CLOUDFLARE®

Event-driven Network Automation and Orchestration

Mircea Ulinic
Cloudflare, London

RIPE 76
Marseille, May 2018

Mircea Ulinic

- Network software engineer at Cloudflare
- Previously research and teaching assistant at EPFL, Switzerland
- Member and maintainer at NAPALM Automation
- SaltStack contributor of the year 2017
- O’Reilly author
- OpenConfig representative
- https://mirceaulinic.net/

mirceaulinic @mirceaulinic

Cloudflare

- How big?
  - 7+ million zones/domains
  - Authoritative for ~40% of Alexa top 1 million
  - 200 million Internet users served
  - 100+ billion DNS queries/day
    - Largest
    - Fastest
    - 35% of the Internet requests
    - Now also a resolver (1.1.1.1)
  - 10 trillion requests / month
  - 10% of the Internet traffic
- 150+ anycast locations globally
  - 74 countries (and growing)
  - Many hundreds of network devices

Agenda

- Why automate and how to start
- Vendor-agnostic automation using Salt
- YANG
- Using *napalm-logs* for event-driven network automation
- Live demo

To automate, I have to learn Python or another programming language.

To automate, I have to learn Python or another programming language.

WRONG!

Do not jump into implementation.
Design first!

What’s the best tool?

Wrong question.

~~What’s the best tool?~~

What’s the best tool for my network?

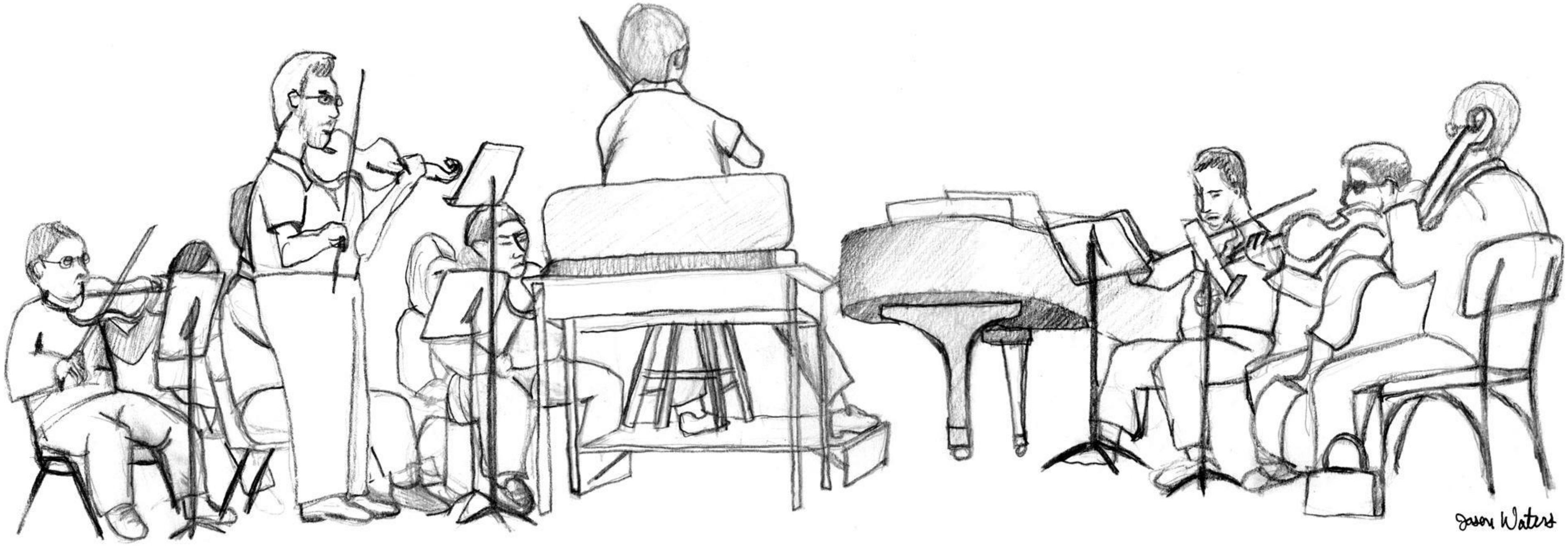
What’s the best tool for my network?

- How large is your network?
- How many platforms / operating systems?
- How dynamic?
- External sources of truth? e.g., IPAM
- Do you need native caching? REST API?
- Event-driven automation?
- **Community**

Why Salt

- Very scalable
- Concurrency
- Event-driven automation
- Easily configurable & customizable
- Native caching and drivers for useful tools
- One of the friendliest communities
- Great documentation

Why Salt
Orchestration vs. Automation



Why Salt

“

In SaltStack, speed isn’t a byproduct, it is a design goal. SaltStack was created as an extremely fast, lightweight communication bus to provide the foundation for a remote execution engine.

SaltStack now provides orchestration, configuration management, event reactors, cloud provisioning, and more, all built around the SaltStack high-speed communication bus.

… + cross-vendor network automation from 2016.11 (Carbon) ”

