CLOUDFLARE®

SaltStack as network orchestrator

Scalable, fast, cross-vendor

Mircea Ulinic
Cloudflare, London

APRICOT 2017
Ho Chi Minh City, VN

Agenda

- So you want to automate
- Meet the tools
- Configure SaltStack
- CLI syntax
- Configuration management (brief intro)
- Real-world orchestration example

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?

- Mind your network
- How many devices?
- How many platforms / operating systems?
- How dynamic?
- Configuration management only?
- Triggered configuration changes?
- External sources of truth? e.g. IPAM
- Do you need native caching? REST API? etc...

Meet the Tools
Why Salt?

- Very scalable
- Concurrency
- Easily configurable & customizable
- Config verification & enforcement
- Periodically collect statistics
- Native caching and drivers for useful tools

Meet the Tools
Why Salt?

“

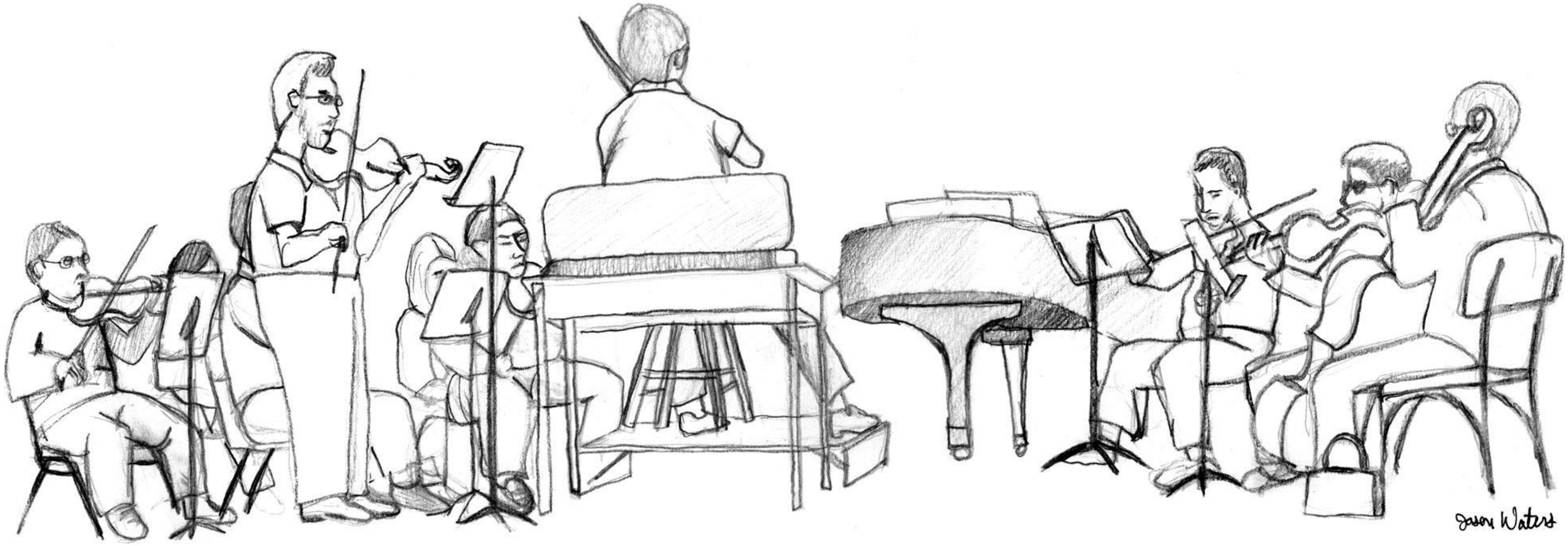
In SaltStack, speed isn’t a byproduct, it is a design goal. SaltStack was created as an extremely fast, lightweight communication bus to provide the foundation for a remote execution engine.

