

Network Automation

Past, present and future

NANOG 71, San Jose, CA

October 2017

Agenda

- Kirk Byers: first steps
- David Barroso: vendor-agnostic automation
- Jeremy Stretch: NetBox IPAM
- Jathan McCollum: NSoT IPAM
- Mircea Ulinic: event-driven automation

Kirk Byers - Bio

- Runs Python for Network Engineers and Ansible Courses
- CCIE (emeritus) in Routing and Switching
- Creator of Netmiko Python library and member of the NAPALM team.
- Runs the SF Network Automation Meetup

Engineers getting started in automation

How to fail at network automation?

1. Start with high-risk, difficult problems.
2. Assume an all-or-nothing mindset (everything has to be automated or nothing can be automated).
3. Try to reinvent everything yourself.
4. Superficially copy code and patterns without comprehension.
5. Fail to learn good debugging processes.
6. [Hugely] over-engineering the solution.

Engineers getting started in automation

How to fail at network automation?

- 7. Fail to apply things that you learned on a small scale.
- 8. Being too busy to automate.
- 9. Fail to learn how to reuse your code [longer term].
- 10. Fail to use available developer tools: Git, linters, unit-testing, CI-tools [longer term].

David Barroso - BIO

- Career
 - Network Systems Engineer @Fastly
 - Network Engineer @Spotify
 - Network Engineer @NTT
 - ...
- NAPALM co-creator



darrosop



@dbarrosop

What is NAPALM?

- Network Automation and Programmability Abstraction Layer with Multivendor support
- Python library
- Abstracts network operations:
 - configuration management
 - retrieving operational state
- Supports many vendors/operating systems:
 - ios, ios-xe, ios-xr, junos, eos, fortios, etc...
- Integrates with ansible, salt, stackstorm, trigger, nsot
- Used by many large-scale networks like Fastly, Linx, Cloudflare, DigitalOcean, Linode and many others which their legal teams don't let us mention.

Why NAPALM?

Focus on your network problems and how to solve them instead of in the gritty details on how to achieve simple tasks like deploying a few lines of configuration for each particular network operating system out there.

```
1 if nos == "junos":
2     from jnpr.junos import Device
3     dev = Device(host='host', user='user', password='passwd' )
4     dev.open()
5     do_junos_things(dev)
6 elif nos == "eos":
7     import pyeapi
8     dev = pyeapi.connect_to('veos01')
9     do_eos_things(dev)
10 elif nos == "ios":
11     from netmiko import ConnectHandler
12     dev = ConnectHandler(**conf)
13     do_ios_things(dev)
```

```
1 from napalm import get_network_driver
2 driver = get_network_driver(nos)
3 with driver(**conf) as dev:
4     do_things(dev)
```


Example (I)

```
>>> with junos_driver(**junos_configuration) as junos:
...     pp.pprint(junos.get_facts())
...
{'fqdn': u'new-hostname',
 'hostname': u'new-hostname',
 'interface_list': [ 'ge-0/0/0',
                     'mt-0/0/0',
                     'vlan'],
 'model': u'FIREFLY-PERIMETER',
 'os_version': u'12.1X47-D20.7',
 'serial_number': u'5b2b599a283b',
 'uptime': 1080,
 'vendor': u'Juniper'}
```

```
>>> with eos_driver(**eos_configuration) as eos:
...     pp.pprint(eos.get_facts())
...
{'fqdn': u'a-new-hostname',
 'hostname': u'a-new-hostname',
 'interface_list': [u'Ethernet1',
                   u'Ethernet2',
                   u'Management1'],
 'model': u'vEOS',
 'os_version': u'4.16.6M',
 'serial_number': u'',
 'uptime': 1217,
 'vendor': u'Arista'}
```

Example (II)

```
>>> with junos_driver(**junos_configuration) as junos:
...     junos.load_merge_candidate(
...         "system {host-name new-hostname;}")
...     )
...     print(device.compare_config())
...
[edit system]
- host-name old-hostname;
+ host-name new-hostname;
>>>
```

```
>>> with eos_driver(**eos_configuration) as eos:
...     eos.load_merge_candidate(
...         'hostname new-hostname'
...     )
...     print(device.compare_config())
...
-hostname old-hostname
+hostname new-hostname
>>>
```

