Automating Networks Using Salt, Without Running Proxy Minions

Mircea Ulinic

RIPE 79, October 2019
Rotterdam, NL

2012 founded in
New York City

$123M+ funding
raised

80M+ Droplets
launched to date

1.3M+ developers and
teams

480+ employees

3rd largest and fastest
growing cloud provider

Investors

12 data centers in 8 global markets

1500+

peers around the world

1.2PB+

of RAM

50PB

of storage

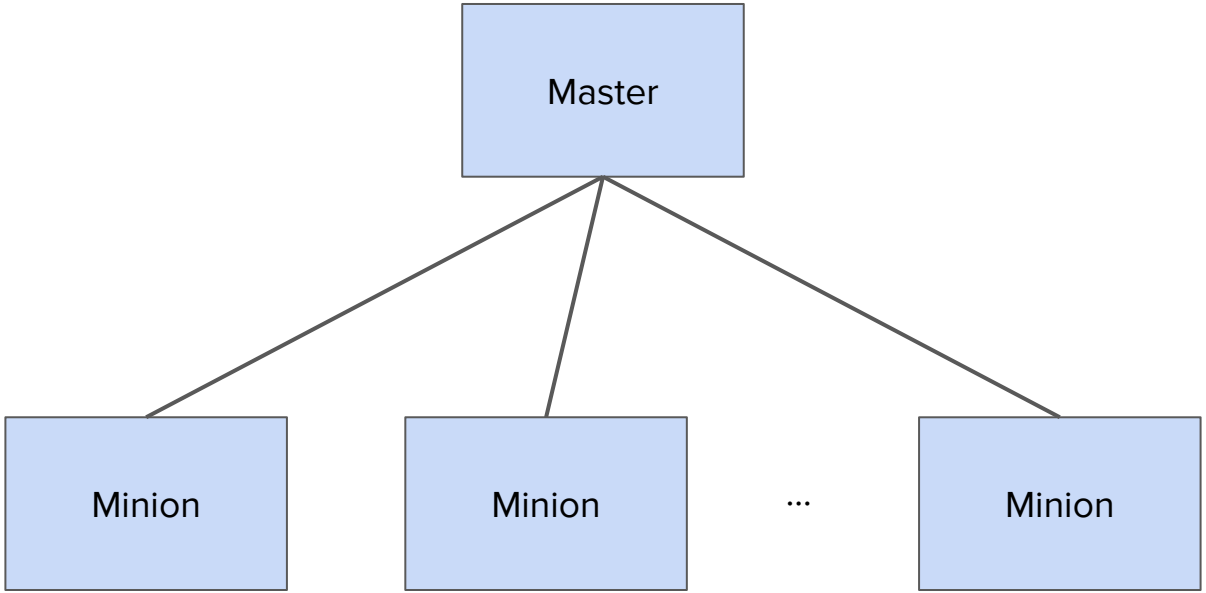
Brief Introduction to Salt

Salt is an event-driven and data-driven configuration management and orchestration tool.

"*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.*"

