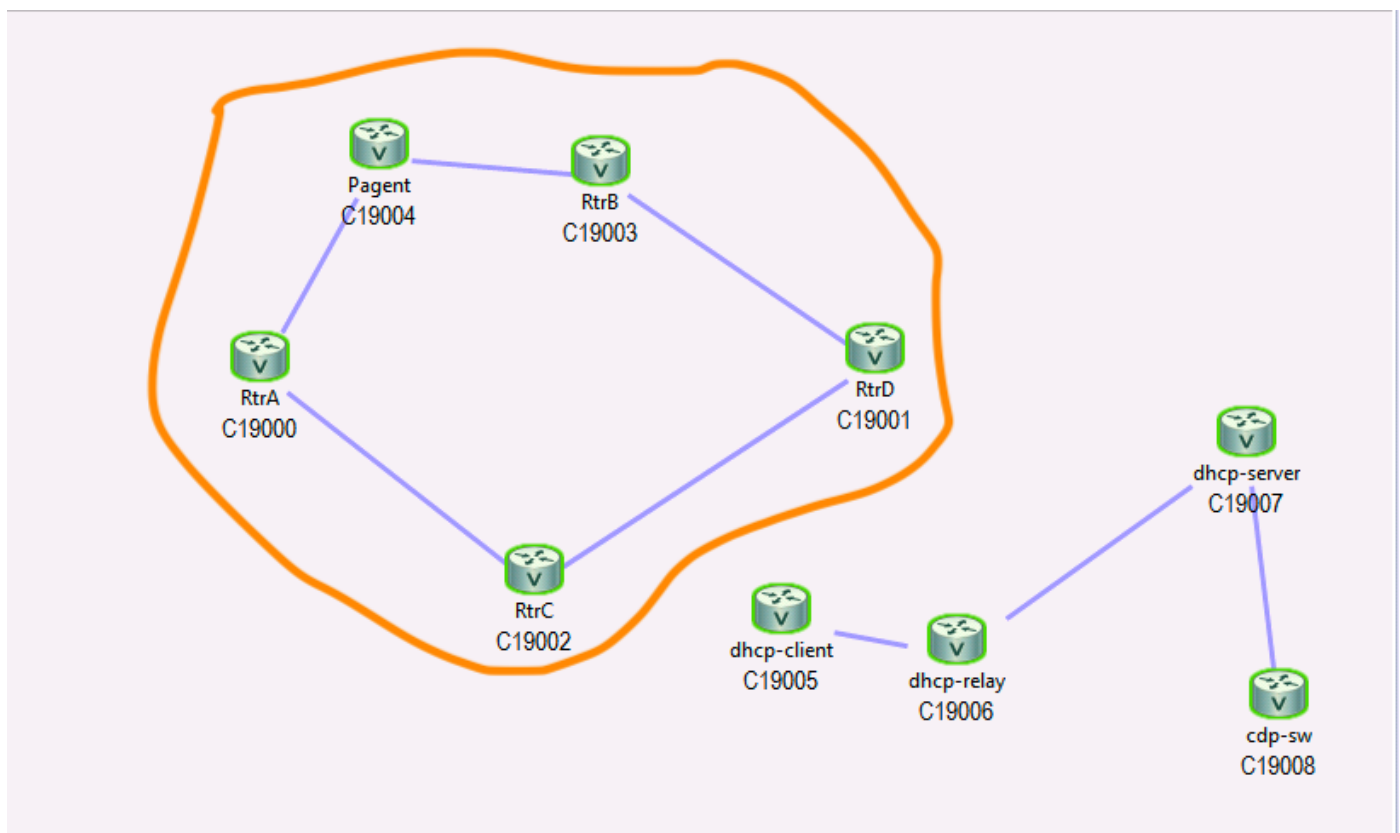
In this task, we will use EEM applets in conjunction with IPSLA collectors to test for latency, and then failover to a redundant link if latency rises to unacceptable levels. When latency has returned to acceptable values for a specified number of polling intervals, traffic will once again be routed over the primary link.

Objective

Failover examples are typically done when total reachability is lost across a given link. This example differs in that the primary link may still be viable to pass traffic, but latency across the link has become unacceptable. For this task, we will use an IPSLA ICMP collector to test the end-to-end latency across the primary link. When the latency threshold is violated, EEM will initiate a failover to our redundant link. When IPSLA reports acceptable latency on the primary link, EEM will continue to monitor to make sure the latency values are stable. Only then will it trigger a fail back to the primary link.

Getting Started

For this task, we will be using a VIRL-based topology. The whole topology is depicted below with the nodes used for this task highlighted.



In order to accomplish this task:

1. We will focus on **RtrA** and **Pagent** in this exercise. The Pagent software will act as a "bump on the wire" that will inject delay into our primary link.

Within PuTTY on your lab computer launch the **RtrA** session. On another computer, telnet to 198.18.1.106 port 19000. This will open a connection to the router's console. Enter enable mode using the password "cisco" (without quotes).

Also open a console connection to **Pagent**. To do this, launch the Pagent session from within PuTTY on your lab computer. From another computer, telnet to 198.18.1.106 port 19004. Use the password "cisco" (without quotes) to enter enable mode.

In EXEC mode on **Pagent**, enter the following commands. You can just paste the list below.

```
pmod
gi0/1
active
gi0/2
active
```

If successful, you should see the following on **Pagent's** console.

```
***Interfaces GigabitEthernet0/1 and GigabitEthernet0/2
are now internally connected by Passthru Modify.

Putting GigabitEthernet0/1 interface into promiscuous mode.
Putting GigabitEthernet0/2 interface into promiscuous mode.
```

Hints

If you prefer not to simply copy and paste the solution, here are some hints to get you started. The Solution section provides the EEM policies that will accomplish this task.

- The IPSLA collector has already been configured on **RtrA** to run every 10 seconds. The thresholds for Round-Trip Time (RTT) have been set to 700 ms for rising and 10 ms for falling. You can see the current results of this collector by running the `show ip sla stat 1` command on **RtrA**. If you do not see any successes, but you do see Number of failures incrementing, check the **Pagent** config as specified in the Getting Started section. Chances are the PMOD config has not been enabled.
- For this task, we will need an EEM applet that checks the result of IPSLA collector 1 (hint: use the ipsla Event Detector) to see if an *rtt* event has occurred. Note: this EEM policy will fire when the event occurs and when it clears. When the event has occurred (meaning the RTT is over 700 ms), the ipsla ED will set the value of the variable `$_ipsla_condition` to Occurred (without quotes). Therefore, add a condition to your applet that checks to see if the threshold has been violated or not.
- If the threshold has been violated, remove the static default route to 10.0.0.17 and add one to 10.0.0.5. By changing the default route, traffic to **RtrD** will flow through **RtrC** instead of **RtrB**. When RTT returns to an acceptable level (i.e., it falls below 10 ms), your applet's else block will create a *nested* EEM watchdog timer applet that will execute `show ip sla stat 1` and count the number of times the output of Latest operation return code is OK. If it is okay for six consecutive checks, then remove the static default route to 10.0.0.5 and re-add the route to 10.0.0.17. Note: be sure that in your Occurred block you take care of removing this nested timer applet to ensure that the timer is only counting consecutive below-threshold instances. You can use either EEM contexts or a counter to track the number of consecutive below-threshold instances.
- Once you have your code written, scroll down to the Testing section to see how you can use **Pagent** to test your applets.

