



Cisco Advanced Services



NSO Test Automation Framework (NSO-TAF)

Cisco Advanced Services

June 2, 2017

Version 0.3

**Cisco Systems, Inc.
Corporate Headquarters
170 West Tasman Drive
San Jose, CA 95134-1706 USA
<http://www.cisco.com>
Tel: 408 526-4000 Toll Free: 800 553-NETS (6387)
Fax: 408 526-4100**

Contents

| | |
|--|----------|
| CONTENTS | 2 |
| ABOUT NSO-TEST AUTOMATION FRAMEWORK | 2 |
| HISTORY | 2 |
| REVIEW | 2 |
| DOCUMENT CONVENTIONS | 2 |
| 1 INTRODUCTION | 4 |
| 1.1 PURPOSE | 4 |
| 1.2 INTENDED AUDIENCE | 4 |
| 1.3 OVERVIEW..... | 4 |
| 2 ARCHITECTURE | 5 |
| 2.1 LOGICAL ARCHITECTURE..... | 5 |
| 2.2 BUILDING BLOCKS | 5 |
| 2.2.1 Ansible..... | 5 |
| 2.2.2 Linux Containers (LXC)..... | 6 |
| 2.2.3 Openstack..... | 7 |
| 2.2.4 Flask..... | 7 |
| 3 HOW TO USE | 9 |
| 3.1 INSTALL NSO-TAF | 9 |
| 3.2 ONBOARD A NEW PROJECT..... | 11 |
| 3.3 RUN NSO-TAF ON A STAND-ALONE ENVIRONMENT | 15 |
| 3.3.1 <i>Web based REST Interface</i> | 15 |
| 3.3.2 <i>CLI based Interface</i> | 17 |
| 3.4 RUN NSO-TAF IN A CI/CD ENVIRONMENT: | 18 |
| 3.5 RUN CUSTOM SCRIPTS FOR A PROJECT USING NSO-TAF | 19 |
| 3.6 UPDATE AND RUN TEST CASES USING NSO-TAF | 20 |
| 3.7 WRITE A TEST SCRIPT OR TEST CASE..... | 20 |



About NSO-Test Automation Framework

Author Prashanth Chintalapudi
Co-Authors Vishesh Kansal
Change Authority Cisco Systems Advanced Services

History

| Version No. | Issue Date | Status | Reason for Change |
|-------------|------------|-----------|----------------------------------|
| 0.1 | 15/05/2017 | Pre-Draft | Initial draft |
| 0.2 | 23/05/2017 | Pre-Draft | Added writing test cases section |
| 0.3 | 30/05/2017 | Pre-Draft | Changes in screenshots |
| | | | |
| | | | |

Review

| Reviewer's Details | Version No. | Date |
|--------------------|-------------|----------------|
| Kiran Kulkarni | 0.1,0.2,0.3 | May 30th, 2017 |
| | | |

Document Conventions



Alerts readers to take note. Notes contain helpful suggestions or references to material not covered in the document.



Alerts readers to be careful. In this situation, you might do something that could result in equipment damage or loss of data.



Alerts the reader that they can save time by performing the action described in the paragraph affixed to this icon.



Alerts the reader that the information affixed to this icon will help them solve a problem. The information might not be troubleshooting or even an action, but it could be useful information similar to a Timesaver.

1 Introduction

1.1 Purpose

The document has been designed to detail about Test Automation Framework and its components. It also covers the framework architecture and how to use it for NSO service applications.

1.2 Intended Audience

This document is primarily for developers and testers who write the unit test cases and want to automate the testing of their code developed by them.

The framework software can be interesting to all but it is accessible for internals only. This framework can be used for reference.

1.3 Overview

TAF (Test Automation Framework) provides a generic framework to automate unit testing of the NSO service applications leveraging the CI/CD process and tools (GitHub, Jenkins, Artifactory, Sonarqube) with a single touch approach.

The objectives of having a test automation framework is to provide a simple, formal & automated platform integrated with CICD for delivering the quality NSO code.

TAF is built using the python language which makes the framework easy to integrate with different tools and run across multiple platforms.