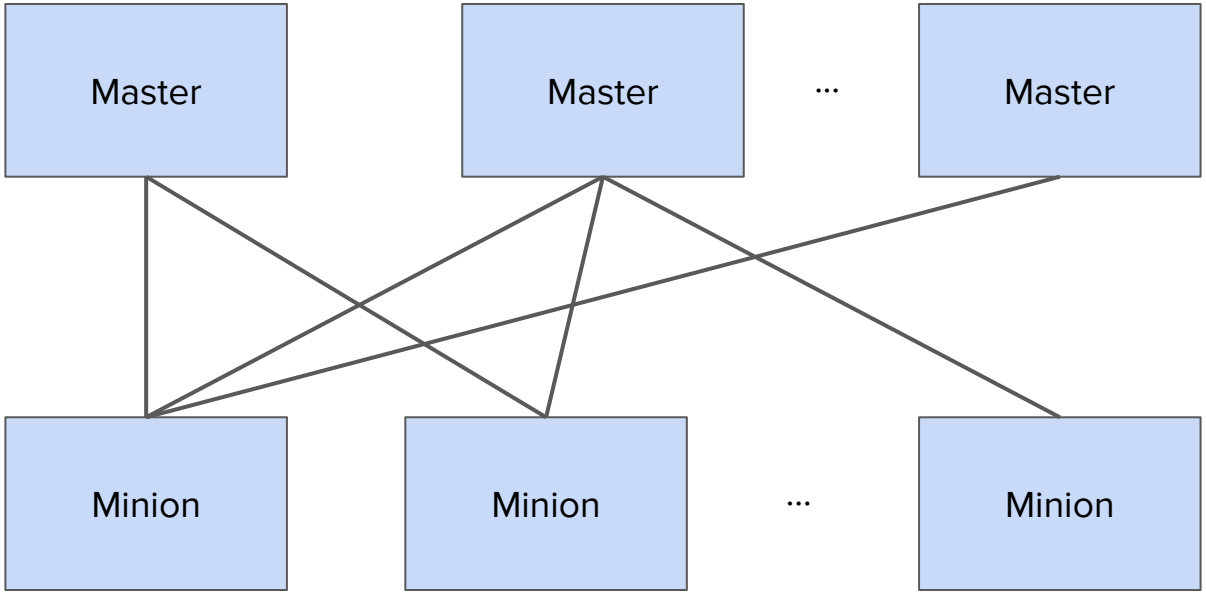
https://docs.saltstack.com/en/getstarted/speed.html

Brief Introduction to Salt: Typical Architecture



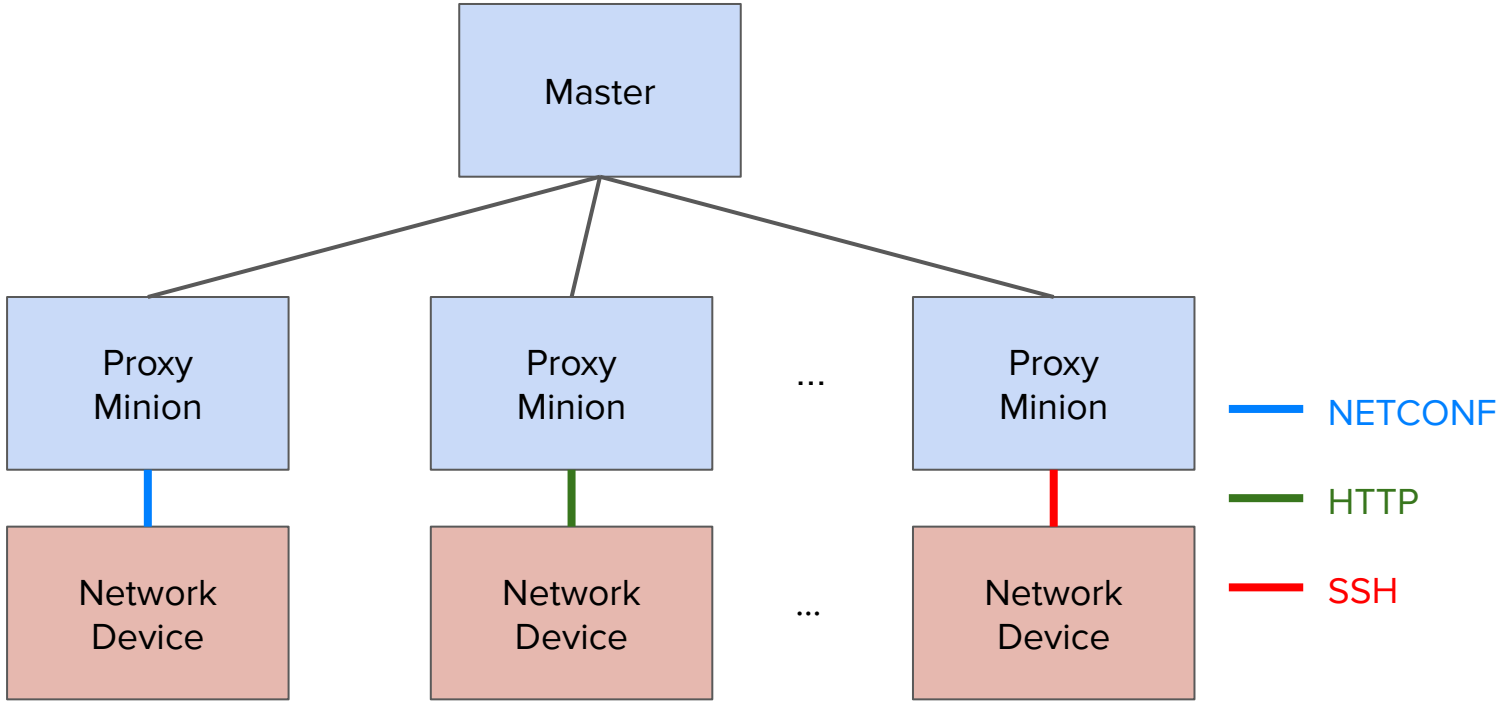
https://docs.saltstack.com/en/latest/topics/topology/index.html

