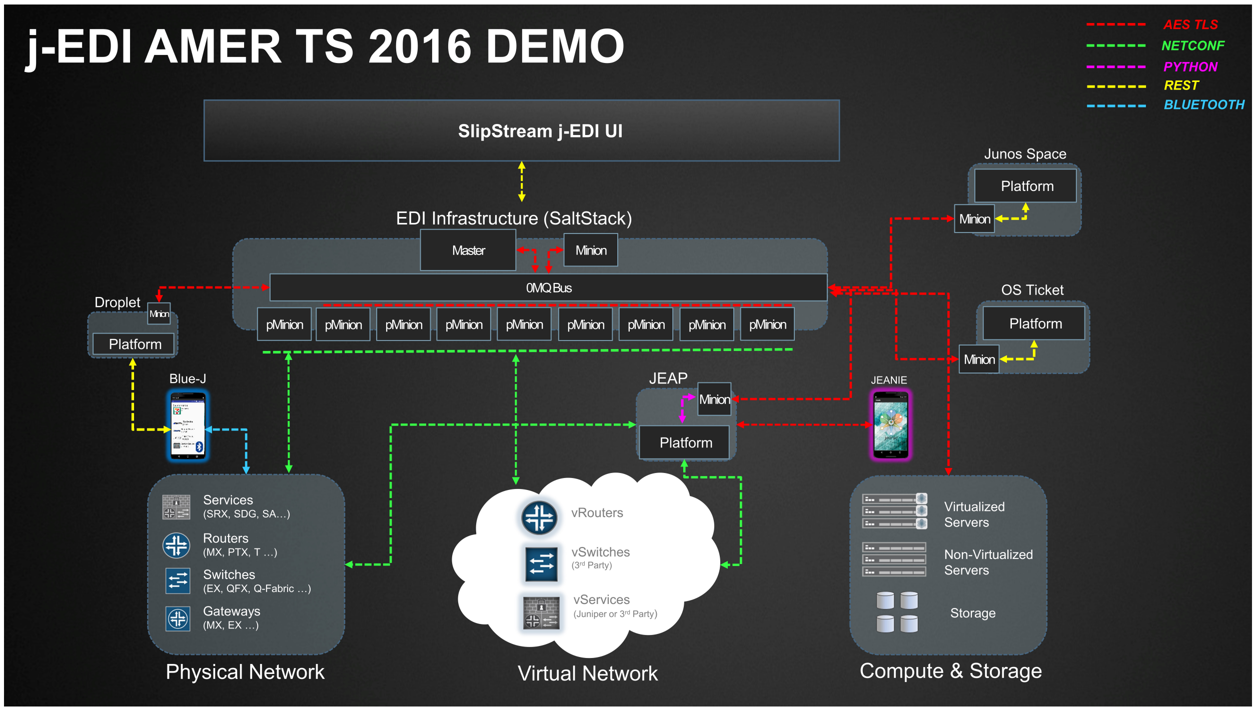
Juniper Event Driven Infrastructure

Introduction:

Juniper currently doesn't have a event driven infrastructure. Event driven infrastructure has the capability of bringing down the operational expenditure of our company. jEDI accomplishes this by means of its two main capabilities.

a. **Cutting down the human interactio**n with the system, for it to function. The jEDI is capable of handling all the issues and scenarios that might arise in the infrastructures.
b. Orchestration of devices is done **simultaneously**, unlike many other technologies/ infrastructures that deal with devices sequentially.

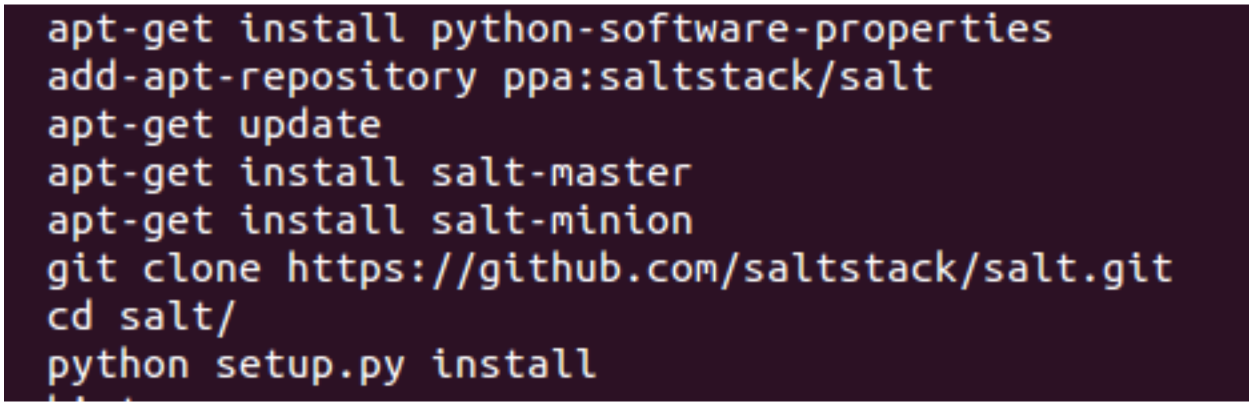
Architecture:



<u>To get the jEDI up and running:</u>

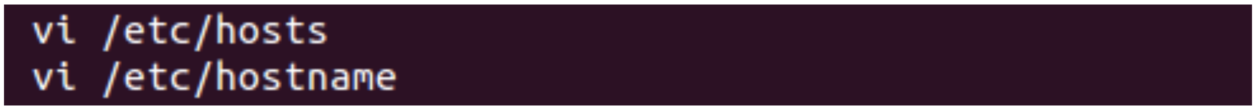
1. **<u>Setting up the jEDI Salt Stack (Master + Minion)</u>**

   Pull the salt stack from the git repo and install it in the Virtual Machine that you have. (we used Ubuntu 14.04 VM installed in VMware Fusion)



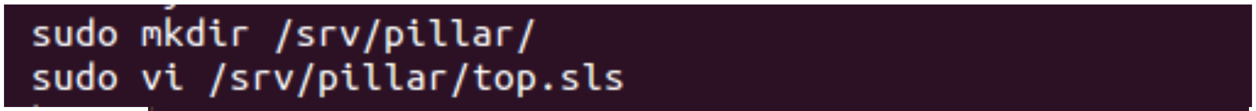
```
apt-get install python-software-properties
add-apt-repository ppa:saltstack/salt
apt-get update
apt-get install salt-master
apt-get install salt-minion
git clone https://github.com/saltstack/salt.git
cd salt/
python setup.py install
```

Also we have to change the hostname to "**SaltStackMinion**" in order for the minion to be populated as "SaltStackMinion" instead of "Ubuntu". So change the hostname in the below 2 files and restart the system to apply changes.