Now with OpenConfig support!!! (I)

Parse native configuration and return an OpenConfig object

```
>>> with eos_device as d:
>>>     running_config = napalm_yang.base.Root()
>>>     running_config.add_model(napalm_yang.models.openconfig_interfaces)
>>>     running_config.parse_config(device=d)
>>>     print(running_config.get(filter=True))
... {
...     "interfaces": {
...         "interface": {
...             "Ethernet1": {
...                 "config": {
...                     "description": "This is a description",
...                     "enabled": True,
...                     "type": "ethernetCsmacd"
...                 },
...             },
...         },
...     }
```

Now with OpenConfig support!!! (II)

Translate
OpenConfig
to native
configuration

```
>>> candidate = napalm_yang.base.Root()
>>> candidate.add_model(napalm_yang.models.openconfig_interfaces())
>>> oc_config = { "interfaces": { "interface": { "et1": {
    "config": {
        "description": "Uplink1",
        "mtu": 9000
    }, ~
    "routed-vlan": { "ipv4": { "addresses": { "address": {
        "192.168.1.1": {
            "config": {
                "ip": "192.168.1.1",
                "prefix-length": 24}}}}}}}}}}}}
>>> candidate.load_dict(oc_config)
>>> print(candidate.translate_config(eos_device.profile))
... interface et1
...   ip address 192.168.1.1/24
...   description Uplink1
...   mtu 9000
...   exit
```

Summary

- NAPALM helps you focus on the “what” rather than on the “how”
- NAPALM brings OpenConfig support to those vendors without support for it (and to those that claim they have)
- NAPALM doesn't pick sides; custom scripts, ansible, salt, stackstorm, trigger, we like you all :)

Jeremy Stretch - Bio

- Sr. Network Developer at DigitalOcean
- Lead maintainer of the NetBox open source IPAM/DCIM application
- Previously known for packetlife.net



jeremystretch



@packetlife

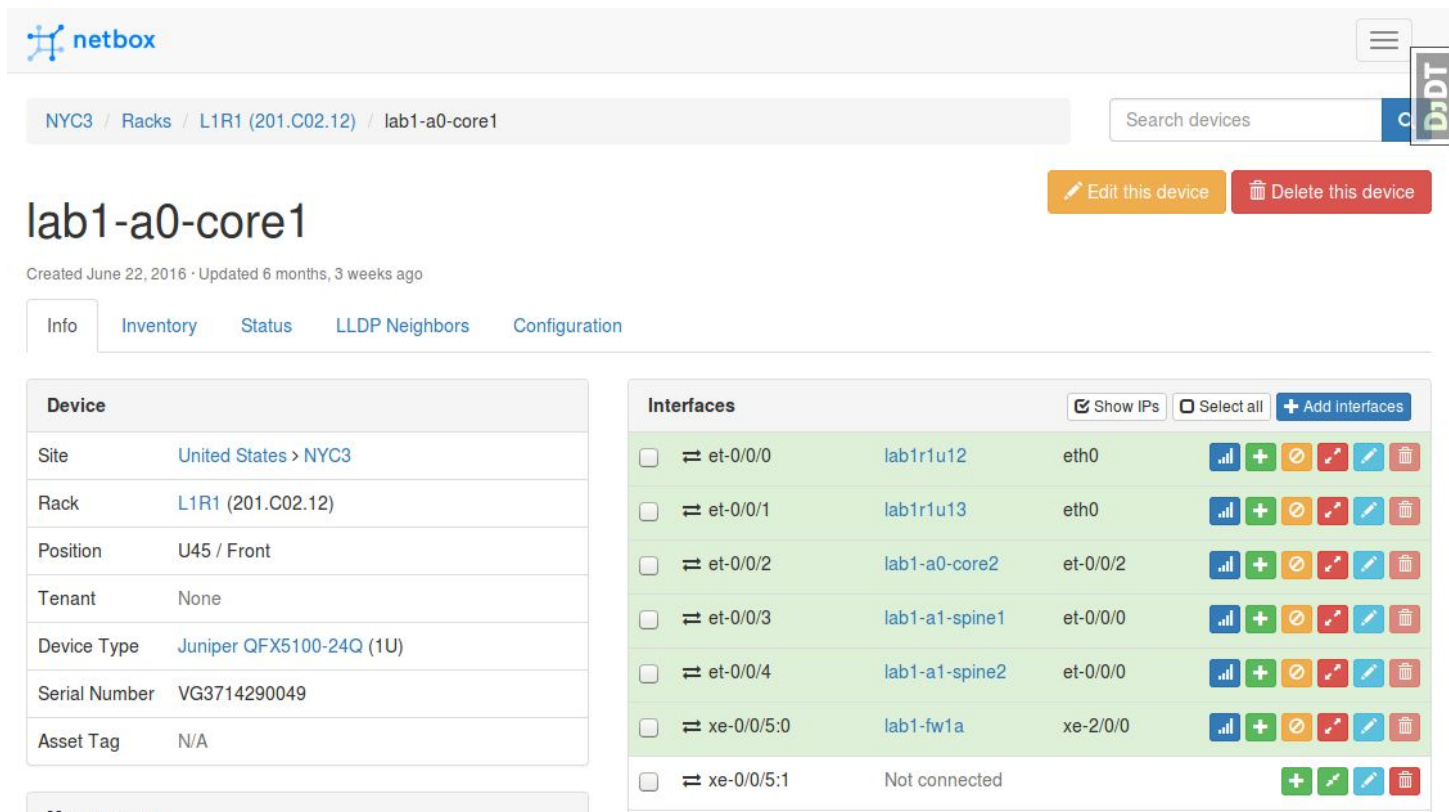
What is IP Address Management (IPAM)?