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

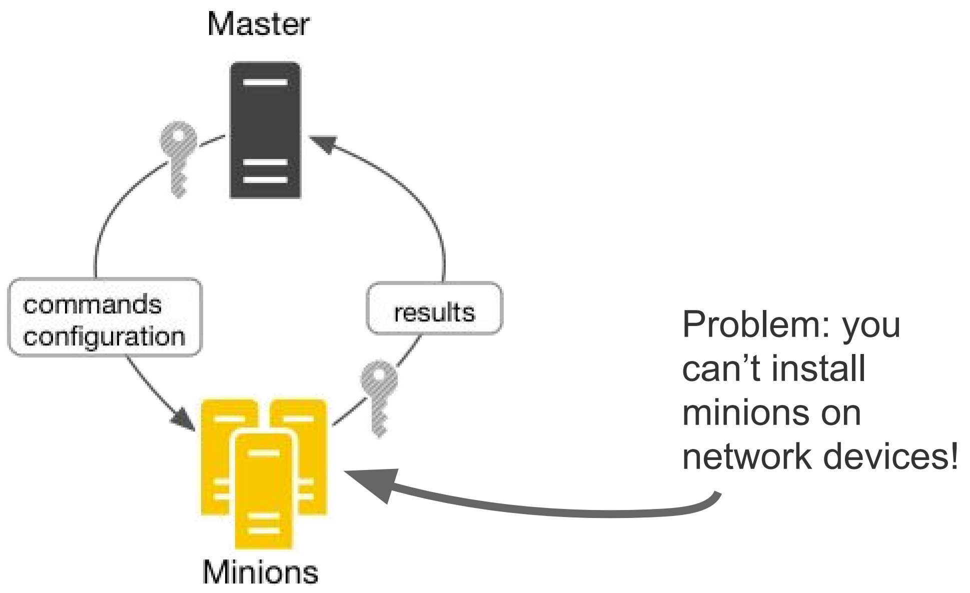
… + cross-vendor network automation from 2016.11 (Carbon) ”

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

Meet the Tools
Orchestration vs. Automation

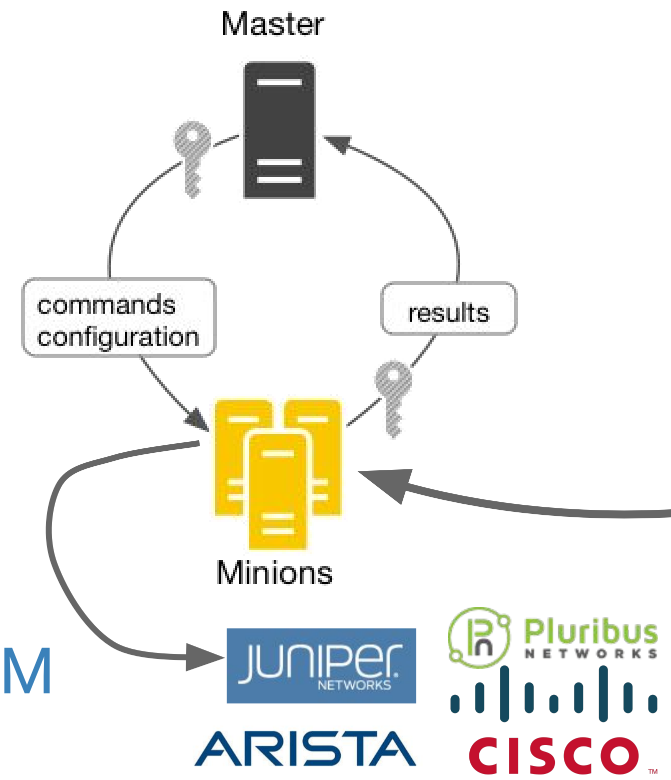


Meet the Tools
Salt Architecture



Problem: you can't install minions on network devices!