https://docs.saltstack.com/en/getstarted/speed.html

Who’s Salty

SAP

Linked in

linode

Adobe

CLOUDFLARE

COMCAST

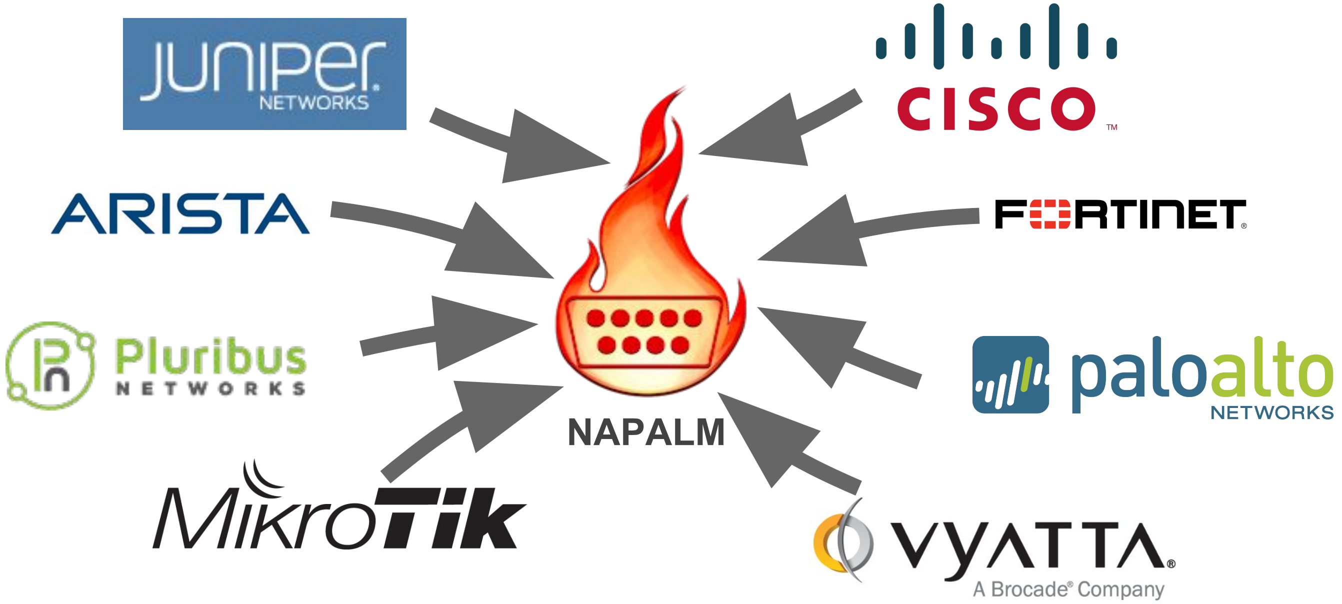
DigitalOcean

KPMG

Bloomberg

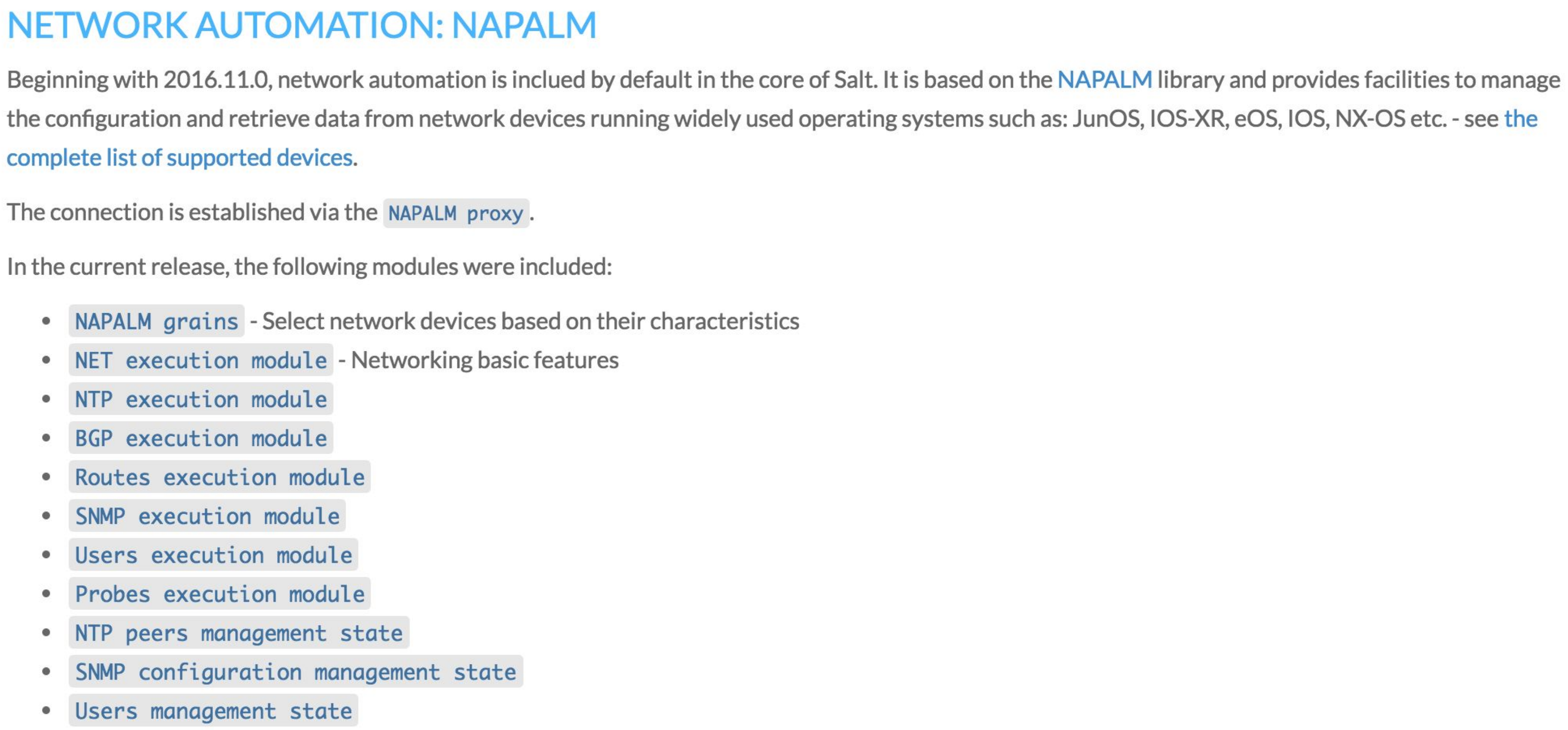
Vendor-agnostic API: NAPALM

(Network Automation and Programmability Abstraction Layer with Multivendor support)



https://github.com/napalm-automation

NAPALM integrated in Salt: Carbon



NETWORK AUTOMATION: NAPALM

Beginning with 2016.11.0, network automation is inclued by default in the core of Salt. It is based on the NAPALM library and provides facilities to manage the configuration and retrieve data from network devices running widely used operating systems such as: JunOS, IOS-XR, eOS, IOS, NX-OS etc. - see the complete list of supported devices.

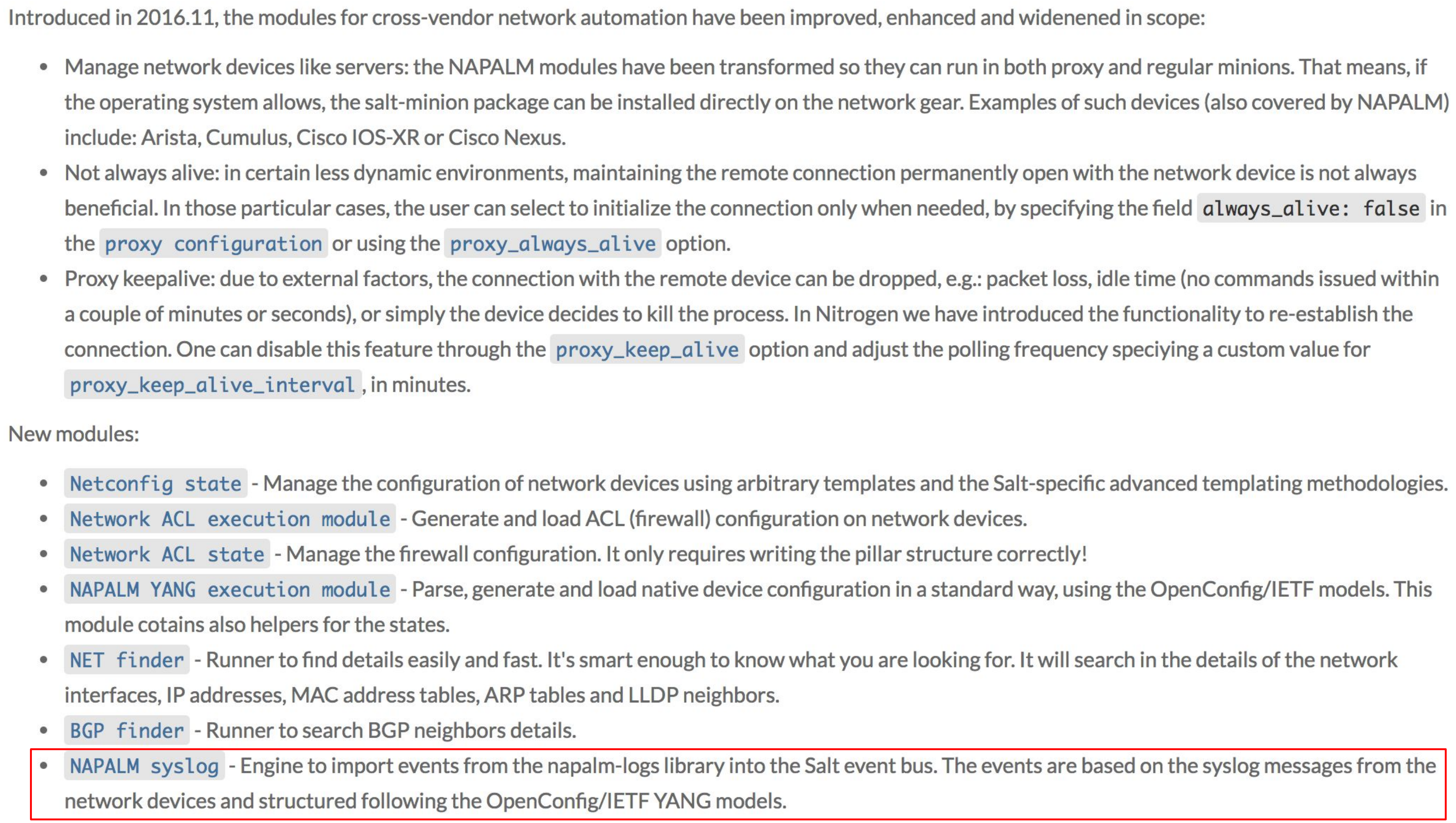
The connection is established via the `NAPALM proxy`.

In the current release, the following modules were included:

- `NAPALM grains` - Select network devices based on their characteristics
- `NET execution module` - Networking basic features
- `NTP execution module`
- `BGP execution module`
- `Routes execution module`
- `SNMP execution module`
- `Users execution module`
- `Probes execution module`
- `NTP peers management state`
- `SNMP configuration management state`
- `Users management state`

https://docs.saltstack.com/en/develop/topics/releases/2016.11.0.html

NAPALM integrated in Salt: Nitrogen



Introduced in 2016.11, the modules for cross-vendor network automation have been improved, enhanced and widenened in scope:

- Manage network devices like servers: the NAPALM modules have been transformed so they can run in both proxy and regular minions. That means, if the operating system allows, the salt-minion package can be installed directly on the network gear. Examples of such devices (also covered by NAPALM) include: Arista, Cumulus, Cisco IOS-XR or Cisco Nexus.
- Not always alive: in certain less dynamic environments, maintaining the remote connection permanently open with the network device is not always beneficial. In those particular cases, the user can select to initialize the connection only when needed, by specifying the field `always_alive: false` in the `proxy configuration` or using the `proxy_always_alive` option.
- Proxy keepalive: due to external factors, the connection with the remote device can be dropped, e.g.: packet loss, idle time (no commands issued within a couple of minutes or seconds), or simply the device decides to kill the process. In Nitrogen we have introduced the functionality to re-establish the connection. One can disable this feature through the `proxy_keep_alive` option and adjust the polling frequency speciying a custom value for `proxy_keep_alive_interval`, in minutes.

New modules:

- `Netconfig state` - Manage the configuration of network devices using arbitrary templates and the Salt-specific advanced templating methodologies.
- `Network ACL execution module` - Generate and load ACL (firewall) configuration on network devices.
- `Network ACL state` - Manage the firewall configuration. It only requires writing the pillar structure correctly!
- `NAPALM YANG execution module` - Parse, generate and load native device configuration in a standard way, using the OpenConfig/IETF models. This module cotains also helpers for the states.
- `NET finder` - Runner to find details easily and fast. It's smart enough to know what you are looking for. It will search in the details of the network interfaces, IP addresses, MAC address tables, ARP tables and LLDP neighbors.
- `BGP finder` - Runner to search BGP neighbors details.
- `NAPALM syslog` - Engine to import events from the napalm-logs library into the Salt event bus. The events are based on the syslog messages from the network devices and structured following the OpenConfig/IETF YANG models.

https://docs.saltstack.com/en/develop/topics/releases/nitrogen.html

Vendor-agnostic automation (1)

```
$ sudo salt junos-router net.arp
junos-router:
    ----------
    out:
        |_
          ----------
          age:
              129.0
          interface:
              ae2.100
          ip:
              10.0.0.1
          mac:
              84:B5:9C:CD:09:73
        |_
          ----------
          age:
              1101.0
```

```
$ sudo salt iosxr-router net.arp
iosxr-router:
    ----------
    out:
        |_
          ----------
          age:
              1620.0
          interface:
              Bundle-Ether4
          ip:
              10.0.0.2
          mac:
              00:25:90:20:46:B5
        |_
          ----------
          age:
              8570.0
```

