SaltStack FAQs

---

1. **Where is the documentation related to junos proxy modules?**

Ans. Here are the links to junos proxy documentation:

   a. https://docs.saltstack.com/en/latest/ref/proxy/all/salt.proxy.junos.html
   b. https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.junos.html
   c. https://docs.saltstack.com/en/latest/ref/states/all/salt.states.junos.html
   d. https://docs.saltstack.com/en/develop/ref/engines/all/salt.engines.junos_syslog.html

---

2. **Is there is SaltStack docker container?**

Ans: Below is a project to help users getting started with SaltStack on junos:
https://github.com/Juniper/docker-saltstack-junos

---

3. **The junos proxy won't start, giving the following error:**

```
# salt '*' junos.ping

7465f044ef32:
'junos' __virtual__ returned False: The junos module could
not be loaded: junos-eznc or jxmlease or proxy could not be
loaded.
proxy_minion:
    Minion did not return. [Not connected]
ERROR: Minions returned with non-zero exit code
```

Ans: The junos proxy as two dependencies which need to be installed

   a. PyEZ
   b. jxmlease

Check that both of them are installed on the system running SaltStack.

---

4. **Some junos modules work, some don't.**

`junos.ping` **and** `junos.facts` **work while modules like** **`junos.rpc`** **don't work.**

Ans: This usually occurs when multiprocessing option is not set in the proxy configuration file. In the proxy config file add the following line:

```
#/etc/salt/proxy
multiprocessing: False
```

From Nitrogen release (2017.x onwards), setting this option is not required as the junos proxy code will take care of it.

---

5. **Junos is proxy not working and giving the following error:**

```
[ERROR] No proxy key found in pillar or opts for id vsrx3. Check
your pillar/opts configuration and contents.  Salt-proxy aborted.
The Salt ProxyMinion is shutdown.
No proxy key found in pillar or opts for id vsrx3. Check your
pillar/opts configuration and contents.  Salt-proxy aborted.
```

Ans: This error usually occurs when the junos proxy doesn't get the data the in pillar or gets the wrong type of data. Check that top.sls of pillars if properly written.

```
#/srv/pillar/top.sls
base:
  vmx:
  -  details
```

Eventhough my file name is details.sls, we mention it without '.sls extension' in the top file.

The pillar file is defined as follows:

```
#/srv/pillar/details.sls
proxy:
   proxytype: junos
   host: x.x.x.x
   username: jnpr
   passwd: pass123
```

Notice 'passwd' is used instead of 'password'.
From Nitrogen release onwards, all options present in PyEZ while connecting to a device are available in junos proxy as well.

This type of error can also be caused by wrong configuration in /etc/salt/master:

```
pillar_roots:
  base:
    - /opt/salt-jnpr/srv/pillar
```

This is often seen when you install Salt in an alternate installation path (e.g. /opt/salt)

---

6. **I want to use port other than 830 to make the connection to junos device. How to do that?**

Ans: Release prior to Nitrogen don't have that option. Nitrogen release (onwards the user can change the port to their choice.

---

7. **I get the following error while gathering facts:**

```
<<<< omitted for brevity >>>>>
File "src/lxml/lxml.etree.pyx", line 2998, in lxml.etree.Element
(src/lxml/lxml.etree.c:76821)
  File "src/lxml/apihelpers.pxi", line 105, in
lxml.etree._makeElement (src/lxml/lxml.etree.c:17387)
  File "src/lxml/apihelpers.pxi", line 1630, in
lxml.etree._tagValidOrRaise (src/lxml/lxml.etree.c:33940)
ValueError: Invalid tag name u'--deepcopy--'
[DEBUG   ] Initializing new Schedule
[DEBUG   ] LazyLoaded timezone.get_offset
<<<< omitted for brevity >>>>>
```

Ans: Check the version of PyEZ on the system. Changes were made in the facts gathering in PyEZ 2.1. If you are using PyEZ with version >= 2.1, you need the SaltStack version 2017.x or above (Nitrogen release or above). In order to check the salt version, use the following command:

```
salt --version
```

---

8. **What is the best way today to save the configuration from a junos device on a file locally with SaltStack today?**

Ans: One can use the junos.rpc function to send a get_config rpc (can also apply filters like in PyEZ). There is an option in the rpc function to store the output in a file on the minion. Remember, the output is stored on the minion. If you want to get the file on the master, use cp.push function of SaltStack

---

9. **How to update junos grains?**

Ans: The junos proxy, stores the facts gathered as salt grains. For releases prior to Nitogen use saltutil.synch_grains function. From Nitrogen release onwards one can use junos.facts_refresh itself to get latest facts as well update the junos grains.