Solution

1. To create the main EEM config that will react to both threshold violations and clearances, enter the following into the config of **RtrA**.

```
event manager environment q "
event manager environment cr $_cli_result
event manager environment rr $_regexp_result
event manager applet ipsla-failover
  event ipsla operation-id 1 reaction-type rtt
  action 001 cli command "enable"
  action 002 if $_ipsla_condition eq "Occurred"
  action 003 cli command "config t"
  action 004 cli command "no ip route 0.0.0.0 0.0.0.0 10.0.0.17"
  action 005 cli command "ip route 0.0.0.0 0.0.0.0 10.0.0.5"
  action 006 cli command "no event manager applet ipsla-failback-timer"
  action 007 cli command "end"
  action 008 counter name "IPSLACNT" op set value 0
  action 009 syslog msg "Failing over as latency is now at $_ipsla_measured_threshold_value"
  action 010 else
  action 011 cli command "config t"
  action 012 cli command "event manager applet ipsla-failback-timer"
  action 013 cli command "event timer watchdog time 10"
  action 014 cli command "action 001 cli command enable"
  action 015 cli command "action 002 cli command $q show ip sla stat 1 | inc Latest operation return
code$q"
  action 016 cli command "action 003 regexp $q code: OK$q $cr"
  action 017 cli command "action 004 if $rr eq 1"
  action 018 cli command "action 005 counter op inc name IPSLACNT value 1"
  action 019 cli command "action 006 end"
```

```

action 020 cli command "end"
action 021 end
event manager applet ipsla-failback-counter
event counter name IPSLACNT entry-val 6 entry-op ge exit-val 0 exit-op eq
action 001 cli command "enable"
action 002 cli command "config t"
action 003 cli command "no event manager applet ipsla-failback-timer"
action 004 cli command "no ip route 0.0.0.0 0.0.0.0 10.0.0.5"
action 005 cli command "ip route 0.0.0.0 0.0.0.0 10.0.0.17"
action 006 cli command "end"
action 007 counter name "IPSLACNT" op set value 0
action 008 syslog msg "Failing back as latency is now at acceptable levels for 60 seconds"

```

NOTE: After configuring this applet, exit to EXEC mode. If you do not do this, the applet will not register itself!

The environment variables **q**, **cr**, and **rr** stand for the double-quote character, the variable `$_cli_result`, and the variable `$_regex_result` respectively. This is needed to support the nesting of applets.

The logic behind these applets is that the first applet will react to the IPSLA collector either violating the threshold (i.e., when `$_ipsla_condition` equals Occurred) or the threshold has cleared (the value of `$_ipsla_condition` will be Cleared in this case). In the violation case this applet will switch the default route so that traffic will flow over the redundant link. However, when the threshold has cleared, this applet will install a nested timer applet to check that the latency has been acceptable for at least the past six consecutive polling cycles (60 seconds in our case). Each time it sees a good threshold value, it will increase an internal EEM counter by one. When the timer hits six the third applet (**ipsla-failback-counter**) will take care of failing back to the primary link.

Scroll down to the Testing section to view the outcome.

Testing

1. Once you have your applets in place, test that they do what they should by going onto the Pagent console and pasting the following commands at EXEC mode.

```

pmod
gi0/1
delay on
delay duration 500 to 2000
delay until-stopped
start

```

The **Pagent** prompt should change to the following to let you know the delay injection is happening.

```
Pagent (PMOD:ON:Gi0/1:0/0) #
```

NOTE: There is an ON in the prompt.

Give it about 20 seconds, and you should see the following on **RtrA's** console (assuming you've used the solution above).

```

%RTT-3-IPSLATHRESHOLD: IP SLAs(1): Threshold exceeded for rtt
%SYS-5-CONFIG_I: Configured from console by on vty0 (EEM:ipsla-failover)
%HA_EM-6-LOG: ipsla-failover: Failing over as latency is now at 1062

```

At this point, check the output of "show ip route" and you should see the default route is now pointing to 10.0.0.5. If you ping 192.168.0.1 from the command line, it should still give you a speedy result. That is because the new route has lower latency. The IPSLA traffic is still going to 10.0.0.17, though. This is done using a route-map to drive policy-based routing. Test using the command "ping ip 192.168.0.1 source 10.0.0.18" and you should see a very slow ping.

To test the fallback, type, "stop" at the **Pagent** prompt. The prompt should then change to the following.

```
Pagent (PMOD:OFF:Gi0/1:0/0) #
```

The OFF indicates the latency injection has stopped.

After about 60 seconds, you should see the following on the console (assuming you've used the solution above).

```
%SYS-5-CONFIG_I: Configured from console by on vty1 (EEM:ipsla-failback-counter)
%HA_EM-6-LOG: ipsla-failback-counter: Failing back as latency is now at acceptable levels for 60 seconds
```

NOTE: You can see how EEM can be used to integrate with other IOS subsystems in order to deliver solutions such as customized failover.

IOS-XE Guestshell: Enable Guestshell

IOS-XE guestshell is an application hosting environment offered via a container that runs in parallel with the IOS-XE image. Using guestshell, you can install your own applications and scripts to help extend network automations and enable new use cases for your network platform.

Objectives

The objective is to enable the IOS-XE guestshell container on the **csr1k-nwp1**, and then enable guestshell to communicate with the internet. The `dohost` command will also be used as a means to run commands on the host IOS-XE instance.

Getting Started

SSH to **csr1k-nwp1** from the Windows workstation or from your own laptop once you have used AnyConnect to connect to the dCloud environment. Use the documented credentials to log into the router.

Hints

If you do not wish to just copy and paste the solution, here are some hints to get you going. If you want to just run through the solution, scroll down to the **Solution** section below.

Guestshell is enabled as part of the **iox** sub-system. You will need to enable **iox** before you can enable the guestshell. In order to run IOS-XE commands from within guestshell, you will need to enable the HTTP server on the router. Finally, in order for guestshell to communicate with the internet, you will need to enable NAT. Once you have guestshell working properly, confirm you can ping the Windows host in your topology, and use the built-in `dohost` command to run a command on the IOS-XE instance from within guestshell.

Solution

1. Enter configuration mode, and configure the following command.

```
iox
```

To verify the status of IOX, use the following command:

```
csr1k-nwp1#show iox
Virtual Service Global State and Virtualization Limits:

Infrastructure version : 1.7
Total virtual services installed : 1
Total virtual services activated : 1

Machine types supported   : LXC
Machine types disabled    : KVM

Maximum VCPUs per virtual service : 0
Resource virtualization limits:
Name                       Quota      Committed  Available
-----
system CPU (%)             75         5          70
memory (MB)                1024       512        512
```

```
bootflash (MB)          20000          756          2908
```

```
IOx Infrastructure Summary:
```

```
-----
```

```
IOx service (CAF)      : Running
IOx service (HA)       : Not Running
IOx service (IOxman)   : Running
LibvirtD               : Running
```

NOTE: it will take a couple of minutes for IOX to fully initialize, but you can continue on with steps 1 – 4 in the meantime.

