DEVNET Create

Building a Troubleshooting Assistant

Combing old and new automation skills to make jobs easier

Hank Preston

Principal Engineer, Learning at Cisco

@hfpreston

Hank Preston

Principal Engineer, Learning at Cisco

If it ain't broke you didn't try hard enough ;-)

A long time network engineer turned automation engineer. I'm passionate about modernizing infrastructure engineering and operations and telling stories about it.

DEVNET Create

You knew being the new engineer on the team would mean getting some “boring work” but this latest assignment is pretty bad. A network interface connected to a critical system has been flapping unexpectedly. You’ve been told to drop everything and just watch for that interface to flap. And whenever it goes down, you need to gather some details before it goes back up. Surely you can automate this?

So what do we really need to do

- Watch for operational state changes of a specific interface
  - An Ethernet interface on a Nexus 9000 switch
- When interface goes down (or up)
  - Log the date/time of the change
  - Gather outputs from the following commands
    - show interface ethernet #/#
    - show logging last 50
    - show ip arp vrf all
    - show mac address-table
    - show ip route vrf all
    - show system internal interface ethernet #/# ethernet #/# event-history
- Store the output of these commands for each event

A common manual approach to this task

1. Create a text file with commands to run
2. Log into the switch
3. Turn on terminal client output logging
4. Do one of the following
    1. Monitor the console for state changes
    2. Run "show interface" command over and over and over again
5. When the change happens, paste the commands into terminal
6. Eventually compile the data from the output log into separate records/files

How an Automation Engineer Can Tackle It

- Use programmability skills to monitor the state, capture command output, and save output
- The goal is to automate what you would manually do anyway
- Test and build your solution in a lab
  - Or DevNet Sandbox if you don't have a Lab
- Result in a solution that can be used again and again

Questions to ask as we start

How can we know when the interface state changes?

Monitor the device log for changes

Where should we run the script?

Run it on the device itself

How do we gather the command output?

Included Cisco cli library

How/Where do we save the command output?

One file per event per command on the device bootflash

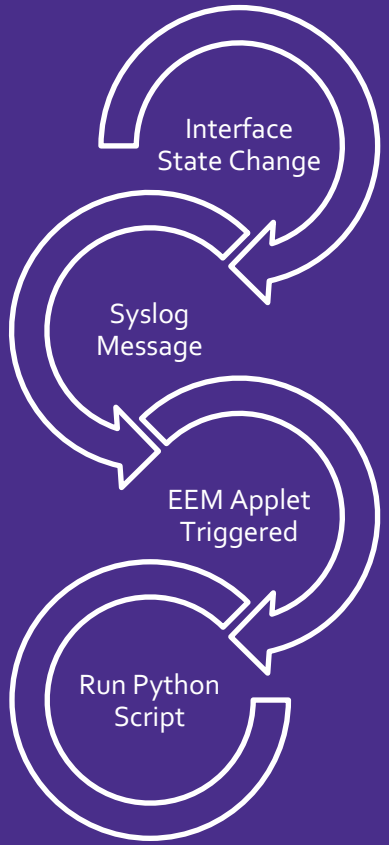
How can we know in real-time of a state change?

Embedded Event Manager (EEM)

Preparing and Planning the Use Case

Overall Project Planning

- Interface state changes and generates syslog message
- EEM sees the syslog message and runs Python script
- Python script
  - Run command list and gather output
  - Create folder for output files
  - Write output from each command to file in folder



Considerations on Building an OnBox Python Script

- Version of Python to use
  - NX-OS 9.3(5)+ include Python 3
  - Earlier versions include Python 2.7
- The Cisco Python CLI API libraries only available when running on-switch
- Interactive Python Shell is a great way to test commands and scripts.
- Recommendation: Keep onbox scripts as simple as possible.

```
! NX-OS 10.1(x)

switch# python3

Python 3.7.3 (default, Nov 20 2019, 14:38:01)
[GCC 5.2.0] on linux
Type "help", "copyright", "credits" or "license"
for more information.

>>> import cisco, cli

################################################

! NX-OS 9.2(3)

dist-sw01# python

Python 2.7.11 (default, Feb 26 2018, 03:34:16)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license"
for more information.

>>> import cisco, cli
```

