Automating Networks Using Salt, Without Running Proxy Minions

Mircea Ulinic

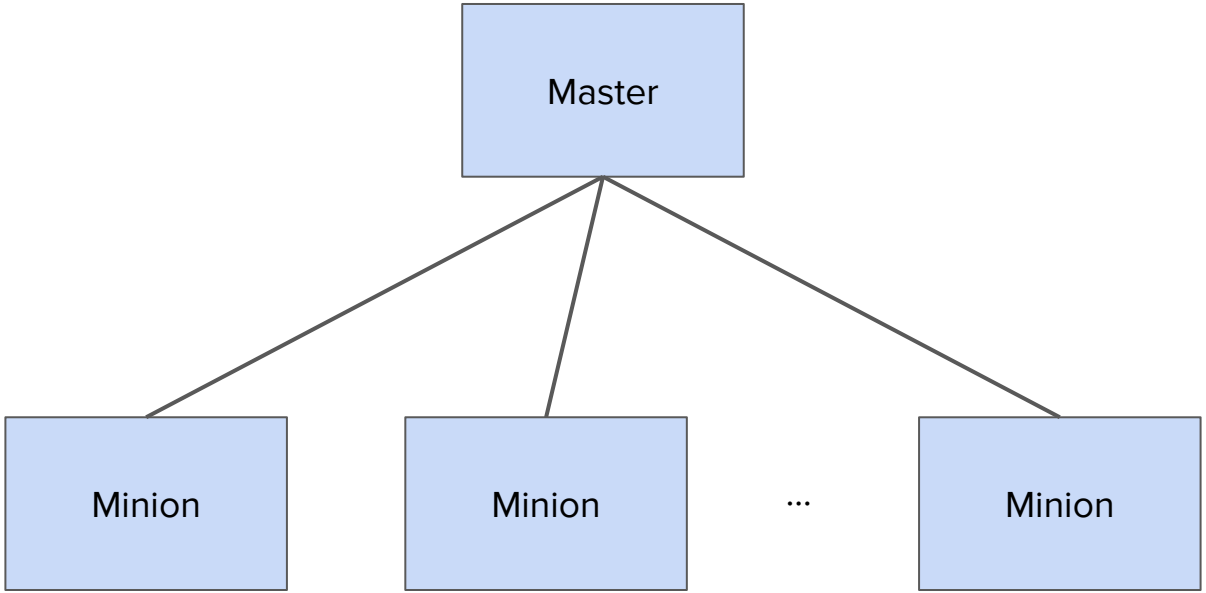
Brief Introduction to Salt

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.*"

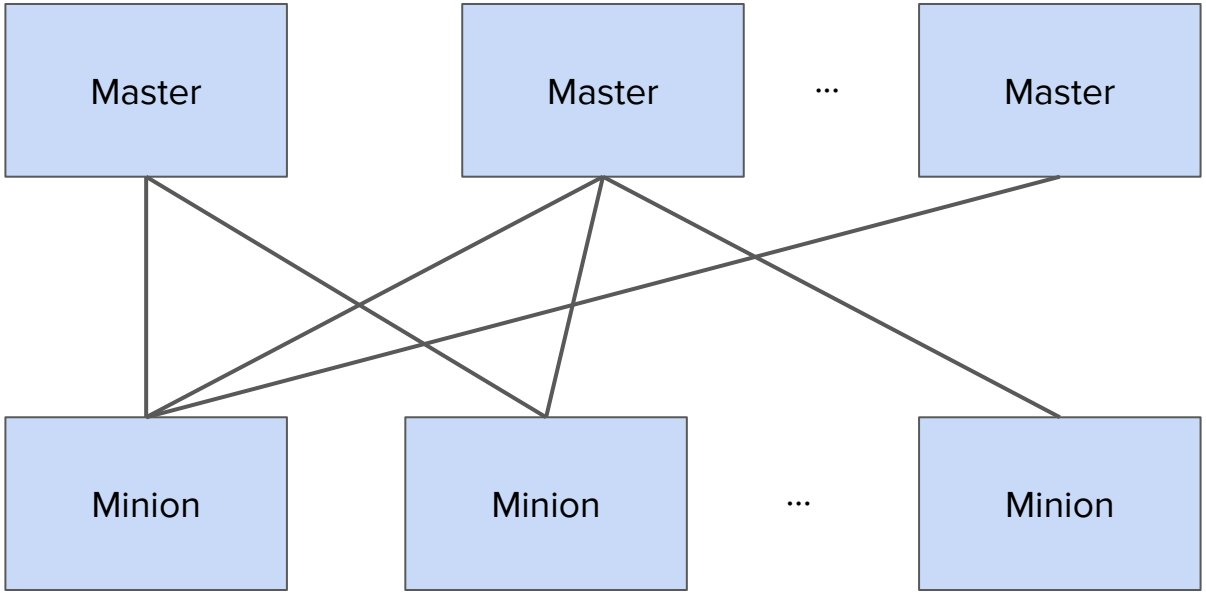
https://docs.saltstack.com/en/getstarted/speed.html