Vendor-agnostic automation (2)

```
$ sudo salt junos-router state.sls ntp
junos-router:
----------
          ID: oc_ntp_netconfig
    Function: netconfig.managed
      Result: True
     Comment: Configuration changed!
     Started: 10:53:25.624396
    Duration: 3494.153 ms
     Changes:
              ----------
              diff:
                  [edit system ntp]
                  -     peer 172.17.17.2;
                  [edit system ntp]
                  +     server 10.10.10.1 prefer;
                  +     server 10.10.10.2;
                  -     server 172.17.17.1 version 2 prefer;
```

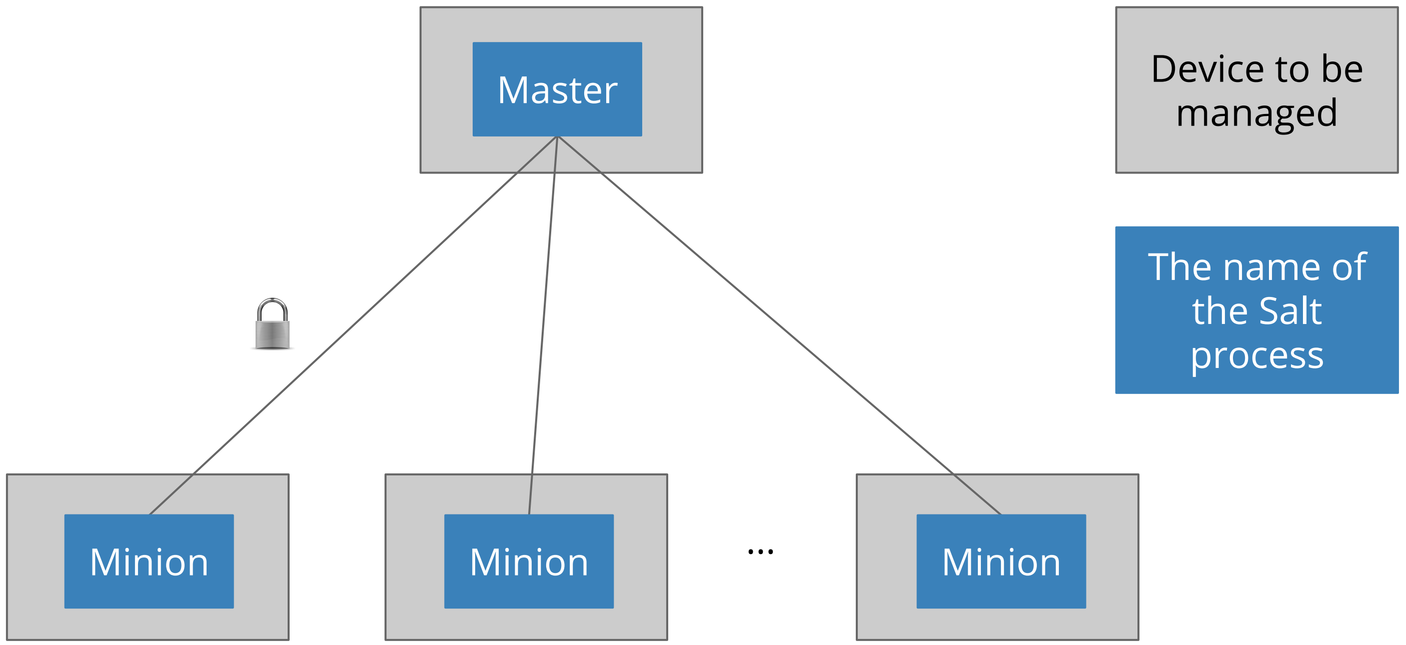
```
$ sudo salt iosxr-router state.sls ntp
iosxr-router:
----------
          ID: oc_ntp_netconfig
    Function: netconfig.managed
      Result: True
     Comment: Configuration changed!
     Started: 11:02:39.162423
    Duration: 3478.683 ms
     Changes:
              ----------
              diff:
                  ---
                  +++
                  @@ -1,4 +1,10 @@
                  +ntp
                  + server 10.10.10.1 prefer
                  + server 10.10.10.2
                   !
```

Vendor-agnostic automation: how to

- Salt in 10 minutes
- Salt fudamentals
- Configuration management
- Network Automation official Salt docs
- Step-by-step tutorial -- up and running in 60 minutes
- Using Salt at Scale

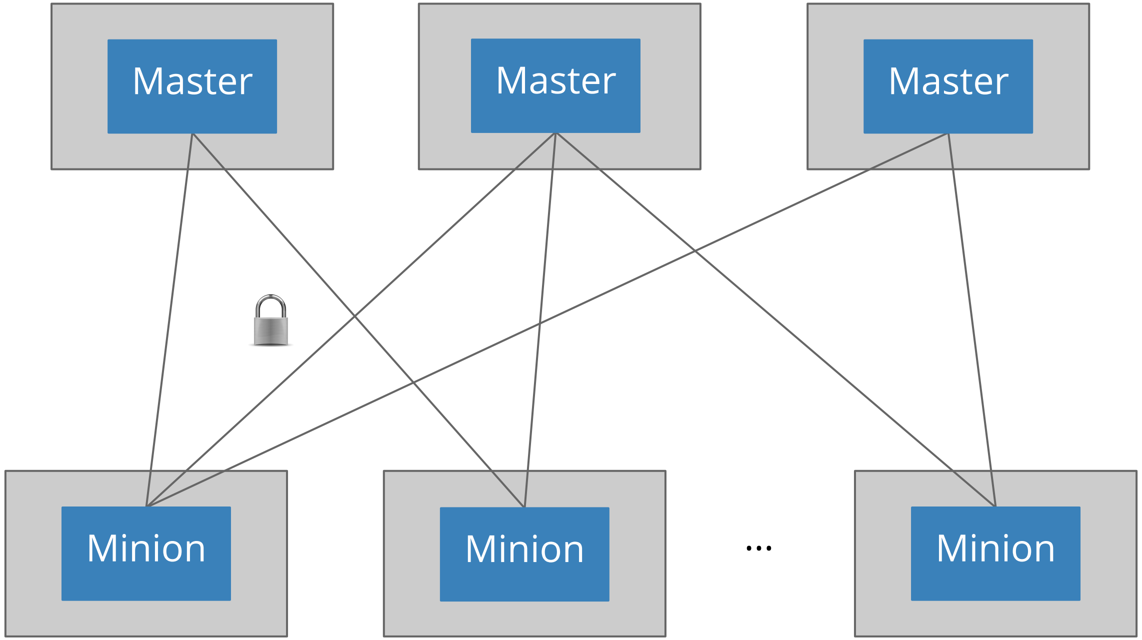
Introduction to Salt

Salt Architectures: typical hub and spoke



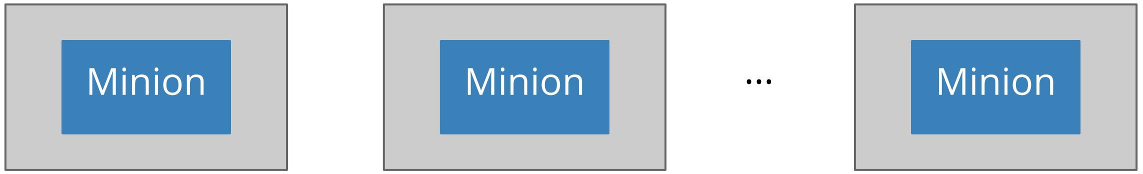
https://docs.saltstack.com/en/latest/topics/topology/index.html

