Network automation at scale
Up and running in 60 minutes

Mircea Ulinic
Cloudflare, London

NANOG 69 Washington, D.C.
February 2017
Why us?

- **How big?**
  - Four+ million zones/domains
  - Authoritative for ~40% of Alexa top 1 million
  - 43+ billion DNS queries/day
    - Second only to Verisign

- **100+ anycast locations globally**
  - 50 countries (and growing)
  - Many hundreds of network devices
To automate, I have to learn Python or another programming language.
To automate, I have to learn Python or another programming language.
Prerequisites

- No programming skills required (but very welcome)!
- Basic system ops
- Networking (of course)
- Basic **YAML** & **Jinja** understanding
  (6 simple rules is all you need for the beginning)
Agenda

- Meet the tools
- Install the tools
- Configure Vagrant VMs (demo only)
- Configure SaltStack
- CLI syntax
- Configuration management
- Advanced topics
Meet the Tools

Live setup

- Access to a remote server
- OR
- Vagrant + VM(s) from your favourite vendor(s)

**NOTE**: Vagrant is used for demo only!
The power of SaltStack can be seen when managing high number of real network devices!
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

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 included 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
Install the tools

Install NAPALM

$ pip install napalm

See Complete installation notes
Install the tools
Install SaltStack

$ sudo apt-get install salt-master
$ sudo apt-get install salt-proxy

See Complete installation notes
Configure Vagrant

This assumes **Vagrant** and **VirtualBox** are already installed

Vagrantfile examples:

**What I use**

**Something simpler**

**NOTE**: skip this section if you are running in a real network environment (preferable)
Configure Vagrant
Download vEOS

Go to Arista software download (account required)

Select any .box file, but make sure that VEOS_BOX matches the name in the Vagrantfile.
Configure Vagrant
Download vSRX

$ vagrant box add juniper/ffp-12.1X47-D20.7-packetmode

==> box: Loading metadata for box 'juniper/ffp-12.1X47-D20.7-packetmode'

box: URL: https://vagrantcloud.com/juniper/ffp-12.1X47-D20.7-packetmode

This box can work with multiple providers! The providers that it can work with are listed below. Please review the list and choose the provider you will be working with.

1) virtualbox
2) vmware_desktop

Enter your choice: 1

==> box: Adding box 'juniper/ffp-12.1X47-D20.7-packetmode' (v0.5.0) for provider: virtualbox

box: Downloading:
https://atlas.hashicorp.com/juniper/boxes/ffp-12.1X47-D20.7-packetmode/versions/0.5.0/providers/virtualbox.box

==> box: Successfully added box 'juniper/ffp-12.1X47-D20.7-packetmode' (v0.5.0) for 'virtualbox'!
Configure Vagrant

Start Vagrant boxes

$ vagrant up vsrx

Bringing machine 'vsrx' up with 'virtualbox' provider...

==> vsrx: Setting the name of the VM: mirucha_vsrx_1483551699725_41640

==> vsrx: Clearing any previously set network interfaces...

==> vsrx: Preparing network interfaces based on configuration...

  vsrx: Adapter 1: nat
  vsrx: Adapter 2: intnet
  vsrx: Adapter 3: intnet
  vsrx: Adapter 4: intnet
  vsrx: Adapter 5: intnet

==> vsrx: Forwarding ports...

  vsrx: 22 (guest) => 12202 (host) (adapter 1)
  vsrx: 830 (guest) => 12830 (host) (adapter 1)
  vsrx: 80 (guest) => 12280 (host) (adapter 1)

==> vsrx: Booting VM...

==> vsrx: Waiting for machine to boot. This may take a few minutes...

  vsrx: SSH address: 127.0.0.1:12202
  vsrx: SSH username: vagrant
  vsrx: SSH auth method: private key

  vsrx:
  
  vsrx: Vagrant insecure key detected. Vagrant will automatically replace
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...

*written by the user, generally one file per device*

**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](https://docs.saltstack.com/en/latest/topics/tutorials/walkthrough.html)
Configure SaltStack
Master config

```yaml
/etc/salt/master

file_roots:
  base:
    - /etc/salt/states

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](https://docs.saltstack.com/en/latest/ref/configuration/master.html)

**Complete salt master config file**
Configure SaltStack
Proxy config

/etc/salt/proxy

- master: localhost
- pki_dir: /etc/salt/pki/proxy
- cachedir: /var/cache/salt/proxy
- multiprocessing: False
- mine_enabled: True

Configure SaltStack Device *pillar*

Under the `pillar_root` 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](https://docs.saltstack.com/en/develop/ref/proxy/all/salt.proxy.napalm.html).

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

- **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.

- **Environment name**
  - Useful to have different envs: prod, qa, develop etc.

- **.sls file to be included**
  - Specify the name of the `.sls` file descriptor (earlier defined).
  - Do **NOT** include the `.sls` extension.
Configure SaltStack

**master systemd file** (optional)