Meet the Tools
Salt Architecture

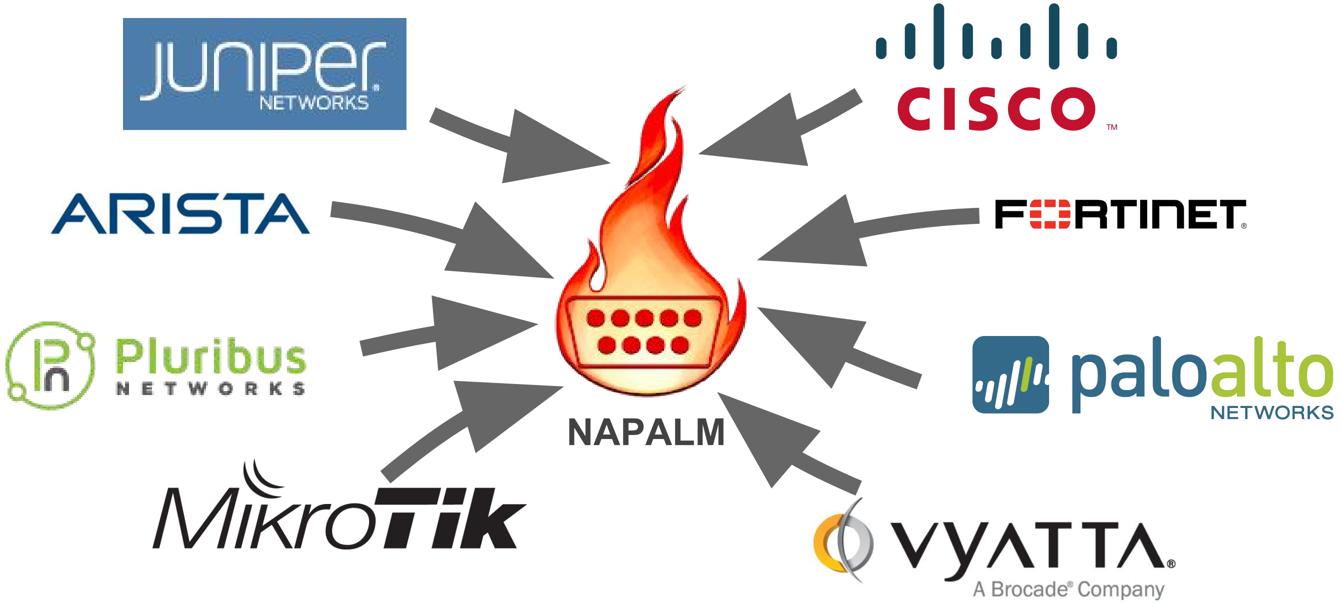


Solution:
proxy minions
They behave like minions, but can talk to network devices