Salt Architectures: multiple Masters



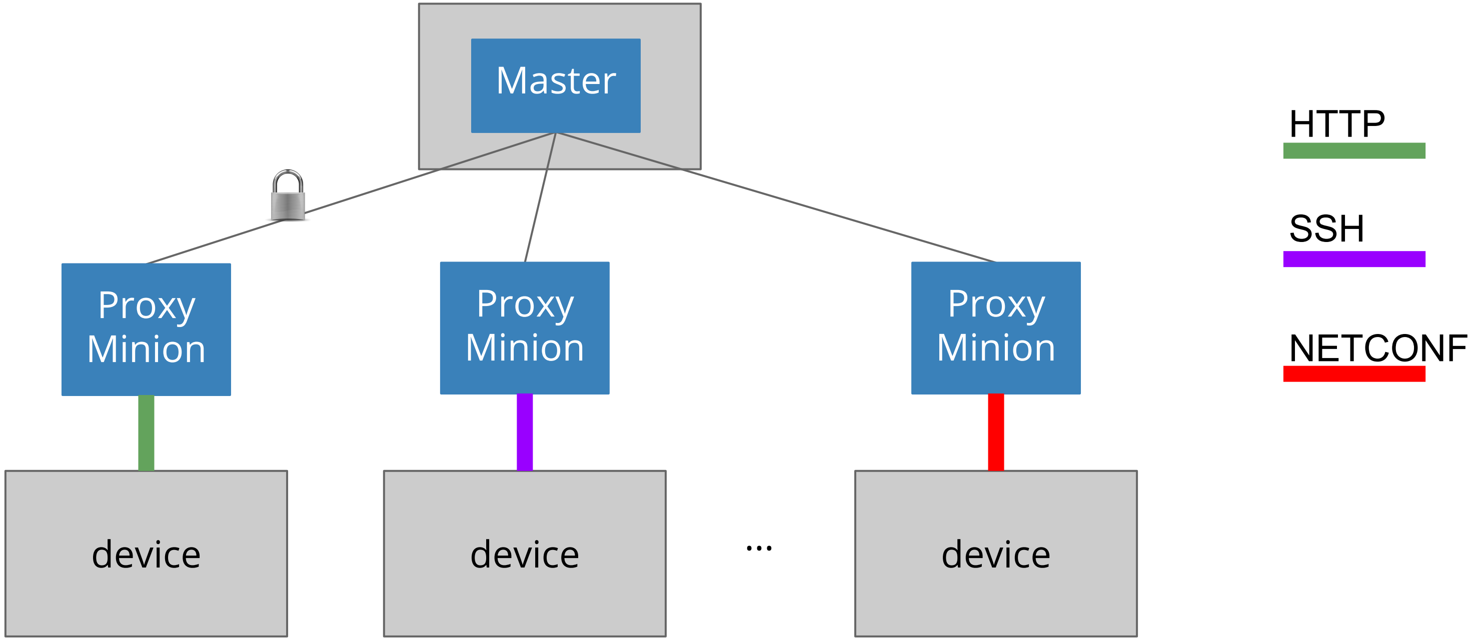
Introduction to Salt

Salt Architectures: Masterless



Introduction to Salt

Salt Architectures: Proxy Minions



https://docs.saltstack.com/en/latest/topics/topology/index.html

Introduction to Salt

Nomenclature

Pillar

Free-form data that can be used to organize configuration values or manage sensitive data, e.g.: interface details, NTP peers, BGP config...

Data provided by the user (as file, HTTP API, database, etc.)

Grains

Data collected from the device, e.g., device model, vendor, uptime, serial number etc.

Salt handles this, you don’t need to do anything

Salt in 10 minutes: https://docs.saltstack.com/en/latest/topics/tutorials/walkthrough.html

Introduction to Salt
Nomenclature

SLS

File format used by Salt in various subsystems.

In can be used for both data and automation logic.

By default, SLS = Jinja + YAML.

Can be changed to any other Renderer combination.

Introduction to Salt

SLS

The following SLS files are equivalent:

Pure YAML

```
list_with_ten_elements:
  - 0
  - 1
  - 2
  - 3
  - 4
  - 5
  - 6
  - 7
  - 8
  - 9
```

Default SLS (Jinja + YAML)

```
list_with_ten_elements:
{%- for i in range(10) %}
  - {{ i }}
{%- endfor %}
```

Other combination: Pure Python

```
#!py

def run():
    return {
        'list_with_ten_elements': [ i
for i in range(10) ]
    }
```

YANG

YANG

- Data modeling language
  - A language of its own
  - It is not XML, JSON, YAML etc.
  - It describes hierarchies and the types of data
- Standardised in RFC 6020
- Aims to solve the vendor discrepancy in terms of operational and configuration data

YANG for dummies:
https://napalm-automation.net/yang-for-dummies/

YANG
A quick example

```
// module name
module napalm-star-wars {

    grouping personal-data {
        leaf name {
            type string;
        }
        leaf age {
            type age;
        }
    }

    // this is the root object defined by the model
    container universe {
        list individual {
            // identify each individual by using the name as key
            key "name";

            uses personal-data;
        }
    }
}
```

YANG
A quick example: the structure

```
$ pyang -f tree napalm-star-wars.yang
module: napalm-star-wars
    +--rw universe
        +--rw individual* [name]
           +--rw name            string
           +--rw age?            age
```

YANG
A quick example

A JSON document following the hierarchy defined in the YANG model defined previously.

```
{
    "universe": {
        "individual": {
            "Obi-Wan Kenobi": {
                "age": 57,
                "name": "Obi-Wan Kenobi"
            },
            "Luke Skywalker": {
                "age": 19,
                "name": "Luke Skywalker"
            },
            "Darth Vader": {
                "age": 42,
                "name": "Darth Vader"
            },
            "Yoda": {
                "age": 896,
                "name": "Yoda"
            }
        }
    }
}
```

YANG
A quick example

A XML document following the hierarchy defined in the YANG model defined previously.

```
<universe>
  <individual>
    <Luke Skywalker>
      <age>
        19
      </age>
      <name>
        Luke Skywalker
      </name>
    </Luke Skywalker>
    <Darth Vader>
      <age>
        42
      </age>
      <name>
        Darth Vader
      </name>
    <Yoda>
      <age>
        896
      </age>
      <name>
        Yoda
      </name>
    </Yoda>
  </individual>
```

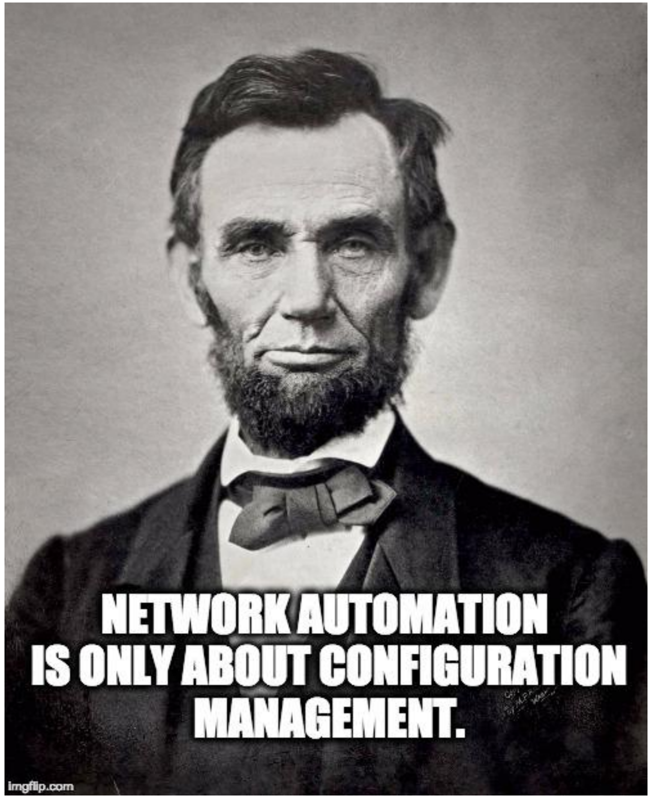
YANG

Standards Organizations

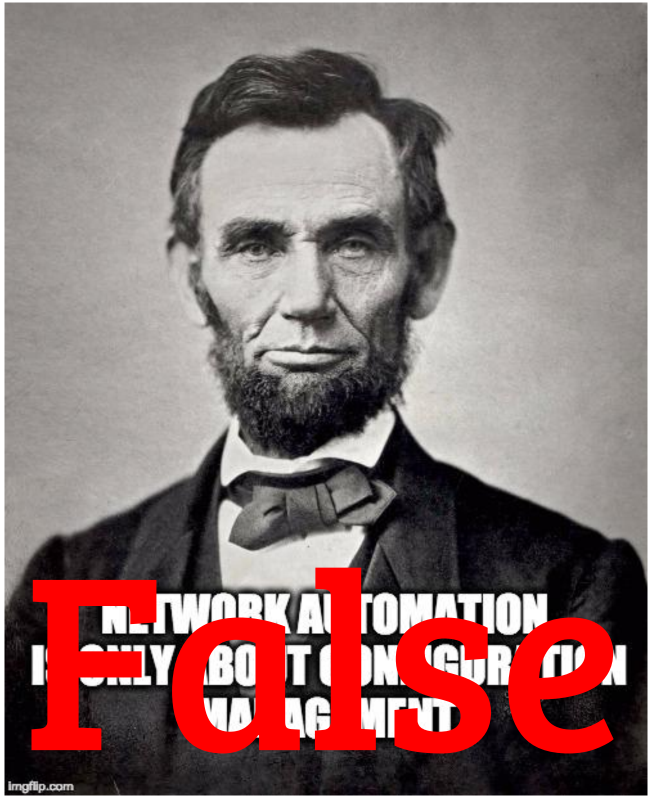
- OpenConfig
    - OpenConfig is an informal working group of network operators
    - YANG models available at https://github.com/openconfig/public
- IETF
- IEEE
- BBF (Broadband Forum)

Event-driven automation

Event-driven network automation (1)



Event-driven network automation (1)



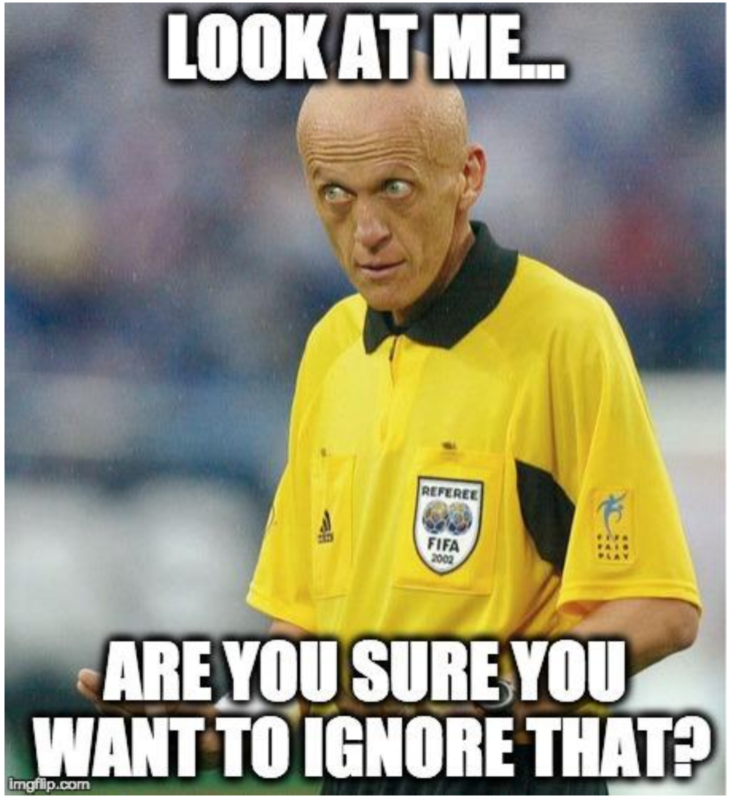
Event-driven network automation (2)