2. Configure a VirtualPortGroup0 interface to act as the management gateway for guestshell:

```
interface VirtualPortGroup0
 ip address 10.1.1.1 255.255.255.0
```

3. Enable the HTTP server on the router:

```
ip http server
```

4. Configure NAT so that guestshell can communicate with outside networks:

```
interface VirtualPortGroup0
 ip nat inside
!
interface GigabitEthernet1
 ip nat outside
!
ip access-list standard GS_NAT
 permit 10.1.1.0 0.0.0.255
!
ip nat inside source list GS_NAT interface GigabitEthernet1 overload
```

5. Configure Guestshell and bind it to the interface **VirtualPortGroup 0**:

```
app-hosting appid guestshell
 vnic gateway1 virtualportgroup 0 guest-interface 0 guest-ipaddress 10.1.1.2 netmask 255.255.255.0
 gateway 10.1.1.1 name-server 8.8.8.8
```

6. Exit configuration mode and enable the guestshell. This command will take a minute or so to complete.

```
csr1k-nwpl#guestshell enable
Interface will be selected if configured in app-hosting
Please wait for completion
guestshell installed successfully
Current state is: DEPLOYED
guestshell activated successfully
Current state is: ACTIVATED
guestshell started successfully
Current state is: RUNNING
Guestshell enabled successfully
```

7. To verify if the Guestshell is running, use the following command:

```
csr1k-nwpl#show app-hosting detail
State          : RUNNING
Author         : Cisco Systems
Application
  Type         : lxc
  App id       : guestshell
  Name         : GuestShell
  Version      : 2.3(0.1)
Activated profile name : custom
```

```
Description      : Cisco Systems Guest Shell XE for x86
Resource reservation
Memory           : 512 MB
Disk              : 1 MB
CPU               : 799 units
VCPUs             : 1
...
<omitted for brevity>
```

8. Once guestshell has been enabled, start the guestshell:

```
csr1k-nwp1#guestshell
[guestshell@guestshell ~]$
```

9. Ping the Windows workstation from guestshell:

```
[guestshell@guestshell ~]$ ping 198.18.133.36
```

10. Use the dohost command to run “show version” from the IOS-XE instance:

```
[guestshell@guestshell ~]$ dohost "show version"
```

IOS-XE Guestshell: Python interpreter

The IOS-XE guestshell environment includes Python 2.7 interpreter by default. This Python environment allows one to run EXEC and configuration commands from the host IOS-XE instance. This is also a fully-functional Python 2.7 environment, so one can also install Python modules and write scripts to run directly on the device.

Objectives

The objective of this task is to start the Python interpreter within guestshell and perform some CLI functions using the built-in `cli` module.

Getting Started

SSH to **csr1k-nwp1** from the Windows workstation or from your own laptop once you have used AnyConnect to connect to the dCloud environment. Use the documented credentials to log into the router. **Make sure you have performed "[IOS-XE Guestshell: Enable Guestshell](#)" first.** Start the guestshell Python by running the following command:

```
csr1k-nwp1#guestshell run python
```

You should now be at a Python prompt that looks like the following:

```
Python 2.7.5 (default, Jun 17 2014, 18:11:42)
[GCC 4.8.2 20140120 (Red Hat 4.8.2-16)] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>>
```

Hints

If you do not wish to just copy and paste the solution, here are some hints to get you going. If you want to just run through the solution, scroll down to the **Solution** section below.

Import the `cli` module and get the built-in help on it. Note the `clip()` function. Use this function to run the EXEC commands “show ip interface brief” and “show app-hosting list”. Use the `configure()` function to configure a loopback interface with a description of “Configured By Python”.

Solution

1. From the Python `>>>` prompt, type the following:

```
import cli
```

2. Python provides built-in help on modules by running the `help()` and `dir()` commands on modules. Test each one to get help on the `cli` module using the following commands. In the `help()` output, scroll down to the bottom to get a Unix manual-style page for the `cli` module. Note the functions “cli”, “clip”, and “configure”. These can be used to get CLI output into a variable, print CLI output, and perform configuration tasks respectively:

```
dir(cli)
```



```
help(cli)
```

3. Print the output of the “show ip interface brief” and “show app-hosting list” commands using one `cli.clip()` call by separating the commands with a ‘;’:

```
cli.clip('show ip interface brief ; show app-hosting list')
```

4. Configure a loopback interface on the router and give it a description of “Configured By Python” using the `configure()` function. Note that the individual lines are separated by a newline, but the whole text block is within a Python multi-line string as indicated by the three single-quote characters:

```
cli.configure('''interface loopback1  
description Configured By Python''')
```

5. Verify that the interface was configured and has the correct description. The output should look like the following:

```
cli.clip('show run int loopback1')  
interface Loopback1  
description Configured By Python  
no ip address  
end
```

NOTE: `cli.clip` function prints the command output to the standard output but returns nothing. In contrast, `cli.cli` function returns the output from the command but does not print anything. This makes `cli` function perfect for interaction with the command output programmatically, while `clip` function may be helpful for human consumption

IOS-XE Guestshell: Prepare Python environment

The IOS-XE guestshell Python environment comes with `yum`, CentOS package manager and `pip`, the Python package management system. Using `yum` command, it is possible to install container wide packages. Using the `pip` command, one can install various Python modules needed to create very robust on-box automation scripts.

Objectives

The objective of this task is to prepare Python environment by installing several modules and packages. It is necessary to install **git** and **nano** applications as well as **libffi-devel** and **python-devel** libraries using **yum**. Additionally, upgrade of **pip** and **setuptools** Python packages using **pip** is needed. This will ensure that installation of manually created packages is working as expected. This module will be used in all subsequent guestshell tasks.

Getting Started

SSH to **csr1k-nwp1** from the Windows workstation or from your own laptop once you have used AnyConnect to connect to the dCloud environment. Use the documented credentials to log into the router. **Make sure you have performed "[IOS-XE Guestshell: Enable Guestshell](#)" first.** Enter the guestshell by running the following command:

```
csr1k-nwp1#guestshell
```

Hints

If you do not wish to just copy and paste the solution, here are some hints to get you going. If you want to just run through the solution, scroll down to the **Solution** section below.

For this task, it is necessary to run commands as root (make sure `sudo` is used). It is advised to update `yum` database first, before running `yum install`. In order to upgrade Python packages use `pip install -U` command

Solution

1. From the guestshell prompt, use the following command to update yum database:

```
sudo yum update
```

2. Install **git** **nano** **libffi-devel** and **python-devel** packages using the following command:

```
sudo yum install -y git nano libffi-devel python-devel
```

3. Now we need to upgrade **pip** and **setuptools**.

```
sudo pip install -U pip setuptools
```

NOTE: **setuptools** upgrade will fail with **OSError**. This is expected and the package has been actually upgraded successfully.

4. Verify that **setuptools** and **pip** have been upgraded. pip version must be `>=9.0.1` and setuptools version `>=38.4.0`:

```
[guestshell@guestshell ~]$ pip list
DEPRECATION: The default format will switch to columns in the future. You can use --
format=(legacy|columns) (or define a format=(legacy|columns) in your pip.conf under the [list] section)
to disable this warning.
iniparse (0.4)
pip (9.0.1)
```

```
pycurl (7.19.0)
pygpme (0.3)
pyliblzma (0.5.3)
pyxattr (0.5.1)
setuptools (38.4.0)
urlgrabber (3.10)
yum-metadata-parser (1.1.4)
```

5. Verify git version, it should be $\geq 1.8.3.1$:

```
[guestshell@guestshell ~]$ git --version
git version 1.8.3.1
```

6. Create a sample Python script with nano:

```
[guestshell@guestshell ~]$ nano hello_world.py
```

Insert the following code:

```
print('Hello World')
BEST_PROGRAMMING_LANGUAGE = 'Python'
print('{} all the things!'.format(BEST_PROGRAMMING_LANGUAGE))
```

Save the script using CTRL-O and exit using CTRL-X

7. Exit to IOS-XE prompt and run the script using the following command:

```
csr1k-nwp1#guestshell run python hello_world.py
Hello World
Python all the things!
```

IOS-XE Guestshell: Python packaging

It is possible to organize Python code into a single package which can be distributed online (via git or PyPi server) or offline using "wheel" (.whl). This allows to write the custom logic in the code, bundle it together and distribute to all devices in an easy way.

Objectives

The objective of this task is to install a Python library from git repository which solves 3 subsequent guestshell tasks.