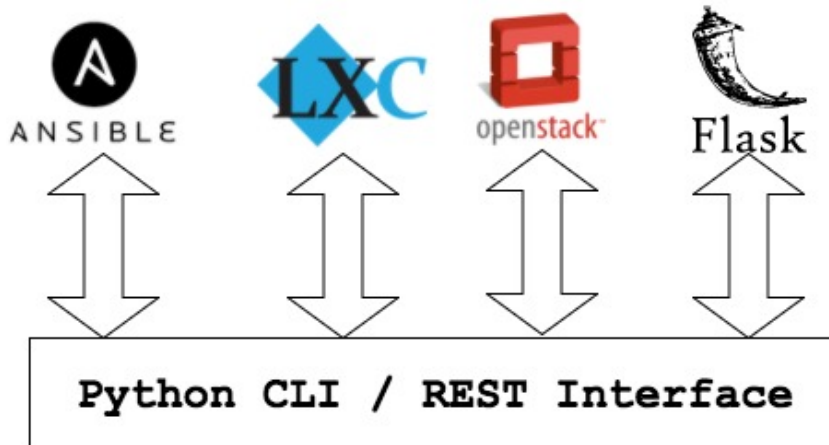
TAF supports unit test cases written in different ways like Python, JSON, LUX and flexible enough to run a test script in any format.

The test automation framework is built on python as the core platform language, making the framework easy to integrate with different tools and run across multiple platforms.

The framework maintains a multi-threaded environment in the running state, executing Ansible, Openstack, flask, Linux containers.

2 Architecture

2.1 Logical Architecture



The framework exposes CLI and REST interfaces to northbound applications. Framework accepts the inputs from the northbound interface and enables multiple threads on top of which either of the four applications (Ansible, LXC, Openstack, Flask) will be running.

2.2 Building Blocks

2.2.1 Ansible

Ansible is an open source automation platform. It is very, very simple to setup and yet powerful. Ansible can help you with configuration management, application deployment, task automation. It can also do IT orchestration, where you have to run tasks in sequence and create a chain of events which must happen on several different servers or devices.

Ansible aims to be:

- Clear - Ansible uses a simple syntax (YAML) and is easy for anyone (developers, sysadmins, managers) to understand. APIs are simple and sensible.

- Fast - Fast to learn, fast to set up—especially considering you don't need to install extra agents or daemons on all your servers
- Complete - Ansible does three things in one, and does them very well. Ansible's 'batteries included' approach means you have everything you need in one complete package.
- Efficient - No extra software on your servers means more resources for your applications. Also, since Ansible modules work via JSON, Ansible is extensible with modules written in a programming language you already know.
- Secure - Ansible uses SSH, and requires no extra open ports or potentially-vulnerable daemons on your servers.

The test automation framework generates the tasks in Ansible's playbook format (YAML) format and uploads them onto the run-time workspace of the specific project.

2.2.2 Linux Containers (LXC)

LXC (Linux Containers) is an operating-system-level virtualization method for running multiple isolated Linux systems (containers) on a control host using a single Linux kernel.

LXC aims to be:

- Lightweight and resource-friendly - Enables running multiple instances of an operating system or application on a single host, without inducing overhead on CPU and memory.
- Comprehensive process and resource isolation - Safely and securely run multiple applications on a single system without the risk of them interfering with each other.
If security of one container has been compromised, the other containers are unaffected
- Run multiple versions of an operating system on a single server.
- Rapid and Easy deployment - Containers can be useful to quickly set up a "sandbox" environment

As part of the Ansible tasks, a Linux container is deployed for each new project run by the framework to isolate the project specific test environments and minimize the infrastructure utilisation.

2.2.3 Openstack

OpenStack is a free and open-source software platform for cloud computing, mostly deployed as an infrastructure-as-a-service (IaaS). The software platform consists of interrelated components that control diverse, multi-vendor hardware pools of processing, storage, and networking resources throughout a datacentre.

Framework supports testing on both network simulators as well as the real devices. For each project, based on the project inputs or the requirement, a real device is spawned using the framework's in-built orchestrator module and once the network is ready, testing happens!

2.2.4 Flask

Flask is a micro web framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine. It is BSD licensed. Flask is called a micro framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself.

TAF can be run either in a stand-alone system environment or in a CI/CD environment integrating multiple things. While running the framework as a stand-alone system, TAF is also provided with a GUI containing a simple JavaScript form for collecting the inputs.

Flask also enables the REST based interface of the test framework, where the framework can be triggered with a REST request over HTTP.

3 How to USE

3.1 Install NSO-TAF

Installing the test automation framework is just a script away to execute.