NAPALM

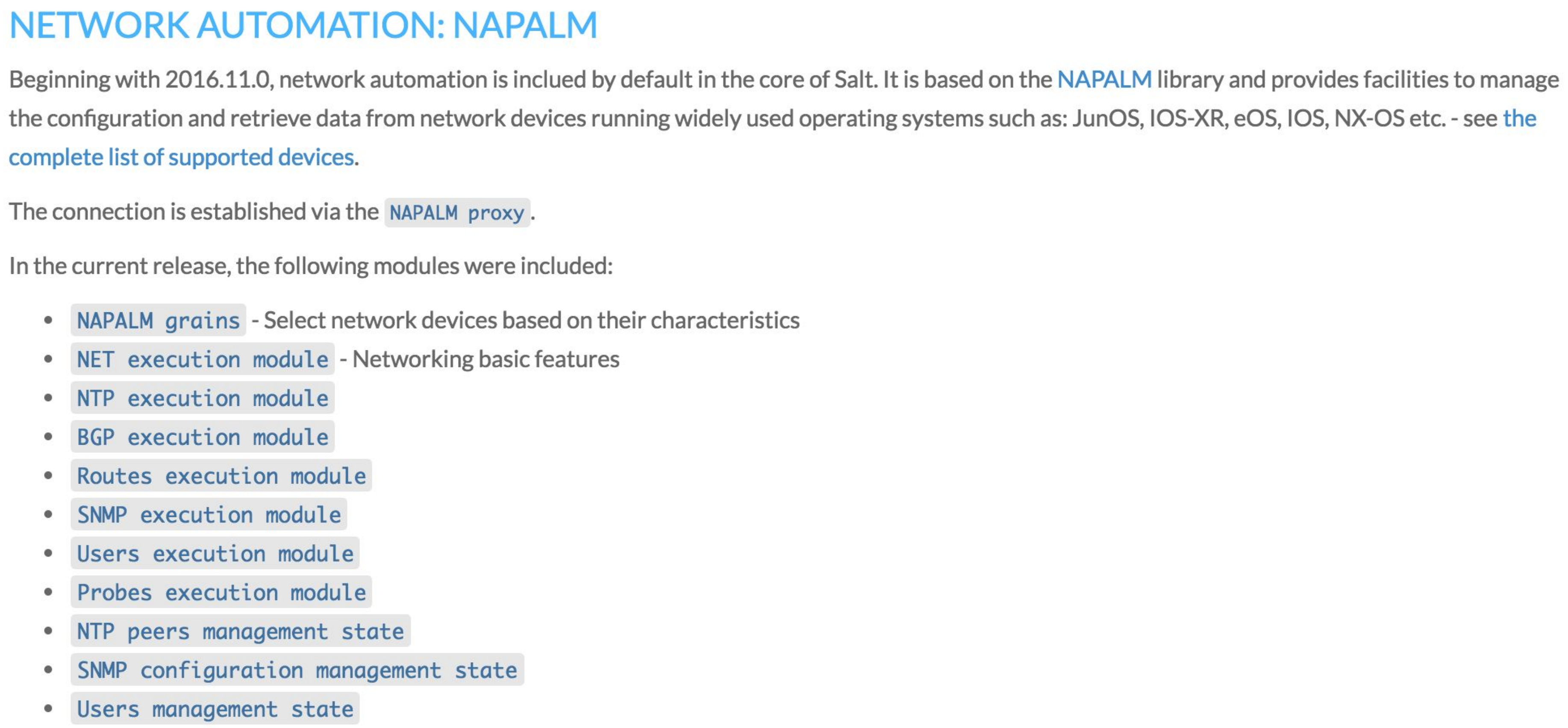
Meet the Tools
Why NAPALM?

(Network Automation and Programmability Abstraction Layer with Multivendor support)



https://github.com/napalm-automation

NAPALM integrated in SaltStack



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

Configure SaltStack

New to Salt?

Pillar

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

YAML file / database / git repository ... ***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

Configure SaltStack
Master config

`/etc/salt/master`

```
file_roots:
  base:
    - /etc/salt/states
    - /etc/salt/reactors
    - /etc/salt/templates
pillar_roots:
  base:
    - /etc/salt/pillar
```

Environment name

Useful to have different environments: prod, qa, develop etc.

For the beginning, let’s focus only on ***file_roots*** and ***pillar_roots***. The others settings are more advanced features: https://docs.saltstack.com/en/latest/ref/configuration/master.html

Complete salt master config file

Configure SaltStack
Device *pillar*

Under the **pillar_roots** directory (as configured in **`/etc/salt/master`**):

`/etc/salt/pillar/device1.sls`

```
proxy:
  proxytype: napalm
  driver: junos
  host: hostname_or_ip_address
  username: my_username
  passwd: my_password
```

Mandatory

Choose between: junos, eos, ios, iosxr, nxos, etc. See the complete list.

Complete documentation at: https://docs.saltstack.com/en/develop/ref/proxy/all/salt.proxy.napalm.html

Configure SaltStack

The *top* file

Under the **pillar_roots** directory (as configured in **`/etc/salt/master`**):

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

```
base:
  device1:
    - device1
  device2:
    - device2
```

Environment name

Useful to have different envs: prod, qa, develop etc.

minion ID

This is how the device will be identified from now on.
It can be anything, does **not** need to match with the *.sls* file or the hostname.