Considerations on Developing OnBox Scripts

- Split “development” and “execution” environments
- How to manage scripts for onbox execution?
- Clone repository to switch with Git
  - Requires git on the switch
  - Requires the switch have access to Git Server (DNS, routing, ACLs, etc)
  - Requires telling switch to “pull”
- Copy scripts to switch as “deployment step”
  - Requires a file transfer protocol
  - Requires a ”build” step that pushes updates

Considerations on Developing OnBox Scripts

- Split "development" and "execution" environments
- How to manage scripts for onbox execution?
- Clone repository to switch with Git
  - Requires git on the switch
  - Requires the switch have access to Git Server (DNS, routing, ACLs, etc)
  - Requires telling switch to "pull"
- Copy scripts to switch as "deployment step"
  - Requires a file transfer protocol
  - Requires a "build" step that pushes updates

```
# Copy the file to the switch
scp troubleshooting_assistant.py cisco@10.10.20.177:

User Access Verification
Password:

troubleshooting_assistant.py  100%  770     7.2KB/s   00:00

# Log into the switch, check for file
ssh cisco@10.10.20.177

User Access Verification
Password:

dist-sw01#
dist-sw01# dir bootflash:///troubleshooting_assistant.py

770    Jun 17 13:36:56 2021  troubleshooting_assistant.py
```

Building the OnBox Python Script to Run Commands and Collect Output

Creating the Script Plan

- As always, start with a plan and outline for the script

```
#! /usr/bin/env python
"""
This is a script that will run "onbox" on a Nexus Switch
with the goal of running a series of show commands and
collecting the output into files stored into date/time
folders. One file per command.


Commands to run:
    show interface ethernet #/#
    show logging last 50
    show ip arp vrf all
    show mac address-table
    show ip route vrf all
    show system internal interface ethernet #/# ethernet
#/# event-history

Command Line Argument: Interface ID
"""


if __name__ == "__main__":
    print("Collecting show commands storing in bootflash.")


    # Collect interface ID as command line argument
    # Run commands and store output
    # Create new folder for output
    # Create a file for each command output
```

What interface to check?

- We could hardcode the interface id into the script
  - But then the script wouldn't be as useful
- Back to our skills with Argparse

Note: f-string is available Python 3.6+. To support pre 9.3.5 NX-OS using .format() instead

```
if __name__ == "__main__":
    print("Collecting show commands and storing in
bootflash.")

    # Collect interface ID as command line argument
    import argparse

    # Use argparse to determine the interface id
    parser = argparse.ArgumentParser(description='Run show
commands to assist with troubleshooting')

    parser.add_argument('--interface', required=True,
     type=str, help='Interface of interest. (example: 1/1')

    args = parser.parse_args()

    print("Interface Ethernet {interface_id} will be
checked.".format(interface_id = args.interface))

    # Run commands and store output

    # Create new folder for output

    # Create a file for each command output
```

Building logic to run the commands

- No actual command running yet
- Two dictionaries
  - One for the commands to run
  - One to save the output
- Create a simple function that will gather raw and json output
- Use a print debug to view the data collected

Note: Even commands that don't include the interface_id variable can have .format() run without error.

```
def run_command(command, interface):
    """
    Run a given command, gather both raw and JSON output.
    Return as a tuple. (output_raw, output_json)
    """

    output_raw = command.format(interface_id=interface)
    output_json = command.format(interface_id=interface)

    return (output_raw, output_json)

#########################################################

# Run commands and store output
# Dict of commands to run. Key will be used for file naming
commands = {
    "show_interface": "show interface ethernet
{interface_id}"
}

# Output Dict
output = {}

# Loop over commands to run function and save output
for label, command in commands.items():
    output[label] = run_command(command, args.interface)

# for debugging, print output
print(output)
```