- A database which contains information about your network's number spaces (IPs, VLANs, VRFs, etc.)
- Functions as an authoritative registry within your organization
- Popular solutions include:
 - Commercial and open source applications
 - Applications developed in-house
 - Spreadsheets
 - Nothing (not a recommended approach)
- https://en.wikipedia.org/wiki/IP_address_management

Why We Built Our Own IPAM Application

- In the beginning, there were spreadsheets
- Started re-evaluating our approach in early 2015
- Common open source limitations
 - Lack of IPv6 and/or VRF support
 - No DCIM functionality (rack elevations, interface connections, etc.)
 - Project no longer actively maintained
- Common commercial limitations
 - Licensed by breadth of IP space/number of objects (\$\$\$!)
 - Paying for features we don't need (DHCP, DNS)
 - No opportunity to expand to meet our needs

Our Solution: NetBox



The screenshot displays the NetBox web interface. At the top, the breadcrumb navigation shows the path: NYC3 / Racks / L1R1 (201.C02.12) / lab1-a0-core1. A search bar for devices is also present. The main title 'lab1-a0-core1' is prominently displayed, with a creation and update timestamp below it. A set of tabs allows switching between different views: Info, Inventory, Status, LLDP Neighbors, and Configuration. The 'Info' tab is currently active, showing a table of device details. To the right, the 'Interfaces' section provides a detailed view of the device's network ports, including their names, connected devices, and interface types. Action buttons for editing and deleting the device are located in the top right corner.

netbox

NYC3 / Racks / L1R1 (201.C02.12) / lab1-a0-core1

Search devices

lab1-a0-core1

Created June 22, 2016 · Updated 6 months, 3 weeks ago

Info Inventory Status LLDP Neighbors Configuration

Device	
Site	United States > NYC3
Rack	L1R1 (201.C02.12)
Position	U45 / Front
Tenant	None
Device Type	Juniper QFX5100-24Q (1U)
Serial Number	VG3714290049
Asset Tag	N/A

