Three years of automating large scale networks using Salt

Mircea Ulinic
Cloudflare, London

FranceIX General Meeting
Paris, September 2018

Mircea Ulinic

- Network software engineer at Cloudflare
- Member and maintainer at NAPALM Automation
- SaltStack contributor of the year 2017
- O'Reilly author
- OpenConfig representative
- https://mirceaulinic.net/

 mirceaulinic

 @mirceaulinic

Automation: definition

- The technique, method, or system of operating or controlling a process by highly automatic means, as by electronic devices, reducing human intervention to a minimum.
- The technique of making an apparatus, a process, or a system operate *automatically*.
    - *Automatically*: Having a self-acting or self-regulating mechanism

Common views on automation

In general (mis)understood as the equivalent of *just* configuration management.

> In simpler terms, this boils down to: generate a configuration based on a template ⇒ load the text blob on the network device.

... but what about the very long list of other manual tasks, e.g.:

- run the command to deploy the config
- same boring email to send to your providers
- Same boring notifications written manually (sometimes with tpyos)
- route leaks you learn about only minutes after it started
- other events you react way too late

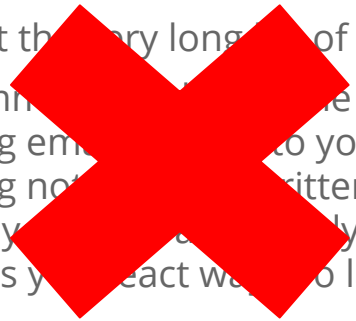
Common views on automation

In general (mis)understood as the equivalent of *just* configuration management.

> In simpler terms, this boils down to: generate a configuration based on a template ⇒ load the text blob on the network device.

... but what about th[illegible]ry long [illegible] of other manual tasks, e.g.:

- run the comm[illegible]e config
- same boring em[illegible]o your providers
- same boring no[illegible]ritten manually (sometimes with tpyos)
- route leaks y[illegible]y minutes after it started
- other events y[illegible]eact w[illegible]o late



This is not automation

Common views on automation

In general (mis)understood as the equivalent of *just* configuration management.

> In simpler terms, this boils down to: generate a configuration based on a template ⇒ load the text blob on the network device.

… but what abou the v st r other manual tasks, e.g.:

- run th config
- same bo ur providers
- same b n manually (sometimes with tpyos)
- route l bou only minutes after it started
- other even you react way too late

Frameworks used in networking before 2016

... but they are not event-driven neither data-driven

Salt had the features to automate *everything*

"

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.

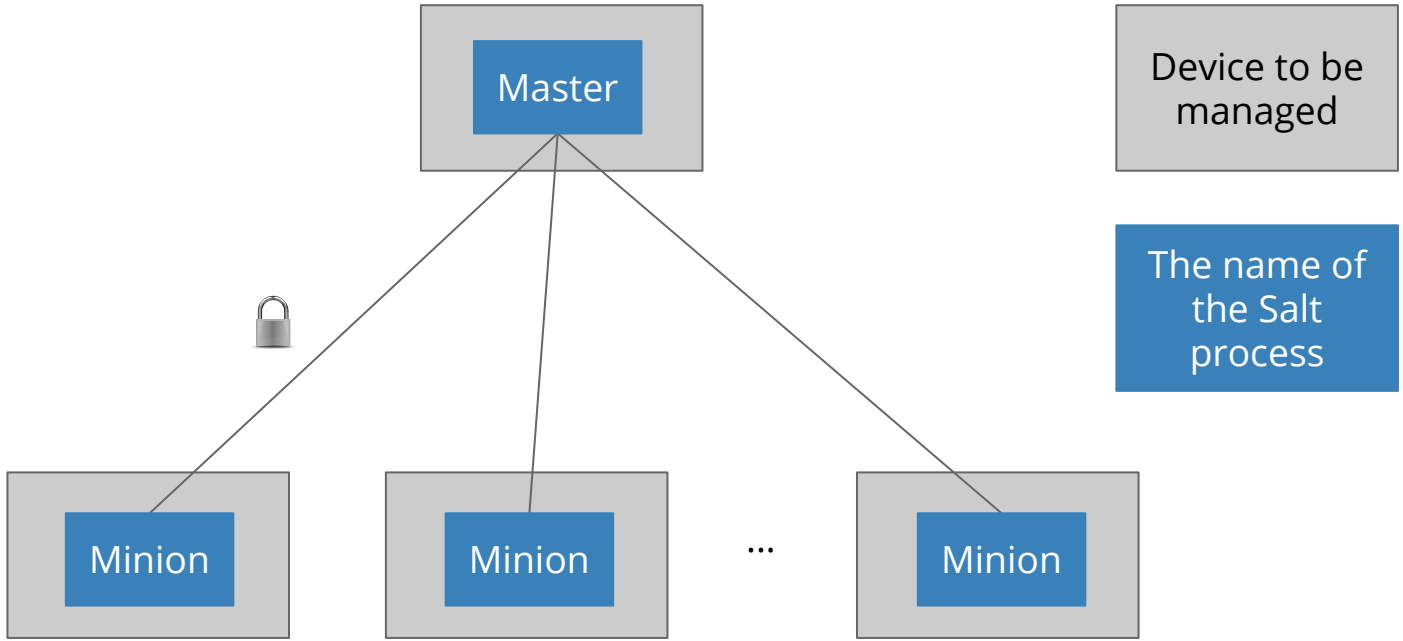
SaltStack now provides orchestration, configuration management, event reactors, cloud provisioning, and more, all built around the SaltStack high-speed communication bus.