Run the show commands with the CLI library

- Import the cli and clid functions
  - Note: Now if you were running the script "off-box", it will no longer work

```
from cli import cli, clid


def run_command(command, interface):
    """
    Run a given command, gather both raw and JSON output.
    Return as a tuple. (output_raw, output_json)
    """

    output_raw = cli(
                     command.format(interface_id=interface)
                    )

    output_json = clid(
                     command.format(interface_id=interface)
                    )

    return (output_raw, output_json)
```

Adding the remaining commands to run

- Even commands that don't include the interface_id variable can have .format() run without error.

```
# Dict of commands to run. Key will be used for file naming
commands = {
    "show_interface": "show interface ethernet {interface_id}",

    "show_logging": "show logging last 50",

    "show_ip_arp": "show ip arp vrf all",

    "show_mac_address_table": "show mac address-table",

    "show_ip_route": "show ip route vrf all",

    "show_system_internal_interface":
    "show system internal interface ethernet {interface_id} ethernet {interface_id} event-history"
}
```

Preparing for the output reports – Folder Name

- Use the datetime module to get the current time
  - Use `.strftime()` function to build a folder_name
- The os library in Python provides functions for working with the host OS – even a switch
  - `os.mkdir()` will make a new directory
- Tell users where to find the data

```
from datetime import datetime
from os import mkdir

# Output Dict
output = {}

# Loop over commands to run function and save output
for label, command in commands.items():
    pass
    # output[label] = run_command(command, args.interface)

# Create new folder for output
now = datetime.now()
report_timestamp = now.strftime("%Y-%m-%d-%H-%M-%S")

folder_name = \
"/bootflash/ts_report_{timestamp}_intf{interface_id}".format(
    timestamp=report_timestamp,
    interface_id=args.interface.replace("/", "_")
    )

print("Output will be stored in folder
{folder_name}/".format(folder_name=folder_name))

mkdir(folder_name)

# Create a file for each command output
```

Logic for writing the result files

- The results of each command is a 2 item tuple
- Unpack them into individual variables
- Only write out the JSON results if they exist

```
print("Output will be stored in folder
{folder_name}/".format(folder_name=folder_name))
mkdir(folder_name)

# Create a file for each command output
for command, results in output.items():
    # Unpack the results
    raw_output, json_output = results

    # write raw data file
    print("Writing file {folder_name}/{command}.txt".format(
         folder_name=folder_name, command=command))

    # if json_output available, write json file
    if json_output:
        print(
          "Writing file {folder_name}/{command}.json".format(
             folder_name=folder_name, command=command))
```

Writing the output files

- Use open() and “w” to create a new writeable file object
- This code is slightly redundant, creating a function here would be a good enhancement

```
# Create a file for each command output
for command, results in output.items():
    # Unpack the results
    raw_output, json_output = results


    # write raw data file
    print("Writing file {folder_name}/{command}.txt".format(folder_name=folder_name, command=command))
    with open("{folder_name}/{command}.txt".format(folder_name=folder_name, command=command), "w") as f_raw:
        f_raw.write(raw_output)


    # if json_output available, write json file
    if json_output:
        print("Writing file {folder_name}/{command}.json".format(folder_name=folder_name, command=command))
        with open("{folder_name}/{command}.json".format(folder_name=folder_name, command=command), "w") as f_json:
            f_json.write(json_output)
```

Running the Script

- No errors…
- But is there data?

```
dist-sw01# python bootflash:troubleshooting_assistant.py --interface 1/11

# OUTPUT
Collecting show commands and storing in bootflash.
Interface Ethernet 1/11 will be checked.
Output will be stored in folder /bootflash/ts_report_2021-06-17-15-37-49_interface1_11/
Writing file /bootflash/ts_report_2021-06-17-15-37-49_interface1_11/show_system_internal_interface.txt
Writing file /bootflash/ts_report_2021-06-17-15-37-49_interface1_11/show_ip_arp.txt
Writing file /bootflash/ts_report_2021-06-17-15-37-49_interface1_11/show_ip_arp.json
Writing file /bootflash/ts_report_2021-06-17-15-37-49_interface1_11/show_interface.txt
Writing file /bootflash/ts_report_2021-06-17-15-37-49_interface1_11/show_interface.json
Writing file /bootflash/ts_report_2021-06-17-15-37-49_interface1_11/show_ip_route.txt
Writing file /bootflash/ts_report_2021-06-17-15-37-49_interface1_11/show_ip_route.json
Writing file /bootflash/ts_report_2021-06-17-15-37-49_interface1_11/show_mac_address_table.txt
Writing file /bootflash/ts_report_2021-06-17-15-37-49_interface1_11/show_mac_address_table.json
Writing file /bootflash/ts_report_2021-06-17-15-37-49_interface1_11/show_logging.txt
```