Step 1: Cloning the source from github3

```
root@vm-pchinta2-002:~/prashant# git clone
https://github3.cisco.com/AS-NSO-Test/Tail-f-NSO.git
Cloning into 'Tail-f-NSO'...
Username for 'https://github3.cisco.com': pchinta2
Password for 'https://pchinta2@github3.cisco.com':
remote: Counting objects: 3100, done.
remote: Total 3100 (delta 0), reused 0 (delta 0),
pack-reused 3100
Receiving objects: 100% (3100/3100), 45.50 MiB |
25.98 MiB/s, done.
Resolving deltas: 100% (1142/1142), done.
Checking connectivity... done.
```


3.2 Onboard a new project

On successful clone and installation of NSO-TAF from GitHub, below mentioned directory should be expected.

```
nso-taf
├── develop
│   ├── common
│   ├── packages
│   ├── scripts
│   ├── src
│   │   ├── app
│   │   ├── static
│   │   │   ├── css
│   │   │   ├── fonts
│   │   │   └── js
│   │   └── templates
│   └── tasks
├── python2-lxc
│   ├── build
│   │   ├── lib.linux-x86_64-2.7
│   │   └── lxc
│   └── temp.linux-x86_64-2.7
└── lxc
```

- ‘develop’ contains the source code and other development operations related sub-directories
 - ‘common’ directory contains the bash/python/Perl scripts which framework uses while running the automation tasks
 - ‘packages’ directory shall contain any third party software/service-packages or installation binaries.
 - ‘scripts’ is a provisional directory where user can upload all the custom scripts or develop custom application and can expect the framework to execute it pre or post testing exercise.
 - ‘src’ directory contains run file for both web and cli based application and flask enabled sub-dirs.

- 'tasks' directory contains Ansible playbook written in YAML format, which framework in a phase based approach.
- 'python2-lxc' is the LXC binding used to deploy Linux containers either locally or on the remote machines.

Step 1: Setup either Github account or CEC account or both by running configure.py script

```
root@vm-pchinta2-002:~/Tail-f-NSO/nso-  
taf/develop/src#  
python configure.py  
***** CONFIGURING GITHUB ACCOUNT:....  
Enter github username: pchinta2  
Get the password:  
***** CONFIGURING CISCO ACCOUNT:....  
Enter CEC-ID: pchinta2  
Enter CEC password:
```

Step 2: (Optional) Fill the configuration file based on the project requirements

- Open nso-taf/develop/config.ini in any preferred editors
- Fill the Project and Test Scripts section of the file
- `neds` and `service-package` can be comma (`,`) separated values

```
# This is the config file read by python application to perform certain kind of automation.
# It needs three sections of inputs, namely: 'Account', 'Project' and 'Test Scripts'
# There are no default values taken for the variables, hence it is expected that all the values are required to be entered

[Account]
# 'machine-ip' is expected to be the IP address of the machine where desired tests need to be performed
# 'cec-id' is required to download any required nso-related packages/files
# as entering 'cec-password' in a file is not a desired approach, cec-password needs to be entered when prompted on running the python application
machine-ip = 127.0.0.1

[Project]
# 'name' can be any valid string with readable chars for a given project
# 'nso-bin' can either be a local path to binary file copied to packages/ or https url to download from the remote repository and copy to project/packages/nso
# 'neds' are the required NEDs for this project and should strictly be a tar.gz file with or without a link.
# 'jars' are the actual developed packages stored remotely for eg: artifactory and can be filled in a similar way as mentioned for above variables
name = vz
nso-bin = /root/nso-4.2.0.1.linux.x86_64.installer.bin
neds = /root/ncs-4.2-cisco-nx-4.1.13.tar.gz,
service-package = /root/vMMEPeerConfigure,
install-type = local

[Test Scripts]
# 'lux-scripts' can either be placed a folder with name 'lux-scripts' in packages/ or provide the https link to download and copy it to the project/packages folder
test-scripts = None

[GITHUB]
username = pchinta2
password = UGMuMjQwMiQk

[CISSCO]
username = pchinta2
password = UGMuMjQwMiQk
```

- Below is an example config file format to be filled up manually

Step 3: Run the nso-taf.py script

- If Step-2 has been done, then execute the nso-taf.py in config-mode as shown below

```
python nso-taf.py config-mode --name test -cf
./config.ini
```

- If the project NEDS, NSO Binary file and the service packages are all present locally in the system, then execute the nso-taf.py in system-mode as shown below

```
python nso-taf.py system-mode --name test --neds /root/ncs-4.2-cisco-nx-4.1.13.tar.gz --nso /root/nso-4.2.0.1.linux.x86_64.installer.bin --packages /root/vMMEPeerConfigure --type local
```