- Several of ways your network is trying to communicate with you
- Millions of messages

Event-driven network automation (3)

- SNMP traps
- Syslog messages
- Streaming telemetry

Event-driven network automation (4)



Event-driven network automation
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)

Event-driven network automation
Syslog messages

- Junos

```
<99>Jul 13 22:53:14 device1 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
```

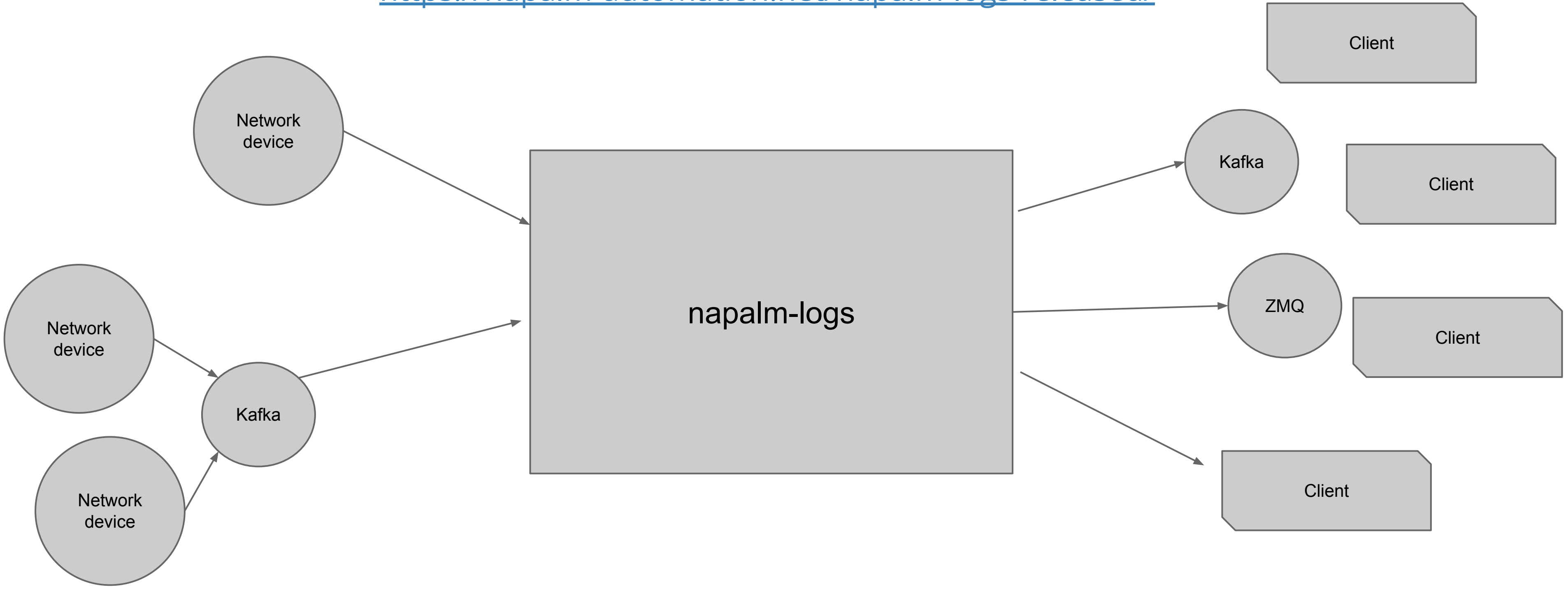
Event-driven network automation
Syslog messages: napalm-logs (1)

https://napalm-logs.com

- 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.

Event-driven network automation
Syslog messages: napalm-logs (2)

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



Event-driven network automation
Syslog messages: napalm-logs startup

```
$ napalm-logs --listener udp --address 172.17.17.1 --port 5514  --publish-address 172.17.17.2 --publish-port 49017

              --publisher zmq --disable-security
```

More configuration options:
https://napalm-logs.readthedocs.io/en/latest/options/index.html

Event-driven network automation
Syslog messages: napalm-logs clients

```python
import zmq  # when using the ZeroMQ publisher
import napalm_logs.utils

server_address = '127.0.0.1'  # IP
server_port = 49017           # Port for the napalm-logs publisher interface

context = zmq.Context()
socket = context.socket(zmq.SUB)
socket.connect('tcp://{address}:{port}'.format(address=server_address,
                                               port=server_port))
socket.setsockopt(zmq.SUBSCRIBE, '')  # subscribe to the napalm-logs publisher

while True:
    raw_object = socket.recv()  # binary object
    print(napalm_logs.utils.unserialize(raw_object))  # deserialize
```

More complete example:
https://github.com/napalm-automation/napalm-logs/blob/master/examples/client_auth.py

Event-driven network automation
Syslog messages (again)

- Junos

```
<99>Jul 13 22:53:14 device1 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
```

Event-driven network automation

Syslog messages: *napalm-logs* structured objects

```
{
  "error": "NTP_SERVER_UNREACHABLE",
  "facility": 12,
  "host": "device1",
  "ip": "127.0.0.1",
  "os": "junos",
  "severity": 4,
  "timestamp": 1499986394,
  "yang_message": {
      "system": {
          "ntp": {
              "servers": {
                  "server": {
                      "172.17.17.1": {
                          "state": {
                              "stratum": 16,
                              "association-type": "SERVER"
                          }
                      }
                  }
              }
          }
      }
  },
  "yang_model": "openconfig-system"
}
```

Other raw syslog message example

- Junos

```
<149>Jun 21 14:03:12  vmx01 rpd[2902]: BGP_PREFIX_THRESH_EXCEEDED: 192.168.140.254 (External AS 4230): Configured maximum
prefix-limit threshold(140) exceeded for inet4-unicast nlri: 141 (instance master)
```

- IOS-XR

```
<149>2647599: xrv01 RP/0/RSP1/CPU0:Mar 28 15:08:30.941 UTC: bgp[1051]: %ROUTING-BGP-5-MAXPFX : No. of IPv4 Unicast prefixes
received from 192.168.140.254 has reached 94106, max 12500
```

Event-driven network automation

Syslog messages: *napalm-logs* structured objects

```
"yang_message": {
      "bgp": {
          "neighbors": {
              "neighbor": {
                  "192.168.140.254": {
                      "afi_safis": {
                          "afi_safi": {
                              "inet4": {
                                  "ipv4_unicast": {
                                      "prefix_limit": {
                                          "state": {
                                              "max_prefixes": 140
                                          }
                                      }
                                  },
                                  "state": {
                                      "prefixes": {
                                          "received": 141
                                      }
                                  }
                              }
                          }
                      },
                      "state": {
                          "peer_as": "4230"
                      }
                  }
              }
          }
      }
  },
  "yang_model": "openconfig-bgp"
}
```

Event-driven network automation
napalm-logs key facts to remember

- Continuously listening to syslog messages
- Continuously publishing structured data
    - Structure following the YANG standards
        - OpenConfig
        - IETF

Event-driven network automation

Salt event system

Salt is a data driven system. 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"
}
```

Unique job tag

Event-driven network automation

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: 172.17.17.2
      port: 49017
      auth_address: 172.17.17.3
      auth_port: 49018
```

Event-driven network automation

Syslog messages: *napalm-logs* structured objects

(from slide #**43**)

```
{
  "error": "NTP_SERVER_UNREACHABLE",
  "facility": 12,
  "host": "device1",
  "ip": "127.0.0.1",
  "os": "junos",
  "severity": 4,
  "timestamp": 1499986394,
  "yang_message": {
      "system": {
          "ntp": {
              "servers": {
                  "server": {
                      "172.17.17.1": {
                          "state": {
                              "stratum": 16,
                              "association-type": "SERVER"
                          }
                      }
                  }
              }
          }
      }
  },
  "yang_model": "openconfig-system"
}
```

Event-driven network automation
Salt event bus

Using the *napalm-syslog* Salt engine you can inject *napalm-logs* events into the Salt event bus.