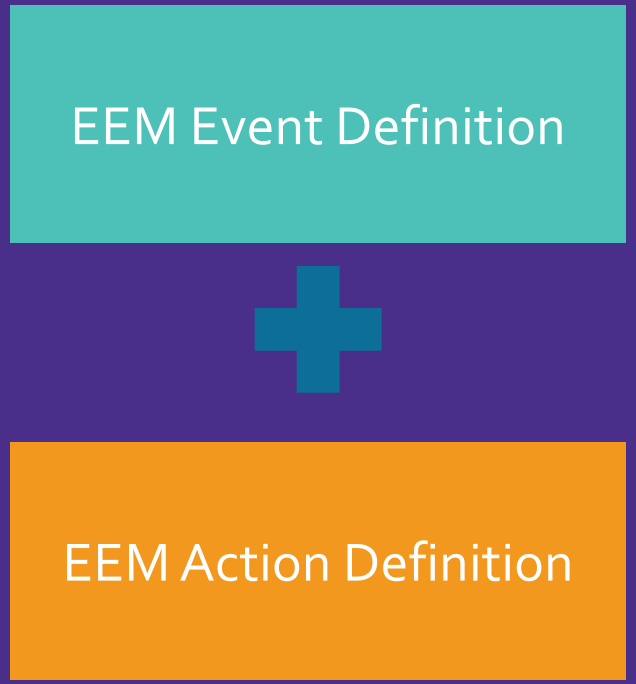
Verifying the File Contents

```
dist-sw01# dir bootflash:ts_report_2021-06-17-15-41-10_interface1_11
       2285    Jun 17 15:41:10 2021  show_interface.json
       1811    Jun 17 15:41:10 2021  show_interface.txt
        947    Jun 17 15:41:10 2021  show_ip_arp.json
        868    Jun 17 15:41:10 2021  show_ip_arp.txt
       9710    Jun 17 15:41:10 2021  show_ip_route.json
       3635    Jun 17 15:41:10 2021  show_ip_route.txt
       3991    Jun 17 15:41:10 2021  show_logging.txt
       2796    Jun 17 15:41:10 2021  show_mac_address_table.json
       1536    Jun 17 15:41:10 2021  show_mac_address_table.txt
      40204    Jun 17 15:41:10 2021  show_system_internal_interface.txt

dist-sw01# show file bootflash:ts_report_2021-06-17-15-41-10_interface1_11/show_ip_route.txt
IP Route Table for VRF "default"
'*' denotes best ucast next-hop
'**' denotes best mcast next-hop
'[x/y]' denotes [preference/metric]
'%<string>' in via output denotes VRF <string>

172.16.101.0/24, ubest/mbest: 1/0, attached
    *via 172.16.101.2, Vlan101, [0/0], 2w1d, direct
172.16.101.1/32, ubest/mbest: 1/0, attached
    *via 172.16.101.1, Vlan101, [0/0], 2w1d, hsrp
172.16.101.2/32, ubest/mbest: 1/0, attached
.

dist-sw01# show file bootflash:ts_report_2021-06-17-15-41-10_interface1_11/show_ip_route.json
{"TABLE_vrf": {"ROW_vrf": [{"vrf-name-out": "default", "TABLE_addrf": {"ROW_addrf": {"addrf": "ipv4", "TABLE_prefix":
{"ROW_prefix": [{"ipprefix": "172.16.101.0/24", "ucast-nhops": "1", "mcast-nhops": "0", "attached": "true", "TABLE_path":
{"ROW_path": {"ipnexthop": "172.16.101.2", "ifname": "Vlan101", "uptime": "P15DT16H28M27S", "pref": "0", "metric": "0",
"clientname": "direct", "ubest": "true"}}}, {"ipprefix":"
.
```