---

10. Can I have multiple devices connected a single proxy minion?

With the following setup:

```
#/srv/salt/pillar
base:
  'junos_vmx':
         - pe1
         - pe2
```

And run both the junos devices with a single command:

```
salt-proxy --proxyid=junos_vmx
```

Ans: Unfortunately, currently this is not possible. Although if you have many number of proxies to run, salt_proxy.configure_proxy comes to rescue.
We can create a state file which starts multiple proxies:

```
#/srv/salt/start_proxy.sls:
  start junos proxy mx:
    salt_proxy.configure_proxy:
      - proxyname: junos_mx
      - start: True

  start junos proxy vMX:
    salt_proxy.configure_proxy:
      - proxyname: junos_vmx
      - start: True

  start junos proxy srx:
    salt_proxy.configure_proxy:
      - proxyname: junos_srx
      - start: True
```

We start the salt minion, on system where we want to start the proxies:

```
sudo salt-minion
```

Then, we run the above state file:

```
sudo salt 'system_minion' state.apply start_proxy
```

---

11. How do we fire events on the salt event bus?

Ans: One can send events from minion to the master event bus via the event.fire_master command:

```
salt '*' event.fire_master '{"data":"my event data"}' 'tag'
```

To fire an event on the local minion event bus, one can use event.fire:

```
        salt '*' event.fire '{"data":"my event data"}' 'tag'
```

---

12. How to write state file?

Ans: State files or sls files are nothing but data structures under the hood. State files are written in YAML by default. Each state file contains one or more tasks. Let us assume each task is one unit. So, in each unit the first line is the name or id associated with that task. The next line is the name of the state module, followed by the arguments which the state function takes. Below is a sample state file. This state file specifies two tasks. First one installs PyEZ (junos-eznc) dependencies. The second task installs PyEZ. The require keyword at the end of second task signifies that the second task is to be run only if the first task was executed successfully.

```
#/srv/salt/install_pyez.sls

install_pyez_deps:
  pkg.installed:
    - pkgs:
        - python-pip
        - python-lxml
        - python-dev
        - libssl-dev
        - libxslt-dev
        - python-paramiko

install_pyez:
  pip.installed:
    - name: junos-eznc
    - require:
      - install_pyez_deps
```

---

13. While running the state file:

```
salt minioncontroller1 state.sls proxy_states.sls
```

I get the following error:

```
minioncontroller1:
    Data failed to compile:
----------
No matching sls found for 'proxy_states.sls' in env 'base'
```

Ans: When specifying the state file in state.sls we omit the '.sls' extension. So, the appropriate command would be:

```
salt minioncontroller1 state.sls proxy_states
```

proxy_states.sls (present at /srv/salt/) being the state file I want to execute.

---

14. What are salt engines? How to start the junos syslog engine?
Ans: Salt Engines are long-running, external system processes that leverage Salt. They are executed in a separate process that is monitored by Salt. If a Salt engine stops, it is restarted automatically. Engines can run on both master and minion. The junos syslog engine listens to syslog message from Junos devices, extracts event information and generates message on SaltStack bus. To start the junos syslog engine, we need to specify engine information in master/minion config file depending on where we want to run the engine. Below is a sample config:

```
#/etc/salt/master
engines:
  - junos_syslog:
      port: xxx
```

Once the engine configuration is added, start the master and minion normally (with salt-master/salt-minion command). The junos syslog engine should have started along with the salt master/minion.
Users can choose the topic/tag associated with the event. The event should start with jnpr/syslog, followed by one or more fields from:

1. hostname
2. hostip
3. daemon
4. event
5. severity
6. priority
7. timestamp
8. message
9. pid
10. raw (the raw event data forwarded from the device)

Example config:

```
#/etc/salt/master
engines:
  - junos_syslog:
      port: xxx
      topic: jnpr/syslog/hostip/daemon/event
```

One can also filter the type of data that will be sent on the event bus. The following configuration will send only events coming from sshd or mgd and with severity 6.

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

```
engines:
  - junos_syslog:
      port: xxx
      topic: jnpr/syslog/hostip/daemon/event
      daemon:
        - mgd
        - sshd
      severity: 6
```

---

15. The junos syslog engine is not working.

Ans: There can be multiple reasons for this-

- If you see a message like this in the debug log

  ```
  <<<< omitted for brevity >>>>>
  [DEBUG] Could not LazyLoad junos_syslog.start:
  'junos_syslog.start' is not available.
  [INFO] Starting Engine
  salt.engines.Engine(salt.loaded.int.engines.reactor)
  [DEBUG] Started
  'salt.engines.Engine(salt.loaded.int.engines.reactor)' with
  pid 22324
  <<<< omitted for brevity >>>>>
  ```

  This means that junos syslog engine is not present in the salt libraries. The junos syslog was introduced in the Nitrogen releases. If you are running a release prior to Nitrogen, you can take the junos syslog code from github. Place this file in the folder of your choice. Then mention the folder path in the master/minion config file depending on where you are running the master.

  ```
  #/etc/salt/master
  engines_dirs:
    - /path/to/the/directory/that/has/junos_syslog.py
  ```

- Another reason that junos syslog engine has not restarted is that the needed dependencies are not installed. Junos syslog engine requires twisted and pyparsing python library. Ensure both of them are installed using:
  - `pip install twisted`
  - `pip install pyparsing`

- The junos syslog engine started but the junos devices are not sending syslog to the engine. You need to configure the junos device to forward syslog data to the engine:

  ```
  set system syslog host <ip-address > port xxxx any any
  ```

  The ip address is that of the device running the syslog engine and port is the port where the engine is listening for events.