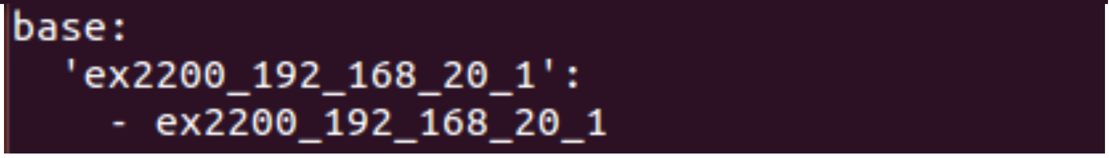


```
vi /etc/hosts
vi /etc/hostname
```

It's time to create the pillar files in which the proxy minion details will be residing. Create the directory and file names as mentioned below.

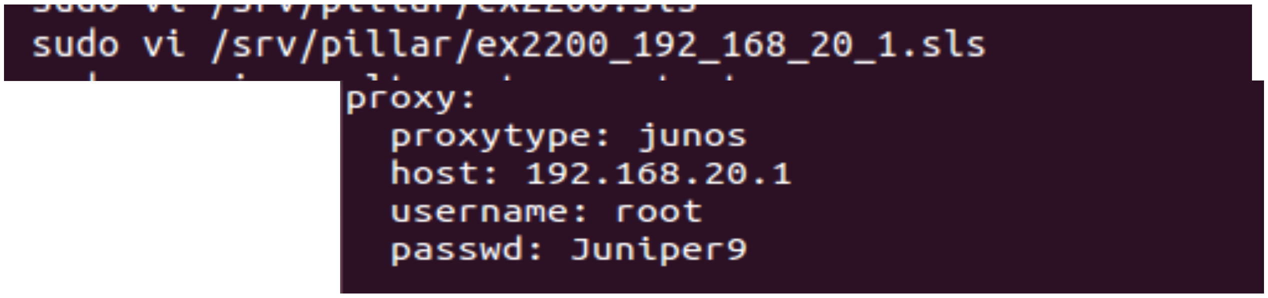


```
sudo mkdir /srv/pillar/
sudo vi /srv/pillar/top.sls
```



```
base:
  'ex2200_192_168_20_1':
    - ex2200_192_168_20_1
```

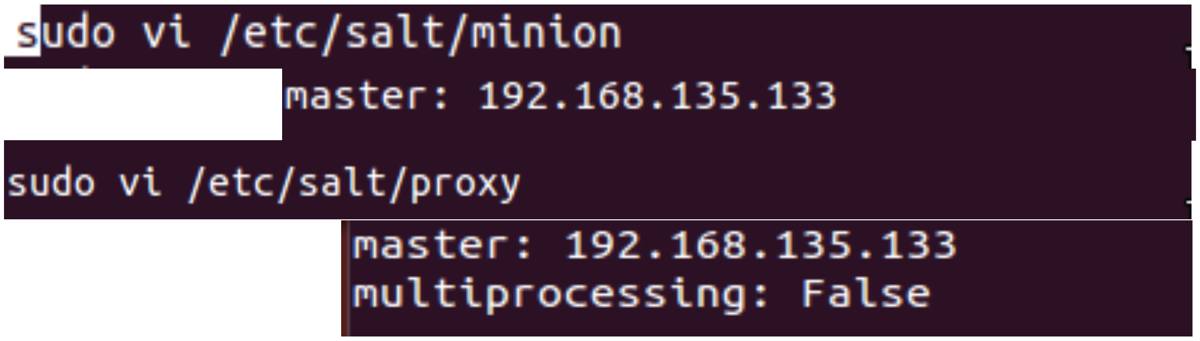
Now create the proxy minion file with the credentials to access the proxy minion.



```
sudo vi /srv/pillar/ex2200_192_168_20_1.sls
```

```
proxy:
  proxytype: junos
  host: 192.168.20.1
  username: root
  passwd: Juniper9
```

Include the below master IP that we got from the ifconfig and include it here in The Minion file and the proxy file.



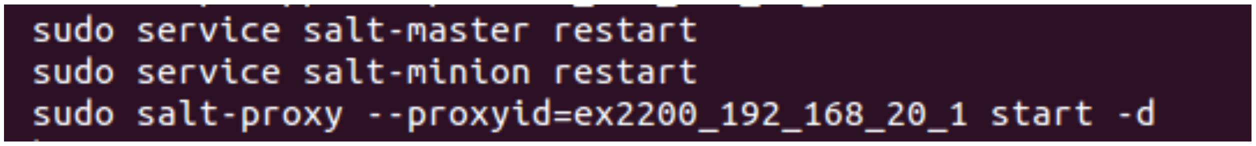
```
sudo vi /etc/salt/minion
```

```
master: 192.168.135.133
```

```
sudo vi /etc/salt/proxy
```

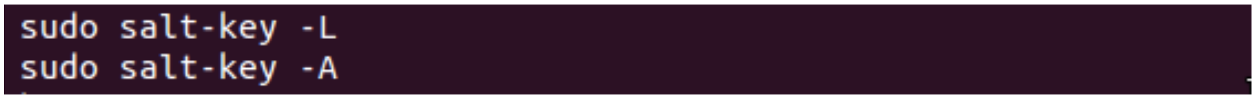
```
master: 192.168.135.133
multiprocessing: False
```

Restart the master and the minion. Also start the proxy minion that was created.