Brief Introduction to Salt: Multi-Master Architecture



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

Brief Introduction to Salt: Network Automation Topology (when using a single Master)



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

Typical Network Automation Topology using Proxies (1)

Proxy Minions are simple processes able to run *anywhere*, as long as:

1) Can connect to the Master.
2) Can connect to the network device (via the channel / API of choice - e.g., SSH / NETCONF / HTTP / gRPC, etc.)

Typical Network Automation Topology using Proxies (2)

Deployment examples include:

- Running as system services
  - On a single server
  - Distributed on various servers
- (Docker) containers
  - E.g., managed by Kubernetes
- Services running in a cloud
  - See, for example, salt-cloud

Typical Network Automation Topology using Proxies (3)

Proxy Minions imply a process always running in the background. That means, whenever you execute a command, Salt is instantly available to run the command. But also means:

- A process always keeping memory busy.
- System services management (one per network device).
- Monitoring, etc.

Not always beneficial, sometimes you just need a one-off command every X weeks / months.

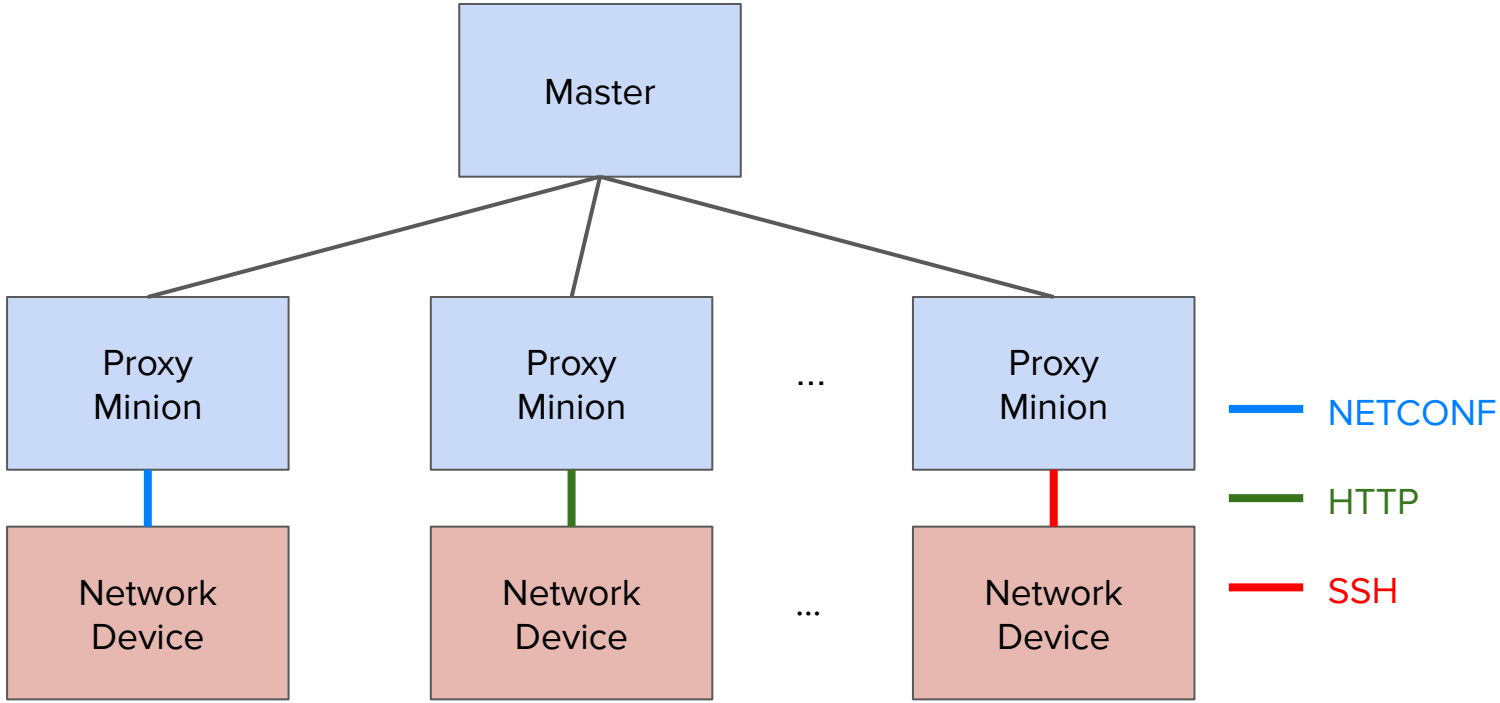
Introducing *salt-sproxy* (Salt Super Proxy)