See
https://napalm-automation.net/napalm-logs-released/
and
https://mirceaulinic.net/2017-10-19-event-driven-network-automation/
For more examples

```
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"
}
```

Event-driven network automation
Fully automated configuration changes

/etc/salt/master

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

Matches the event tag

napalm/syslog/junos/NTP_SERVER_UNREACHABLE/edge01.bjm01

/etc/salt/reactor/exec_ntp_state.sls

```
triggered NTP state:
  cmd.state.sls:
    - tgt: {{ data.host }}
    - arg:
      - ntp
```

CLI Equivalent:
(see slide **#20**)

```
$ sudo salt edge01.bjm01 state.sls ntp
```

Event-driven network automation
Fully automated configuration changes & more

`/etc/salt/master`

```
reactor:
  - 'napalm/syslog/*/INTERFACE_DOWN/*':
    - salt://reactor/if_down_shutdown.sls
    - salt://reactor/if_down_send_mail.sls
```

Shutdown the interface

Send an email notification

Matches the event tag

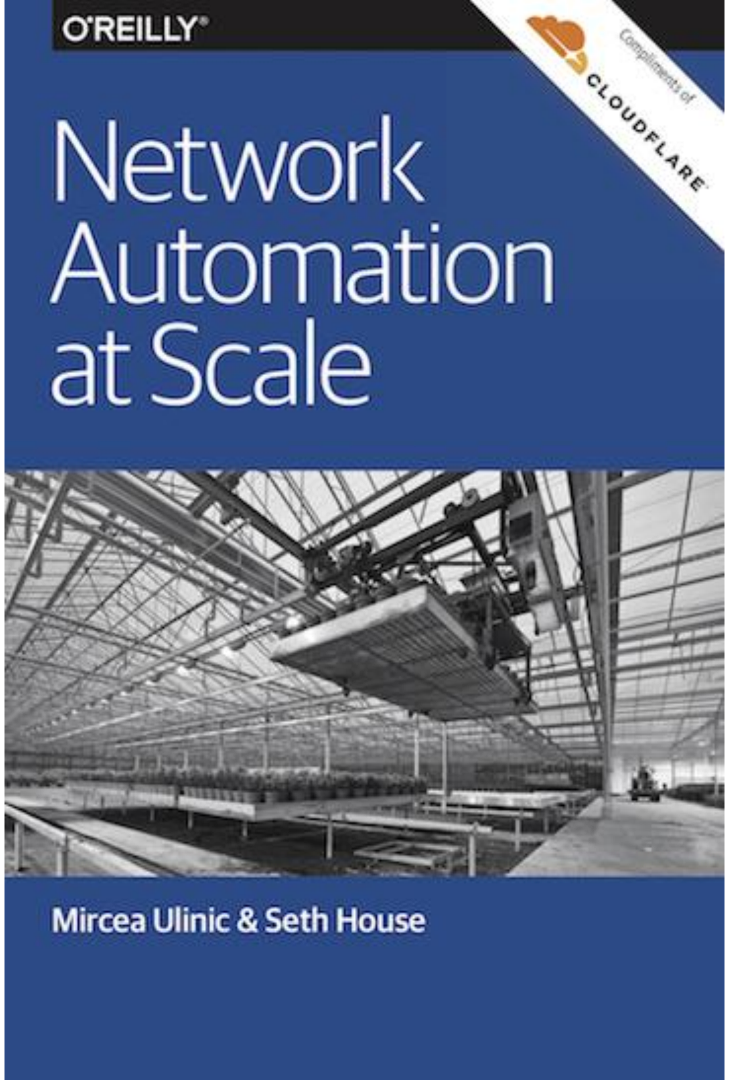
`napalm/syslog/junos/INTERFACE_DOWM/edge01.bjm01`

(Event pushed when an interface is operationally down)

More details at:
https://mirceaulinic.net/2017-10-19-event-driven-network-automation/

Network Automation at Scale: the book

Free download:
https://www.cloudflare.com/network-automation-at-scale-ebook/

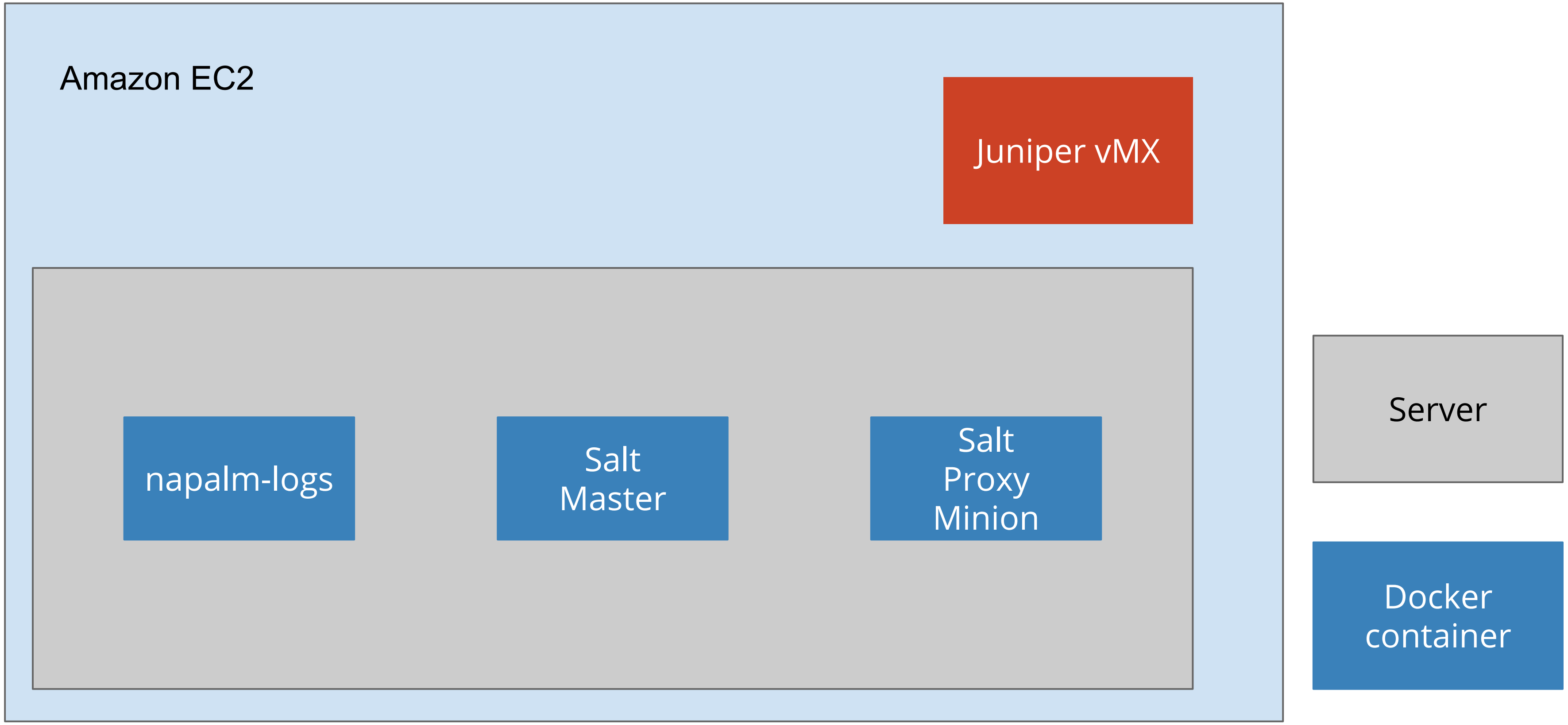

O'REILLY
Compliments of CLOUDFLARE
Network Automation at Scale
Mircea Ulinic & Seth House

Live demo

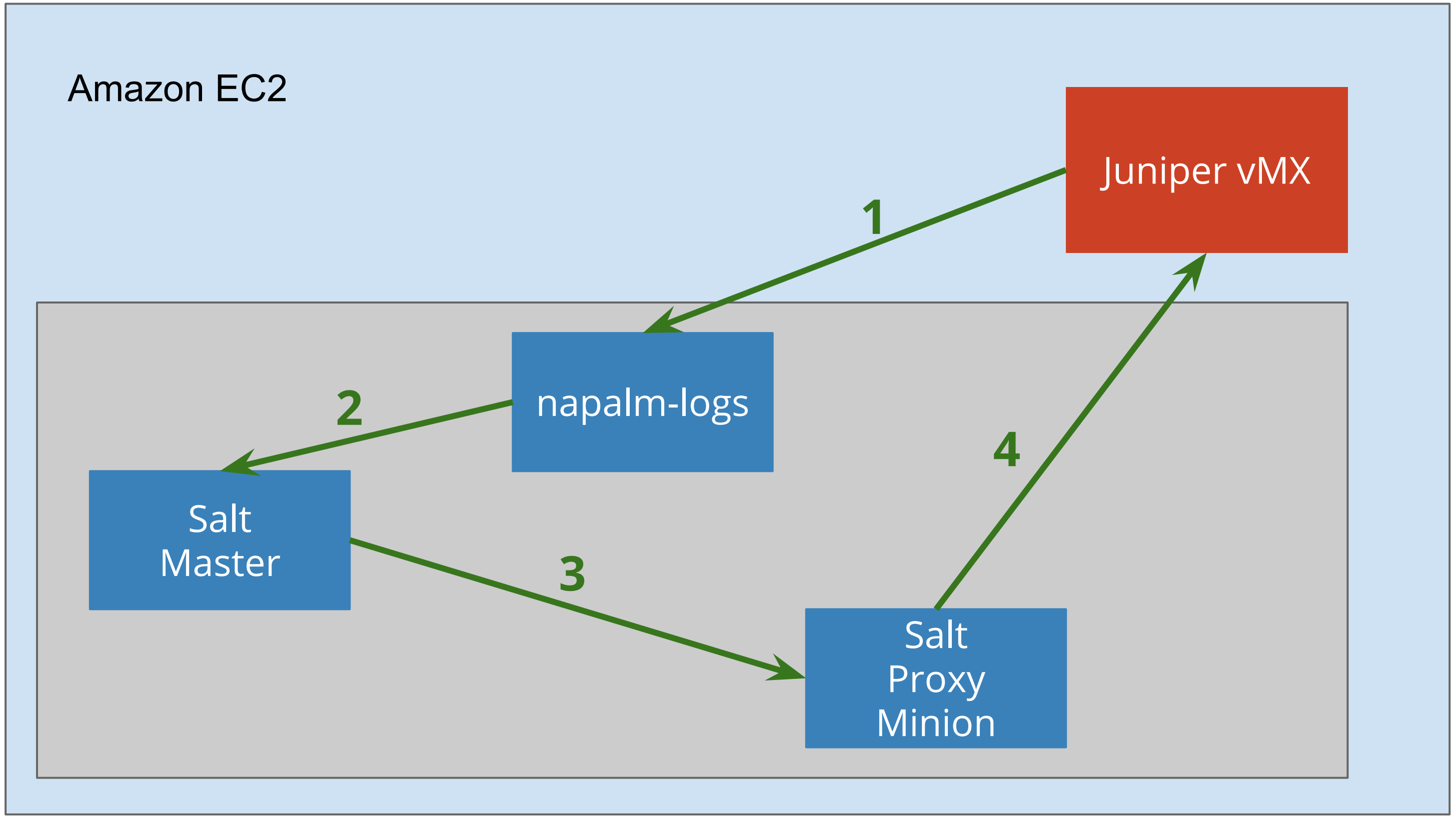
All the files are available on GitHub
https://github.com/mirceaulinic/ripe76-tutorial

Live demo

Topology



Live demo
Topology



1. Syslog notification sent to napalm-logs.
2. Structured napalm-logs message on the Salt bus.
3. Triggered job execution.
4. Configuration changed on the device.