```bash
[Unit]
Description=Salt Master
Requires=network.target
After=network.target

[Service]
Type=forking
PIDFile=/var/run/salt-master.pid
ExecStart=/usr/bin/salt-master -d
Restart=on-failure
RestartSec=15

[Install]
WantedBy=multi-user.target
```
Configure SaltStack *proxy systemd file* (optional)

```ini
[Unit]
Description=Salt proxy minion
After=network.target

[Service]
Type=简单
# ***NOTE*** the virtualenv here! Your location may vary!
ExecStart=/usr/bin/salt-proxy -l debug --proxyid %I
User=root
Group=root
Restart=always
RestartPreventExitStatus=SIGHUP
RestartSec=5

[Install]
WantedBy=default.target
```
Configure SaltStack
Start the salt-master

● With systemd:
  ○ $ sudo systemctl start salt-master

● Without systemd:
  ○ $ sudo salt-master -d
Configure SaltStack

Start the salt-proxy processes

• With systemd:
  ○ $ sudo systemctl start salt-proxy@device1
  ○ $ sudo systemctl start salt-proxy@device2

• Without systemd:
  ○ $ sudo salt-proxy -d --proxyid device1
  ○ $ sudo salt-proxy -d --proxyid device2

As configured in the top file.
Configure SaltStack
Accept the proxies connection to the master

For each device, accept the minion key:

```bash
$ sudo salt-key -a device1
The following keys are going to be accepted:
Unaccepted Keys:
device1
Proceed? [n/Y] y
Key for minion device1 accepted.
```

minion ID
As configured in the top file.

This is due to security reasons.
More about salt-key: https://docs.saltstack.com/en/latest/ref/cli/salt-key.html

**NOTE:** Accepting the minion keys can be automated as well.
Done!
You are now ready to automate your network!
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.
$ 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.
Salt CLI syntax

Outputters

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

Other outputters: [https://docs.saltstack.com/en/develop/ref/output/all/index.html](https://docs.saltstack.com/en/develop/ref/output/all/index.html)
Configuration management
Load static config

$ sudo salt -G 'vendor:arista' net.load_config text='ntp server 172.17.17.1'
edge01.bjm01:
    ----------
    already_configured:
        False
    comment:
    diff:
        @@ -42,6 +42,7 @@
        ntp server 10.10.10.1
        ntp server 10.10.10.2
        ntp server 10.10.10.3
+ntp server 172.17.17.1
        ntp serve all
    !
    result:
        True
edge01.pos01:
    ----------
    already_configured:
        True
    comment:
    diff:
    result:
        True

Match all Arista devices from the network.

No changes required on this device.

Config diff
Configuration management
Load static config: dry-run

$ sudo salt edge01.bjm01 net.load_config text='ntp server 172.17.17.1' test=True

edge01.bjm01:
-------
already_configured:
False
comment:
  Configuration discarded.
diff:
@@ -42,6 +42,7 @@
 ntp server 10.10.10.1
 ntp server 10.10.10.2
 ntp server 10.10.10.3
+ntp server 172.17.17.1
 ntp serve all
!
result:
  True

Dry-run mode

Changes are discarded.
```
$ sudo salt edge01.bjm01 net.load_config /home/mircea/arista_ntp_servers.cfg test=True
edge01.bjm01:

--------

already_configured:
False

comment:
Configuration discarded.

diff:
@@ -42,6 +42,10 @@
  ntp server 10.10.10.2
  ntp server 10.10.10.3
 +ntp server 172.17.17.1
 +ntp server 172.17.17.2
 +ntp server 172.17.17.3
 +ntp server 172.17.17.4
  ntp serve all
!