Interfaces				
<input type="checkbox"/>	⇌ et-0/0/0	lab1r1u12	eth0	[Status Icons]
<input type="checkbox"/>	⇌ et-0/0/1	lab1r1u13	eth0	[Status Icons]
<input type="checkbox"/>	⇌ et-0/0/2	lab1-a0-core2	et-0/0/2	[Status Icons]
<input type="checkbox"/>	⇌ et-0/0/3	lab1-a1-spine1	et-0/0/0	[Status Icons]
<input type="checkbox"/>	⇌ et-0/0/4	lab1-a1-spine2	et-0/0/0	[Status Icons]
<input type="checkbox"/>	⇌ xe-0/0/5:0	lab1-fw1a	xe-2/0/0	[Status Icons]
<input type="checkbox"/>	⇌ xe-0/0/5:1	Not connected		[Status Icons]

IPAM as a Source of Truth

- Desired vs. operational network state
 - Desired: What you want the network to look like
 - Operational: What it *actually* looks like
 - Very rarely (if ever) are these values the same
- When these states differ, the IPAM/DCIM database functions as the authority to assert what is “correct”

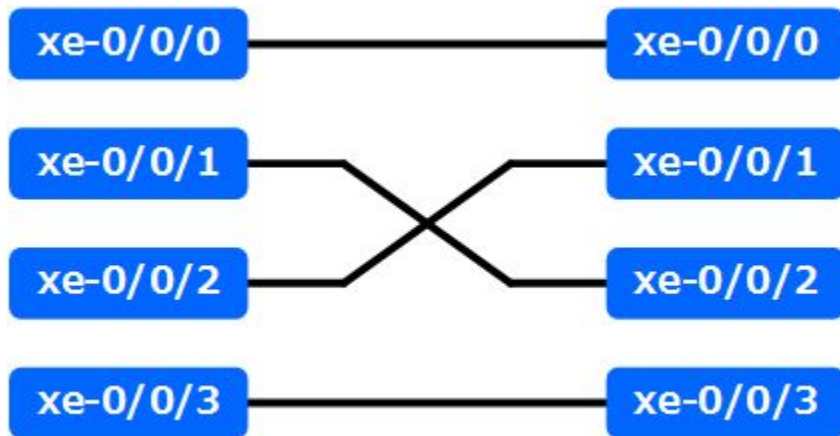
**Maintaining the integrity of
IPAM/DCIM data is crucial**

Populating Initial Data

- Populating the database
 - CSV import (spreadsheet migration)
 - REST API
 - Command line shell
 - Direct database manipulation (use with caution)
- Avoid importing data directly from devices
 - Desired state != operational state
 - Don't blindly grep from network devices
 - Ensure that all data is validated by a human before import