Using EEM to Monitor and React to Syslog

Embedded Event Manager: Old School Automation

- EEM monitors events on the device and takes configured actions
  - Event Examples: Syslog Message, Command Ran, Hardware Changes
  - Action Possibilities: Execute CLI, Generate Log, Generate SNMP, Call Home Action
- Available on IOS, IOS XE, IOS XR, NX-OS
  - System specific differences do exist



What Syslog Patterns to Monitor?

- Need string match for Interface going down, and up
- Cause the event using CLI (or unplug cable if able)
- Balance the exactness of message for match

```
dist-sw01(config-if)# int eth1/11
dist-sw01(config-if)# shut


2021 Jun 17 17:35:39 dist-sw01 %ETHPORT-5-IF_DOWN_CFG_CHANGE: Interface Ethernet1/11 is down(Config change)
2021 Jun 17 17:35:39 dist-sw01 %ETHPORT-5-IF_DOWN_ADMIN_DOWN: Interface Ethernet1/11 is down (Administratively down)




dist-sw01(config-if)# no shut


2021 Jun 17 17:35:56 dist-sw01 %ETHPORT-5-IF_ADMIN_UP: Interface Ethernet1/11 is admin up .
2021 Jun 17 17:35:57 dist-sw01 %ETHPORT-5-SPEED: Interface Ethernet1/11, operational speed changed to auto
2021 Jun 17 17:35:57 dist-sw01 %ETHPORT-5-IF_DUPLEX: Interface Ethernet1/11, operational duplex mode changed to unknown
2021 Jun 17 17:35:57 dist-sw01 %ETHPORT-5-IF_UP: Interface Ethernet1/11 is up in mode access
```

Configuring EEM to Monitor Syslog

- The documentation provides examples you can build from
  - Monitor Syslog
  - Generate Syslog
    (great way to test EEM)
  - CLI Events and Actions

```
! This file contains EEM configurations for this use case


! EEM Verification Examples:
!   These EEM applets will monitor for the Syslog Events for the interface
!   up/down and generate a Syslog Message. These are meant to verify that
!   EEM is working, seperate from the Python Script.


! Monitor for "down"
event manager applet TS_Bot_Eth1_11_DOWN
  event syslog pattern "Interface Ethernet1/11 is down"
  action 1 cli python bootflash:troubleshooting_assistant.py --interface 1/11


! Monitor for "up"
event manager applet TS_Bot_Eth1_11_UP
  event syslog pattern "Interface Ethernet1/11 is up"
  action 1 cli python bootflash:troubleshooting_assistant.py --interface 1/11
```

Testing EEM Action running Python Script

- 14 second difference from event to folder. Time to run the commands
- Timeout error – Python command takes longer to "finish" than EEM expects

```
dist-sw01(config-if)# int eth1/11
dist-sw01(config-if)# shut

dist-sw01# show event manager events action-log

eem_event_time:06/17/2021,18:12:45 event_type:cli event_id:14 slot:active(1) vdc:1 severity:minor applets:TS_Bot_Eth1_11_DOWN
eem_param_info:_syslog_msg = "%ETHPORT-5-IF_DOWN_ADMIN_DOWN: Interface Ethernet1/11 is down (Administratively down)"
Execution timed out for cmd(s):
          python bootflash:troubleshooting_assistant.py --interface 1/11

dist-sw01# dir bootflash: | grep ts_report

       4096    Jun 17 18:12:59 2021  ts_report_2021-06-17-18-12-59_interface1_11/

dist-sw01# dir bootflash:ts_report_2021-06-17-18-12-59_interface1_11
       2277    Jun 17 18:12:59 2021  show_interface.json
       1803    Jun 17 18:12:59 2021  show_interface.txt
        946    Jun 17 18:12:59 2021  show_ip_arp.json
.
.
       1536    Jun 17 18:12:59 2021  show_mac_address_table.txt
     123514    Jun 17 18:12:59 2021  show_system_internal_interface.txt
```