Getting Started

SSH to **csr1k-nwp1** from the Windows workstation or from your own laptop once you have used AnyConnect to connect to the dCloud environment. Use the documented credentials to log into the router. **Make sure you have performed "[IOS-XE Guestshell: Prepare Python environment](#)" first.** Start the guestshell by running the following command:

```
csr1k-nwp1#guestshell
```

Steps

1. Install the library using the following command:

```
sudo -E pip install "git+https://github.com/xorrkaz/enterprise-network-programmability.git#egg=dna-scripts&subdirectory=tasks/guestshell"
```

2. Verify it has been installed:

```
[guestshell@guestshell ~]$ pip freeze
certifi==2018.1.18
chardet==3.0.4
dna-scripts==0.1
idna==2.6
iniparse==0.4
pycurl==7.19.0
pygpgme==0.3
pyliblzma==0.5.3
pyxattr==0.5.1
requests==2.18.4
urlgrabber==3.10
urllib3==1.22
yum-metadata-parser==1.1.4
```

3. Verify that the console script `outputs_collector` has been installed:

```
[guestshell@guestshell ~]$ which outputs_collector
/usr/bin/outputs_collector
```

4. Set environmental variables for Spark: your API token and room ID (room ID was posted to spark in the very first lab task).

```
export SPARK_API_TOKEN=<your-token>
echo 'export SPARK_API_TOKEN='$SPARK_API_TOKEN >> ~/.bashrc
export SPARK_ROOM_ID=<spark-room-id>
echo 'export SPARK_ROOM_ID='$SPARK_ROOM_ID >> ~/.bashrc
```

5. Verify that `dna` library can be imported now from the Python interpreter and a spark message can be sent from the router:

```
[guestshell@guestshell ~]$ python
```

```
Python 2.7.5 (default, Aug  4 2017, 00:39:18)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-16)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import dna
>>> dna.send_spark_message(u'\u2757 These days even your router can send a spark message!')
Spark message has been posted.
```



You 01:49

! These days even your router can send a spark message!

Additional information (optional)

This section covers some details about Python packaging.

To create a Python package, just include `__init__.py` into the directory with python files. It is also advisable to change all package level imports either to absolute or to explicit relative using dot notation.

This creates a Python package, but not a pip-installable package. To make it pip-installable, `setup.py` should be added to the root of your project next to the directory with the package.

Generally speaking, Python package does not include executable scripts that can be called — a package is a number of functions and classes that can be used in another code using import. It will be explained in a moment how to include executable scripts with the package.

Refer to The [Hitchhiker's Guide to Packaging](#) for more details about Python packaging

Below is an example of the directory structure of a Python package:

```
# tree guestshell
guestshell
├── README.md
├── dna
│   ├── __init__.py
│   ├── helper.py
│   └── scripts
│       ├── __init__.py
│       └── outputs_collector.py
└── setup.py
```

- **dna** is the name to use during import in Python
- `__init__.py` in **dna** folder can be empty. In this case it contains the following statement:

```
from .helper import *
```

This allows to access functions in **helper.py** using `dna.<function_name>` notation instead of `dna.helper.<function_name>`

- `__init__.py` in **dna/scripts** folder is empty
- **helper.py** contains functions that you would like to expose as part of your package. For example, functions for: uploading file to FTP, sending a message to Spark, etc.

- Folder **scripts** contains executable console scripts which will be installed together with the package.

In this case **outputs_collector.py** imports functions from **dna** package. This script has command line argument parser and calls different imported helper functions.

- **README.md** file contains longer description of your project in Markdown. Can be left empty.

Let's see now the content of **setup.py**:

```
from setuptools import setup, find_packages

setup(
    name='dna-scripts',
    version='0.1',
    packages=find_packages(),
    license='All rights reserved',
    long_description=open('README.md').read(),
    author='<author name>',
    author_email='<email>',
    entry_points={
        'console_scripts': [
            'outputs_collector = dna.scripts.outputs_collector:main',
        ],
    }
)
```

name reflects how the package will be named in **pip** (you can check installed packages using `pip freeze` command)

Take a look on the variables **entry_points** and **console_scripts**. Here we instruct to create a shell script

`/usr/bin/outputs_collector` which will call the function **main** in the module `dna.scripts.outputs_collector`

To test code locally, we can install our package in editable mode (if any changes are done to the code, no need to reinstall). To do that, go to the directory with **setup.py** file and run:

```
# pip install -e .
```

Once you are happy with your library, there are multiple ways to distribute it:

- 1) Upload the package on PyPi to make it installable using pip via the Internet. If you want to know more, see [this](#) or [this](#) article
- 2) Commit to git repository, either public or private, and use `pip install git+https://<path>` syntax. For example, in this task the following command was used:

```
# pip install "git+https://github.com/xorrkaz/enterprise-network-programmability.git#egg=dna-  
scripts&subdirectory=tasks/guestshell"
```

- 3) Create a package as a single file (called wheel) and distribute it offline. On the machine with the source code in the directory with **setup.py** execute:

```
# python setup.py bdist_wheel
```

This creates a ***.whl** file in **dist/** directory

Transfer the file to the target machine and use the following command to install the package:

```
# pip install <package-name>.whl
```

IOS-XE Guestshell and EEM: Record commands to the database

The IOS-XE guestshell Python environment can be invoked directly from within IOS-XE with a script parameter, and that script will run directly. This also allows to run Python scripts based on EEM events

Objectives

The objective of this task is to log all CLI commands run by a user to an SQLite database on flash. EEM will be used to react to each CLI command as it is run, and Python will be used to log the resulting command and user to an SQLite database.

Getting Started

SSH to **csr1k-nwp1** from the Windows workstation or from your own laptop once you have used AnyConnect to connect to the dCloud environment. **Make sure you have performed ["IOS-XE Guestshell: Python packaging"](#) first.** Use the documented credentials to log into the router.

Hints

If you do not wish to just copy and paste the solution, here are some hints to get you going. If you want to just run through the solution, scroll down to the **Solution** section below.

Create an EEM applet that uses the CLI event detector to react to *all* CLI commands being executed. The applet should be synchronous but it should not prevent the CLI command from running. For its actions, the applet should enable itself, and then invoke a Python script. This Python script is called **record_commands.py**. The script will take a number of arguments: the command that was executed, the user that executed it, the remote host, the privilege level, and the result code. You can execute this Python script directly with the command `guestshell run python record_commands.py $_cli_username $_cli_host $_cli_privilege "$_cli_msg" $_cli_error_code`. Your script should save the commands to the file **/flash/aaa.db**, and this should be an sqlite3 database. From within guestshell, you can use the command `sqlite3 /flash/aaa.db` to view the contents of your database.

Solution

NOTE: The code for this task can be found on the [GitHub](#). It is included as a part of the package installed in ["IOS-XE Guestshell: Python packaging"](#) task

1. Enter IOS-XE configuration mode and configure the following EEM environment variable and applet:

```
event manager environment q "
!
event manager applet log-to-db
  event cli pattern ".*" sync no skip no
  action 1.0 cli command "enable"
  action 2.0 cli command "guestshell run python record_commands.py $_cli_username $_cli_host $_cli_privilege
$_cli_msg$_cli_error_code"
  action 3.0 regexp "ERROR" "$_cli_result"
  action 4.0 if $_regexp_result eq "1"
  action 5.0 syslog msg "$_cli_result"
  action 6.0 end
```

NOTE: The environment variable "q" is there to stand for the nested double-quote character. This is the same that we saw in the DHCP EEM applet example from above. Now that the applet is configured, exit configure mode and run some CLI commands (e.g., "show version", "show ip route", etc.).

2. After you have run a few commands, run the following commands to connect to the local SQLite database file:

```
csr1k-nwp1# guestshell  
[guestshell@guestshell ~]$ sqlite3 /flash/aaa.db
```

3. From the `sqlite>` prompt, run the following SQL query to show the current contents of the database:

```
SELECT * FROM command_history;
```


IOS-XE Guestshell: Outputs collector

Every so often we need to collect a list of outputs and send them to some server or a person. For ages, this task has been done in the following way: logging into the device via SSH or console, executing every command manually, saving the session log to the computer and sending it to the destination. With the introduction of Guestshell and on-box Python we can simplify this task.