A Cautionary Example

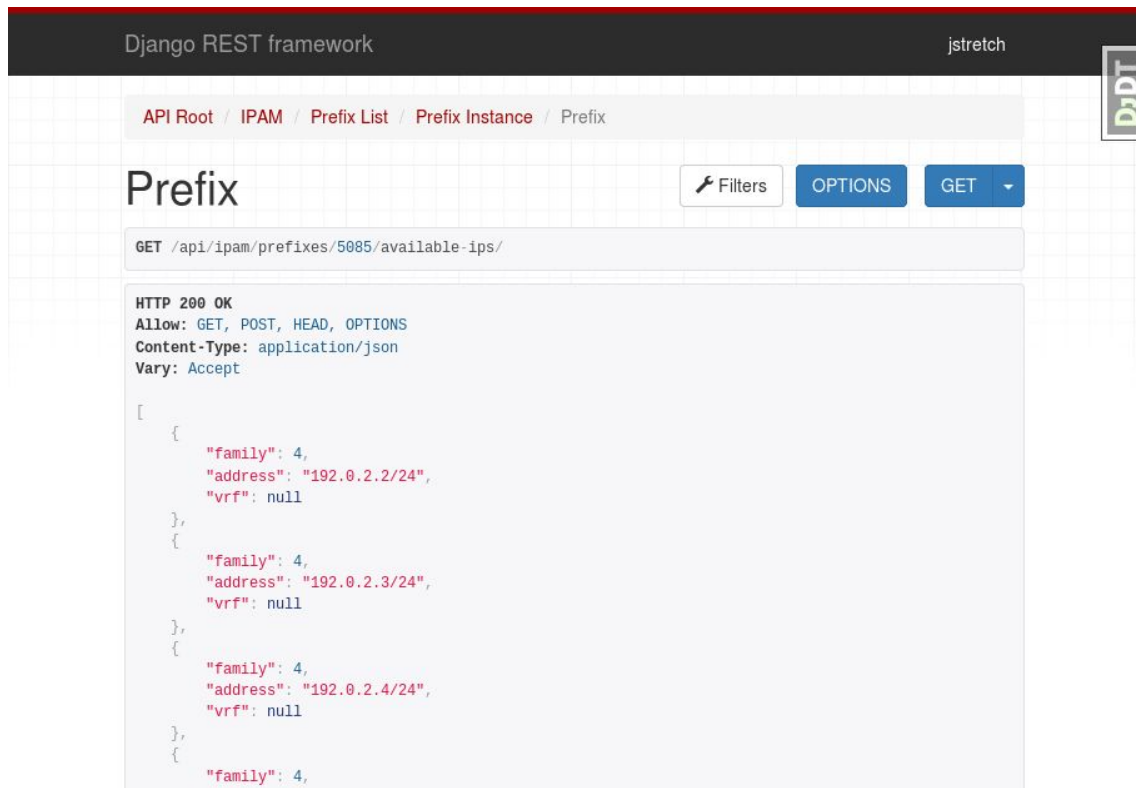
- Two switches with a 4x10GE LAG



IPAM as an Automation Enabler

- Stores information you need to effect automation
 - Device IPs, platform, NAPALM driver, etc.
- Render device configurations from template by providing IPAM data as context
 - Interfaces, IP addresses, VLANs, etc.
- Validate operational state against desired state
 - Example: Compare LLDP data pulled via NAPALM against physical connections defined in NetBox

API Integration



- Leverage REST APIs to integrate with existing applications and processes
- Example: POST to “available IPs” endpoint from ticketing system to provision new IPs

Summary

- Pick an IPAM solution that meets your needs and fits your budget
- Protect your source of truth
 - Always validate data before import
- Pull data from IPAM via its API to generate device configs and validate operational state

Jathan McCollum - Bio

- Network Reliability Engineer at Dropbox
- Maintainer of Network Source of Truth (NSoT), an API-first IPAM and network inventory app
- Maintainer of Trigger, a network automation framework
- Previously in NetEng at AOL and Salesforce



What is NSoT?

- Source of truth
- Inventory
- IP Address Management (IPAM)
- Metadata
- API-first

API-first?

- REST API is first-class citizen
- Everything uses the API
- Browsable API
- Client/CLI
- Bring your own UI

Design Principles

- Ease of install/setup
- It should be easy to get your data in and out
- Feature parity & UX are top priorities
- Customization for any environment
- Loose-coupling between components

Data Model

- Sites (namespaces)
- Attributes (and values)
- Networks (IPAM)
- Devices
- Interfaces
- Circuits
- Changes (event log)

Use it how you want!

- Objects are minimal
- Attributes are where the power lies
- Searching w/ set queries (unions, intersections, differences)
- Intended-state (model-driven) networking
- Discovered data
- Sites as namespaces

NSoT Resources

- NSoT (server)
 - nsot.readthedocs.io
- pyNSoT (client)
 - pynsot.readthedocs.io
- Support
 - Slack (#nsot in slack.networktocode.com)
 - IRC (#nsot on Freenode)

Mircea Ulinic

- Network engineer at Cloudflare
- Prev research and teaching assistant at EPFL, Switzerland
- Member and maintainer at NAPALM Automation
- Integrated NAPALM in Salt
- OpenConfig representative
- <https://mirceaulinic.net/>