https://salt-sproxy.readthedocs.io/

Salt plugin to automate the management and configuration of network devices at scale, without running (Proxy) Minions.

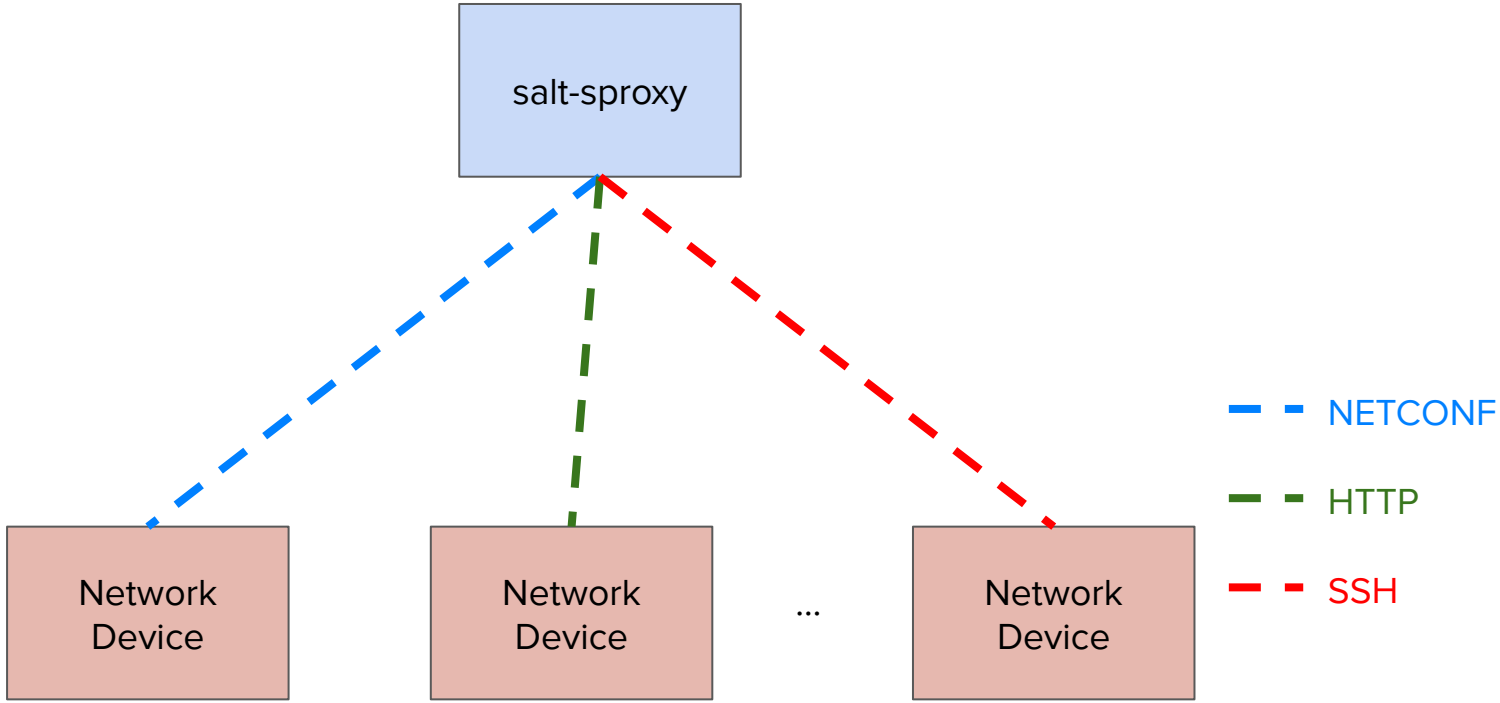
Using *salt-sproxy*, you can continue to benefit from the scalability, flexibility and extensibility of Salt, while you don't have to manage thousands of (Proxy) Minion services. However, you are able to use both *salt-sproxy* and your (Proxy) Minions at the same time.