Objectives

The objective of this task is to implement a script that collects a list of outputs from the device, stores them on the **bootflash** in a single file, sends them to FTP server. Once it is done, the message is posted to spark, containing the following details:

- requested CLI commands
- name of the generated file on the bootflash
- if FTP operation was successful

The script must accept the following command line parameters:

- **-c, --commands** - list of commands separated by "; ". The list should be limited by double quotes
- **-H, --host** - hostname or IP address of FTP server
- **-u, --username** - username for FTP server
- **-p, --password** - password for FTP server

Example of usage:

```
python outputs_collector.py -c "show clock ; show version ; show memory statistics ; show ip interface brief" -H 198.18.1.107 -u cisco -p cisco
```

Getting Started

SSH to **csr1k-nwp1** from the Windows workstation or from your own laptop once you have used AnyConnect to connect to the dCloud environment. **Make sure you have performed ["IOS-XE Guestshell: Python packaging"](#) first.** Use the documented credentials to log into the router. Start the guestshell by running the following command:

```
csr1k-nwp1#guestshell
```

Hints

If you do not wish to just copy and paste the solution, here are some hints to get you going. If you want to just run through the solution, scroll down to the **Solution** section below.

- For parsing command line arguments **argparse** standard library is very useful
- **cli.cli** looks like a good candidate to collect outputs
- It would make sense to add the timestamp to the file stored on the bootflash. For that, check **datetime** module and the function **strftime**
- To send a file via FTP, use **ftplib** and the function **storbinary**
- To transfer the code from the development machine to the network device, **git** can be used.

- To send the message via spark, API key is required. Because our code is tracked in Git, API key can't be a part of git repository. There are multiple ways to approach this: environmental variables (check **os.environ** function), an additional command line argument or a separate file with the key which is not tracked in git (added to **.gitignore**)
- Because only network devices have **cli** library, the code will crash when run locally. There are several approaches to overcome this. Here is one: create a wrapper around **cli.cli** which verifies if **cli** library is installed (check **pkgutil.find_loader** function). If it isn't, then either return an empty string or read test files containing sample CLI commands outputs.

Solution

NOTE: The code for this task can be found on the [GitHub](#). It is included as a part of the package installed in [IOS-XE Guestshell](#): [Python packaging](#) task

1. Because the package was already installed in the earlier task, the script can be invoked from the guestshell:

```
[guestshell@guestshell ~]$ guestshell run outputs_collector -c "show clock ; show version ; show memory
statistics ; show ip interface brief" -H 198.18.1.107 -u cisco -p cisco

Successfully written file /bootflash/20180112-232514_outputs.txt
Successfully uploaded file outputs/20180112-232514_outputs.txt to FTP 198.18.1.107
Spark message has been posted.
```

Here is what can be observed in Spark:



You Yesterday, 00:25

The file containing outputs `show clock ; show version ; show memory statistics ; show ip interface brief` has been uploaded to FTP `198.18.1.107` and is available here: `outputs/20180112-232514_outputs.txt`

2. Let's verify that the file is indeed present on FTP. Log in via SSH using Putty profile to Ubuntu FTP server and run the following command:

```
cisco@linux-host:~$ ls outputs
20180112-232514_outputs.txt
```

3. Check the content of the file using more command:

```
cisco@linux-host:~$ more outputs/<file-name>.txt
```

IOS-XE Guestshell and EEM: Config Diff To Spark

Because of the richness of available Python modules, one can combine services to build a very dynamic user experience, especially in terms of device notifications. For example, using EEM's syslog event detector, one can react to messages that indicate the config changed, diff the current and previous configs, and then send the results to a Spark room.

Objectives

The objective of this task is to react to configuration change syslog messages, determine if anything in the configuration actually changed, and if it did, post the diffs to a Spark room. This task will make use of EEM and on-box Python scripting.

Getting Started

SSH to **csr1k-nwp1** from the Windows workstation or from your own laptop once you have used AnyConnect to connect to the dCloud environment. **Make sure you have performed ["IOS-XE Guestshell: Python packaging"](#) first.** Use the documented credentials to log into the router.

Hints

If you do not wish to just copy and paste the solution, here are some hints to get you going. If you want to just run through the solution, scroll down to the **Solution** section below.

Create an EEM applet that uses the syslog event detector to react to "SYS-5-CONFIG_I" syslog messages. For its actions, the applet should enable itself, and then invoke a Python script on flash. This Python script is called **config_diff_to_spark.py**. You can execute this Python script directly with the command `guestshell run python config_diff_to_spark.py`. The script will compare the current running configuration with the previous running configuration (**hint**: you can save the current configuration to a separate file on flash). The comparison can be done with the `show archive config diff flash:previous.cfg system:running-config`. The output of this command should be posted to the lab Spark room.

Solution

NOTE: The code for this task can be found on the [GitHub](#). It is included as a part of the package installed in ["IOS-XE Guestshell: Python packaging"](#) task

NOTE: Before executing the steps, disable EEM script from the earlier task using: `no event manager applet log-to-db`

1. Enter IOS-XE configuration mode and configure the following EEM applet:

```
event manager applet config-diff-to-spark
  event syslog pattern SYS-5-CONFIG_I
  action 1.0 cli command "enable"
  action 2.0 cli command "guestshell run python config_diff_to_spark.py"
  action 3.0 regexp "ERROR" "$_cli_result"
  action 4.0 if $_regexp_result eq "1"
  action 5.0 syslog msg "$_cli_result"
  action 6.0 end
```

2. Re-enter configure mode, and configure the following under `interface Loopback1` and exit configure mode:

```
interface Loopback1
  description TEST
```

3. You should see a similar message posted to the Spark room:



You 13:24

Configuration differences between the running config and last backup:

```
1 | !Contextual Config Diffs:
2 | interface Loopback1
3 |   +description TEST
4 | interface Loopback1
5 |   -description TEST123
```

NOTE: Make sure to remove this EEM script once you have finished the task

IOS-XE NETCONF: Configure an ACL

In this task, we will see how we can make programmatic config changes using the NETCONF protocol, IOS-XE *native* YANG-based data models, and the Python **ncclient** package. NETCONF provides a way to pass structured, model-based configuration to the device in XML format and receive back clear results as to whether or not the configuration succeeded.

Objective

The objective of this task is to protect your IOS-XE router with an ACL that will only allow the dCloud Windows workstation (i.e., 198.18.133.36) to ping **csr1k-nwp1**. ICMP traffic from other sources will be denied. All other protocols will be allowed. ACL should be applied on the interface GigabitEthernet1 inbound. But this configuration cannot just be entered using configure mode in the CLI. Instead, the NETCONF protocol will be used via a Python script to deploy the new ACL, apply it to an interface, and verify the configuration was successful.

Getting Started

We will use the IOS-XE router, **csr1k-nwp1** to complete this task. Therefore, you will need:

- Open PuTTY on the lab computer or the dCloud Windows workstation. Select the **csr1k-nwp1** session. Consult the topology diagram for the IP address and credentials for **csr1k-nwp1**.
- Enter configure mode on the **csr1k-nwp1** and type `netconf-yang`. This enables the NETCONF subsystem.

NOTE: You may need to wait for a few minutes before NETCONF is fully initialized.

- To verify use the following command:

```
csr1k-nwp1# show platform software yang-management process
```

NETCONF service is enabled when all subsystems show **Running**

Hints

If you prefer not to simply copy and paste the sample solution, here are some hints to get you started. The **Solution** section provides the Python script that will accomplish this task.