Brief Introduction to Salt: Typical Architecture



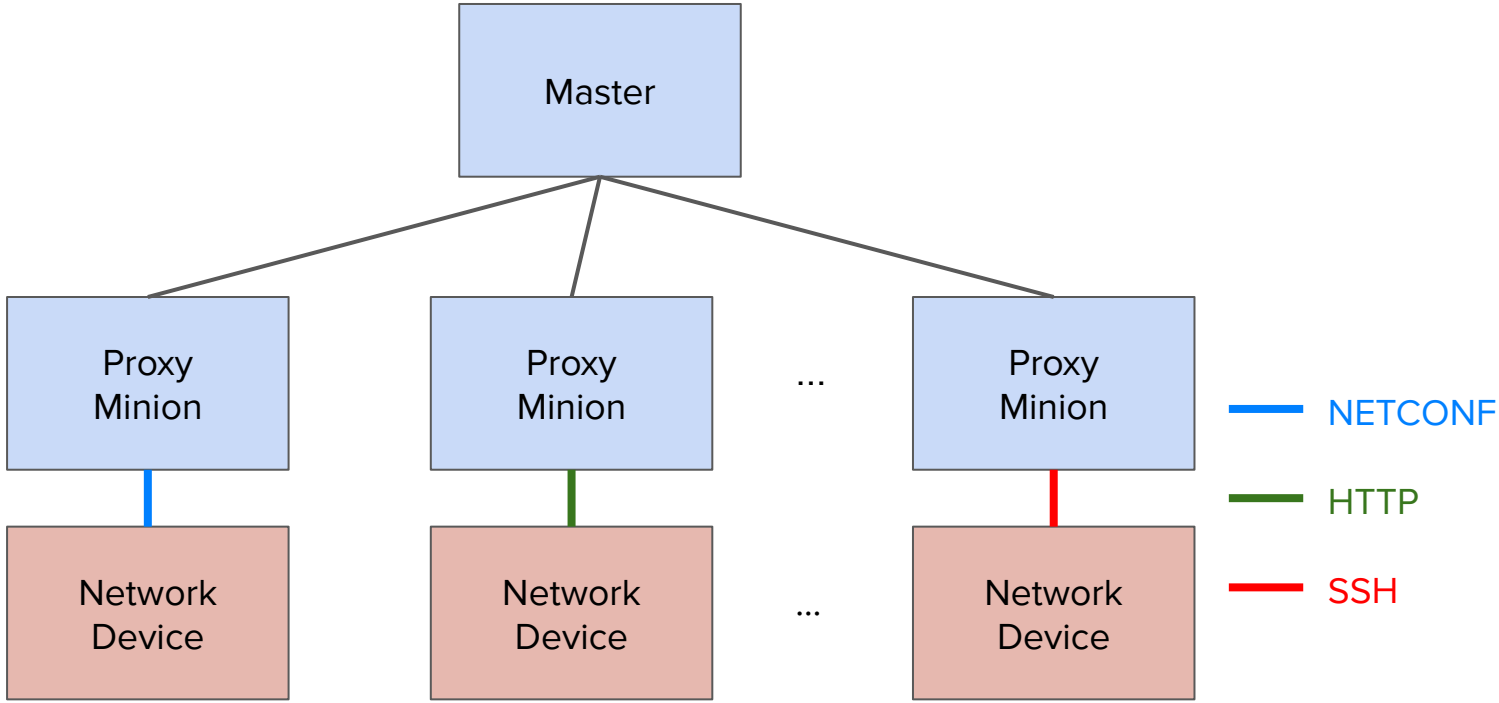
https://docs.saltstack.com/en/latest/topics/topology/index.html

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.

