

Simplified Network Troubleshooting through API Scripting

NANOG 87: February 13th, 2023

Cat Gurinsky

What we'll cover

- Why automate our troubleshooting?
- Why API and not screen scraping?
- Examples of typical, repeatable, troubleshooting
- Example outputs of grabbing the data via API
- What skills do I need? Who has APIs?
- Examples of some actual code
- Q&A

Why automate?

- *“Your network is a crime scene, and you are the detective. You need better ways to investigate what happened, and prove guilt or innocence.”*
- Jeremy Schulman
- Most failures have repeatable troubleshooting steps to root cause.
- Repeatable means we can automate and code against these expectations to find our culprit.
- Why waste time typing out the same sets of commands every time you have a similar failure?

Why API versus SSH/screen scraping?

- API calls are significantly faster
 - A former colleague and I decided to both try our hands at writing a script to install an extension for a security hotfix we needed to install.
 - The colleague wrote theirs with netmiko/ssh calls, mine was with pyeapi.
 - My script consistently ran faster than my colleagues did. We used mine to update the entire fleet.
- Most APIs return all the data to you in JSON, making parsing much easier compared to screen scraping

Typical Troubleshooting Examples

- Link down / Errors on a link
- Switch rebooted unexpectedly
- Power supply failure alert
- Dump show tech and other common outputs
- Many more possible!

Link Down / Link Errors

- Determine both sides of the bad link, if not already known
- Validate light levels for both sides to see if there is an obvious failure in Tx versus Rx
- Validate rate of bouncing if applicable, and if seen by both sides or not.
- Grab data on optics and serial numbers in case a replacement is warranted

Get relevant show version / inventory quickly

- In case we need to swap a part or open a TAC, grabbing the inventory is a useful first step so we don't have to worry about this later

```
+-----+-----+-----+-----+
|   Hostname   |   Model   | OS Version | Serial Number |
+-----+-----+-----+-----+
| my-switch-hostname | DCS-7508N | 4.26.5M   | HSH11122333   |
+-----+-----+-----+-----+

my-switch-hostname # show inventory | inc 4/13/1
+-----+-----+-----+-----+-----+
|   Hostname   |   Port   | Manufacturer | Model | Serial Number |
+-----+-----+-----+-----+-----+
| my-switch-hostname | Ethernet4/13/1 | KAIAM CORP | XQX5004 | BL1228888CC |
+-----+-----+-----+-----+-----+
```

Check for common outputs

- These are 3 common outputs I look at every time I'm checking a bad link.
- In this example for Arista's pyeapi, the first two return JSON, the last returns text. So I parse the JSON outputs into a nice table, and return the original plain text for the last.

```
my-switch-hostname# show interfaces counters errors
+-----+-----+-----+-----+-----+-----+-----+-----+
| Port | FCS | Align | Symbol | Rx | Runts | Giants | Tx |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Ethernet50/1 | 4332416 | 0 | 330965 | 4344306 | 11890 | 0 | 0 |
+-----+-----+-----+-----+-----+-----+-----+-----+

my-switch-hostname# show interfaces transceiver
+-----+-----+-----+-----+-----+-----+
| Port | Temp(C) | Voltage | Bias Current | Tx (dBm) | Rx (dBm) |
+-----+-----+-----+-----+-----+-----+
| Ethernet50/1 | 55.75 | 3.30 | 41.06 | 1.84 | -7.59 |
| Ethernet50/2 | 55.75 | 3.30 | 38.41 | 1.84 | -7.62 |
| Ethernet50/3 | 55.75 | 3.30 | 39.65 | 1.86 | -7.51 |
| Ethernet50/4 | 55.75 | 3.30 | 40.14 | 1.79 | -6.33 |
+-----+-----+-----+-----+-----+-----+

my-switch-hostname# show interfaces Ethernet 50/1 mac detail
Current System Time: Wed Jul 22 15:28:59 2020

Ethernet50/1
Current State      Changes      Last Change
PHY State          linkUp       44           3:46:28 ago
Interface State    linkUp       55           3:46:27 ago
MAC Rx Local Fault False        60           3:46:28 ago
MAC Rx Remote Fault False        12           20:21:30 ago
```


Lather, Rinse, Repeat!

- In the case of bad links, I want to validate the other side, so I use LLDP or descriptions/inventory database if down, to see what the far end is and run the same command sets there.

```
my-switch-hostname# show lldp neighbors
```

Hostname	Local Port	LLDP Neighbor	Remote Port
my-switch-hostname	Ethernet50/1	my-core-hostname	Ethernet3/26/1

Unexpected switch reboot

- When a switch reboots there are 2 things I always check
- Current uptime and code version of the switch (show version)
- What the switch thinks the reload reason was (show reload cause)
- Sure it doesn't take long to login and do this, but it's just super fast with API instead!