Remember slide #7?



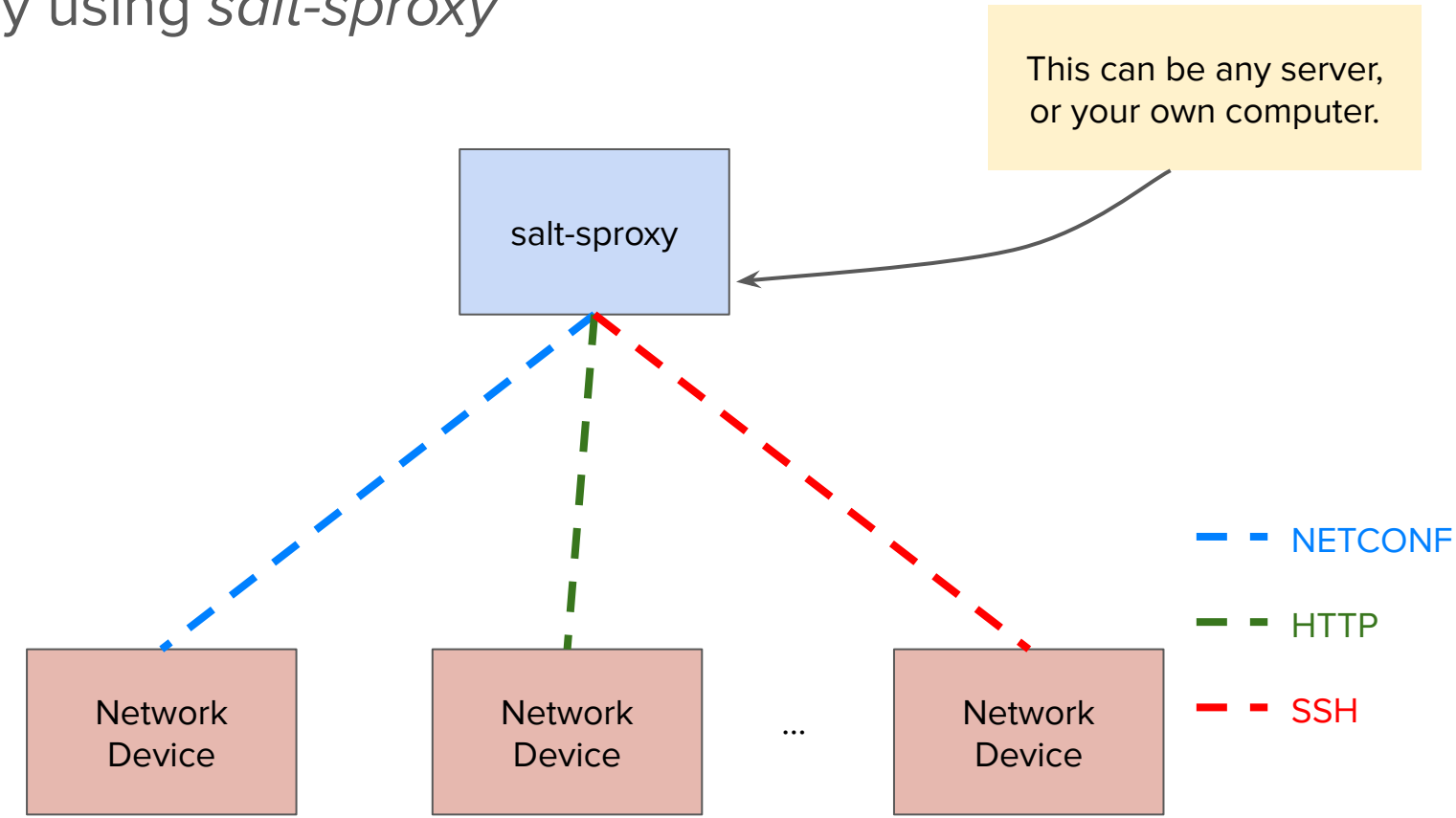
https://docs.saltstack.com/en/latest/topics/topology/index.html