"

… but no features for network automation before 2016

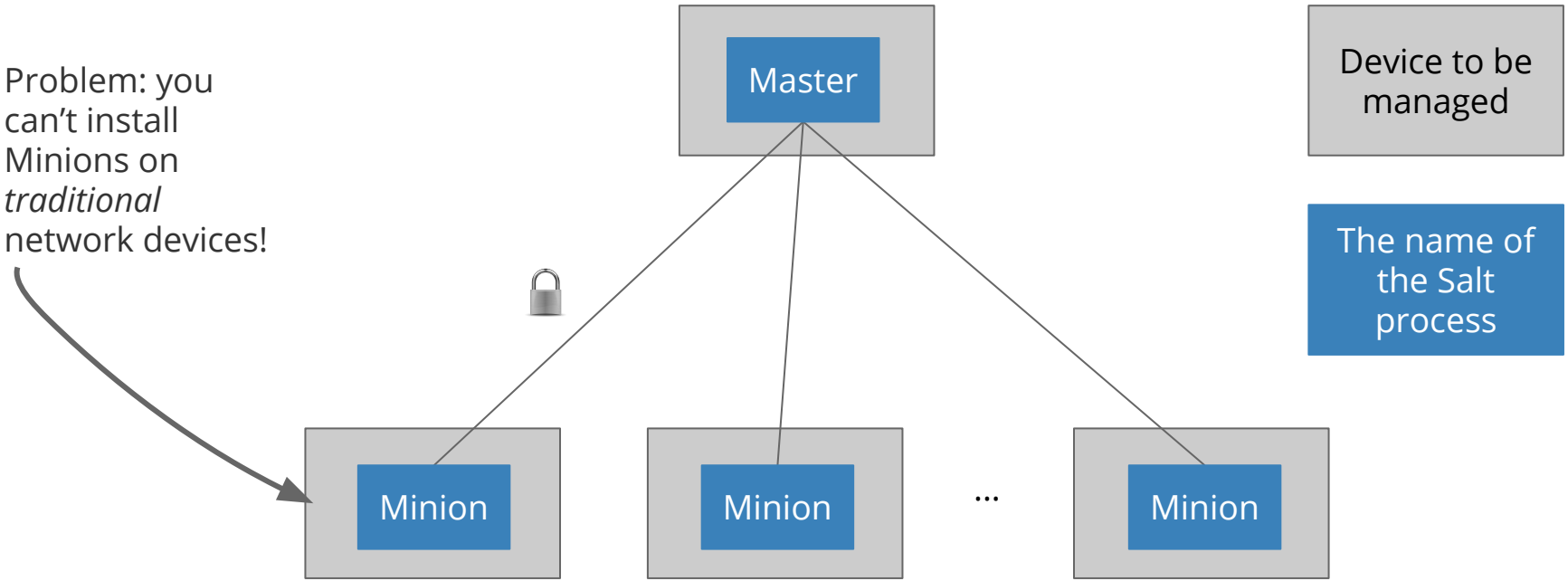
https://docs.saltstack.com/en/getstarted/speed.html