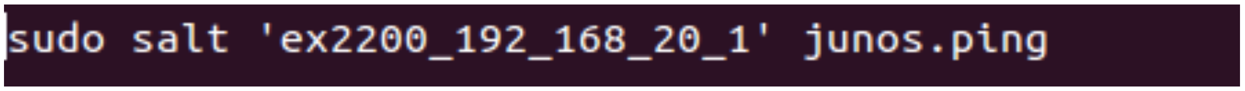
```
sudo service salt-master restart
sudo service salt-minion restart
sudo salt-proxy --proxyid=ex2200_192_168_20_1 start -d
```

The first command gives us the list of keys accepted and not accepted. By the second command, we can accept all the keys of the new minions, that were pending.



```
sudo salt-key -L
sudo salt-key -A
```

Now if you do a junos.ping, we can successfully ping the device.



```
sudo salt 'ex2200_192_168_20_1' junos.ping
```

2. **<u>Installing pyEZ:</u>**

```
sudo apt-get install python-setuptools python-dev build-essential
pip install --user setuptools==20.3.1
sudo apt-get install libffi-dev
pip install cryptography --force-reinstall
pip install junos-eznc
```

Create the below script and try running it to ensure that the PyEZ was installed as expected.

Python Script:

```
from pprint import pprint
from jnpr.junos import Device
dev = Device(host='192.168.20.1', user='root', password='Juniper9' )
dev.open()
pprint( dev.facts )
dev.close()
```

python pythonScript.py (Should run successfully and fetch the device details)

3. **<u>REACTOR SYSTEMS:</u>**

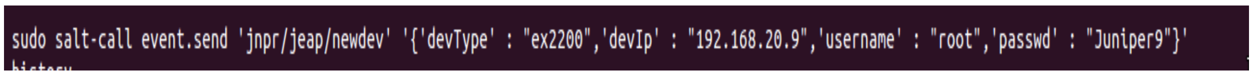
We have 2 reactor systems designed in the jEDI.

a. Auto Proxy Minion Creation System.
b. Jsnapy Reactor System.

A. Auto Proxy Minion Creation System:

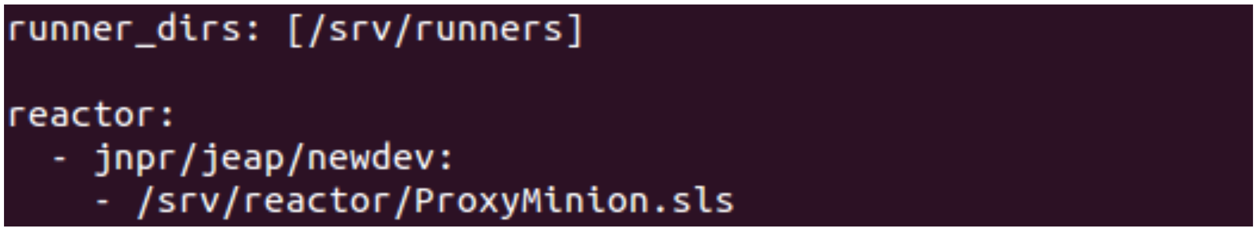
When a new device is added to the infrastructure, the device is brought up by using ZTP (ZTP Progress App) and an event is sent to the jEDI system with the credentials. The jEDI detects this event and spins up a new proxy minion immediately.

Event: sudo salt-call event.send 'myco/custom/event' '{'devType' : "ex2200",'devIp' : "192.168.20.6",'username' : "root",'passwd' : "Juniper9"}'



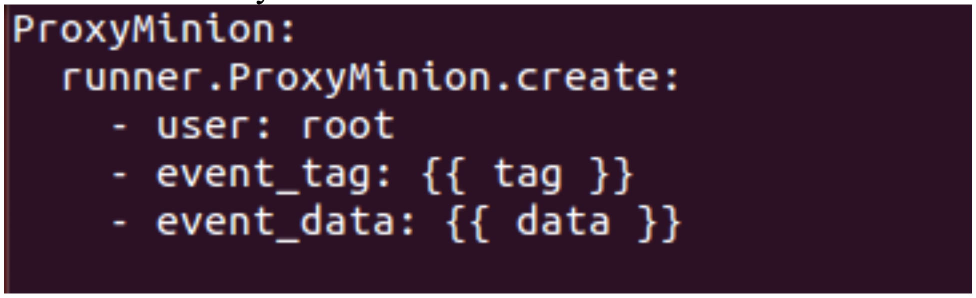
```
sudo salt-call event.send 'jnpr/jeap/newdev' '{'devType' : "ex2200",'devIp' : "192.168.20.9",'username' : "root",'passwd' : "Juniper9"}'
```

Reactor:
/etc/salt/master.d/reactor.conf



```
runner_dirs: [/srv/runners]

reactor:
  - jnpr/jeap/newdev:
    - /srv/reactor/ProxyMinion.sls
```

/srv/reactor/ProxyMinion.sls

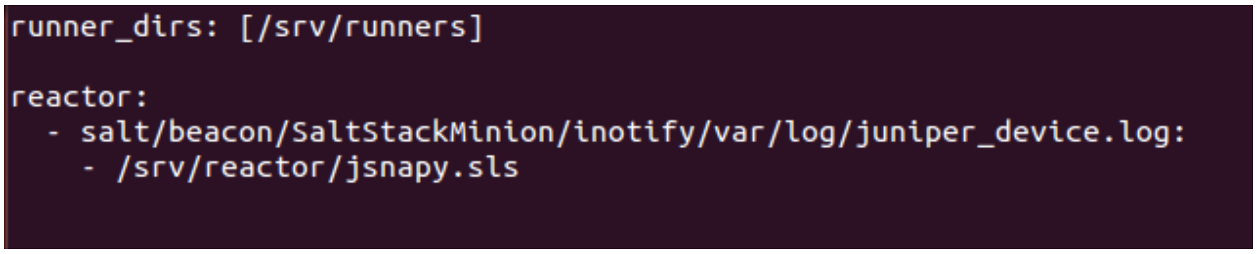


```
ProxyMinion:
  runner.ProxyMinion.create:
    - user: root
    - event_tag: {{ tag }}
    - event_data: {{ data }}
```

The reaction finally calls the "**Create function**" defined in the below file and Proxy minion is created automatically.
/srv/runners/ProxyMinion.py

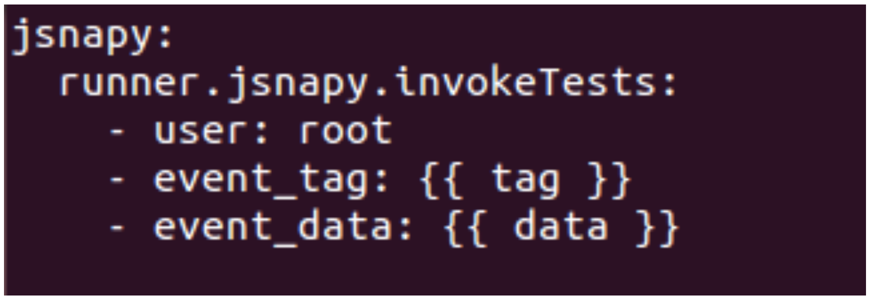
B. JSNAPY Reactor System:
When anybody commits in the device in the jEDI infrastructure the jEDI immediately verifies whether the changes were valid or not valid (eg: golden config change check). When the changes are not valid, an email will be sent to the Administrator saying that an invalid config was checked in and it has to be reverted.