- If the project NEDS, NSO Binary file and service packages all need to be copied from a http source, the execute nso-taf.py in github-mode as show below

```
python nso-taf.py github-mode --name test --neds https://...../ncs-pkgs/cisco-ios/4.1/ncs-4.1-cisco-ios-4.0.2.tar.gz https://.../ncs-pkgs/cisco-iosxr/4.1/ncs-4.1-cisco-iosxr-4.0.1.1.tar.gz --nso https://.../ncs/nso-4.1.linux.x86_64.installer.bin -g https://github3.cisco.com/AS-ATT-CCS/ATT_CCS.git
```

- If the project environment has already been setup and want to re-use the same for further testing, then execute nso-taf.py in run-mode as shown below

```
python nso-taf.py run-mode --name vzw
```

3.3 Run NSO-TAF on a stand-alone environment

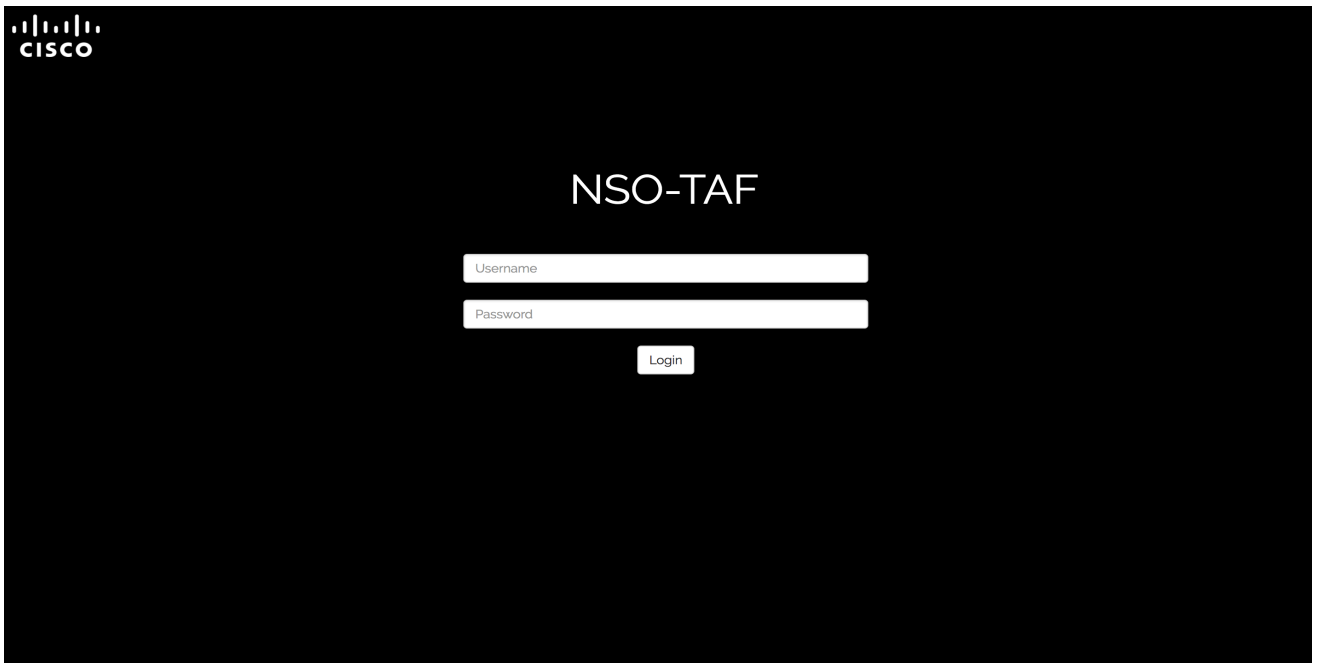
There are two ways provisioned by the framework to get started on a stand-alone environment:

3.3.1 Web based REST Interface

- This enables the framework to get started with a GUI interface.
- User needs to provide the inputs on the html form and click submit
- Execute the 'run.py' located in develop/src/

```
root@vm-nprabhal-001:~/prashant/nso-taf/develop/src# python
run.py
* Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger pin code: 320-368-168
```

- Upon error-free execution, type management IP address of the machine where framework is running on the browser.