Salt Architecture



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

Salt Architecture

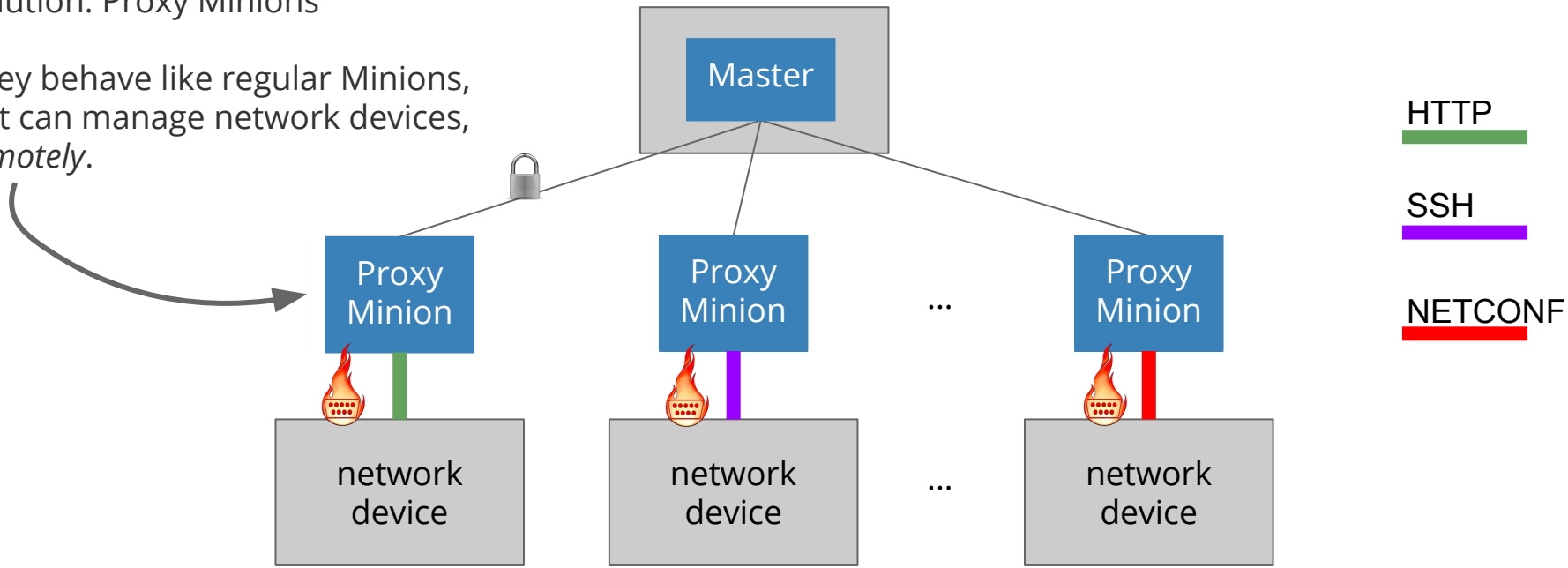


https://docs.saltstack.com/en/latest/topics/topology/index.html

Salt Architecture: Proxy Minions

Solution: Proxy Minions

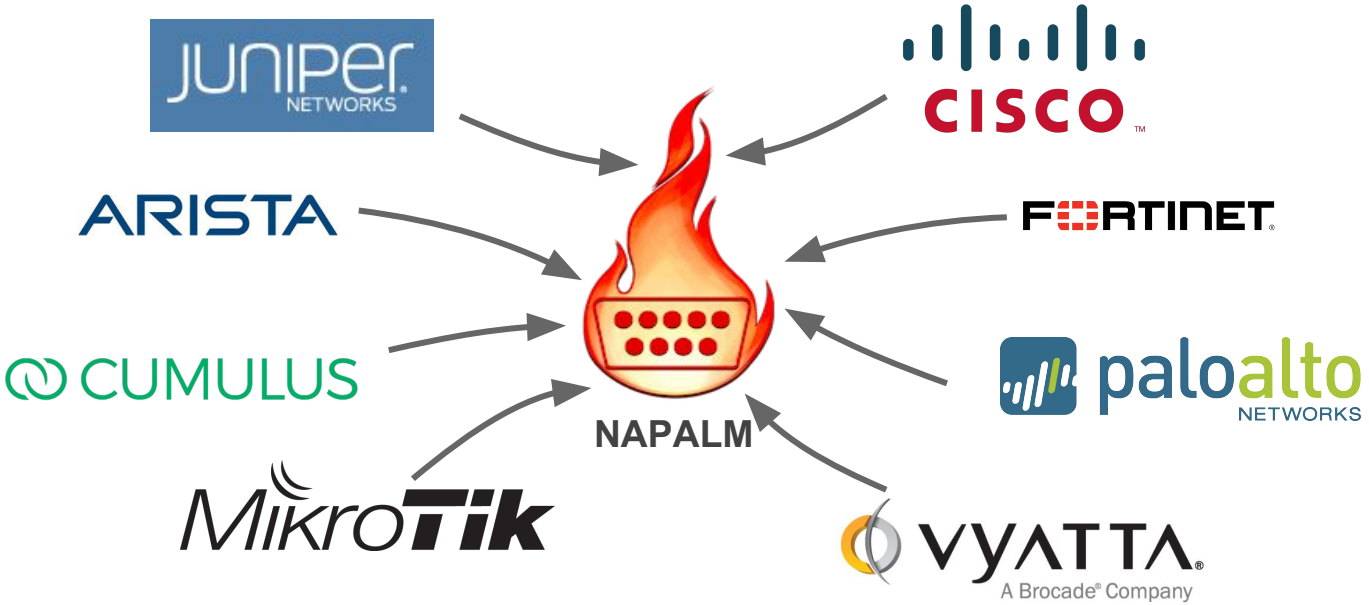
They behave like regular Minions, but can manage network devices, *remotely*.



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

Vendor-agnostic API: NAPALM

Network Automation and Programmability Abstraction Layer with Multivendor support



https://github.com/napalm-automation

NAPALM integrated in Salt: Carbon (2016.11)

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

2016: Vendor-agnostic automation is here!

```
$ 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 configuration management