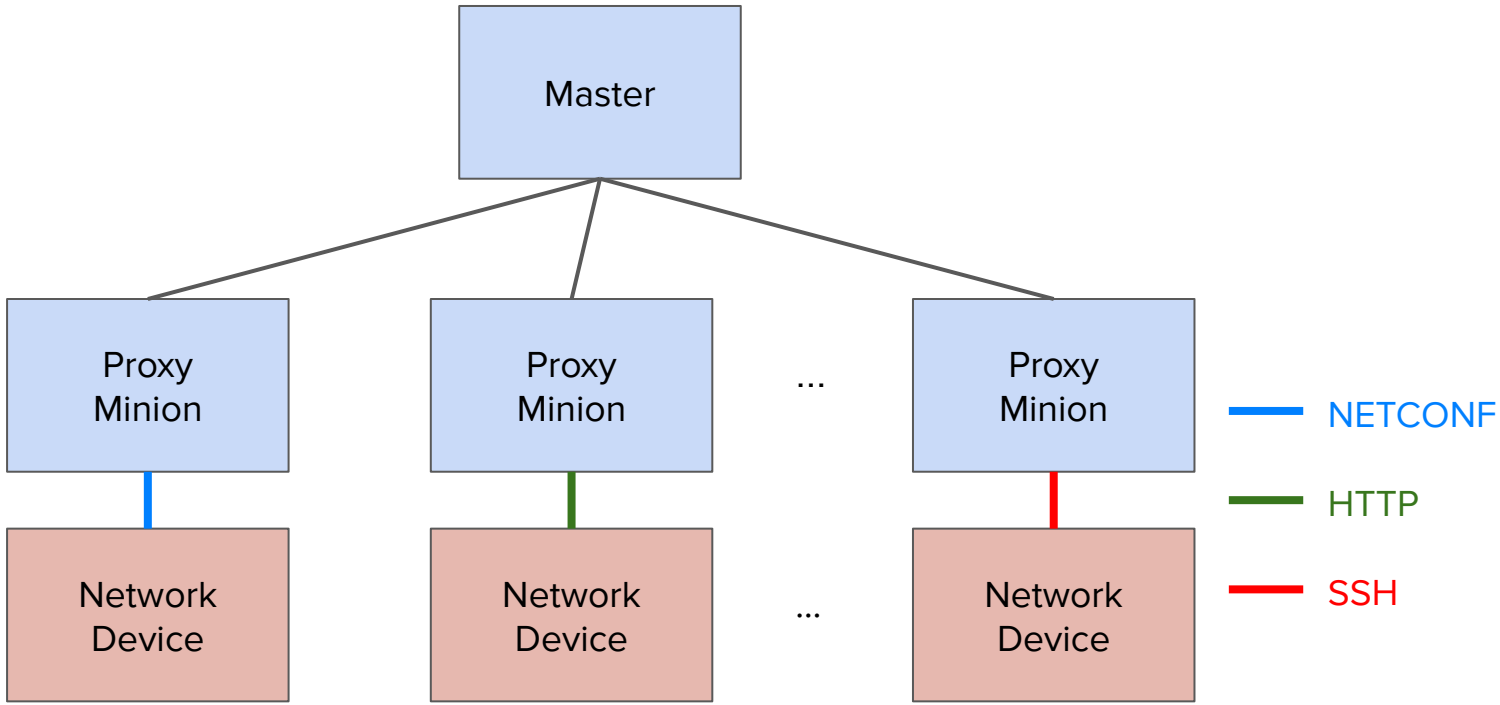
Introducing *salt-sproxy* (Salt Super Proxy)