/etc/salt/master.d/reactor.conf



```
runner_dirs: [/srv/runners]

reactor:
  - salt/beacon/SaltStackMinion/inotify/var/log/juniper_device.log:
    - /srv/reactor/jsnapy.sls
```

/srv/reactor/jsnapy.sls



```
jsnapy:
  runner.jsnapy.invokeTests:
    - user: root
    - event_tag: {{ tag }}
    - event_data: {{ data }}
```

This action of committing will invoke the JSNAPY "invokeTests" function which is in the below file.
/srv/runners/jsnapy.py

<u>**Device Config**</u>:
For the device to push the commit log to the minioin server, the following has to be configured in the juniper device:

```
syslog {
        host 192.168.20.2 {
                any any;
                match "commit complete";
        }
        source-address 192.168.20.1;
}
```

<u>**Minion Server Config**</u>:
<u>Install the syslog server in the minion:</u>

```
        sudo apt-get install syslog-ng-core
        sudo vi /etc/syslog-ng/syslog-ng.conf
                        source s_udp {
                        udp();
                        };
                        destination juniper_device {
                        file("/var/log/juniper_device.log");
                        };
                        log {
                        source(s_udp);
                        destination(juniper_device);
                        };
        sudo salt 'SaltStackMinion' pkg.install python-pyinotify
```

Create a file where the comit logs will be pushed:

```
sudo vi /var/log/juniper_device.log
```

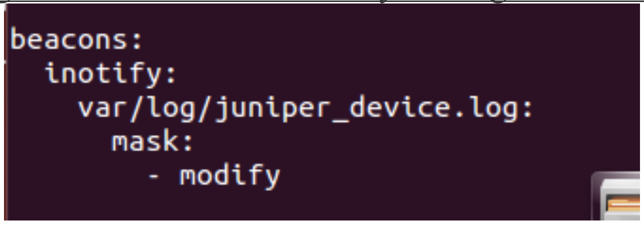
Install the python inotify:

```
sudo salt 'SaltStackMinion' pkg.install python-inotify
```

Restart the Syslog Server:

```
sudo service syslog-ng restart
```

Config beacons in the minion by adding the below in /ect/salt/minion:



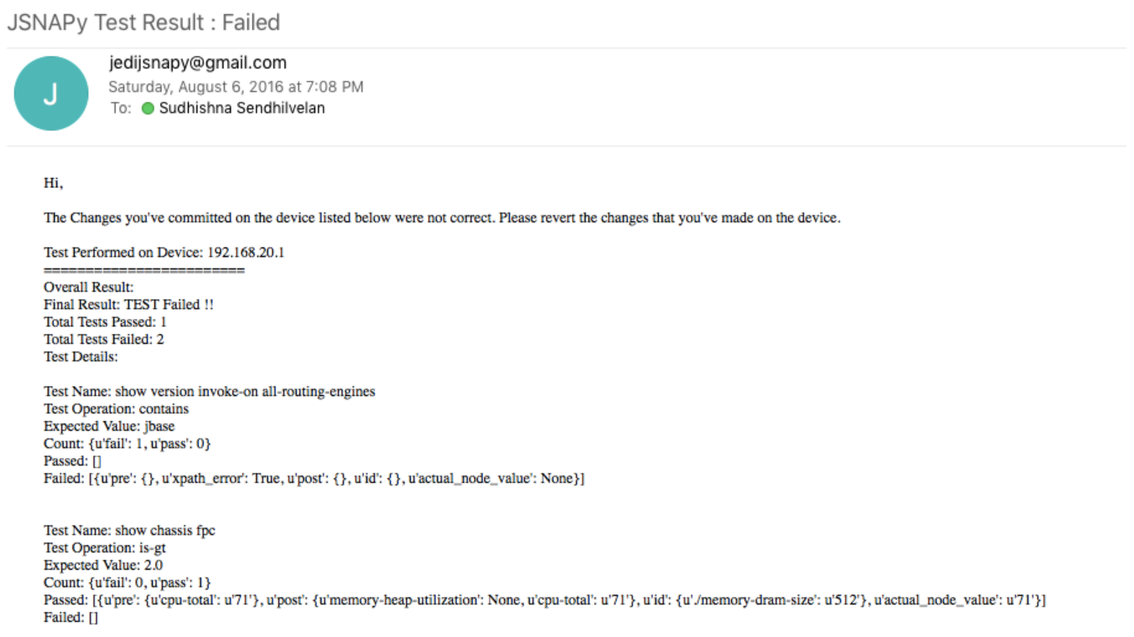
```
beacons:
  inotify:
    var/log/juniper_device.log:
      mask:
        - modify
```

Restar the minion to apply changes:

```
sudo service salt-minion restart
```

Now all the commit happening in the devices are monitored and tested by the jEDI.