```
$ sudo salt device1 state.sls ntp
device1:
----------
          ID: Manage the NTP config
    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;
                  +     server 10.10.10.2;
                  -     server 172.17.17.1;
```

```
$ sudo salt device2 state.sls ntp
device2:
----------
          ID: Manage the NTP config
    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
                  + server 10.10.10.2
                   !
```

NAPALM integrated in Salt: Nitrogen (2017.7)

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

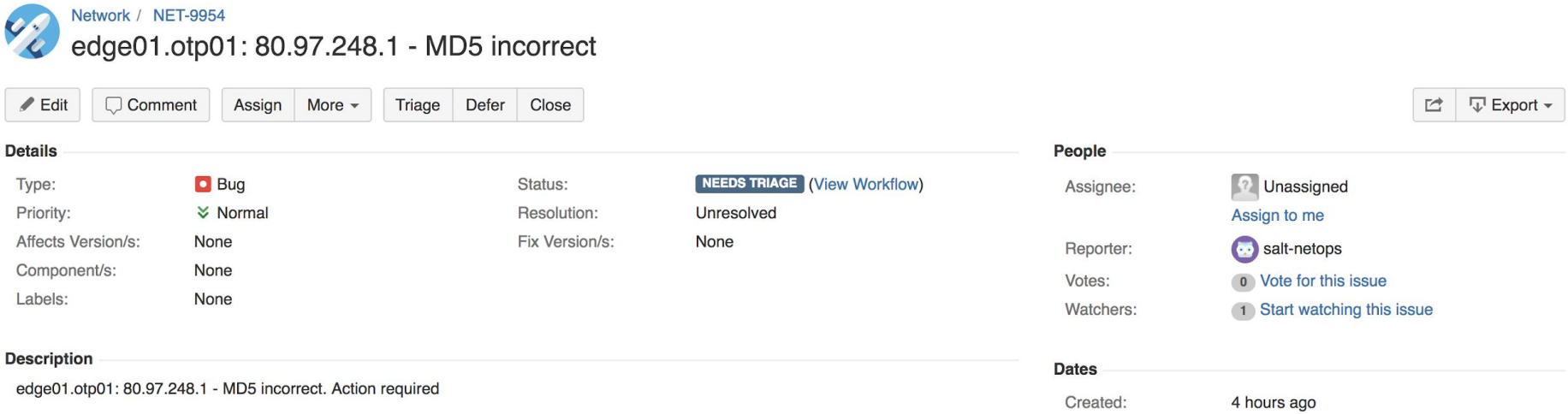
2017: event-driven network automation

Salt BOT Wed 4:52 PM

edge01.jnb01: 2001:43f8:1f0::121 (AS32437 - CYBERTEK-): Increased the prefix limit to 500:

```
[edit protocols bgp group 6-PUBLIC-ANYCAST-PEERS neighbor 2001:43f8:1f0::121 family inet6 unicast prefix-limit]
-         maximum 100;
+         maximum 500;
```

2017: event-driven network automation



NAPALM integrated in Salt: Fluorine (2018.11)

NAPALM

COMMIT AT AND COMMIT CONFIRMED

Beginning with this release, NAPALM users are able to execute scheduled commits (broadly known as "commit at") and "commit confirmed" (revert the configuration change unless the user confirms by running another command). These features are available via the `commit_in`, `commit_at`, `revert_in`, or `revert_at` arguments for the `net.load_config` and `net.load_template` execution functions, or `netconfig.managed`.

The counterpart execution functions `net.confirm_commit`, or `net.cancel_commit`, as well as the State functions `netconfig.commit_cancelled`, or `netconfig.commit_confirmed` can be used to confirm or cancel a commit.

Please note that the commit confirmed and commit cancelled functionalities are available for any platform whether the network devices supports the features natively or not. However, be cautious and make sure you read and understand the caveats before using them in production.

MULTIPLE TEMPLATES RENDERED SIMULTANEOUSLY

The `template_name` argument of the `net.load_template` Execution and `netconfig.managed` State function now supports a list of templates. This is particularly useful when a very large Jinja template is split into multiple smaller and easier to read templates that can eventually be reused in other States. For example, the following syntax is not correct to manage the configuration of NTP and BGP simultaneously, using two different templates and changing the device configuration through one single commit:

YAML

```
manage_bgp_and_ntp:
  netconfig.managed:
    - template_name:
        - salt://templates/bgp.jinja
        - salt://templates/ntp.jinja
    - context:
        bpg: {{ pillar.bgp }}
        ntp: {{ pillar.ntp }}
```

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

NAPALM integrated in Salt: Fluorine (2018.11)

CONFIGURATION REPLACE FEATURES

To replace various configuration chunks, you can use the new `net.replace_pattern` execution function, or the `netconfig.replace_pattern` State function. For example, if you want to update your configuration and rename a BGP policy referenced in many places, you can do so by running:

BASH

```
salt '*' net.replae_pattern OLD-POLICY-CONFIG new-policy-config
```

Similarly, you can also replace entire configuration blocks using the `net.blockreplace` function.

CONFIGURATION SAVE FEATURES

The `net.save_config` function can be used to save the configuration of the managed device into a file. For the State subsystem, the `netconfig.saved` function has been added which provides a complete list of facilities when managing the target file where the configuration of the network device can be saved.

For example, backup the running configuration of each device under its own directory tree:

YAML

```
/var/backups/{{ opts.id }}/running.cfg:
  netconfig.saved:
    - source: running
    - makedirs: true
```

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

NAPALM integrated in Salt: Fluorine (2018.11)

All the new network automation modules mentioned above are directly exposed to the NAPALM users, without requiring any architectural changes, just eventually install some requirements:

JUNOS

The features from the existing `junos` Execution Module are available via the following functions:

- `napalm.junos_cli`: Execute a CLI command and return the output as text or Python dictionary.
- `napalm.junos_rpc`: Execute an RPC request on the remote Junos device, and return the result as a Python dictionary, easy to digest and manipulate.
- `napalm.junos_install_os`: Install the given image on the device.
- `napalm.junos_facts`: The complete list of Junos facts collected by the `junos-eznc` underlying library.

Note

To be able to use these features, you muse ensure that you meet the requirements for the `junos` module. As `junos-eznc` is already a dependency of NAPALM, you will only have to install `jxmlease`.

Usage examples:

BASH