result:
True
```
$ sudo salt edge01.bjm01 net.load_template set_hostname template_source='hostname {{ host_name }}' host_name='arista.lab'

Observe the function name is: 
net.load_template

Inline template

Template var

NOTE: the template is evaluated on the minion
$ sudo salt edge01.bjm01 net.load_template set_hostname template_source='hostname {{ grains.model }}.lab'
edge01.bjm01:
    --------------
    already_configured: False
    comment: 
    diff:
@@ -35,7 +35,7 @@
  logging console emergencies
  logging host 192.168.0.1
  
  -hostname edge01.bjm01
  +hostname DCS-7280SR-48C6-M-R.lab
  
  result:
  True

Router model is collected from the grains
Configuration management
Cross vendor templating (1)

Get the device vendor from the grains

Hostname already specified in the pillar.
Many vendors, one simple template!
$ sudo salt edge01.flw01 net.load_template /home/mircea/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

Configuration management

Debug mode

Absolute path

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.
$ sudo salt edge01.flw01 net.load_template salt://templates/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

Translated to **file_roots**, as specified in the master config file - see slide #21.

E.g.: if **file_roots** is configured as `/etc/salt/states/`, the physical location of the template is
`/etc/salt/states/templates/example.jinja`

**Note:**
Under **file_roots**, one can also add:
`/etc/salt/templates`, define the template file under the path: `/etc/salt/templates/example.jinja` and call using:
`salt://example.jinja`

```
```

Matches all devices running IOS

Loads external template from [http://bit.ly/2gKOj20](http://bit.ly/2gKOj20) which shortens the link to the NAPALM native template for IOS.
Configuration management

Advanced templating: reusing existing data (1)

Retrieving the ARP table using the `net.arp` function.
$ 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
Retrieving the static route data using the \texttt{route.show} function.

This requires appending a new line in the device pillar:

\texttt{default_route_nh: 1.2.3.4}
**Configuration management**

**Advanced templating: reusing existing data (2)**

```bash
$ 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
```
Homework: 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.
Advanced topics
States, schedulers, reactors, beacons, API

These are advanced topics, that require the user to read carefully the documentation.

Using these types of modules, one can control the configuration based on events, either external or internal, e.g.:

- BGP neighbor down triggers a BGP configuration change
- Git pull-request merged triggers configuration update
- High temperature alert triggers a notification post in a Slack channel
- ChatOps
- etc.
Advanced topics

State

A state ensures that on the devices you have configured what you expect to be. What’s not defined in the pillar, it will be removed; what’s not on the device, but it’s defined in the pillar, will be added.

Integrated states:

- netntp
- netsnmp
- netusers
- probes
- netconfig (very important; will be added in the next release: Nitrogen)
Advanced topics
State example: 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
```

```
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!
Advanced topics

State example: update NTP peers (1)

As configured under *file_roots*

```
/etc/salt/states/router/ntp.sls
```

Take the NTP peers/servers from the pillar (earlier defined)

Pass them as state arguments

**Best practice:**
Although not mandatory, use the *json()* filter to explicitly serialize objects.

This is the state virtualname, more doc:
Advanced topics

State example: update NTP peers (3)

Include:
- ntp

This file is mandatory.

Include the earlier defined state SLS file.

```
$ sudo salt <target> state.sls router.ntp
```
Advanced topics

State output example:
update NTP peers (3)

$ 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
Total run time: 16.813 s
Ensure the configuration is consistent, without running commands manually.

```
/etc/salt/proxy
```

```
schedule:
  keep_ntp_config_updated:
  function: state.sls
  args: router.ntp
  days: 1
```

The previous command will be executed automatically every day and ensures the NTP config is as expected.
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
Advanced topics
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/states/reactor/example.sls

invoke_orchestrate_file:
  runner.state.orchestrate:
  - mods: orch.do_complex_thing
  - pillar:
    event_tag: {{ tag }}
    event_data: {{ data | json() }}
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` (using the same example as in slides #52-#54)
Using the reactor system, one can match these event tags and take actions when they happen.

```
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"
}
```

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

Beacon event tag example

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

/etc/salt/master

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

/etc/salt/states/reactor/run_ntp_state_when_file_changed.sls

```yaml
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.
... 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.
Embedded caching

`mine_functions`
- `net.interfaces`: []
- `net.lldp`: []
- `net.arp`: []

`mine_interval`: 5

What to cache

How often to update (in minutes)

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

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

```
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"
  ]]'
```

/etc/salt/master

rest_cherrypy:
  port: 8001

ssl_crt: /etc/nginx/ssl/my_certificate.pem

ssl_key: /etc/nginx/ssl/my_key.key
More advanced topics

- Orchestration: define complex workflows
- 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
Real world example:
Cloudflare’s self-resilient network
Monitoring carriers (transit providers)

```bash
mircea@re0.edge01.iad01> show configuration services rpm | display set | match 1299 | match probe-type
set services rpm probe transit test t-edge01.scl01-1299-12956-4 probe-type icmp-ping
set services rpm probe transit test t-edge01.eze01-1299-6762-4 probe-type icmp-ping
set services rpm probe transit test t-edge01.lax01-1299-1299-4 probe-type icmp-ping
set services rpm probe transit test t-edge01.eze01-1299-12956-4 probe-type icmp-ping
set services rpm probe transit test t-edge01.mia01-1299-1299-4 probe-type icmp-ping
set services rpm probe transit test t-edge01.lhr01-1299-1299-4 probe-type icmp-ping
set services rpm probe transit test t-edge01.ams01-1299-1299-4 probe-type icmp-ping
set services rpm probe transit test t-edge01.fra03-1299-1299-4 probe-type icmp-ping
set services rpm probe transit test t-edge01.dfw01-1299-1299-4 probe-type icmp-ping
set services rpm probe transit test t-edge01.sea01-1299-1299-4 probe-type icmp-ping
```