Live demo
Start a Salt Master and Proxy Minion (1)

My Pillar Top file

`/etc/salt/pillar/top.sls`

```
base:
  '*':
    - ntp
  {{ opts.id }}:
    - {{ opts.id }}_pillar
```

How to read this: each Minion will load the contents from a file based on it's ID, e.g., Minion ID *dummy* will load the contents from *dummy_pillar.sls*, etc.
The *ntp.sls* Pillar is spread to all the Minions (thanks to the *).

Live demo
Start a Salt Master and Proxy Minion (2)

The shared Pillar for the NTP configuration, structured as defined in the *openconfig-system* model

`/etc/salt/pillar/ntp.sls`

```
openconfig-system:
  system:
    ntp:
      config:
        servers:
          server:
            17.253.2.125:
             config:
               association_type: SERVER
               prefer: true
```

Live demo
Start a Salt Master and Proxy Minion (3)

The Pillar for the *device1* Proxy Minion

`/etc/salt/pillar/device1_pillar.sls`

```
proxy:
  proxytype: napalm
  driver: junos
  host: example.com
  username: test
  password: test
```

Live demo
Start a Salt Master and Proxy Minion (4)

The Salt Master config file

`/etc/salt/master`

```
open_mode: true

pillar_roots:
  base:
    - /etc/salt/pillar

file_roots:
  base:
    - /etc/salt
    - /etc/salt/states

engines:
  - napalm_syslog:
      address: example.com
      port: 17171
      disable_security: true

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

The configuration for the napalm-syslog Salt Engine, to ingest events from napalm-logs.

Reactor configuration, to trigger a State execution when there's a NTP_SERVER_UNREACHABLE notification.

Live demo
Start a Salt Master and Proxy Minion (1)

The Proxy config file

`/etc/salt/proxy`

```
master: salt-master
open_mode: true
multiprocessing: false
```

Live demo
Start a Salt Master and Proxy Minion (1)

One Docker container each, using Docker Compose

Share filesystems with the containers, mounting the files as volumes. This way, we can edit the Pillars / States / Reactor etc. files without restarting the container, but simply edit them locally and the changes will be reflected inside the container.

docker-compose.yml (simplified version)

```
services:
  salt-master:
    image: mirceaulinic/salt-master:2017.7.5
    hostname: salt-master
    container_name: salt-master
    environment:
      - LOG_LEVEL
    volumes:
      - ./master:/etc/salt/master
      - ./pillar/:/etc/salt/pillar/
      - ./states/:/etc/salt/states/
      - ./reactor/:/etc/salt/reactor/
    network_mode: host
  salt-proxy:
    image: mirceaulinic/salt-proxy:2017.7.5
    hostname: salt-proxy
    container_name: salt-proxy
    volumes:
      - ./proxy:/etc/salt/proxy
    environment:
      - LOG_LEVEL
      - PROXYID
    network_mode: host
```

Live demo
Start a Salt Master and Proxy Minion (2)

One Docker container each, using Docker Compose

The Proxy Minion ID can be sent from the Makefile

docker-compose.yml (live version)

```yaml
services:
  salt-master:
    image: mirceaulinic/salt-master:2017.7.5
    hostname: salt-master
    container_name: salt-master
    environment:
      - LOG_LEVEL
    volumes:
      - ./master:/etc/salt/master
      - ./pillar/:/etc/salt/pillar/
      - ./states/:/etc/salt/states/
      - ./reactor/:/etc/salt/reactor/
    network_mode: host
  salt-proxy:
    image: mirceaulinic/salt-proxy:2017.7.5
    hostname: ${PROXYID}
    container_name: salt-proxy-${PROXYID}
    volumes:
      - ./proxy:/etc/salt/proxy
    environment:
      - LOG_LEVEL
      - PROXYID
    network_mode: host
```

Live demo
Start a Salt Master and Proxy Minion (1)

The Salt Proxy config file

`/etc/salt/proxy`

```
master: salt-master
open_mode: true
multiprocessing: false
```

Live demo
Starting napalm-logs (1)

The *napalm-logs* config file

`/etc/napalm/logs`

```
log_level: info
log_file: cli
port: 17171
listener:
  - udp: {}
publisher:
  - zmq: {}
disable_security: true
```

Live demo
Starting napalm-logs (2)

Similarly, in a Docker container

```
docker-compose.yml  (live version)
```

```
services:
  napalm-logs:
    image: mirceaulinic/napalm-logs:0.5.0
    hostname: napalm-logs
    container_name: napalm-logs
    environment:
      - LOG_LEVEL
    volumes:
      - ./napalm-logs.conf:/etc/napalm/logs
    network_mode: host
    ports:
      - "49017"
```

Live demo

Starting napalm-logs: configure the network box

```
set system syslog host 10.10.10.1 port 17171 any any
```

Where napalm-logs will be running (listening to the syslog messages)

As configured in the *napalm-logs.conf* config file

For more configuration details for other network operating systems, check the [Supported devices and configuration]() section in the napalm-logs documentation.

Live demo
Starting all the containers (1)

The Makefile

```
export PROXYID ?= dummy

all:
	docker-compose up -d
```

Live demo
Starting all the containers (2)

Simply execute

```
$ make PROXYID=device1
docker-compose up -d
Creating napalm-logs          ... done
Creating salt-proxy-device1 ... done
Creating salt-master          ... done
```

Live demo

Starting all the containers (3)

All 3 containers should be up and running

```
$ sudo docker ps
CONTAINER ID        IMAGE                               COMMAND                  CREATED             STATUS
PORTS               NAMES
df9019904036        mirceaulinic/salt-master:2017.7.5   "/bin/sh -c \"/usr/lo…"   8 minutes ago       Up 8 minutes
salt-master
9feb2f32f864        mirceaulinic/salt-proxy:2017.7.5    "/bin/sh -c \"/usr/lo…"   8 minutes ago       Up 8 minutes
salt-proxy-device1
bc693428c207        mirceaulinic/napalm-logs:0.5.0      "/bin/sh -c 'napalm-…"    8 minutes ago       Up 8 minutes
napalm-logs
```

Live demo

Starting all the containers (4)

From the Master can execute:

```
$ docker exec -it salt-master bash
root@salt-master:/# salt device1 test.ping
device1:
    True
```

```
$ docker exec -it salt-master bash
root@salt-master:/# salt device1 route.show
0.0.0.0/0
device1:
    ----------
    comment:
    out:
        ----------
        0.0.0.0/0:
            |_
              ----------
              age:
                  113525
              current_active:
                  True
              inactive_reason:
              last_active:
                  True
              next_hop:
                  172.31.0.1
```

Live demo
Starting all the containers

We should notice napalm-logs events on the Salt bus:

```
$ docker exec -it salt-master bash
root@salt-master:/# salt-run state.event pretty=True
```

```
napalm/syslog/junos/USER_ENTER_CONFIG_MODE/edge01.mrs01 {
    "_stamp": "2018-05-09T14:05:10.669616",
    "error": "USER_ENTER_CONFIG_MODE",
    "facility": 23,
    "host": "edge01.mrs01",
    "ip": "172.31.13.150",
    "os": "junos",
    "severity": 5,
    "timestamp": 1525874708,
    "yang_message": {
        "users": {
            "user": {
                "mircea": {
                    "action": {
                        "enter_config_mode": true
                    }
                }
            }
        }
    },
    "yang_model": "NO_MODEL"
}
```

Live demo
Reproduce this yourself

```
$ git clone https://github.com/mirceaulinic/ripe76-tutorial.git
$ cd ripe76-tutorial
### edit pillar/device1_pillar.sls file, and add your authentication details
$ make PROXYID=device1
```

Docker and Docker Compose are assumed to be already installed. Otherwise, follow the installation notes: https://docs.docker.com/install/ and https://docs.docker.com/compose/install/

Live demo

Applying a configuration change

```
[edit]
napalm@device1# set system ntp server 1.2.3.4

[edit]
napalm@device1# show | compare
[edit system]
+   ntp {
+       server 1.2.3.4;
+   }

[edit]
napalm@device1# commit
commit complete
```

Live demo

Applying a configuration change: as seen on the Salt bus