---

16. How to listen for events sent on the master event bus?
Ans: There two ways. One is using following command on master CLI:

```
salt-run state.event pretty=True
```

Another is to write a script which prints event details coming on master event bus. Here is a sample script.

```
#!/usr/bin/python

import salt.config
import salt.utils.event
import pprint
import time

opts = salt.config.client_config('/etc/salt/master')

event = salt.utils.event.get_event(
        'master',
        sock_dir=opts['sock_dir'],
        transport=opts['transport'],
        opts=opts)

print ""
print ""
print
"=========================================================
====="
print ""
print "Listening for events .... (infinite loop) ...."
print ""
print
"=========================================================
====="
print ""
print ""

while True:
    evdata = event.get_event(full=True)
    if evdata is None:
        continue

    tag, data = evdata['tag'], evdata['data']

    pprint.pprint(tag)
    pprint.pprint(evdata)

    # we don't need to kill the CPU
    time.sleep(1)
```

17. What are reactor files? How to configure a trigger for an event using Salt Reactor?

Ans: Reactors files are executed when an event with a specific tag comes on the master event bus. This mapping between event tag and the reactor file to execute is present in the master config file. Here a sample config.

```
#/etc/salt/master
reactor:
 - '/my/event/tag'
    - 'path/to/reactor.sls'
```

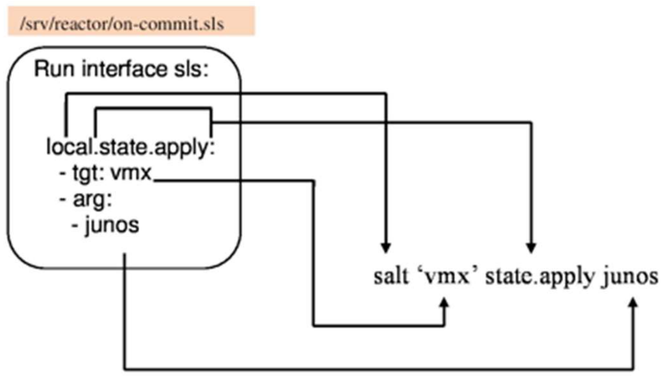
Suppose we want to write a reactor which runs a 'junos.sls' state file on the junos proxy minion when a UI_COMMIT_COMPLETED event occurs.
The corresponding master config would look something like this:

```
#/etc/salt/master:
reactor:
- 'jnpr/syslog/*/mgd/UI_COMMIT_COMPLETED'
 - '/srv/reactor/react_to_commit.sls'
```

Since the reactor file would be executed at the master, we need to inform the master to run junos.sls on your junos proxy. The reactor file would be:

```
#/srv/reactor/on_commit.sls
   Run interface sls:
     local.state.apply:
       - tgt: vmx
       - arg:
         - junos
```

on_commit.sls will be executed on the salt master. The format of on_commit.sls is similar to the CLI command which we would otherwise execute on salt master.



The correspondi ing junos.rpc state module.

```
#/srv/salt/junos.sls
get-interface-information:
 junos:
   - rpc
```

```
          - dest: /tmp/rpc.log
          - interface_name: lo0
```

So, when an UI_COMMIT_COMPLETED event comes on the salt master bus. The reactor system will execute on_commit.sls locally. on_commit.sls informs the salt master to run the state file, junos.sls on the vmx proxy minion.

---

18. While running a reactor file the master gives the following log:

```
[DEBUG] Could not find file 'salt://junos/sls.sls' in saltenv 'base'
[DEBUG] Could not find file 'salt://junos/sls/init.sls' in saltenv
'base'
```

Here is the corresponding reactor file:

```
#/salt/reactor/react.sls
Run interface sls:
  local.state.apply:
    - tgt: vmx
    - arg:
      - junos.sls
```

Ans: In the last line we see that in the arg section we mentioned the sate file name as 'junos.sls'. But we need to mention the name of the sls file without the '.sls' extension. This makes sense because even on the salt master CLI we mention the name of state file without the '.sls' extension. So, the correct reactor would be:

```
#/salt/reactor/react.sls
Run interface sls:
  local.state.apply:
    - tgt: vmx
    - arg:
      - junos
```

---

19. I want to fire events on salt master from the minion using a python script.

```
#send_event.py:
import salt.client
caller = salt.client.Caller()
caller.sminion.functions['event.send'](
    'myco/myevent/success',
    {
       'success': True,
       'message': "It works!",
    }
   )
```

But no event is fired is on the master.

Ans: For such a script to work, the minion should be started (with salt-minion command). Make sure both salt master and minion are running properly. Also, one must run the script as the same user as they run the salt-minion. (root by default)

---

20. How do I write modules of my own?
Ans: There is a tutorial available at:
https://git.juniper.net/nembery/salt_custom_junos_module

---

21. How do I contribute to the junos module?
Ans: We have SaltStack in our Juniper github repo. All the contributions are to be to this repository. This repo is periodically synced with the main SaltStack code.