JunOS: RPM
https://www.juniper.net/documentation/en_US/junos12.1x46/topics/concept/security-rpm-overview.html

IOS-XR: ISPLA
How many probes?

$ sudo salt-run transits.probes show_count=True
Generated 7248 probes.

Generated using:

- net.ipaddrs
- net.interfaces
- bgp.neighbors
- bgp.config

All integrated by default in SaltStack.
How are they installed?

$ sudo salt 'edge*' state.sls router.probes
edge01.dfw01:
----------
ID: cf_probes
Function: probes.managed
Result: True
Comment: Configuration updated
Started: 23:00:17.228171
Duration: 10.206 s
Changes:
----------
added:
----------
transit:
----------
t-edge01.sjc01-1299-1299-4:
  source: 1.2.3.4
target: 5.6.7.8
t-edge01.den01-1299-1299-4:
  source: 10.11.12.13
target: 14.15.16.17
t-edge01.den01-174-174-4:
  source: 18.19.20.21
target: 22.23.24.25
t-edge01.den01-4436-4436-4:
  source: 26.27.28.29
target: 30.31.32.33
----------
removed:
----------
updated:
----------

$ cat /etc/salt/pillar/probes_edge01_dfw01.sls
probes.config:
  transit:
    t-edge01.sjc01-1299-1299-4:
      source: 1.2.3.4
target: 5.6.7.8
t-edge01.den01-1299-1299-4:
      source: 10.11.12.13
target: 14.15.16.17
t-edge01.den01-174-174-4:
      source: 18.19.20.21
target: 22.23.24.25
t-edge01.den01-4436-4436-4:
      source: 26.27.28.29
target: 30.31.32.33
Spaghetti
Retrieving probes results

```bash
$ sudo salt 'edge*' probes.results

dge01.dfw01:
    -----------
    out:
        -----------
        transit:
            -----------
            t-edge01.sjc01-1299-1299-4:
                -----------
                current_test_avg_delay: 24.023
                current_test_max_delay: 28.141
                current_test_min_delay: 23.278
                global_test_avg_delay: 23.936
                global_test_max_delay: 480.576
                global_test_min_delay: 23.105
```
How the Internet looks like nowdays
Self-resilient network
Self-resilient network: HipChat alerts

**Event-action-script** · Sep-30 07:37
Cogent: Disabled in EU
Current alerts per router:
   Routers and their active alerts on transit:
   - edge01.cdg01: 5
   - edge01.otp01: 5
   - edge01.man01: 5
   - edge01.sof01: 5

**Netperf** · Oct-5 10:36
[netperf] Anycast disabled on edge01.mde01

**Event-action-script** · Oct-1 17:26
Comcast: Disabled in NA
Current alerts per router:
   Routers and their active alerts on transit:
   - edge01.dfw01: 3
   - edge01.bos01: 6
   - edge01.den01: 4
   - edge01.phl01: 4
   - edge01.atl01: 2
How often?

1688 request-reply pairs during a random window of 7 days
~ 120 config changes / day in average
0 human intervention
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

By email:

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

Arista Software download
Authentication system
Beacons
Engines
Event System
Grains
Jinja
load_template documentation
Master config file, default
Master config file, example
Master configuration options
Master systemd file
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
NAPALM SNMP execution module functions
NAPALM users execution module functions
Nested outputter
NETAPI Modules
Netconfig state
Node Groups
NTP state
Orchestration
Output modules
Pillar
Pillar modules
Proxy config file, default
Proxy config file, example
Proxy Minion
Proxy systemd file
Reactor
REST CherryPy
References

Returners
Runners
Salt 2016.11 (Carbon) release notes
Salt Get Started
Salt Installation
Salt Walkthrough
Salt-key
SaltStack Package Repo
SNMP state
States
Targeting minions
The Top file
Users state
Vagrant boxes, HashiCorp
Vagrant Installation
Vagrantfile example 1
Vagrantfile example 2
VirtualBox Installation
YAML