- To complete this task, we will operate with YANG data models: **Cisco-IOS-XE-acl.yang** and **Cisco-IOS-XE-native.yang**. All YANG models can be found on [YANG GitHub](#) repository.
- To explore YANG models there is a number of useful tools: **Yang Explorer**, **pyang** and many others.
- To perform NETCONF operations from Python, use **ncclient** library.
- Verification: **linux-host** should no longer ping or SSH to csr1k-nwp's IP address.
- SSH to csr1k-nwp and verify the ACL has been applied to the running configuration and looks like what you would expect it to be if the configuration had been applied using the CLI.

Solution

A sample solution for this task is provided in `acl.py` within the **tasks/netconf** directory of the [GitHub](#) repository.

NOTE: If you are running this script from a host other than the Windows workstation within dCloud, modify the IP address of 198.18.133.36 within the script to be the address on which the script is running (so you block the right host).

1. Run the script from Windows workstation using the following command from **tasks/netconf** directory:

```
C:\Users\administrator\Documents\GitHub\enterprise-network-programmability\tasks\netconf> python acl.py
Successfully configured ACL on 198.18.133.212
```

2. The CLI-based configuration should look like the following when the ACL is applied:

```
ip access-list extended WIN_ONLY
 permit icmp host 198.18.133.36 host 198.18.133.212
 deny    icmp any host 198.18.133.212
 permit ip any any
!
interface GigabitEthernet1
 ip access-group WIN_ONLY in
```

3. To test, open the **linux-host** session in Putty on the lab computer or dCloud Windows workstation. From the linux-host you should not be able to ping **csr1k-nwp1**.

IOS-XE RESTCONF: Configure an ACL

In this task, we will see how we can make programmatic config changes using the RESTCONF protocol, IOS-XE *native* YANG-based data models with the help of Python requests library. RESTCONF, unlike NETCONF, is a stateless protocol, which interacts with YANG models using REST interface and supports both JSON and XML as a payload.

Objective

The objective of this task is to protect your IOS-XE router with an ACL that will only permit the **linux-host** (i.e., 198.18.1.107) to ping **csr1k-nwp1**. ICMP traffic from other sources will be denied. All other protocols will be allowed. ACL should be applied on the interface GigabitEthernet1 inbound. This configuration change should be performed using **RESTCONF** protocol via Python.

This task is very similar to the previous task [IOS-XE NETCONF: Configure an ACL](#) and is designed to show the differences between RESTCONF and NETCONF

Getting Started

We will use the IOS-XE router, **csr1k-nwp1** to complete this task. Therefore, you will need:

- Open PuTTY on the lab computer or the dCloud Windows workstation. Select the **csr1k-nwp1** session. Consult the topology diagram for the IP address and credentials for **csr1k-nwp1**.
- Enter configure mode on the **csr1k-nwp1** and type the following commands:

```
ip http secure-server
restconf
```

This enables the RESTCONF subsystem.

NOTE: You may need to wait for a few minutes before RESTCONF is fully initialized.

NOTE: Even though this task is similar to [IOS-XE NETCONF: Configure an ACL](#), these two tasks do not depend on each other.

Hints

If you prefer not to simply copy and paste the sample solution, here are some hints to get you started. The **Solution** section provides the Python script that will accomplish this task.

- To complete this task, we will operate with YANG data models: **Cisco-IOS-XE-acl.yang** and **Cisco-IOS-XE-native.yang**. All YANG models can be found on YANG GitHub repository.
- To explore YANG models there is a number of useful tools: **Yang Explorer**, **pyang** and others.
- RESTCONF uses basic authentication, hence it is required to provide credentials of a user with the privilege level 15 in the header to perform RESTCONF operation
- To test RESTCONF you can use **curl** command line utility or **Postman** application - this is a GUI to perform any REST operations.

For example, this is how to get configuration of every interface on the router for Native YANG model using curl:

```
cisco@linux-host:~$ curl -k -u admin:Cisco12345 -H "Accept: application/yang-data+json"
https://198.18.133.212/restconf/data/native/interface
```

- Useful RESTCONF URI for this task

Extended ACL:

```
https://198.18.133.212/restconf/data/native/ip/access-list/extended
```

Access-list applied on the interface GigabitEthernet1 inbound:

```
https://198.18.133.212/restconf/data/native/interface/GigabitEthernet=1/ip/access-group/in/acl
```

- Use the **requests** library to perform RESTCONF operations.
- Verification: **linux-host** should be able ping **csr1k-nwp**, Windows workstation should not be able to ping **csr1k-nwp**
- SSH to csr1k-nwp and verify the ACL has been applied to the running configuration and looks like what you would expect it to be if the configuration had been applied using the CLI.

Solution

A sample solution for this task is provided in **tasks/restconf/acl.py** of the [GitHub](#) repository.

1. Run the script from Windows workstation using the following command from **tasks/restconf** directory:

```
C:\Users\administrator\Documents\GitHub\enterprise-network-programmability\tasks\restconf> python acl.py
ACL WIN_ONLY is currently applied on the interface GigabitEthernet1 in "IN" direction
ACL WIN_ONLY was successfully removed from the interface GigabitEthernet1 in "IN" direction
Successfully configured extended ACL LINUX_ONLY
ACL LINUX_ONLY was successfully applied on the interface GigabitEthernet1 in "IN" direction
```

2. The CLI-based configuration should look like the following when the ACL is applied:

```
ip access-list extended LINUX_ONLY
 permit icmp host 198.18.1.107 host 198.18.133.212
 deny icmp any host 198.18.133.212
 permit ip any any
!
interface GigabitEthernet1
 ip access-group LINUX_ONLY in
```

3. To test, open the **linux-host** session in Putty on the lab computer or dCloud Windows workstation. From the linux-host ping to **csr1k-nwp1** should succeed. Ping from dCloud Windows workstation should fail.

IOS-XE RESTCONF: Operational data to Spark

In this task, we will see how we can extract data from one API, and then push that data into another API. Specifically, we will look at how we can grab operational data using a device's RESTCONF interface. Once we have the data, we will post it to a Cisco Spark room using Spark's REST API.

Objective

Network operation is a very collaborative process. Our operators work in different shifts and in different geographies. We want to be able to easily disseminate network data to all of our operators without duplicating processes that would put undue burden on the network and could introduce inconsistencies. Cisco Spark is an excellent tool that can bring together our operators into a single room with an ever-growing history of content. If we post the network data we want to that room, all operators will see it. New operators added to the room later will also be able to view the previous data posts.

In this task, we will extract hostname and interface GigabitEthernet1 counters (number of received and transmitted bytes) from **csr1k-nwp1** using its RESTCONF interface, and then post this data to the Spark room.

Getting Started

Make sure you have performed "[IOS-XE RESTCONF: Configure an ACL](#)" first.

We will use the IOS-XE router, **csr1k-nwp1** to complete this task. Therefore, you will need:

- Open PuTTY on the lab computer or the dCloud Windows workstation. Select the **csr1k-nwp1** session. Consult the topology diagram for the IP address and credentials for **csr1k-nwp1**.

Hints

If you prefer not to simply copy and paste the sample solution, here are some hints to get you started. The Solution section provides the Python script that will accomplish this task.

- The hostname can be accessed using the following RESTCONF URI:

```
https://198.18.133.212/restconf/data/native/hostname
```

- Interface counters can be accessed using the following RESTCONF URI

```
https://198.18.133.212/restconf/data/interfaces-state/interface
```

Solution

A sample solution for this task is provided in **tasks/restconf/acl.py** of the [GitHub](#) repository.

1. Run the script from Windows workstation using the following command from **tasks/restconf** directory:

```
C:\Users\administrator\Documents\GitHub\enterprise-network-programmability\tasks\restconf> python interface_stats.py
```

2. Shortly you should see a Spark message similar to the following:



You 05:53

Interface **GigabitEthernet1** on the device **csr1k-nwp1** has received **138200 bytes** and transmitted **102535 bytes**.