Topology using *salt-sproxy*



https://salt-sproxy.readthedocs.io/

Topology using *salt-sproxy*



https://salt-sproxy.readthedocs.io/

Getting started with *salt-sproxy*: Installation

```
$ pip install salt-sproxy
```

See a recorded demo at:
https://asciinema.org/a/247697?autoplay=1

https://salt-sproxy.readthedocs.io/

Getting started with *salt-sproxy*: Setup example (1)

Build the database of devices you want to manage. For example, as a file:

/srv/pillar/devices.sls

```
devices:
  - name: router1
    driver: junos
  - name: router2
    driver: iosxr
  - name: router3
    proxytype: junos
  - name: switch1
    driver: eos
  - name: fw1
    driver: panos
    host: fw1.firewall.as1234.net
```

https://salt-sproxy.readthedocs.io/en/latest/quick_start.html

Getting started with *salt-sproxy*: Setup example (1)

When working with SLS files, make sure to reference it into the *Pillar top*:

/srv/pillar/top.sls

```
base:
  '*':
    - devices
```

https://docs.saltstack.com/en/latest/topics/tutorials/pillar.html

Getting started with *salt-sproxy*: Setup example (2)

Prepare the connection credentials:

/srv/pillar/proxy.sls

```
proxy:
  proxytype: <proxy type>
  username: <username>
  password: <password>
  [... other params - see doc ...]
```

Where *<proxy type>* is the name of the Proxy Module of choice, see https://docs.saltstack.com/en/latest/ref/proxy/all/index.html; each proxy module may have different arguments required for the connection.

https://salt-sproxy.readthedocs.io/en/latest/quick_start.html

Getting started with *salt-sproxy*: Setup example (2)

For example, using the NAPALM Proxy Module:

/srv/pillar/proxy.sls

```
proxy:
  proxytype: napalm
  username: salt
  password: SaltSPr0xyRocks!
  host: {{ opts.id }}.as1234.net
```

https://salt-sproxy.readthedocs.io/en/latest/quick_start.html

Getting started with *salt-sproxy*: Setup example (2)

For example, using the NAPALM Proxy Module:

/srv/pillar/proxy.sls

```
proxy:
  proxytype: napalm
  username: salt
  password: SaltSPr0xyRocks!
  host: {{ opts.id }}.as1234.net
```

SLS by default means Jinja + YAML. This can be a very powerful feature.

The *host* field is rendered individually per device. For example, the host will be ***router1.as1234.net*** for the device name *router1*, etc.

https://salt-sproxy.readthedocs.io/en/latest/quick_start.html

Getting started with *salt-sproxy*: Setup (2)

Tip: when you want to use your own credentials to manage the device

/srv/pillar/proxy.sls

```
proxy:
  proxytype: napalm
  username: {{ salt.environ.get('USER') }}
  password: ''
  host: {{ opts.id }}.as1234.net
```

The *username* field renders to the username currently logged in (and executing the command).