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

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

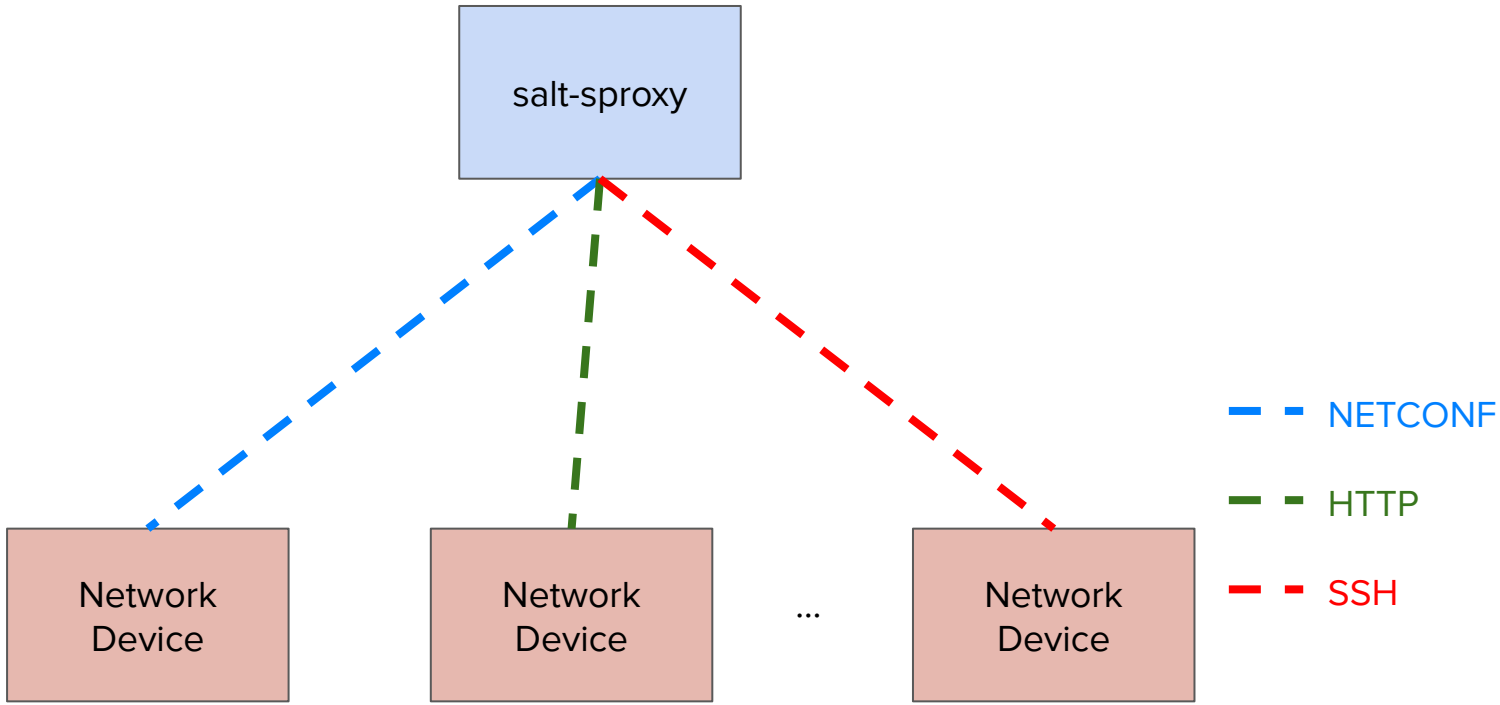
Using *salt-sproxy*, you can continue to benefit from the scalability, flexibility and extensibility of Salt, while you don't have to manage thousands of (Proxy) Minion services. However, you are able to use both *salt-sproxy* and your (Proxy) Minions at the same time.

Remember slide #7?



