

Experiência com Saltstack

Utilizando Salt + NAPALM em equipamentos de rede

Victor Cerqueira Leal

Intro

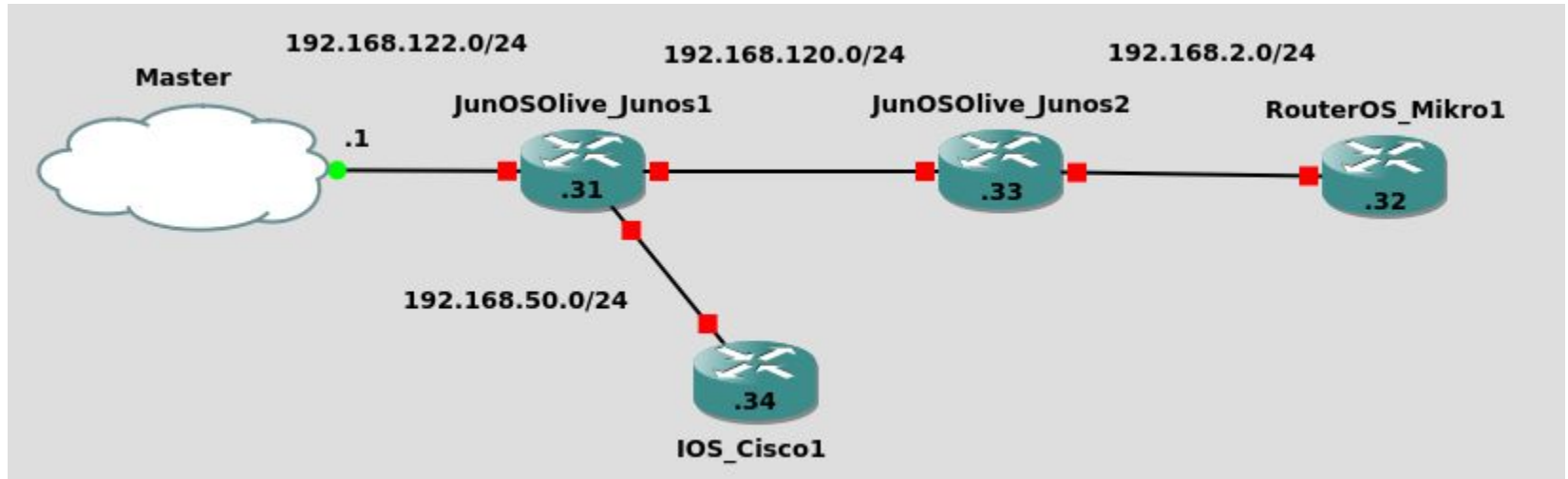
Primeiro contato com Saltstack e bibliotecas NAPALM

Tentativa de implementar o básico na comunicação com equipamentos de redes em um ambiente virtual

Testes voltados para necessidades de backup e gerenciamento da configuração

Ambiente

Máquinas virtuais (VirtualBox) e imagens de roteadores no GNS3



Instalação

- Salt

Ubuntu:

- `apt install salt-master`
- `apt install salt-minion`

(Alt.) Salt bootstrap script:

- (<https://github.com/saltstack/salt-bootstrap>)

- NAPALM

- `apt install libffi-dev
libssl-dev python-dev
python-cffi libxslt1-dev
python-pip`
- `pip install --upgrade cffi`

Bibliotecas:

- `pip install napalm-junos
napalm-iosxr napalm-ios
ou`
- `pip install napalm`

Configuração (master e proxy)

/etc/salt/master

```
file_roots:
  base:
    - /etc/salt
    - /etc/salt/states

runner_dirs:
  - /etc/salt/runners

nodegroups:
  gns3:
    - cisco1
    - junos1
    - junos2

interface: 192.168.122.1

schedule:
  backup_job:
    function: backup.all
    days: 1
```

/etc/salt/proxy

```
master: 192.168.122.1
multiprocessing: False
mine_enable: True

mine_functions:
  net.config: []
mine_interval: 1440

pki_dir: /etc/salt/pki/proxy
cachedir: /var/cache/salt/proxy

schedule:
  mine_backup_config:
    function: mine.update
    hours: 1
    splay: 10
    kwargs:
      mine_functions:
        net.config: []
```

Configuração arquivos SLS - YAML (Pillar e top file)

Em /etc/salt/master (padrão)

```
pillar_roots:
```

```
  base:
```

```
    - /srv/pillar
```