```
napalm/syslog/junos/CONFIGURATION_COMMIT_REQUESTED/device1 {
    "_stamp": "2018-05-09T14:19:40.733766",
    "error": "CONFIGURATION_COMMIT_REQUESTED",
    "facility": 23,
    "host": "device1",
    "ip": "172.31.13.150",
    "os": "junos",
    "severity": 5,
    "timestamp": 1525875578,
    "yang_message": {
        "users": {
            "user": {
                "napalm": {
                    "action": {
                        "comment": "none",
                        "requested_commit": true
                    }
                }
            }
        }
    },
    "yang_model": "NO_MODEL"
}
```

Live demo

Applying a configuration change: as seen on the Salt bus

```
napalm/syslog/junos/CONFIGURATION_COMMIT_COMPLETED/device1 {
    "_stamp": "2018-05-09T14:19:50.235894",
    "error": "CONFIGURATION_COMMIT_COMPLETED",
    "facility": 23,
    "host": "device1",
    "ip": "172.31.13.150",
    "os": "junos",
    "severity": 4,
    "timestamp": 1525875587,
    "yang_message": {
        "system": {
            "operations": {
                "commit_complete": true
            }
        }
    },
    "yang_model": "NO_MODEL"
}
```

Live demo
The device is unsynchronised

```
napalm@device1> show ntp associations
   remote           refid           st t when poll reach   delay   offset  jitter
==============================================================================
 1.2.3.4         .INIT.          16 -    -   64    0    0.000    0.000 4000.00
```

Live demo

An NTP server becomes unreachable

```
napalm/syslog/junos/NTP_SERVER_UNREACHABLE/device1      {
    "error": "NTP_SERVER_UNREACHABLE",
    "facility": 12,
    "host": "device1",
    "ip": "172.31.13.150",
    "os": "junos",
    "severity": 3,
    "timestamp": 1525942185,
    "yang_message": {
        "system": {
            "ntp": {
                "servers": {
                    "server": {
                        "1.2.3.4": {
                            "state": {
                                "association-type": "SERVER",
                                "stratum": 16
                            }
                        }
                    }
                }
            }
        }
    },
    "yang_model": "openconfig-system"
}
```

Live demo

The reactor kicks a State execution (configuration change): Salt Master logs

```
[DEBUG   ] Sending event: tag = napalm/syslog/junos/NTP_SERVER_UNREACHABLE/device1; data = {'_stamp':
'2018-05-10T08:49:58.296950', 'yang_message': {'system': {'ntp': {'servers': {'server': {'1.2.3.4': {'state':
{'association-type': 'SERVER', 'stratum': 16}}}}}}}, 'message_details': {'processId': '22637', 'severity': 3, 'facility':
12, 'hostPrefix': None, 'pri': '99', 'host': 'device1', 'tag': 'xntpd', 'time': '09:13:23', 'date': 'May 10', 'message':
'NTP Server 1.2.3.4 is Unreachable'}, 'facility': 12, 'ip': '172.31.13.150', 'error': 'NTP_SERVER_UNREACHABLE', 'host':
'device1', 'yang_model': 'openconfig-system', 'timestamp': 1525943603, 'os': 'junos', 'severity': 3}
[DEBUG   ] Gathering reactors for tag napalm/syslog/junos/NTP_SERVER_UNREACHABLE/device1
[DEBUG   ] Compiling reactions for tag napalm/syslog/junos/NTP_SERVER_UNREACHABLE/device1
[DEBUG   ] Rendered data from file: /var/cache/salt/master/files/base/reactor/exec_ntp_state.sls:
triggered NTP state:
  local.state.sls:
    - tgt: device1
    - arg:
      - ntp
[DEBUG   ] Results of YAML rendering:
OrderedDict([('triggered NTP state', OrderedDict([('local.state.sls', [OrderedDict([('tgt', 'device1')]),
OrderedDict([('arg', ['ntp'])])])]))])
```

Live demo

The reactor kicks a State execution (configuration change): Salt Master logs

```
[DEBUG   ] Sending event: tag = 20180510084958473615; data = {'_stamp': '2018-05-10T08:49:58.474571', 'minions':
['device1']}
[DEBUG   ] Sending event: tag = salt/job/20180510084958473615/new; data = {'tgt_type': 'glob', 'jid':
'20180510084958473615', 'tgt': 'device1', '_stamp': '2018-05-10T08:49:58.474852', 'user': 'root', 'arg': ['ntp'], 'fun':
'state.sls', 'minions': ['device1']}
[DEBUG   ] Adding minions for job 20180510084958473615: ['device1']
[INFO    ] User root Published command state.sls with jid 20180510084958473615
[DEBUG   ] Published command details {'tgt_type': 'glob', 'jid': '20180510084958473615', 'tgt': 'device1', 'ret': '',
'user': 'root', 'arg': ['ntp'], 'fun': 'state.sls'}
[INFO    ] Got return from device1 for job 20180510084958473615
[DEBUG   ] Sending event: tag = salt/job/20180510084958473615/ret/device1; data = {'fun_args': ['ntp'], 'jid':
'20180510084958473615', 'return': {'netconfig_|-oc_ntp_netconfig_|-oc_ntp_netconfig_|-managed': {'comment': 'Configuration
changed!\n', 'pchanges': {'diff': '[edit system ntp]\n+    server 17.253.2.125 prefer;\n-    server 1.2.3.4;'}, 'name':
'oc_ntp_netconfig', 'start_time': '08:49:58.671645', 'result': True, 'duration': 10704.875, '__run_num__': 0, '__sls__':
'ntp.netconfig', 'changes': {'diff': '[edit system ntp]\n+    server 17.253.2.125 prefer;\n-    server 1.2.3.4;'},
'__id__': 'oc_ntp_netconfig'}}, 'retcode': 0, 'success': True, 'cmd': '_return', '_stamp': '2018-05-10T08:50:09.966462',
'fun': 'state.sls', 'id': 'device1', 'out': 'highstate'}
```

Live demo
The reactor kicks a State execution (configuration change): Salt bus

```
20180510084958473615       {
    "_stamp": "2018-05-10T08:49:58.474571",
    "minions": [
        "device1"
    ]
}
salt/job/20180510084958473615/new {
    "_stamp": "2018-05-10T08:49:58.474852",
    "arg": [
        "ntp"
    ],
    "fun": "state.sls",
    "jid": "20180510084958473615",
    "minions": [
        "device1"
    ],
    "tgt": "device1",
    "tgt_type": "glob",
    "user": "root"
}
```

Live demo

The reactor kicks a State Execution (configuration change): result

```
salt/job/20180510084958473615/ret/device1{
    "_stamp": "2018-05-10T08:50:09.966462",
    "cmd": "_return",
    "fun": "state.sls",
    "fun_args": [
        "ntp"
    ],
    "id": "device1",
    "jid": "20180510084958473615",
    "out": "highstate",
    "retcode": 0,
    "return": {
"netconfig_|-oc_ntp_netconfig_|-oc_ntp_netconfig_|-managed": {
            "changes": {
                "diff": "[edit system ntp]\n+    server
17.253.2.125 prefer;\n-    server 1.2.3.4;"
            },
            "comment": "Configuration changed!\n",
            "duration": 11288.952,
            "name": "oc_ntp_netconfig",
            "result": true,
            "start_time": "08:49:58.671645"
        }
    },
    "success": true
}
```

Live demo

Side note: the previous job output, when executed from the CLI

```
root@salt-master:/# salt device1 state.sls ntp
device1:
----------
          ID: oc_ntp_netconfig
    Function: netconfig.managed
      Result: True
     Comment: Configuration changed!
     Started: 11:22:21.364131
    Duration: 10644.564 ms
     Changes:
              ----------
              diff:
                  [edit system ntp]
                  +    server 17.253.2.125 prefer;
                  -    server 1.2.3.4;

Summary for device1
------------
Succeeded: 1 (changed=1)
Failed:    0
------------
Total states run:     1
Total run time:  10.645 s
root@salt-master:/#
```

Live demo

The device is synchronised

```
napalm@device1> show configuration system ntp
server 17.253.2.125 prefer;

napalm@device1>

napalm@device1> show ntp associations
   remote           refid           st t when poll reach   delay   offset  jitter
===============================================================================
*17.253.2.125     .GPSs.             1 -    8   64   77   39.015   12.527  27.595
```

Other examples using napalm-logs

- Using BGP_NEIGHBOR_STATE_CHANGED, can send alerts (e.g., chat, SMS, email, etc.)
- On OSPF_NEIGHBOR_DOWN can adjust the route metric cost.
- On BGP_SESSION_NOT_CONFIGURED, send an email to your peer as a reminder.
- This list can be nearly infinite, and depends on your own use case.

Key takeaways

- *napalm-logs* can be used for event-driven network automation.
- There can be many other good sources of internal and external events, e.g., streaming telemetry, BGPmon, Prometheus Alertmanager, emails from your partners, or other in-house that make sense to your own business model.

“

The only limit to your impact is your imagination and commitment.

”

Tony Robbins

(besides time, mood, human resources, YouTube, Facebook, etc.)

Need help/advice?

slack

Join https://networktocode.slack.com/

rooms: #**saltstack** #**napalm**

New: https://saltstackcommunity.slack.com

rooms: #**networks**

How can you contribute?

- napalm-logs: https://github.com/napalm-automation/napalm-logs
- NAPALM Automation: https://github.com/napalm-automation
- Salt https://github.com/saltstack/salt

Questions

?

mircea@cloudflare.com

CLOUDFLARE