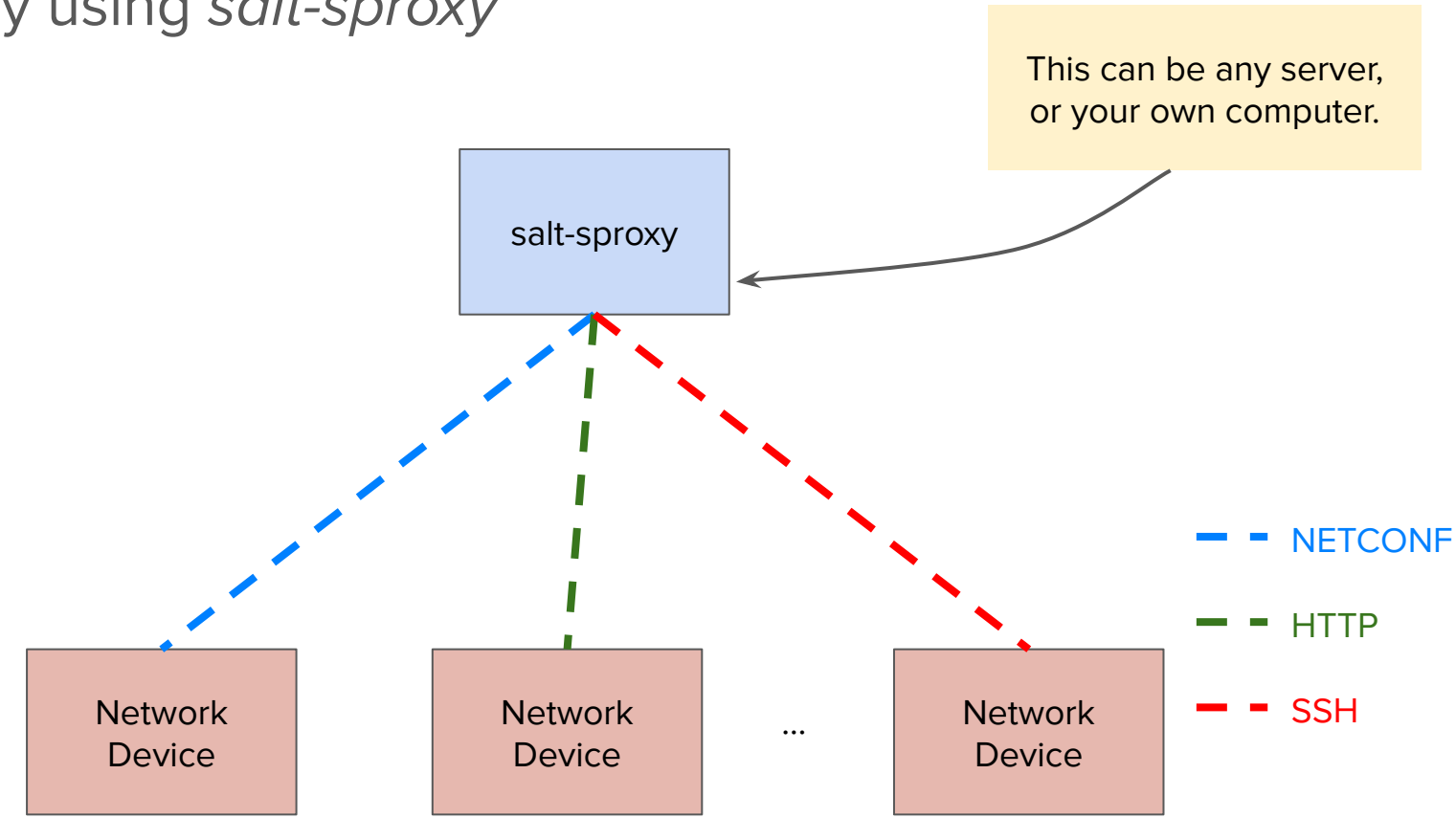
https://docs.saltstack.com/en/latest/topics/topology/index.html

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:

/etc/salt/roster

```
router1:
  driver: junos
router2:
  driver: iosxr
spine1:
  driver: junos
leaf1:
  driver: eos
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 (2)

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

/etc/salt/master

```
roster: file

proxy:
  proxytype: napalm
  username: <username>
  password: <password>
```

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

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

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

/etc/salt/master

```
roster: file

proxy:
  proxytype: napalm
  username: <username>
  password: <password>
```

There are different flavours of Roster sources, including NetBox, Pillar (i.e., retrieve data from HTTP APIs, MySQL / PostgreSQL databases, etc.).
File is the easiest to understand and demo.

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

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

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

/etc/salt/master