.sls file to be included

Specify the name of the .sls file descriptor (earlier defined).
Do **NOT** include the *.sls* extension.

Salt CLI syntax

Selecting the devices we need to run the command.

Targeting can be complex:
https://docs.saltstack.com/en/latest/topics/targeting/

```
$ sudo salt <target> <function> [<arguments>]
```

Function name, as specified in the module documentation.

For example if we need BGP-related commands, we’ll look at the BGP module.

Other examples: dnsutil.A, net.arp, net.lldp, net.traceroute etc.

Function arguments, as specified in the module documentation.
Some functions do not require any arguments.

Salt CLI syntax
Examples

```
$ sudo salt 'edge*' net.traceroute 8.8.8.8
# execute traceroute on all devices whose minion ID starts with ‘edge’
$ sudo salt -N NA transit.disable cogent
# disable Cogent in North-America
$ sudo salt -G 'os:junos' net.cli “show version”
# execute ‘show version’ on all devices running JunOS
$ sudo salt -C 'edge* and G@os:iosxr and G@version:6.0.2' net.arp
# get the ARP tables from devices whose ID starts with edge*, running IOS-XR 6.0.2
$ sudo salt -G 'model:MX480' probes.results
# retrieve the results of the RPM probes from all Juniper MX480 routers
```

‘NA’ is a nodegroup:

https://docs.saltstack.com/en/latest/topics/targeting/nodegroups.html

Salt CLI syntax
Output example

Default output style: nested.

```
$ sudo salt edge01.iad01 net.arp
edge01.iad01:
    ----------
    out:
        |_
          ----------
          age:
              129.0
          interface:
              ae2.100
          ip:
              10.0.0.1
          mac:
              00:0f:53:36:e4:50
        |_
          ----------
          age:
              1101.0
          interface:
              xe-0/0/3.0
          ip:
              10.0.0.2
          mac:
              00:1d:70:83:40:c0
```

Salt CLI syntax
Outputters

```
$ salt --out=json edge01.iad01 net.arp
[
  {
    "interface": "ae2.100",
    "ip": "10.0.0.1",
    "mac": "00:0f:53:36:e4:50",
    "age": 129.0
  },
  {
    "interface": "xe-0/0/3.0",
    "ip": "10.0.0.2",
    "mac": "00:1d:70:83:40:c0",
    "age": 1101.0
  },
```

Using the --out optional argument, one can select the output format.

```
$ salt --out=yaml edge01.iad01 net.arp
edge01.iad01:
  comment: ''
  out:
  - age: 129.0
    interface: ae2.100
    ip: 10.0.0.1
    mac: 00:0f:53:36:e4:50
  - age: 1101.0
    interface: xe-0/0/3.0
    ip: 10.0.0.2
    mac: 00:1d:70:83:40:c0
```

Other outputters: https://docs.saltstack.com/en/develop/ref/output/all/index.html

Configuration management
Cross vendor templating (1)

/etc/salt/templates/example.jinja

```
{%- set router_vendor = grains.vendor -%}
{%- set hostname = pillar.proxy.host -%}
{%- if router_vendor|lower == 'juniper' %}
system {
    host-name {{ hostname }}.lab;
}
{%- elif router_vendor|lower in ['cisco', 'arista'] %}
{# both Cisco and Arista have the same syntax for hostname #}
hostname {{ hostname }}.lab
{%- endif %}
```

Get the device vendor from the grains.

Hostname taken from the pillar.

Multiple templating systems supported (not only Jinja):
https://docs.saltstack.com/en/latest/ref/renderers/all/index.html

Configuration management
Cross vendor templating (2)

```
$ sudo salt '*' net.load_template /etc/salt/templates/example.jinja
edge01.bjm01:
    ----------
    already_configured:
        False
    comment:
    diff:
        @@ -35,7 +35,7 @@
         logging console emergencies
         logging host 192.168.0.1
         !
        -hostname edge01.bjm01
        +hostname edge01.bjm01.lab
         !
    result:
        True
```

Absolute path

Arista device

```
edge01.flw01:
    ----------
    already_configured:
        False
    comment:
    diff:
        [edit system]
        -   host-name edge01.flw01;
        +   host-name edge01.flw01.lab;
    result:
        True
```

Juniper device

Many vendors, one simple template!