Pulling Reports off the Switch

Downloading all Report Files

- Having the report files on the switch isn't that useful to diagnose
- Use scp to download all report directories in one go

```
scp -r "cisco@10.10.20.177:ts_report_*"  ./

show_logging.txt                                      100% 5116    47.6KB/s   00:00
show_mac_address_table.txt                            100% 1536    15.4KB/s   00:00
show_mac_address_table.json                           100% 2796    27.9KB/s   00:00
show_system_internal_interface.txt                    100%  121KB 136.5KB/s   00:00
show_ip_arp.txt                                       100%  868     9.1KB/s   00:00
show_ip_route.json                                    100% 9602    93.1KB/s   00:00
show_ip_route.txt                                     100% 3681    36.4KB/s   00:00
show_interface.txt                                    100% 1803    18.8KB/s   00:00
show_ip_arp.json                                      100%  946     9.7KB/s   00:00
show_interface.json                                   100% 2277    21.8KB/s   00:00
show_logging.txt                                      100% 5255    47.8KB/s   00:00
show_mac_address_table.txt                            100% 1536    16.5KB/s   00:00
show_mac_address_table.json                           100% 2796    28.7KB/s   00:00
show_system_internal_interface.txt                    100%  142KB 136.0KB/s   00:01
show_ip_arp.txt                                       100%  868     9.3KB/s   00:00
show_ip_route.json                                    100% 9617    92.1KB/s   00:00
show_ip_route.txt                                     100% 3681    36.5KB/s   00:00
show_interface.txt                                    100% 1803    18.1KB/s   00:00
show_ip_arp.json                                      100%  937     9.2KB/s   00:00
show_interface.json                                   100% 2277    21.1KB/s   00:00
```

Downloading all Report Files

- Having the report files on the switch isn't that useful to diagnose
- Use scp to download all report directories in one go

```
ls -l ts_report_*

ts_report_2021-06-17-18-12-59_interface1_11:
total 172
-rw-r--r-- 1 hpreston hpreston   2277 Jun 17 18:42 show_interface.json
-rw-r--r-- 1 hpreston hpreston   1803 Jun 17 18:42 show_interface.txt
-rw-r--r-- 1 hpreston hpreston    946 Jun 17 18:42 show_ip_arp.json
-rw-r--r-- 1 hpreston hpreston    868 Jun 17 18:42 show_ip_arp.txt
-rw-r--r-- 1 hpreston hpreston   9602 Jun 17 18:42 show_ip_route.json
-rw-r--r-- 1 hpreston hpreston   3681 Jun 17 18:42 show_ip_route.txt
-rw-r--r-- 1 hpreston hpreston   5116 Jun 17 18:42 show_logging.txt
-rw-r--r-- 1 hpreston hpreston   2796 Jun 17 18:42 show_mac_address_table.json
-rw-r--r-- 1 hpreston hpreston   1536 Jun 17 18:42 show_mac_address_table.txt
-rw-r--r-- 1 hpreston hpreston 123514 Jun 17 18:42 show_system_internal_interface.txt

ts_report_2021-06-17-18-16-27_interface1_11:
total 240
-rw-r--r-- 1 hpreston hpreston   2277 Jun 17 18:42 show_interface.json
-rw-r--r-- 1 hpreston hpreston   1803 Jun 17 18:42 show_interface.txt
-rw-r--r-- 1 hpreston hpreston    937 Jun 17 18:42 show_ip_arp.json
-rw-r--r-- 1 hpreston hpreston    868 Jun 17 18:42 show_ip_arp.txt
-rw-r--r-- 1 hpreston hpreston   9617 Jun 17 18:42 show_ip_route.json
-rw-r--r-- 1 hpreston hpreston   3681 Jun 17 18:42 show_ip_route.txt
.
```