The email that will be sent during jsnapy check failure is as below:



JSNAPy Test Result : Failed

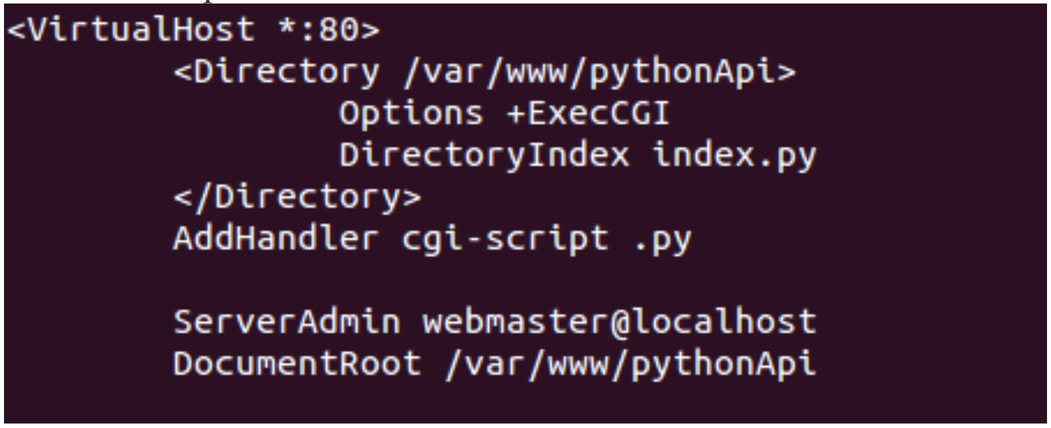
jedijsnapy@gmail.com
Saturday, August 6, 2016 at 7:08 PM
To: Sudhishna Sendhilvelan

```
Hi,

The Changes you've committed on the device listed below were not correct. Please revert the changes that you've made on the device.

Test Performed on Device: 192.168.20.1
========================
Overall Result:
Final Result: TEST Failed !!
Total Tests Passed: 1
Total Tests Failed: 2
Test Details:

Test Name: show version invoke-on all-routing-engines
Test Operation: contains
Expected Value: jbase
Count: {u'fail': 1, u'pass': 0}
Passed: []
Failed: [{u'pre': {}, u'xpath_error': True, u'post': {}, u'id': {}, u'actual_node_value': None}]


Test Name: show chassis fpc
Test Operation: is-gt
Expected Value: 2.0
Count: {u'fail': 0, u'pass': 1}
Passed: [{u'pre': {u'cpu-total': u'71'}, u'post': {u'memory-heap-utilization': None, u'cpu-total': u'71'}, u'id': {u'./memory-dram-size': u'512'}, u'actual_node_value': u'71'}]
Failed: []
```

4. **PYTHON CGI – Rendering data to the Slipstream Front End**

Install Apache2 and make the required config for python to render in browser:

sudo apt-get install python-pip
sudo apt-get install apache2
sudo apt-get install python-setuptools libapache2-mod-wsgi
sudo vi /etc/apache2/sites-available/000-default.conf



```
<VirtualHost *:80>
        <Directory /var/www/pythonApi>
                Options +ExecCGI
                DirectoryIndex index.py
        </Directory>
        AddHandler cgi-script .py

        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/pythonApi
```

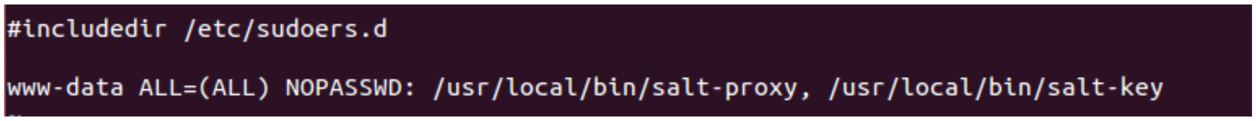
Install py curl to make the rest api calls from the python script:

sudo apt-get install python-pycurl

Create the python APIs directory:

sudo mkdir /var/www/pythonApi
sudo chmod 755 /var/www/pythonApi

Set the required permissions in /etc/sudoers:



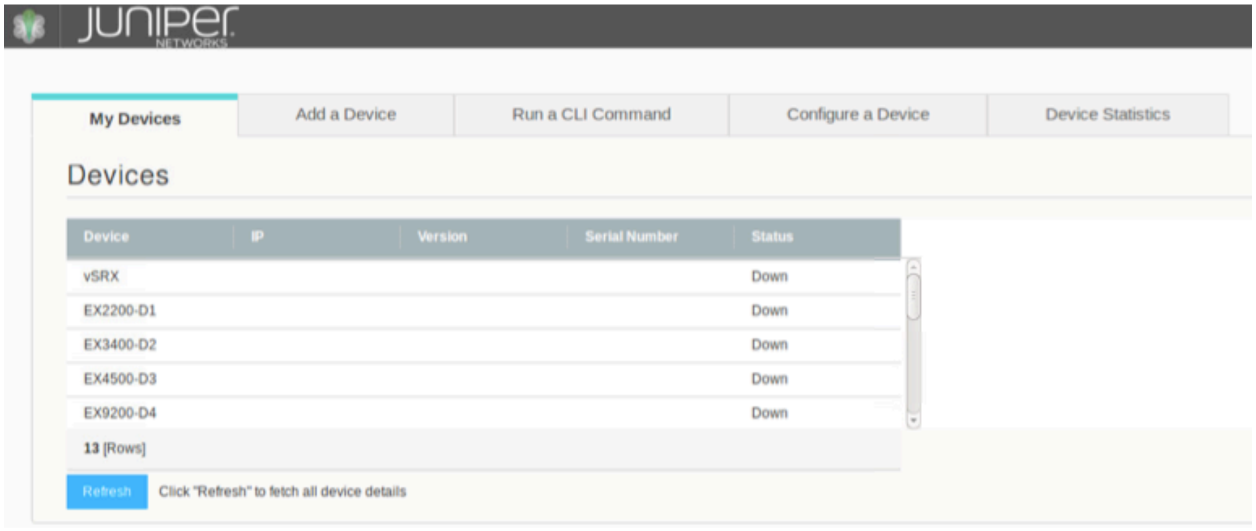
```
#includedir /etc/sudoers.d

www-data ALL=(ALL) NOPASSWD: /usr/local/bin/salt-proxy, /usr/local/bin/salt-key
```

Restart Apache to apply changes:

sudo service apache2 restart

Below are the python APIs residing in /var/www/pythonApi/

1. **DeviceListing.py:**
   This script fetches all the devices list in the jEDI infrastructure.
   Output: Json
   Front End Display:



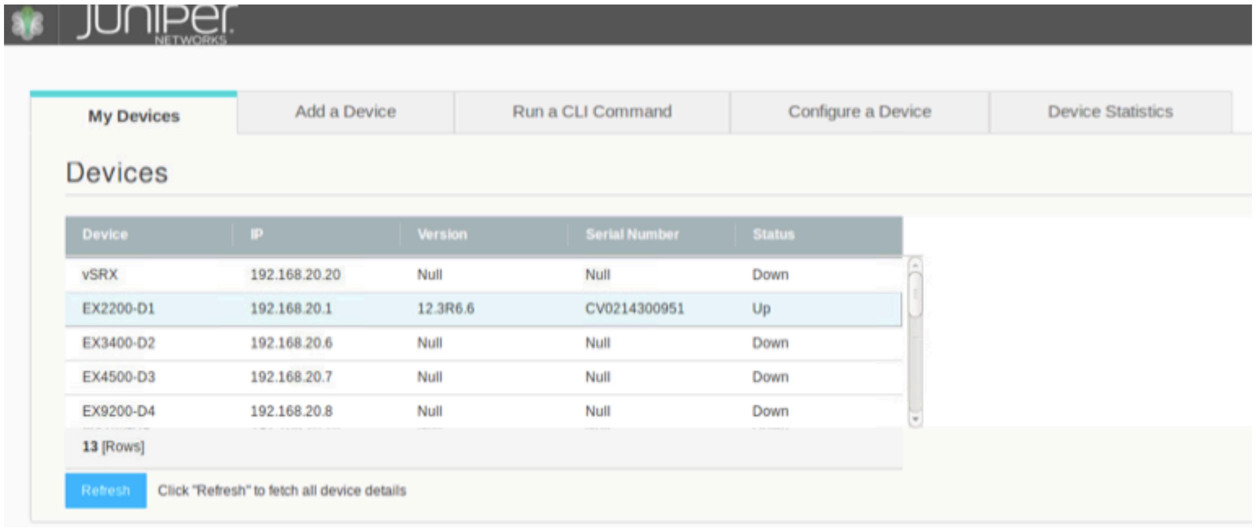
Devices

Device	IP	Version	Serial Number	Status
vSRX				Down
EX2200-D1				Down
EX3400-D2				Down
EX4500-D3				Down
EX9200-D4				Down

13 [Rows]

Click "Refresh" to fetch all device details

2. **GetDeviceDetails.py:**
   This script is used to fetch the details of the listed device
   Output: Json
   Front End Display:



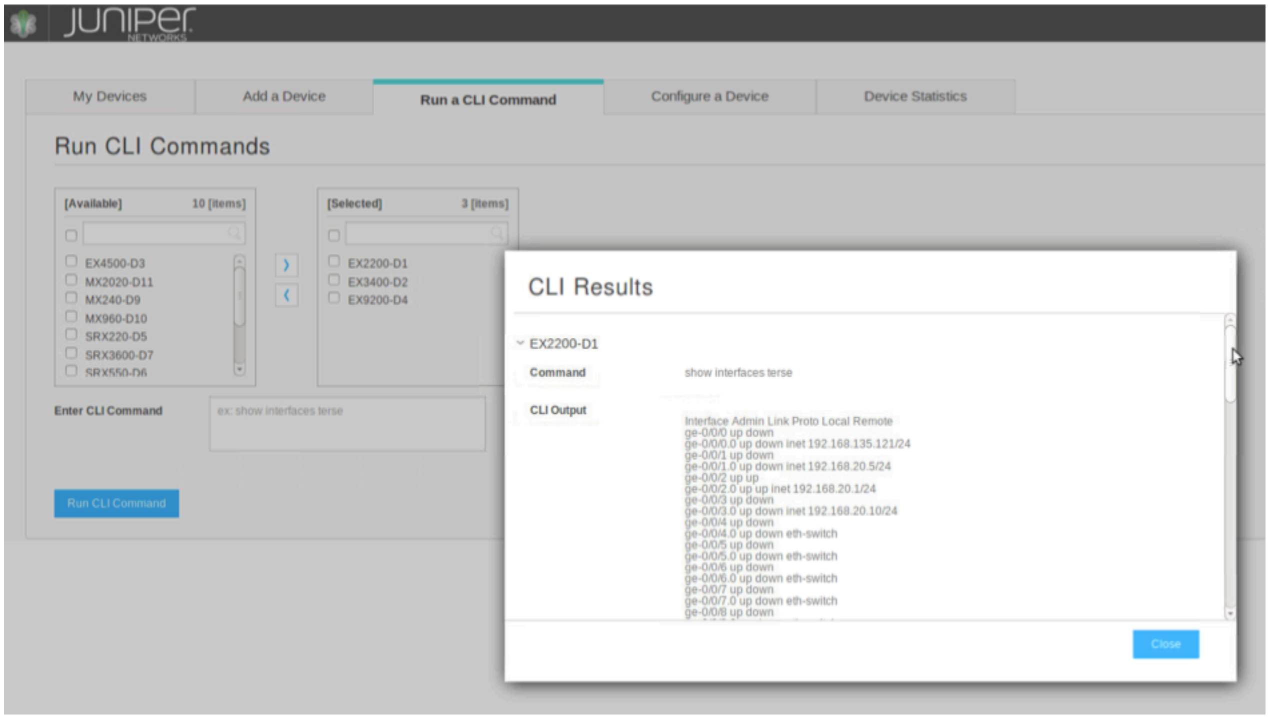
Devices

Device	IP	Version	Serial Number	Status
vSRX	192.168.20.20	Null	Null	Down
EX2200-D1	192.168.20.1	12.3R6.6	CV0214300951	Up
EX3400-D2	192.168.20.6	Null	Null	Down
EX4500-D3	192.168.20.7	Null	Null	Down
EX9200-D4	192.168.20.8	Null	Null	Down

13 [Rows]

Click "Refresh" to fetch all device details

3. **<u>ExecuteCli.py:</u>**
This script is used to run the command that was sent as parameter in the list of devices sent along and returns back the json data with the CLI output for each of those devices.
<u>Output:</u> Json
<u>Front End Display:</u>