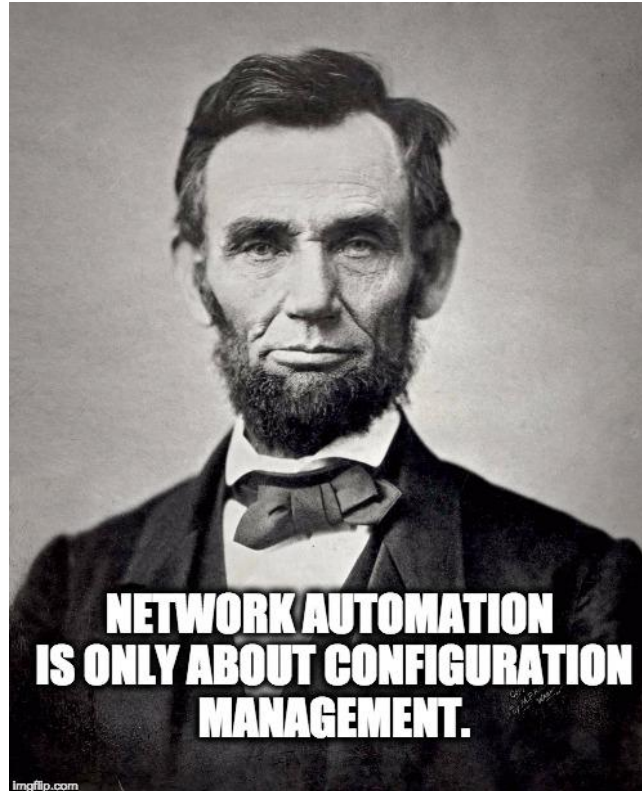


mirceaulinic

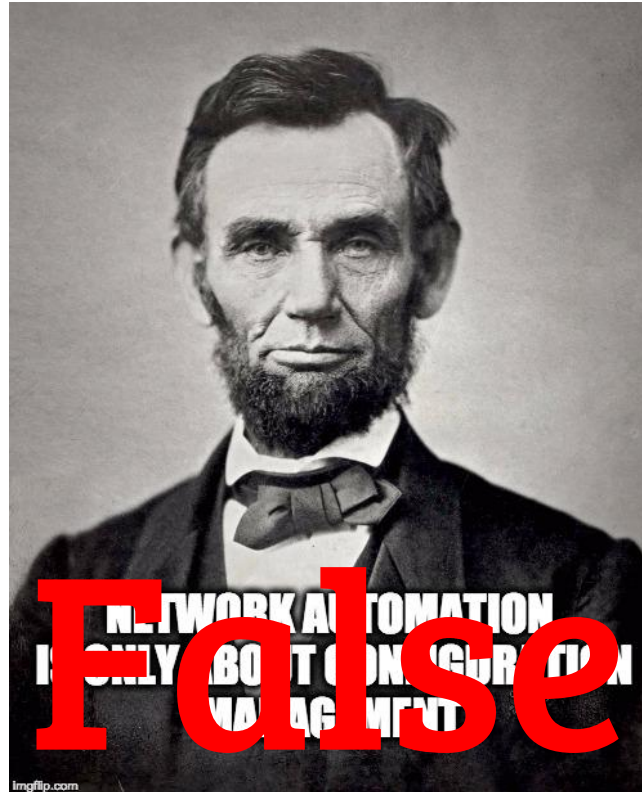


@mirceaulinic

Event-driven network automation (1)



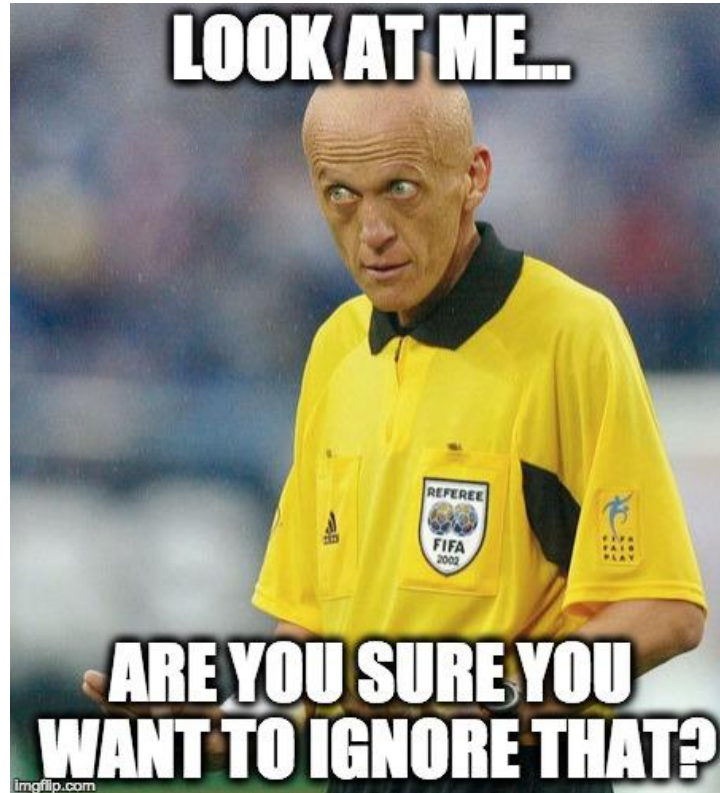
Event-driven network automation (1)