```
salt '*' napalm.junos_cli 'show arp' format=xml
salt '*' napalm.junos_rpc get-interface-information
```

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

NAPALM integrated in Salt: Fluorine (2018.11)

ARISTA PYEAPI

For various operations and various extension modules, the following features have been added to gate functionality from the `pyeapi` module:

- `napalm.pyeapi_run_commands`: Execute a list of commands on the Arista switch, via the `pyeapi` library.
- `napalm.pyeapi_config`: Configure the Arista switch with the specified commands, via the `pyeapi` Python library. Similarly to `napalm.netmiko_config`, you can use both local and remote files, with or without templating.

Usage examples:

BASH

```
salt '*' napalm.pyeapi_run_commands 'show version' 'show interfaces'
salt '*' napalm.pyeapi_config config_file=salt://path/to/template.jinja
```

CISCO NX-API

In the exact same way as above, the user has absolute control by using the following primitives to manage Cisco Nexus switches via the NX-API:

- `napalm.nxos_api_show`: Execute one or more show (non-configuration) commands, and return the output as plain text or Python dictionary.
- `napalm.nxos_api_rpc`: Execute arbitrary RPC requests via the Nexus API.
- `napalm.nxos_api_config`: Configures the Nexus switch with the specified commands, via the NX-API. The commands can be loaded from the command line, or a local or remote file, eventually rendered using the templating engine of choice (default: `jinja`).

Usage examples:

BASH

```
salt '*' napalm.nxos_api_show 'show bgp sessions' 'show processes' raw_text=False
```

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

NAPALM integrated in Salt: Fluorine (2018.11)

CISCOCONFPARSE

The following list of function may be handy when manipulating Cisco IOS or Junos style configurations:

- `napalm.config_filter_lines`: Return a list of detailed matches, for the configuration blocks (parent-child relationship) whose parent and children respect the regular expressions provided.
- `napalm.config_find_lines`: Return the configuration lines that match the regular expression provided.
- `napalm.config_lines_w_child`: Return the configuration lines that match a regular expression, having child lines matching the child regular expression.
- `napalm.config_lines_wo_child`: Return the configuration lines that match a regular expression, that don't have child lines matching the child regular expression.

Note

These functions require the `ciscoconfparse` Python library to be installed.

Usage example (find interfaces that are administratively shut down):