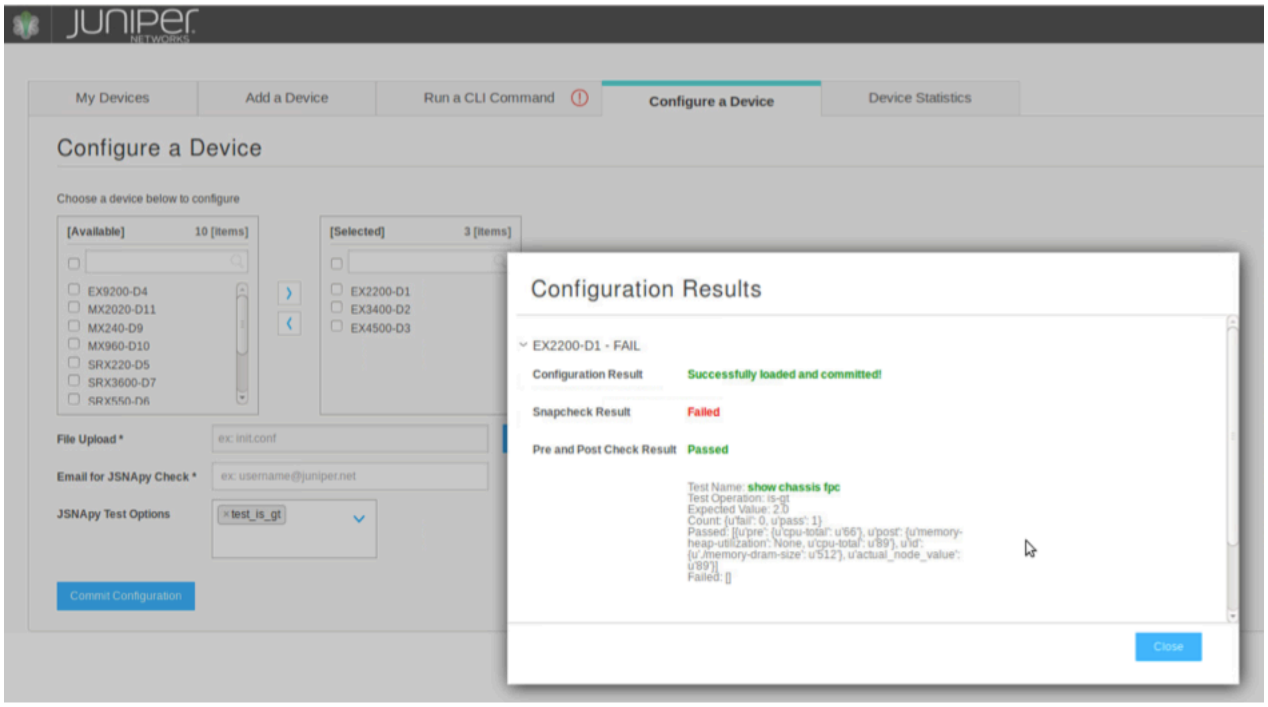
Command: show interfaces terse

CLI Output:

```
Interface Admin Link Proto Local Remote
ge-0/0/0 up down
ge-0/0/0.0 up down inet 192.168.135.121/24
ge-0/0/1 up down
ge-0/0/1.0 up down inet 192.168.20.5/24
ge-0/0/2 up up
ge-0/0/2.0 up up inet 192.168.20.1/24
ge-0/0/3 up down
ge-0/0/3.0 up down inet 192.168.20.10/24
ge-0/0/4 up down
ge-0/0/4.0 up down eth-switch
ge-0/0/5 up down
ge-0/0/5.0 up down eth-switch
ge-0/0/6 up down
ge-0/0/6.0 up down eth-switch
ge-0/0/7 up down
ge-0/0/7.0 up down eth-switch
ge-0/0/8 up down
```

4. **<u>ConfigDevice.py:</u>**
   a. This script configures the list of device with the config data in the file
   b. It also reports the golden config output by reading the file to which the reactor system writes the result to
   c. It also performs the JSNAPy pre and post check and returns the result in the json.
   d. <u>Output: </u>Json

   <u>Front End Display:</u>



Configure a Device

Choose a device below to configure

[Available] 10 [items]	[Selected] 3 [items]
EX9200-D4	EX2200-D1
MX2020-D11	EX3400-D2
MX240-D9	EX4500-D3
MX960-D10	
SRX220-D5	
SRX3600-D7	
SRX550-D6	

File Upload * ex: init.conf

Email for JSNApy Check * ex: username@juniper.net

JSNApy Test Options ×test_is_gt

Configuration Results

EX2200-D1 - FAIL

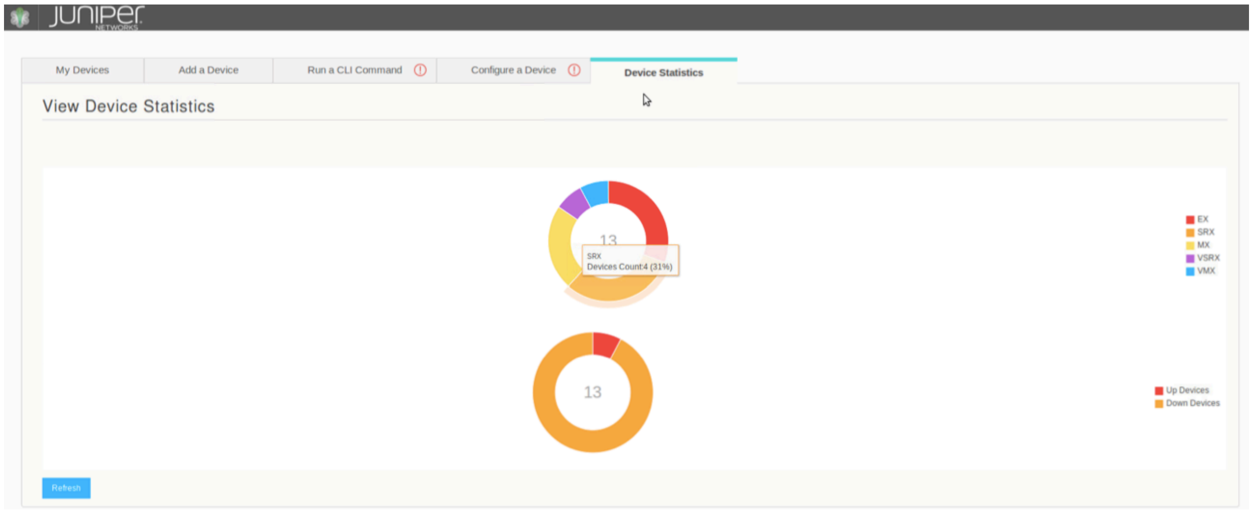
Configuration Result	Successfully loaded and committed!
Snapcheck Result	Failed
Pre and Post Check Result	Passed

```
Test Name: show chassis fpc
Test Operation: is-gt
Expected Value: 2.0
Count: {u'fail': 0, u'pass': 1}
Passed: [{u'pre': {u'cpu-total': u'66'}, u'post': {u'memory-
heap-utilization': None, u'cpu-total': u'89'}, u'id':
{u'./memory-dram-size': u'512'}, u'actual_node_value':
u'89'}]
Failed: []
```