Event-driven network automation (2)

- Several ways your network is trying to communicate with you
 - SNMP traps
 - Syslog messages
 - Streaming telemetry
- Millions of messages

Event-driven network automation (3)



Streaming Telemetry

- Push notifications
 - vs. pull (SNMP)
- Structured data
 - Structured objects, using the [YANG](#) standards
 - [OpenConfig](#)
 - [IETF](#)
- Supported on very new operating systems
 - IOS-XR >= 6.1.1
 - Junos >= 15.1 (depending on the platform)

Syslog messages

- Junos

```
<99>Jul 13 22:53:14 re0.edge01.bjm01 xntpd[16015]: NTP Server 172.17.17.1 is Unreachable
```

- IOS-XR

```
<99>2647599: device3 RP/0/RSP0/CPU0:Aug 21 09:39:14.747 UTC: ntpd[262]: %IP-IP_NTP-5-SYNC_LOSS : Synchronization lost :  
172.17.17.1 : The association was removed
```

Syslog messages: napalm-logs (1)

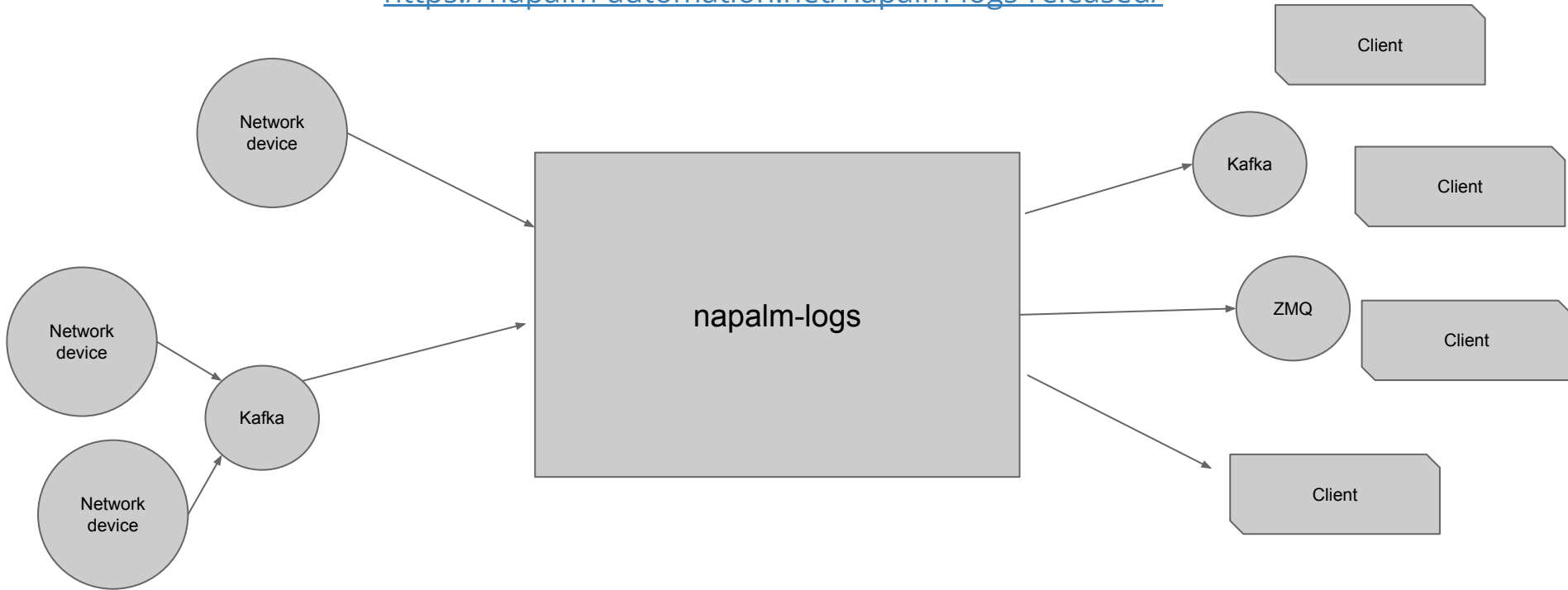
<https://napalm-automation.net/napalm-logs-released/>