Scenario 3. Controller-Level Programmability

Device Count in APIC-EM Inventory

APIC-EM offer a northbound REST API for your scripts and applications to interact. The Cisco Apps use this same REST API as well and extend the REST API with their additional functionality.

Objective

In this task, we use Inventory Device Count as an example for how to approach using the REST API for the first time.

Hints

The API is used for a reason, a scenario. With this in mind, a typical developer sequence is to:

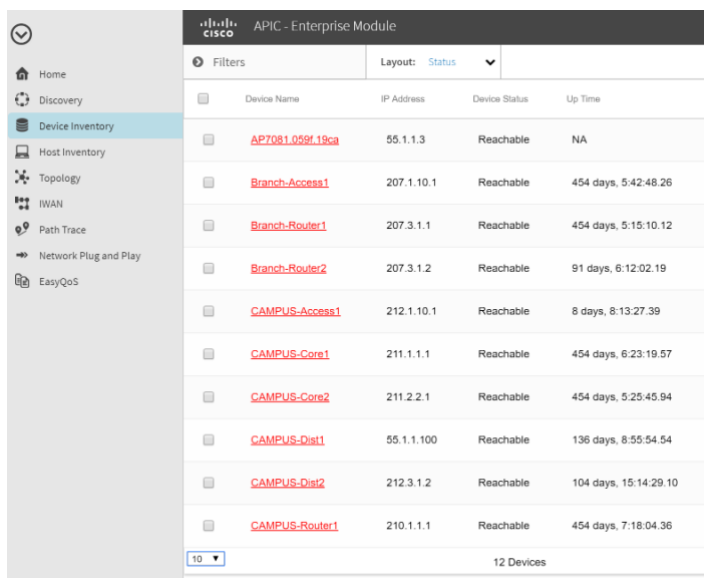
- Navigate through the sequence using the GUI of the controller
- Explore the corresponding REST API calls via APIC-EM GUI
- Prototype a possible sequence using Chrome/POSTMAN
- Craft and refine a script

We will follow this logical sequence to get the number of devices in the APIC-EM Inventory.

Steps

1. **APIC-EM GUI:** Login to APIC-EM using your Chrome Browser and navigate to Device Inventory and note the number of devices at the bottom center of the page (12 Devices in our case).

Figure 18. APIC-EM



Device Name	IP Address	Device Status	Up Time
AP7081.059f.19ca	55.1.1.3	Reachable	NA
Branch-Access1	207.1.10.1	Reachable	454 days, 5:42:48.26
Branch-Router1	207.3.1.1	Reachable	454 days, 5:15:10.12
Branch-Router2	207.3.1.2	Reachable	91 days, 6:12:02.19
CAMPUS-Access1	212.1.10.1	Reachable	8 days, 8:13:27.39
CAMPUS-Core1	211.1.1.1	Reachable	454 days, 6:23:19.57
CAMPUS-Core2	211.2.2.1	Reachable	454 days, 5:25:45.94
CAMPUS-Dist1	55.1.1.100	Reachable	136 days, 8:55:54.54
CAMPUS-Dist2	212.3.1.2	Reachable	104 days, 15:14:29.10
CAMPUS-Router1	210.1.1.1	Reachable	454 days, 7:18:04.36

12 Devices

NOTE: Optionally, if you are familiar with Chrome's developer tools, observe the client-server communications upon loading this page – this will give you extra information about the API calls used.

APIC-EM Live API Documentation (Swagger): Click on the API link in the top-right corner of the APIC-EM GUI. This leads to the live API documentation. Then navigate to the desired API (in our example /network-device/count within the Inventory API section).

Open the GET call for GET /network-device/count and select ALL as the RBAC Scope – then click on Try it out. You should get the same number of devices via a JSON formatted response from the API.

Figure 19. Inventory - /network-device/count API

APIC - Enterprise Module

Available APIs

- Discovery
- File Service
- Host
- Inventory**
- Policy
- Policy Analysis
- QoS
- Task
- Topology Service
- Zero Touch Deployment

APIC-EM API Documentation

APIC-EM API based on the Swagger™ 1.2 specification

[Terms of service](#)
[Cisco DevNet](#)

API Name	Show/Hide	List Operations	Expand Operations	Raw
device-credential : Device Credential API				
discovery : Discovery API				
discovery-frequency : Discovery Frequency API				
interface : Interface API				
link : Link API				
location : Location API				
network-device : network-device API				
GET /network-device/tag				getNetworkDeviceTag
GET /network-device/location/{startIndex}/{recordsToReturn}				getNetworkDeviceLocationByRange
POST /network-device				addNetworkDevice
PUT /network-device				updateNetworkDevice
GET /network-device/ip-address/{ipAddress}				getNetworkDeviceByIp
GET /network-device/serial-number/{serialNumber}				getNetworkDeviceBySerialNumber

Figure 20. Network Device Count

GET /network-device/count

Retrieves network device count by filters

Implementation Notes

Gets the count of network devices filtered by management IP address, mac address, hostname and location name

Response Class

Model | Model Schema

CountResult {
 version (string, optional),
 response (integer, optional)
}

Response Content Type: application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
scope	ALL	Authorization Scope for RBAC	header	List

Error Status Codes

HTTP Status Code	Reason
200	This Request is OK
403	This user is Forbidden Access to this Resource
401	Not Authorized Yet, Credentials to be supplied
404	No Resource Found

Try it out | [Hide Response](#)

Request URL

https://sandboxapic.cisco.com:9443/api/v1/network-device/count

Response Body

```
{
  "response": 12,
  "version": "1.0"
}
```

NOTE: In the APIC-EM GUI login was completed via the browser at the beginning of your session. When accessing the API directly from an external tool, script or application, a separate login is required

Chrome/Postman: Especially when more complex REST calls need to be tested and combined, the Postman browser plugin in Chrome is a great way to get started and generate code stubs. For our simple call this is not needed – but we will use it to illustrate how to get started with Postman and APIC-EM.

First we need to login to RBAC via the REST API to obtain the **Session Token** which is required for APIC-EM northbound REST API calls.

- Use the **POST/ticket** API call to request a token
- Copy ticket and add to **X-Auth-Token** Header for use in subsequent API calls