Example reload diagnostic

- Simple and to the point, with relevant data in case a TAC needs to be opened for follow up.
- Also if debug information was returned, we write it to a file on the local computer so it's ready to go.

```
Hostname: my-switch-hostname.com
Model: DCS-7010T-48-R Hardware Revision: 12.03
Serial Number: SGD22299999
OS Version: 4.22.11M
+-----+-----+
| Reload Reason          | The system rebooted due to a watchdog |
| Recommendation        | This may indicate a software or hardware problem. |
|                        | Contact your customer support representative. |
| Last Reboot Date (UTC) | 2022-02-01 18:47:18 |
| Time Since Last Reboot | 349 days, 10:25:21.296788 |
| Total Current Uptime   | 349 days, 10:33:51.150000 |
+-----+-----+
Switch online for: 0 years, 11 months, 15 days, 10 hours, 25 minutes
No debug information available
```

Example FPGA error script output

- Sometimes switches require a reboot due to an uncorrectable FPGA error, quick script to validate the error is still there

```
Hostname: my-switch-hostname.com
Model: DCS-7050SX3-48YC8-R, Hardware Revision: 11.15
Serial Number: SGD22233333
OS Version: 4.22.11M
Uptime: 322 days, 5:19:48.190000

+-----+-----+
| pciFpga0 Error Count | 1 |
| pciFpga0 First Occurence (UTC) | 2022-12-19 03:08:06.297093 |
| pciFpga0 Last Occurence (UTC) | 2022-12-19 03:08:06.297093 |
+-----+-----+
```

Power Supply Failure

- Show inventory and grab the power supply section
- Show version in case TAC needed
- Show environment power details

```
+-----+-----+-----+
| PSU Number | PSU Model | Serial Number |
+-----+-----+-----+
| 1          | PWR-3KT-AC-RED | L330000000AVP |
| 3          | PWR-3KT-AC-RED | L330000000AVP |
| 2          | PWR-3KT-AC-RED | L330000000AVP |
| 4          | PWR-3KT-AC-RED | L330000000AVP |
| 7          | PWR-3KT-AC-RED | L330000000AVP |
| 6          | PWR-3KT-AC-RED | L330000000AVP |
+-----+-----+-----+

Hostname: my-switch-hostname.com|
Model: DCS-7508N Hardware Revision: 13.00
Serial Number: HSH11122222
OS Version: 4.26.5M

my-switch-hostname# show environment power

Power                               Input  Output  Output
Supply Model                        Capacity Current Current  Power Status
-----
1      PWR-3KT-AC-RED      3000W   3.34A  57.25A  691.0W Ok
2      PWR-3KT-AC-RED      3000W   3.39A  58.62A  708.0W Ok
3      PWR-3KT-AC-RED      3000W   4.02A  69.75A  842.0W Ok
4      PWR-3KT-AC-RED      3000W   0.00A   0.00A   0.0W Power Loss
6      PWR-3KT-AC-RED      3000W   3.83A  66.38A  801.0W Ok
7      PWR-3KT-AC-RED      3000W   3.53A  60.25A  726.0W Ok
Total  --                    15000W   --      --      3768.0W --

                                Uptime
-----
179 days, 12:43:04
179 days, 12:42:54
 61 days, 21:17:42
  Offline
179 days, 12:42:34
 61 days, 21:17:39
  --
```

Dump show tech & other common requested data for TAC

- You may find that TAC consistently asks you for the same set of files
- Most of these commands (at least in Arista world) return plain text , you may need to strip the “\n”s for the newlines when generating to a proper text file
- Use “`command | json`” to see if you need to request plain text or regular API json results.

What skills do I need to do this?

- For my examples, a working knowledge of python and some time with your vendors API module to understand anything special about their commands.
- Most vendors have APIs and some have python modules to make your interactions even easier.
- My examples are Arista, other platforms have similar options (see last slides for useful links)

Vendor	Python Module
Arista	pyeapi
Juniper	py-junos-eznc
Cisco	No official (off box) one but plenty of user created ones
Nokia	gRPC + profobufs

JSON

- Understanding JSON (JavaScript Object Notation) formatting is useful as most APIs will return data to you in a JSON format
- Data is stored in name/value pairs and separated by commas
- Curly braces hold objects, square brackets hold arrays

Example code (pyeapi)

- Plan to re-use switch API bits in multiple scripts?
 - Consider using a shared library file + class
- Create your “node” aka switch device w/ pyeapi
- Timeout is optional
 - For slow commands (like show tech) add this to reduce timeout errors in the scripts

```
class AristaPyeapi:
    def __init__(self, username, password, switch_hostname, logger=None, timeout=180):
        self.switch_hostname = switch_hostname.strip()
        self.node = pyeapi.connect(
            transport="https",
            host=self.switch_hostname,
            username=username,
            password=password,
            timeout=timeout,
            return_node=True,
        )
```

Show inventory examples (pyeapi)

- Once you get used to the json formatting, finding the data you want is very quick.
- On Arista, from CLI of the switch, you can see what the JSON format for the data looks like by typing `command | json`
- Example extracting inventory from sub-sections of the output

```
def get_inventory(self, type):
    inventory = self.try_eapi_command("show inventory", "enable")
    if type == "interfaces":
        return inventory["xcvrSlots"]
    elif type == "power":
        return inventory["powerSupplySlots"]
    elif type == "storage":
        return inventory["storageDevices"]
    elif type == "system":
        return inventory["systemInformation"]
    elif type == "linecards":
        return inventory["cardSlots"]
    else:
        return inventory
```