- Listen for syslog messages
 - Directly from the network devices, via UDP or TCP
 - Other systems: Apache Kafka, ZeroMQ, etc.
- Publish encrypted messages
 - Structured documents, using the [YANG](#) standards
 - [OpenConfig](#)
 - [IETF](#)
 - Over various channels: ZeroMQ, Kafka, etc.

Syslog messages: napalm-logs (2)

<https://napalm-automation.net/napalm-logs-released/>



Syslog messages: napalm-logs structured objects

```
{
  "error": "NTP_SERVER_UNREACHABLE",
  "facility": 12,
  "host": "edge01.bjm01",
  "ip": "10.10.0.1",
  "os": "junos",
  "timestamp": 1499986394,
  "yang_message": {
    "system": {
      "ntp": {
        "servers": {
          "server": {
            "172.17.17.1": {
              "state": {
                "association-type": "SERVER",
                "stratum": 16
              }
            }
          }
        }
      }
    }
  },
  "yang_model": "openconfig-system"
}
```


Salt event system

Salt is a [data driven automation framework](#). Each action (job) performed (manually from the CLI or automatically by the system) is uniquely identified and has an identification *tag*:

```
$ sudo salt junos-router net.arp
# output omitted
```



```
$ sudo salt-run state.event pretty=True
salt/job/20170110130619367337/new {
  "_stamp": "2017-01-10T13:06:19.367929",
  "arg": [],
  "fun": "net.arp",
  "jid": "20170110130619367337",
  "minions": [
    "junos-router"
  ],
  "tgt": "junos-router",
  "tgt_type": "glob",
  "user": "mircea"
}
```

Tag

Syslog messages: *napalm-syslog* Salt engine (1)

https://docs.saltstack.com/en/latest/ref/engines/all/salt.engines.napalm_syslog.html

Imports messages from *napalm-logs* into the Salt event bus

```
/etc/salt/master
```

```
engines:  
  - napalm_syslog:  
    transport: zmq  
    address: 10.10.0.1  
    port: 49017  
    auth_address: 10.10.0.2  
    auth_port: 49018
```

Syslog messages: Napalm-syslog Salt engine (2)

Salt event bus:

```
napalm/syslog/junos/NTP_SERVER_UNREACHABLE/edge01.bjm01 {  
  "error": "NTP_SERVER_UNREACHABLE",  
  "facility": 12,  
  "host": "edge01.bjm01",  
  "ip": "10.10.0.1",  
  "os": "junos",  
  "timestamp": 1499986394,  
  "yang_message": {  
    "system": {  
      "ntp": {  
        "servers": {  
          "server": {  
            "172.17.17.1": {  
              "state": {  
                "association-type": "SERVER",  
                "stratum": 16  
              }  
            }  
          }  
        }  
      }  
    }  
  },  
  "yang_model": "openconfig-system"  
}
```

Fully automated configuration changes

/etc/salt/master

reactor:

- 'napalm/syslog/*/NTP_SERVER_UNREACHABLE/*':
- salt://reactor/exec_ntp_state.sls

/etc/salt/reactor/exec_ntp_state.sls

triggered NTP state:

cmd.state.sls:

- tgt: {{ data.host }}
- arg:
- ntp

Matches the event tag

napalm/syslog/junos/NTP_SERVER_UNREACHABLE/edge01.bjm01

CLI Equivalent:

\$ sudo salt edge01.bjm01 state.sls ntp