Figure 21. Request Service Ticket

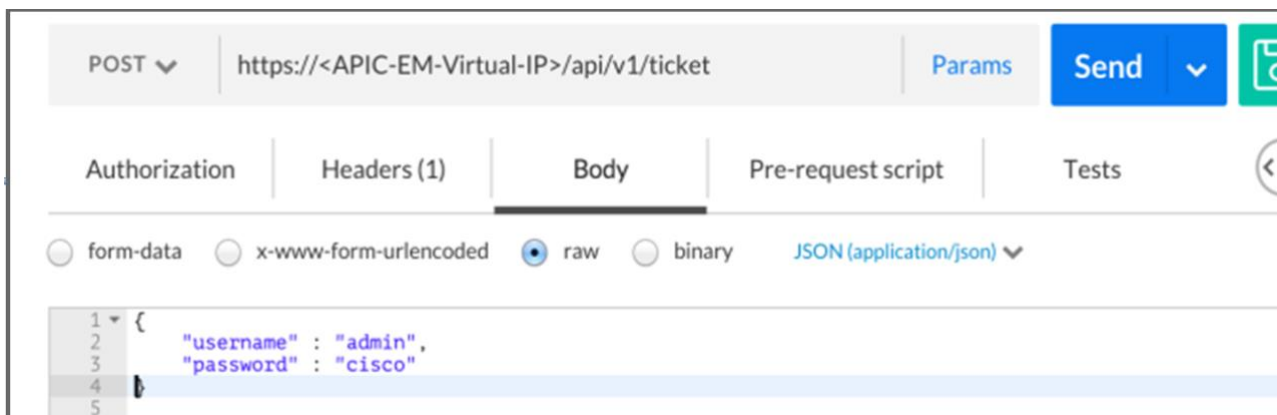


Figure 22. API Response

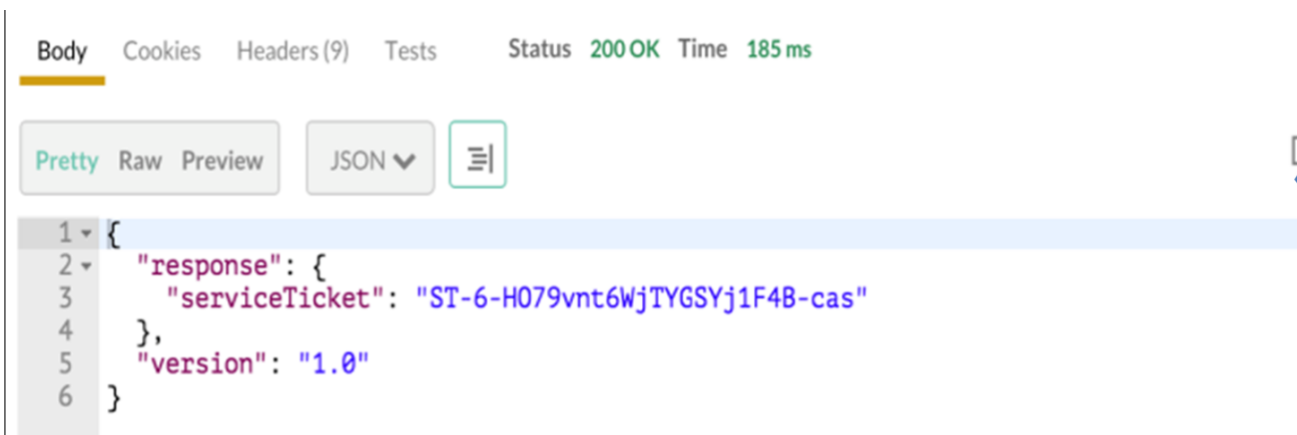
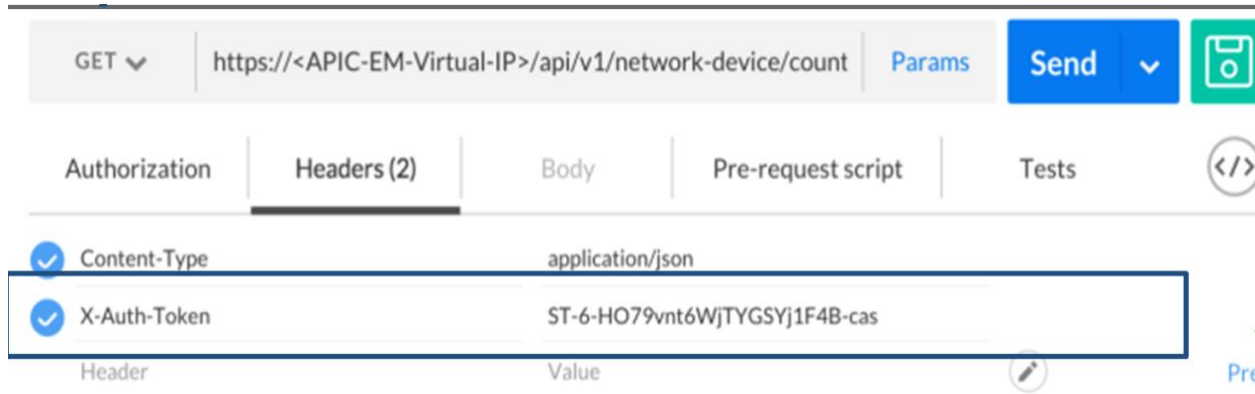


Figure 23. Add Ticket to X-Auth-Token Header Field

You should now see the exact same JSON formatted response from /network-device/count

Python Script: Once the correct sequence and format of REST API calls is understood, it makes sense to start putting together a script. Before being able to use the API, we need to login and get a session token via the /ticket API call – the basic steps for this are:

a. Request Session Ticket.

```
apic_credentials = json.dumps({'username': APIC_LOGIN,
                              'password': APIC_PASSWD})
q = requests.post('https://198.18.129.100/api/v1/ticket',
                  data=apic_credentials,
                  headers='Content-type': 'application/json')
```

b. Put Ticket into X-Auth-Token Header.

```
apic_session_ticket = q.json()['response']['serviceTicket']
apic_headers = {'Content-type': 'application/json',
                'X-Auth-Token': apic_session_ticket}
```

c. Use in subsequent API Calls.

```
q = requests.get('https://198.18.129.100/api/v1/any-other-api-call',
                 headers=apic_headers)
```

A sample solution is provided in `apic-em-1.1-device-count-to-spark-sample.py`

Path Trace via APIC-EM REST APIs

The APIC-EM Path Trace App provides 5-tuple specific path information across a network. While the built in GUI is great for rapid incident response and troubleshooting, the REST API on top of it offers an opportunity for automated proactive harvesting of Path Trace information.

Objectives

In this task, we will use the APIC-EM northbound REST API to initiate a Path Trace task, export the resulting path information and format/display it in various ways.

Hints

Use the following:

- Source IP Address: 10.1.15.117
- Destination IP Address: 10.1.12.20

The logical developer sequence introduced in the previous task is useful here too.

Step by Step

1. **APIC-EM GUI:** Start by using the Path Trace App via it's graphical user interface and notice the sequence of input and output steps.

NOTE: Optionally, use Chrome's developer tools, to observe the REST API calls made.

APIC-EM Live API Documentation (Swagger): Navigate to the /flow-analysis POST call. You will notice this call requires a JSON formatted input document. For description of mandatory vs. optional parameters and for a sample document, click on the Model and Model Schema links to the right.

Provide an input JSON document:

```
{
  "sourceIP": "10.1.15.117",
  "destIP": "10.1.12.20"
}
```

Then click on Try It Out and capture the flowAnalysisId from the resulting JSON formatted response

```
{
  "response": {
    "flowAnalysisId": "53023afa-2ed5-4de1-a2eb-c84853f7e3eb",
    "taskId": "3bda8afb-899a-44d9-87d3-eb2d45889b47",
    "url": "/api/v1/flow-analysis/53023afa-2ed5-4de1-a2eb-c84853f7e3eb"
  },
  "version": "1.0"
}
```

Finally, the `flowAnalysisId` can be used to retrieve the actual path information via the `/flog-analysis` GET call.

NOTE: Running a Path Trace will take some time – a few seconds to a minute or two. The status parameter within the JSON response will indicate whether the Path Trace has completed or is still in progress (in which case a partial path will be available and you may want to do a subsequent GET call at a later stage)

Chrome/Postman: If you are familiar with Chrome/Postman, replicate the sequence and generate some python code samples

Python Script: Now it's time to combine things and make the three API calls from a your own python script:

- POST call `/ticket` to get a session token
- POST call `/flow-analysis` to initiate a Path Trace
- GET call `/flow-analysis` to query the resulting path information

NOTE: A sample solution for this is provided in `apic-em-1.3-path-trace-sample.py`.

Once this works, consider parsing the resulting JSON documents for specific information and using the information to interact with humans – for example:

- Run a Path Trace via the APIC-EM REST API → extract a list of host names.
- Run a Path Trace via the APIC-EM REST API → post host names to Spark.
- Run a Path Trace via the APIC-EM REST API → extract host names and links.
- Run a Path Trace via the APIC-EM REST API → create a NeXt Data.jsRun Path Trace via the UI

NOTE: Some examples are provided in `apic-em-1.3-path-trace-to-spark-sample.py` and further information on NeXt and Spark can be found within the Human Interaction Perspective of this lab guide.



Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at www.cisco.com/go/offices.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: www.cisco.com/go/trademarks. Third party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)