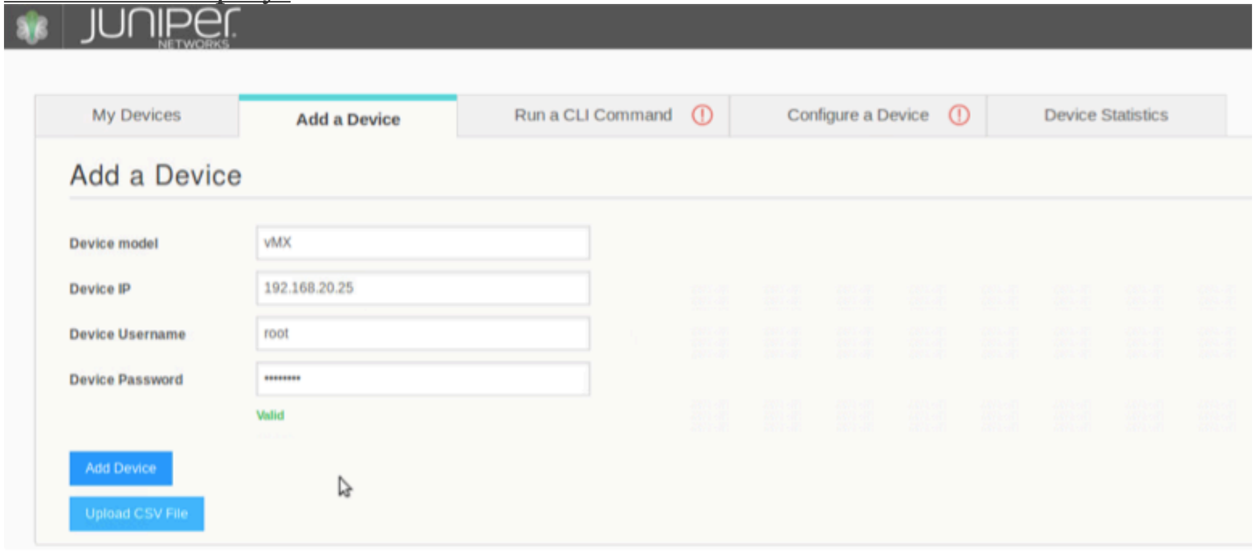
5. **StatusStatistics.py:**
   The devices status whether they are up or down is fetched and returned in this script.
   Output: Json

6. **TypeStatistics.py**
   The devices types (eg: srx, vsrx, mx, vmx, ex. Etc..) is fetched and returned in this script.
   Output: Json

   Front End Display for both Statistics:



7. **AddNewDevice.py:**
   If we have a already up and running device then we have another option adding them through GUI along with the reactor system.
   Output: Json
   Front End Display:

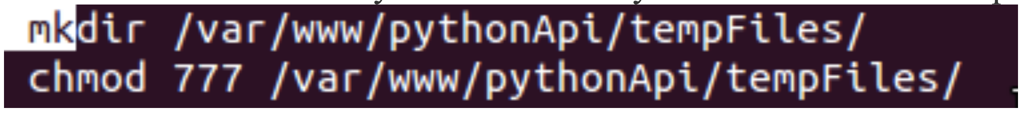


Add a Device

Device model	vMX
Device IP	192.168.20.25
Device Username	root
Device Password	********
	Valid

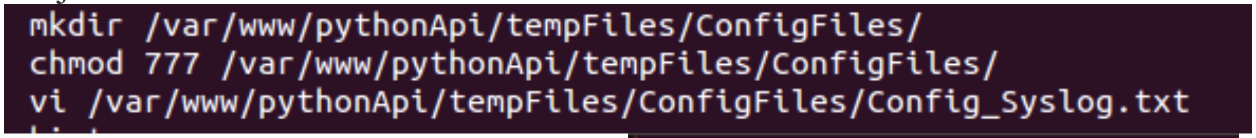
Note:
Remember to set all the python API files to have a chmod 755 permissions

Create the below directory which is used by the APIs to create temporary files.

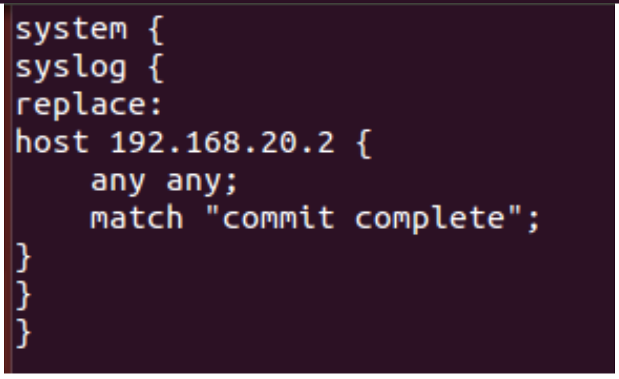


```
mkdir /var/www/pythonApi/tempFiles/
chmod 777 /var/www/pythonApi/tempFiles/
```

Also Create the below folder in the above directory and put the required base configuration config files that has to be installed when a new device is added to the jEDI infrastructure.



```
mkdir /var/www/pythonApi/tempFiles/ConfigFiles/
chmod 777 /var/www/pythonApi/tempFiles/ConfigFiles/
vi /var/www/pythonApi/tempFiles/ConfigFiles/Config_Syslog.txt
```



```
system {
syslog {
replace:
host 192.168.20.2 {
    any any;
    match "commit complete";
}
}
}
```

5. **Future Scope:**
    a. When a device goes down we can make the jEDI detect that react accordingly to bring the device back.
    b. Instead of informing the administrator when a golden config change was made, we can make the jEDI revert the changes by itself or file a ticket automatically.