Viewing the Data Files

summer2021-devasc-prep-troubleshooting-assistant > ts_report_2021-06-17-18-12-59_interface1_1 > show_mac_address_table.json >


```
{
    "TABLE_mac_address": {
        "ROW_mac_address": [
            {
                "disp_mac_addr": "0000.0c07.ac0a",
                "disp_type": "static",
                "disp_vlan": "101",
                "disp_age": "-",
                "disp_is_secure": "F",
                "disp_is_ntfy": "F",
                "disp_port": "vPC Peer-Link(R)"
            },
            {
                "disp_mac_addr": "0000.0c07.ac0a",
                "disp_type": "static",
                "disp_vlan": "102",
                "disp_age": "-",
                "disp_is_secure": "F",
                "disp_is_ntfy": "F",
                "disp_port": "vPC Peer-Link(R)"
            },
            {
                "disp_mac_addr": "0000.0c07.ac0a",
                "disp_type": "static",
                "disp_vlan": "103",
                "disp_age": "-",
                "disp_is_secure": "F",
                "disp_is_ntfy": "F",
                "disp_port": "vPC Peer-Link(R)"
            },
```


summer2021-devasc-prep-troubleshooting-assistant > ts_report_2021-06-17-18-12-59_interface1_1 > show_system_internal_interface.txt


```
Showing output for single interface: Eth1/1
lacp]
  FSM:<Ethernet1/1> Transition at 131304 usecs after Tue Jun  1 23:12:14 2021
    Previous state: [LACP_ST_PORT_IS_DOWN_OR_LACP_IS_DISABLED]
    Triggered event: [LACP_EV_LACP_ENABLED_AND_PORT_UP]
    Next state: [LACP_ST_DETACHED_LAG_NOT_DETERMINED]


lacp]
  FSM:<Ethernet1/1> Transition at 133006 usecs after Tue Jun  1 23:12:15 2021
    Previous state: [LACP_ST_DETACHED_LAG_NOT_DETERMINED]
    Triggered event: [LACP_EV_PERIODIC_TRANSMIT_TIMER_EXPIRED]
    Next state: [FSM_ST_NO_CHANGE]


lacp]
  FSM:<Ethernet1/1> Transition at 136146 usecs after Tue Jun  1 23:12:16 2021
    Previous state: [LACP_ST_DETACHED_LAG_NOT_DETERMINED]
    Triggered event: [LACP_EV_PERIODIC_TRANSMIT_TIMER_EXPIRED]
    Next state: [FSM_ST_NO_CHANGE]


lacp]
  FSM:<Ethernet1/1> Transition at 139970 usecs after Tue Jun  1 23:12:17 2021
    Previous state: [LACP_ST_DETACHED_LAG_NOT_DETERMINED]
    Triggered event: [LACP_EV_PERIODIC_TRANSMIT_TIMER_EXPIRED]
    Next state: [FSM_ST_NO_CHANGE]


lacp]
```

Closing

Considerations on this Use Case

- EEM + Python is not the only way this could have been done
  - There is nearly always more than one way to accomplish a task
- Important to balance reusable code vs hard coded elements
  - Interface ID is an argument to script
  - Interface ID is hard coded into EEM
  - Command list is hard coded into script
- Consider the impact of running a script like this. Overwhelming the management plane is possible.
- Deploying the EEM configuration could have been automated
  - Ansible, pyATS, NETCONF/RESTCONF
- When to automate depends on the overall value in scale/consistency
- This use case could be done with other platforms with adjustments to EEM configuration and Python code
  - Building a platform agnostic solution is likely possible

Webinar Resources

- Code for this use case on GitHub and Code Exchange
  - Cisco NSO Reservable Sandbox
- DevNet Associate Prep Program
  - More detailed walkthrough of this use case available along with other example use cases
- DevNet Associate Exam Topics List
- Cisco NX-OS EEM Guide
- Cisco NX-OS Python API Guide

DEVNET

Create