```bash
salt '*' napalm.config_lines_w_child 'interface' 'shutdown'
```

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

Salt for network automation: not only NAPALM

NETBOX

Added in the previous release, 2018.3.0, the capabilities of the `netbox` Execution Module have been extended, with a much longer list of available features:

- `netbox.create_circuit`
- `netbox.create_circuit_provider`
- `netbox.create_circuit_termination`
- `netbox.create_circuit_type`
- `netbox.create_device`
- `netbox.create_device_role`
- `netbox.create_device_type`
- `netbox.create_interface`
- `netbox.create_interface_connection`
- `netbox.create_inventory_item`
- `netbox.create_ipaddress`
- `netbox.create_manufacturer`
- `netbox.create_platform`
- `netbox.create_site`
- `netbox.delete_interface`
- `netbox.delete_inventory_item`
- `netbox.delete_ipaddress`
- `netbox.get_circuit_provider`
- `netbox.get_interfaces`
- `netbox.get_ipaddresses`
- `netbox.make_interface_child`
- `netbox.make_interface_lag`
- `netbox.openconfig_interfaces`
- `netbox.openconfig_lacp`
- `netbox.update_device`
- `netbox.update_interface`

Besides this Execution Module, Salt users can load data directly from NetBox into the device Pillar, via the `netbox` External Pillar module.

First framework with official OpenConfig integrations

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

Salt for network automation: not only NAPALM

NETMIKO

Netmiko, the multi-vendor library to simplify Paramiko SSH connections to network devices, is now officially integrated into Salt. The network community can use it via the `netmiko` Proxy Module or directly from any Salt Minions, passing the connection credentials - see the documentation for the `netmiko` Execution Module.

ARISTA

Arista switches can now be managed running under the `pyeapi` Proxy Module, and execute RPC requests via the `pyeapi` Execution Module.

CISCO NEXUS

While support for SSH-based operations has been added in the release codename Carbon (2016.11), the new `nxos_api` Proxy Module and `nxos_api` allow management of Cisco Nexus switches via the NX-API.

It is important to note that these modules don't have third party dependencies, therefore they can be used straight away from any Salt Minion. This also means that the user may be able to install the regular Salt Minion on the Nexus switch directly and manage the network devices like a regular server.

GENERAL-PURPOSE MODULES

The new `ciscoconfparse` Execution Module can be used for basic configuration parsing, audit or validation for a variety of network platforms having Cisco IOS style configuration (one space indentation), as well as brace-delimited configuration style.

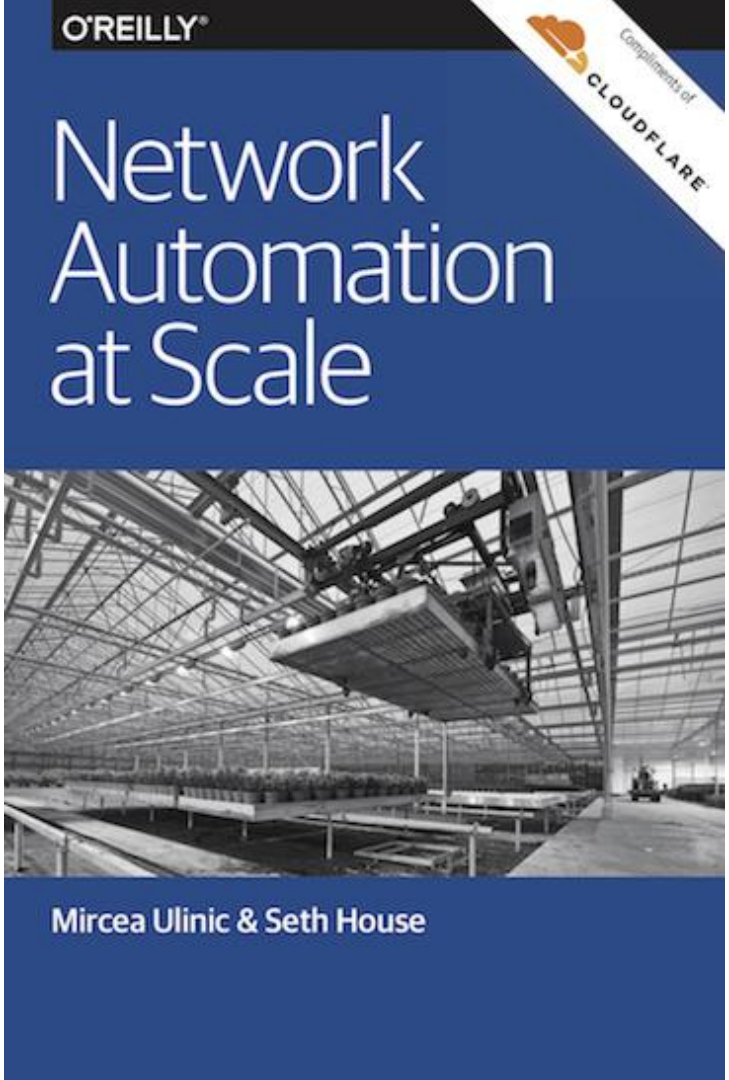
The `iosconfig` can be used for various configuration manipulation for Cisco IOS style configuration, such as: `configuration cleanup`, `tree representation of the config`, etc.

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

Who’s Salty today

Network Automation at Scale: the book

Free download:
http://www.oreilly.com/webops-perf/free/network-automation-at-scale.csp



Everything is open sourced

GitHub

- Salt
  https://github.com/saltstack/salt

- NAPALM Automation:
  https://github.com/napalm-automation

Need help/advice?

Join https://networktocode.slack.com/
rooms: #**saltstack** #**napalm**

New: https://saltstackcommunity.slack.com
rooms: #**networks**

Over 600 members

Questions

mircea@cloudflare.com

CLOUDFLARE