```
roster: file

proxy:
  proxytype: napalm
  username: <username>
  password: <password>
```

You can choose between a variety of Proxy Modules natively available in Salt.

If none available for your use case, developing a new Proxy Module in your own environment is easy and straighforward.

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

$ 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 SLS data from a specific file (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

Why *salt-sproxy*

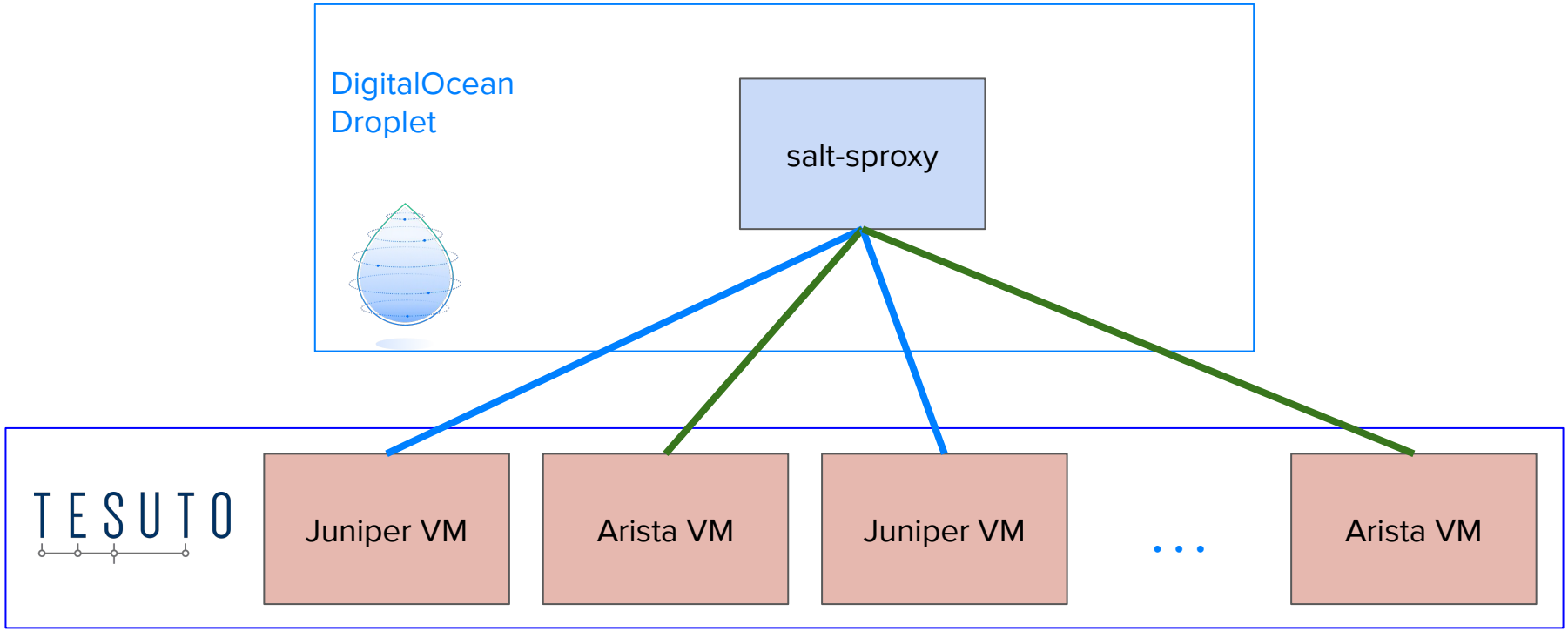
- *salt-sproxy* is much easier to install (compared to the typical Salt setup).
- Say goodbye to the burden of managing hundreds of system services for the Proxy Minion processes.
- You can run it locally, on your own computer.
- Integrates easily with your existing Salt environment (if you have), by installing the package on your Salt Master.
- Can continue to leverage the event-driven automation and orchestration methodologies.
- REST API, see also Using the Salt REST API documentation.
- Python programming made a breeze - might go well with the ISalt package.