Show version example (pyeapi)

- Show commands come back with json formatting, so manipulating them is very simple
- In this example we get show version details and then manipulate them into a new dictionary that is easier to use in our scripts

```
def get_version(self):
    show_version = self.try_eapi_command("show version", "enable")
    switch_eos_version = show_version["version"]
    switch_hardware_rev = show_version["hardwareRevision"]
    switch_model = show_version["modelName"]
    switch_serial_number = show_version["serialNumber"]
    switch_system_mac = show_version["systemMacAddress"]
    switch_uptime = show_version["uptime"]
    if switch_eos_version.startswith(("4.19", "4.20")) is True:
        switch_cli_commands = "old"
    else:
        switch_cli_commands = "new"
    show_version_dict = {
        "eos_version": switch_eos_version,
        "hardware_rev": switch_hardware_rev,
        "model": switch_model,
        "serial_number": switch_serial_number,
        "system_mac": switch_system_mac,
        "uptime": switch_uptime,
        "cli_commands": switch_cli_commands,
    }
    return show_version_dict
```

Current uptime example

- Once we have values, like time of a last reload, it's easy to use normal python to manipulate that and get values like uptime in days.

```
show_version = eapi.get_version()
reload_cause = eapi.get_reload_cause()
if reload_cause["resetCauses"]:
    reload_reason = reload_cause["resetCauses"][0]["description"]
    reload_timestamp = str(
        datetime.datetime.utcnow() -
        datetime.datetime.utcfromtimestamp(reload_cause["resetCauses"][0]["timestamp"])
    )
    now = datetime.datetime.utcnow()
    then = datetime.datetime.utcfromtimestamp(
        reload_cause["resetCauses"][0]["timestamp"]
    )
    difference = str(timedelta(seconds=(now - then).total_seconds()))
    rel_diff = relativedelta(now, then)
    reload_recommendation = reload_cause["resetCauses"][0]["recommendedAction"]
show_version_uptime = show_version["uptime"]
show_version_uptime = str(timedelta(seconds=show_version_uptime))
```

Other use cases for API scripting

- We don't have to limit ourselves to API scripting for troubleshooting efficiency.
- Tedious Tasks:
 - Update port descriptions based on LLDP, ARP and IPv6 Neighbor data (remove human error, and validate patch plans)
 - Pre-upgrade flight checks (LACP on MLAG's, lots of show commands)
- Remediation:
 - Installing extensions, including Security Hot Fixes with validation of if they are already on the box or not
 - Pushing baseline and ACL config remediations
- The possibilities are only limited by your creativity!

No API? There's always SNMP...

- Not all devices have API available
- Next best option is probably SNMP polling (at least for errors, discards, link state)
- For links you'll have to map out the OID's for your ifDescr -> Errors/Discards/Bits etc
- Once logic has been written, SNMP is still a faster way to get this data compared to SSH screen scraping

Optimization

- Try to run your tools scripts locally to the site for the best return times on API, SNMP and other calls
- Cache data when possible to speed up script runs even more (interface names for example)
 - Some companies have Network Source of Truth (NSoT) databases with interfaces stored, query this first before polling the device for actual counters and state based information
- Create outputs formatted in the way best suited for your ticketing systems and tools
 - Example: in one ticket system I use, I can add the following output around my outputs to keep it pre-formatted as a code block:
 - [code]<pre> </pre>[/code]

Security Best Practices

- Don't save API/SSH passwords or SNMP communities hard coded in your scripts
- Prompt with getpass or store in secure environment variables
- Try not to write “current employer” specific code
- Plan ahead for multi-vendor environments

What troubleshooting do you frequently repeat?

- I'm curious about everyone else's use cases
- I love to talk and brainstorm on best practices
- We don't get better by isolating ourselves, and now even more than before it is important for us to stay connected and share ideas, use cases, etc.
- I can be reached at cat@gurinsky.net and I also sit on the #networktocode slack.
- Stay tuned on <https://github.com/shimamizu/> where these scripts will be shared later this week

Q&A



Thank you

Cat Gurinsky

cat@gurinsky.net

Useful Links to Get Started

- Junos
 - https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/rest-api/rest-api.html
- Arista EOS
 - <https://eos.arista.com/arista-eapi-101/>
 - <https://www.arista.com/en/support/hands-on-training>
- NX-OS
 - https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus3000/sw/python/api/python_api/getting_started.html
 - <https://developer.cisco.com/docs/nx-os/#cisco-nexus-9000-series-python-sdk-user-guide-and-api-reference>

Useful links continued

- Cisco + Python examples:
 - https://developer.cisco.com/codeexchange/github/repo/CiscoDevNet/python_code_samples_network/
- Module docs
 - <https://junos-pyez.readthedocs.io/en/2.6.3/>
 - <https://pypi.org/project/junos-eznc/>
 - <https://pyeapi.readthedocs.io/en/latest/>
 - <https://pypi.org/project/pyeapi/>
 - <https://github.com/arista-eosplus/pyeapi>