Configuration management
Debug mode

```
$ sudo salt edge01.flw01 net.load_template salt://example.jinja debug=True
edge01.flw01:
    ----------
    already_configured:
        False
    comment:
    diff:
        [edit system]
        -  host-name edge01.flw01;
        +  host-name edge01.flw01.lab;
    loaded_config:
        system {
            host-name edge01.flw01.lab;
        }
    result:
        True
```

Salt path
Template stored under the file_roots

Debug mode

The result of template rendering. Not necessarily equal to the diff.

Note: Jinja is painful to debug. This option is very helpful.
See more debugging tools

Configuration management
Remote templates

Yes, they can also be elsewhere.
Available options: *salt://*, *ftp://*, *http://*, *https://*,
version control, cloud storage providers etc.

```
$ sudo salt -G 'os:ios' net.load_template http://bit.ly/2gKOj20 peers="['172.17.17.1', '172.17.17.2']"
```

Matches all devices running IOS

Loads external template from http://bit.ly/2gKOj20 which shortens the link to the NAPALM native template for IOS.

Configuration management

Advanced templating: reusing existing data (1)

/etc/salt/templates/arp_example.jinja

```
{%- set arp_output = salt.net.arp() -%}
{%- set arp_table = arp_output['out'] -%}

{%- if grains.os|lower == 'iosxr' %} {# if the device is a Cisco IOS-XR #}
  {%- for arp_entry in arp_table %}
arp {{ arp_entry['ip'] }} {{ arp_entry['mac'] }} arpa
  {%- endfor -%}
{%- elif grains.vendor|lower == 'juniper' %} {# or if the device is a Juniper #}
interfaces {
  {%- for arp_entry in arp_table %}
  {{ arp_entry['interface'] }} {
    family inet {
      address {{ arp_entry['ip'] }} {
       arp {{ arp_entry['ip'] }} mac {{ arp_entry['mac'] }};
      }
    }
  }
  {%- endfor %}
}
{%- endif %}
```

Retrieving the ARP table using the net.arp function.

Configuration management

Advanced templating: reusing existing data (1)

```
$ sudo salt edge01.flw01 net.load_template salt://arp_example.jinja
edge01.flw01:
    ----------
    already_configured:
        False
    comment:
    diff:
        [edit interfaces xe-0/0/0 unit 0 family inet]
        +       address 10.10.2.2/32 {
        +           arp 10.10.2.2 mac 0c:86:10:f6:7c:a6;
        +       }
        [edit interfaces ae1 unit 1234]
        +      family inet {
        +          address 10.10.1.1/32 {
        +              arp 10.10.1.1 mac 9c:8e:99:15:13:b3;
        +          }
        +      }
    result:
        True
```

Configuration management
Advanced templating: reusing existing data (2)

/etc/salt/templates/route_example.jinja

```
{%- set route_output = salt.route.show('0.0.0.0/0', 'static') -%}
{%- set default_route = route_output['out'] -%}

{%- if not default_route -%} {# if no default route found in the table #}
  {%- if grains.vendor|lower == 'juniper' -%}
routing-options {
    static {
        route 0.0.0.0/0 next-hop {{ pillar.default_route_nh }};
    }
}
  {%- elif grains.os|lower == 'iosxr' -%}
  router static address-family ipv4 unicast 0.0.0.0/0 {{ pillar.default_route_nh }}
  {%- endif %}
{%- endif -%}
```

Retrieving the static route data using the route.show function.