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

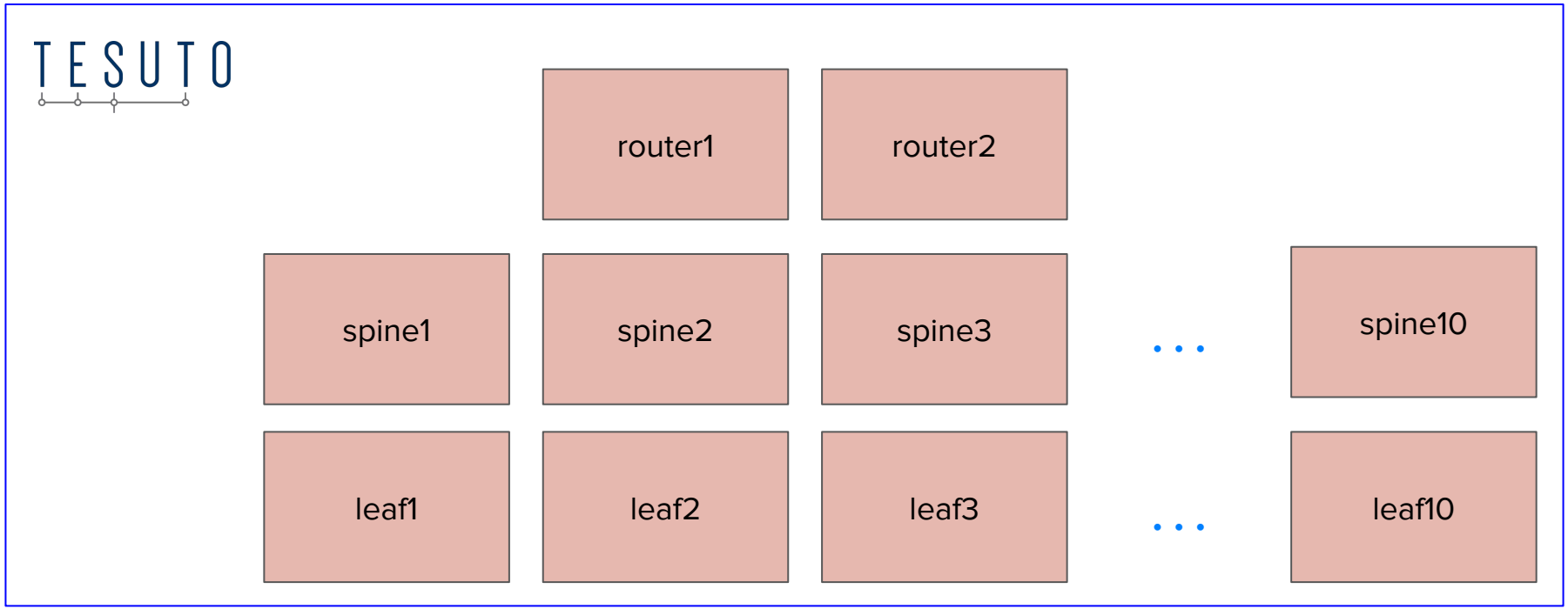
Questions?

Live Demo

Live Demo setup



Tesuto topology



Tesuto topology

- *2 routers (Junos VM)*
- *10 leafs (Arista VM)*
- *10 spines (Junos VM)*

Salt-sproxy configuration

Configuration files, installation script, and demo CLI available at
https://github.com/mirceaulinic/iNOG14v-demo

Salt-sproxy configuration essentials

Master config

```
roster: file

proxy:
  proxytype: napalm
  username: tesutonet
  password: <password>
```

Roster config

```
router1:
  driver: junos
  host: juniper.iNOG14v...
router2:
  driver: junos
  driver: junos0.iNOG14v..

{%- for i in range(1, 11) %}
leaf{{ i }}:
  driver: eos
  host:  eos{{ i }}.iNOG14v..
spine{{ i }}:
  driver: junos
  host: junos{{ i }}.iNOG14v
{%- endfor %}
```

Running salt-sproxy

```
$ salt-sproxy * --preview-target
    - leaf2
    - leaf10
    - leaf5
    - spine9
    - leaf4
    - spine1
    - spine6
    - spine4
    - spine5
    - leaf1
    - spine3
    - leaf9
    - spine8
    - leaf6
      ... [snip] ...
```

Running salt-sproxy

More examples in the cli.sh script

Thanks!

mu@do.co

DigitalOcean