- /srv/pillar/top.sls

Identificação
do minion
proxy

```
base:
  cisco1:
    - cisco1
  mikrol:
    - mikrol
  junos1:
    - junos1
  junos2:
    - junos2
```

Arquivo .sls

Arquivos sls incluídos no top.sls

- /srv/pillar/junos1.sls

```
proxy:
  proxytype: napalm
  driver: junos
  host: 192.168.122.31
  username: admin
  passwd: admin123

default_route_nh: 200.128.12.42

snmp_test:
  snmp_name: '"UNI - Campus"'
  community: ropopcom

include:
  - ntp_config
```

Problemas configuração

- Mikrotik
 - Erro conexão sem senha de acesso
 - Não suporta todas funções no NAPALM
- Juniper
 - Habilitar SSH para conexão com salt
 - Alteração código biblioteca (junos.py) para testar retorno de RE0 (desprezando uptime)
- Cisco
 - Habilitar SSH e senha de enable
 - Privilégio username
 - Secret (senha de enable) como optional_args no pillar
 - Incluir disk0 no gns3
 - Configuração de conexão e timeout line vty

Chaves

Primeira conexão

- Iniciar salt-master:
 - `systemctl start salt-master`
- Iniciar salt-proxy:
 - `salt-proxy --proxyid=junos1 -l debug`
 - `salt-proxy --proxyid=junos1 -d`
 - `systemctl start salt-proxy@junos1`
- Aceitar a chave associada ao minion (proxy)
 - `salt-key -a -y junos1`
- Aceitar todas
 - `salt-key -A -y`

- Listar chaves
 - `salt-key -L`

```
~ $ sudo salt-key -L
Accepted Keys:
ciscot
junos1
junos2
mikrot
Denied Keys:
Unaccepted Keys:
vcl-HP
Rejected Keys:
```


Módulos e execução

- Módulos NAPALM
 - net module
 - ntp module
 - bgp module
 - snmp module
 - route module
 - users module
 - probes module
 - network ACL module

salt '*' test.rand_sleep 120

target module.function arguments

- Estados NAPALM
 - ntp state
 - snmp state
 - users state
 - netconfig state
 - network ACL state

Informações e alterações

- Grains
- Execution modules
- States

Testar conectividade:

- `salt * net.connected`

Dados dos grains:

- `salt * grains.items`

Configuração:

- `salt * net.config`

Alterar configurações:

- Diretamente
 - `salt 'junos*' net.load_config txt='system ntp server 172.17.17.1;'`
- Arquivo de configuração
 - `salt -L 'junos1, junos2' net.load_config /path/to/file`
- Template
 - `salt -G 'os:junos' net.load_template salt://path_to_template`
- States
 - `salt -N gns3 state.apply router.ntp`

Flags de configuração

- `test` (padrão = `False`)
 - Não aplica as mudanças, apenas mostra as diferenças e descarta alterações
- `commit` (padrão = `True`)
 - Não aplica as mudanças imediatamente, mas salt mostra como se tivesse alterado, só entra em vigor de fato após `commit` (`net.commit`, `net.discard_config` ou `net.config_control`)
- `replace` (padrão = `False`)
 - Descarta toda a configuração anterior substituindo pela nova enviada

Usando Jinja templates

```
{% set router_vendor = grains.vendor -%}{# info dos grains #}  
{% set hostname = pillar.proxy.host -%}{# info estática do pillar #}  
{% if router_vendor|lower == 'juniper' %}  
system {  
    host-name {{hostname}}.novo;  
}  
{% elif router_vendor|lower in ['cisco', 'arista'] %}  
hostname {{hostname}}.novo  
{% endif %}
```

Exemplos uso Jinja

```
{% set router_vendor = grains.vendor -%}{# info dos grains #}
```

```
{%- if grains.vendor|lower == 'cisco' %}
```

```
...
```

```
{%- elif grains.os|lower == 'junos' %}
```

```
{% set nome = pillar.snmp_test.snmp_name %}
```

```
{% set comm_name = pillar.snmp_test.community %}
```

```
{% set servers = pillar.get('ntp.servers', []) %}
```

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

```
{%- set default_route = route_output['out'] -%}
```

Dados no Pillar

Em `/srv/pillar/junos1.sls`

...

include:

- `ntp_config`



Arquivos sls incluídos no pillar

- `/srv/pillar/ntp_config.sls`

`ntp.servers:`

- `200.128.0.21`

`ntp.peers:`

- `200.128.0.211`

- `200.128.0.212`

Usando states

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

```
{% set ntp_peers = pillar.get('ntp.peers', []) %}  
{% set ntp_servers = pillar.get('ntp.servers', []) %}
```

update_ntp_config:	State	update_ntp_config:
netntp.managed:		netconfig.managed:
- peers: {{ ntp_peers json() }}		- template_name: salt://ntp.jinja
- servers: {{ ntp_servers json() }}		- peers: {{ ntp_peers }}
		- servers: {{ ntp_servers }}

Aplicar estado:

```
salt \* state.apply router.ntp
```

** pode aplicar vários estados criando top.sls e init.sls dentro dos diretórios de states

Schedule em states

/etc/salt/master

```
schedule:
  backup_job:
    function: backup.all
    days: 1
```

/etc/salt/proxy

```
schedule:
  mine_backup_config:
    function: mine.update
    hours: 1
    splay: 10
    kwargs:
      mine_functions:
        net.config: []
```

/srv/pillar/schedule/init.sls

```
schedule:
  update_test_config:
    function: state.sls
    args:
      - router.ntp
    seconds: 30
    splay: 1
```

Incluir em /srv/pillar/top.sls referente ao minion que deseja executar state programado.

Ex.:

```
base:
  junos1:
    - junos1
    - schedule
```

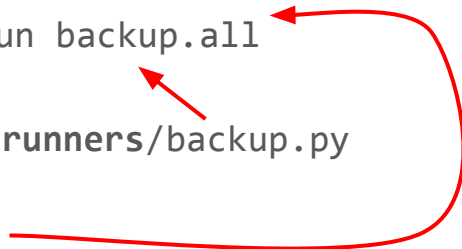

Salt Runners

Aplicações executadas no master através do comando salt-run

Ex.: salt-run backup.all

/etc/salt/runners/backup.py

```
...
def all():
    config = client.cmd('*', 'net.config')
    for key in config:
        if config[key] is False:
            ret.update({key: "Minion did not return. [No response]"})
        elif config[key]['result'] is False:
            ret.update({key: config[key]['comment']})
        else:
            with open(key+"-"+hoje+".cfg",'w') as file:
                file.write(config[key]['out']['running'])
                ret.update({key: config[key]['result']})
    return ret
```



Salt Beacons + Reactors

- Beacons geram eventos no Salt para algum processo não relacionado ao Salt, ex. alterar um arquivo no sistema de arquivos.
- Reactors executam alguma ação baseado em algum evento no Salt.

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

```
...
```

```
beacons:
```

```
  inotify:
```

```
    /srv/pillar/ntp_config.sls:
```

```
      mask:
```

```
        - modify
```

```
      disable_during_state_run: True
```

Exemplo de Beacon usando inotify (apt install inotify-tools e pip install inotify)

Salt Beacons + Reactors

```
/etc/salt/master
```

```
...
```

```
reactor:
```

- 'salt/beacon/*/inotify//srv/pillar/ntp_config.sls':
 - salt://ntp_changed.sls

```
/etc/salt/reactors/ntp_changed.sls
```

```
run_ntp_state:
```

```
  local.state.sls:
```

- tgt: {{ data['id'] }}
- arg:
 - router.ntp

Comandos úteis (grains, pillars e config)

- `saltutil.refresh_pillar`
 - Atualizar informações do pillar
- `pillar.items`
 - Listar conteúdo do pillar
- `pillar.get`
 - Obter algum elemento do pillar
- `grains.ls`
 - Listar grains
- `grains.items`
 - Dados dos grains
- `net.config_changed`
 - Verificar se configuração foi alterada
- `net.compare_config`
 - Comparar configurações antes de commit
- `net.config_control`
 - Realiza commit ou rollback se algum erro
- `net.rollback`
 - Desfaz última alteração

Referências

Saltstack Documentation

<https://docs.saltstack.com/en/latest/contents.html>

Napalm

<https://github.com/napalm-automation/napalm-salt>

<https://mirceaulinic.net/>

Apresentação NAPALM+Salt na RIPE-72 (Mircea Ulinic)

<https://ripe72.ripe.net/presentations/58-RIPE72-Network-Automation-with-Salt-and-NAPALM-Mircea-Ulinic-CloudFlare.pdf>

Apresentação NAPALM+Salt na RIPE-74 (Mircea Ulinic)

<https://ripe74.ripe.net/presentations/18-RIPE-74-Network-automation-at-scale-up-and-running-in-60-minutes.pdf>

Obrigado!

Victor Leal
FEEC - Unicamp

vcleal@gmail.com

[GitHub - vcleal](#)