This requires appending a new line in the device pillar:

```
default_route_nh: 1.2.3.4
```

Configuration management

Advanced templating: reusing existing data (2)

```
$ sudo salt 'edge01.oua01' net.load_template salt://route_example.jinja debug=True
edge01.oua01:
    ----------
    already_configured:
        False
    comment:
    diff:
        ---
        +++
        @@ -3497,6 +3497,7 @@
         !
         router static
          address-family ipv4 unicast
        +  0.0.0.0/0 1.2.3.4
           172.17.17.0/24 Null0 tag 100
    loaded_config:
        router static address-family ipv4 unicast 0.0.0.0/0 1.2.3.4
    result:
        True
```

Other simple examples

- Using postgres.psql_query populate a table in a Postgres database with the network interfaces details (retrieved using net.interfaces)
- Using bgp.neighbors remove from the BGP config neighbors in *Active* state
- Using ntp.stats, remove unsynchronised NTP peers
- Using net.environment, push high temperature notifications in Slack

The list can be nearly infinite - depends only on your own use case.

There are thousands of functions already available:

https://docs.saltstack.com/en/develop/ref/modules/all/index.html

Note: the examples above are implemented more elegant using states, beacons, reactors, etc.

Real-world orchestration example
State: update NTP peers (1)

Append directly these lines in the device pillar, or define in external file and include:

/etc/salt/pillar/ntp_config.sls

```
ntp.peers:
  - 10.10.1.1
  - 10.10.2.2
ntp.servers:
  - 172.17.17.1
  - 172.17.19.1
```

/etc/salt/pillar/device1.sls

```
proxy:
  proxytype: napalm
  driver: junos
  host: hostname_or_ip_address
  username: my_username
  passwd: my_password
include:
  - ntp_config
```

Better to use the *include*, as multiple devices can have the same NTP peers etc.

When including, strip the *.sls* extension!

Real-world orchestration example

State output: update NTP peers (2)

```
$ sudo salt 'edge01.jnb01' state.sls router.ntp
edge01.jnb01:
----------
          ID: update_my_ntp_config
    Function: netntp.managed
      Result: True
     Started: 09:50:41.228728
    Duration: 16813.319 ms
     Changes:
              ----------
              peers:
                  ----------
                  removed:
                      - 10.10.1.1
              servers:
                  ----------
                  added:
                      - 172.17.17.1
                      - 172.17.19.1
Summary for edge01.jnb01
------------
Succeeded: 1 (changed=1)
Failed:    0
------------
Total states run:     1
```

Real-world orchestration example
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-run state.event pretty=True
salt/job/20170110130619367337/new    {
    "_stamp": "2017-01-10T13:06:19.367929",
    "arg": [],
    "fun": "probes.results",
    "jid": "20170110130619367337",
    "minions": [
        "edge01.bjm01"
    ],
    "tgt": "edge01.bjm01",
    "tgt_type": "glob",
    "user": "mircea"
}
```

Unique job tag

Real-world orchestration example
Reactor

Using the job tags, you can identify events (triggers) and react (action):

/etc/salt/master

```
reactor:
  - 'salt/job/*/ret/*':
    - salt://reactor_example.sls
```

Unique job tags (regular expression): in this example will match any job returns

When this event occurs, execute this reactor descriptor.

/etc/salt/reactors/reactor_example.sls

```
invoke_orchestrate_file:
  runner.state.orchestrate:
    - mods: orch.do_complex_thing
    - pillar:
        event_tag: {{ tag }}
        event_data: {{ data | json() }}
```

Real-world orchestration example
Beacon example: inotify

Beacons let you use the Salt event system to monitor non-Salt processes.

/etc/salt/proxy

```
beacons:
  inotify:
    /etc/salt/pillar/ntp_config.sls:
      mask:
        - modify
    disable_during_state_run: True
```

Will fire an event when updating */etc/salt/pillar/ntp_config.sls*

Real-world orchestration example
Beacon event tag example

This event is fired when a change is made and saved to */etc/salt/pillar/ntp_config.sls*:

```
salt/beacon/device1/inotify//etc/salt/pillar/ntp_config.sls       {
 "_stamp": "2017-01-09T15:59:37.972753",
 "data": {
     "change": "IN_IGNORED",
     "id": "device1",
     "path": "/etc/salt/pillar/ntp_config.sls"
 },
 "tag": "salt/beacon/device1/inotify//etc/salt/pillar/ntp_config.sls"
}
```

Using the reactor system, one can match these event tags and take actions when they happen.

Real-world orchestration example
Beacon event tag example

React when the */etc/salt/pillar/ntp_config.sls* is changed

/etc/salt/master

```
reactor:
  - 'salt/beacon/*/inotify//etc/salt/pillar/ntp_config.sls':
    - salt://run_ntp_state_on_pillar_update.sls
```

/etc/salt/reactors/run_ntp_state_on_pillar_update.sls

```
run_ntp_state:
  local.state.sls:
    - tgt: {{ data['id'] }}
    - arg:
      - router.ntp
```

This is how the reactor system knows that a state execution is required.

Run the state against the minion ID that triggered the event

Run the ntp state defined earlier (slides #35-#36).

Real-world orchestration example

Beacon event tag example

… and that’s it!
From now on, whenever you update */etc/salt/pillar/ntp_config.sls*,
it will automatically update your routers’ config.

And you maintain entities of data, not pseudo-formatted text files,
regardless on the device vendor.

Advanced topics

Mine

Embedded caching

/etc/salt/pillar/device1.sls

```
mine_functions:
  net.interfaces: []
  net.lldp: []
  net.arp: []
mine_interval: 5
```

What to cache

How often to update (in minutes)

Read more: https://docs.saltstack.com/en/latest/topics/mine/

Advanced topics
The Salt API

You can also execute commands remotely, via HTTPS

Easy to setup, easy to use

/etc/salt/master

```
rest_cherrypy:
  port: 8001
  ssl_crt: /etc/nginx/ssl/my_certificate.pem
  ssl_key: /etc/nginx/ssl/my_key.key
```

```
curl -sSk
https://salt-master-ns-or-ip:8001/run \
    -H 'Content-type: application/json' \
    -d '[{
        "client": "local",
        "tgt": "<target>",
        "fun": "net.arp",
        "username": "my username",
        "password": "my password",
        "eauth": "pam"
    }]'
```

Sources

https://github.com/mirceaulinic/talks/tree/master/APRICOT2017/sources

More advanced topics

- Orchestration: define complex workflows
  https://docs.saltstack.com/en/latest/topics/orchestrate/index.html
- Publish events to external services (e.g.: logstash, hipchat)
  https://docs.saltstack.com/en/develop/ref/engines/all/index.html
- Pillar: load data from external services, not just static
  https://docs.saltstack.com/en/develop/ref/pillar/all/
- Custom authentication methods for the minions
  https://docs.saltstack.com/en/develop/ref/auth/all/index.html
- Forward outputs in external data systems on runtime
  https://docs.saltstack.com/en/develop/ref/returners/all/index.html

How can you contribute?

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

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

Need help/advice?

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

By email:

- Mircea Ulinic: mircea@cloudflare.com
- Jerome Fleury: jf@cloudflare.com

Questions

?

References

Authentication system
Beacons
Engines
Event System
Grains
Jinja
load_template documentation
Master config file, example
Master configuration options
Mine
NAPALM
NAPALM BGP execution module functions
NAPALM Grains
NAPALM Installation
NAPALM network execution module functions
NAPALM NTP execution module functions
NAPALM Proxy
NAPALM route execution module functions
Nested outputter
NETAPI Modules
Netconfig state
Node Groups
NTP state
Orchestration
Pillar
Pillar modules
Proxy Minion
Reactor
REST CherryPy
Returners
Runners
Salt 2016.11 (Carbon) release notes
Salt Get Started
Salt Installation
Salt Walkthrough
SaltStack Package Repo
SNMP state
States
Targeting minions
The Top file
Users state
YAML