When *password* is empty, it'll use your SSH key for authentication.

https://salt-sproxy.readthedocs.io/en/latest/quick_start.html

Getting started with *salt-sproxy*: Setup example (2)

Again, make sure to reference the */srv/pillar/proxy.sls* file into the *Pillar top*:

/srv/pillar/top.sls

```
base:
  '*':
    - proxy
    - devices
```

https://docs.saltstack.com/en/latest/topics/tutorials/pillar.html

Getting started with *salt-sproxy*: Setup example (3)

And, finally, let *salt-sproxy* know that the data is loaded from the Pillar:

/etc/salt/master

```
roster: pillar
```

https://salt-sproxy.readthedocs.io/en/latest/quick_start.html

Getting started with *salt-sproxy*: Usage

After these three easy steps, you can start running commands:

```
$ salt-sproxy ‘router*’ --preview-target
- router1
- router2
- router3

$ salt-sproxy ‘router*’ net.arp
... snip ...

$ salt-sproxy ‘router*’ net.load_config \
      text=’set system ntp server 10.0.0.1’ test=True
... snip ...
```

https://salt-sproxy.readthedocs.io/en/latest/quick_start.html

Getting started with *salt-sproxy*: Usage

After these three easy steps, you can start running commands:

```
$ salt-sproxy 'router1' net.load_config \
    text='set system ntp server 10.0.0.1' test=True
router1:
        ----------
        already_configured:
            False
        comment:
            Configuration discarded.
        diff:
            [edit system]
            +   ntp {
            +    server 10.0.0.1;
            +   }
        loaded_config:
        result:
            True
```

https://salt-sproxy.readthedocs.io/en/latest/quick_start.html

Getting started with *salt-sproxy*: Alternative setup

In the previous examples, we used Pillar data (i.e., information that we maintain ourselves) as SLS files , to build the list of devices.

But there can be plenty of other sources where to load this data from, see https://docs.saltstack.com/en/latest/ref/pillar/all/index.html, examples include:

- HTTP API
- Postgres / MySQL database
- Etcd, Consul, Redis, Mongo, etc.
- CSV file :-(

https://salt-sproxy.readthedocs.io/en/latest/roster.html

Getting started with *salt-sproxy*: Alternative setup - NetBox

Update */etc/salt/master* to let *salt-sproxy* know that you want to load the list of devices from NetBox:

/etc/salt/master

```
roster: netbox

netbox:
  url: https://netbox.live/
  token: <token>
```

https://salt-sproxy.readthedocs.io/en/latest/examples/netbox.html

Using salt-sproxy via the Salt REST API

Salt has a natively available a REST API, which can be used in combination with *salt-sproxy* to invoke commands over HTTP, without running Proxy Minions.

Enable the API:

/etc/salt/master

```
rest_cherrypy:
  port: 8080
  ssl_crt: /path/to/crt
  ssl_key: /path/to/key
```

https://salt-sproxy.readthedocs.io/en/latest/salt_api.html

Using salt-sproxy via the Salt REST API

After these three easy steps, you can start running commands:

```
$ curl -sS localhost:8080/run -H 'Accept: application/x-yaml' \
  -d eauth='pam' \
  -d username='mircea' \
  -d password='pass' \
  -d client='runner' \
  -d fun='proxy.execute' \
  -d tgt=router1 \
  -d function='test.ping' \
  -d sync=True
return:
  router1: true
```

https://salt-sproxy.readthedocs.io/en/latest/quick_start.html

Getting started with *salt-sproxy*

Everything available in Salt is possible through *salt-sproxy*, just that:

- *salt-sproxy* is much easier to install (compared to the typical Salt setup).
- You don't have any Proxy Minions to manage.
- *salt-sproxy* is specially tailored for network automation use (but not limited to).

See another example at:
https://asciinema.org/a/247726?autoplay=1

-

https://github.com/mirceaulinic/salt-sproxy

Thank You!

Questions?

mu@do.co

DigitalOcean