- Login with 'admin' and 'admin' for username and password
- Upon successful login, below html form will be shown up

- Enter the required inputs in the form and click submit
- Framework then forms the Ansible playbook tasks and starts executing each phase of testing starting from deploying a container for the project to creating and running the lab environment.
- Below is a kind of output expect on the browser.

```

/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: WARNING: All keys were skipped because they already exist on the remote system.

PLAY [nso-test] *****
TASK [setup] *****
ok: [127.0.0.1]

TASK [Run apt-get update] *****
ok: [127.0.0.1]

TASK [Install linux container dependencies] *****
ok: [127.0.0.1] => (item=[u'lxco', u'lxco-dev', u'lxctl', u'lxco-templates'])

TASK [Install other dependencies] *****
ok: [127.0.0.1] => (item=[u'python-software-properties', u'python-dev', u'python-pip', u'python-yaml', u'git', u'opensl'])

TASK [Start clean-up container] *****
changed: [127.0.0.1]

TASK [Destroy existing container] *****
changed: [127.0.0.1]

TASK [Creating new container setup] *****
changed: [127.0.0.1]

TASK [Wait for the network to be setup in the containers] *****
Pausing for 10 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [127.0.0.1]

TASK [Get containers info now that IPs are available] *****
ok: [127.0.0.1]

TASK [debug] *****
ok: [127.0.0.1] => {
  "msg": {
    "changed": false,
    "lxc_container": {
      "init_pid": 3518,
      "interfaces": {
        "eth0": {
          "lo",
          "lo"
        },
        "ips": {
          "10.0.3.93"
        },
        "name": "telstra",
        "state": "running"
      }
    }
  }
}

```

3.3.2 CLI based Interface

- On the console, the framework can be run in either of the four modes shown below

```
usage: nso-taf-test.py [-h] {run-mode,config-mode,system-mode,github-mode}
...
positional arguments:
  {run-mode,config-mode,system-mode,github-mode}
  run-mode              if running-directory already exists
  config-mode           if config-file is existing to replicate a project
  system-mode           system paths for all the project requirements
  github-mode           refers to project from github url
optional arguments:
  -h, --help            show this help message and exit
```

- run-mode, config-mode, system-mode, github-mode
- ‘run-mode’ is to execute an existing framework by passing the project name as the parameter.

```
root@vm-nprabhal-001:~/prashant/nso-taf/develop/src# python nso-taf-
test.py run-mode -h
usage: nso-taf-test.py run-mode [-h] [--name NAME]
```

- ‘config-mode’ is to replicate an existing project config to a new project

```
root@vm-nprabhal-001:~/prashant/nso-taf/develop/src# python nso-taf-
test.py config-mode -h
usage: nso-taf-test.py config-mode [-h] [--name NAME] -cf CONFIG_FILE
```

- ‘system-mode’ is to create a new project with all the required parameters

```

root@vm-nprabhal-001:~/prashant/nso-taf/develop/src# python nso-taf-
test.py system-mode -h
usage: nso-taf-test.py system-mode [-h] [--name NAME] [--neds NEDS [NEDS
...]]
                                [--nso NSO]
                                [--packages PACKAGES [PACKAGES ...]]
                                [--test_scripts TEST_SCRIPTS]

```

- ‘github-mode’ is to integrate a remote git repository to the framework

```

root@vm-nprabhal-001:~/prashant/nso-taf/develop/src# python nso-taf-
test.py github-mode -h
usage: nso-taf-test.py github-mode [-h] [--name NAME] [--neds NEDS [NEDS
...]]
                                [--nso NSO] -g GITHUB_URL -u USERNAME -
p
                                PASSWORD

```

3.4 Run NSO-TAF in a CI/CD environment:

- In a CI/CD environment, the framework needs to be integrated with the Jenkins by configuring the Jenkins job for calling the script via cli based interface or making a HTTP REST request.
- Once a Jenkins job is created, build actions needs to be configured as shown below

```
Execute shell
Command GIT_URI=https://github3.cisco.com/...{[PROJECT_NAME]}.git
PROJECT_NAME=TEST
IOS_NED=https://.../ncs-pkgs/cisco-ios/4.1/ncs-4.1-cisco-ios-4.0.2.tar.gz
IOSXR_NED=https://.../ncs-pkgs/cisco-iosxr/4.1/ncs-4.1-cisco-iosxr-4.0.1.1.tar.gz
NSO_BIN=https://.../ncs/nso-4.1.linux.x86_64.installer.bin
SCRIPT=/root/Tail-f-NSO/nso-taf/develop/src/nso-taf.py
VM_INSTANCE=vm-gdc-blr-001
sshpass -p Cisco123 ssh root@[VM_INSTANCE] "python $SCRIPT github-mode --name $PROJECT_NAME --neds $IOS_NED $IOSXR_NED --nso $NSO_BIN -g $GIT_URL"
```

See [the list of available environment variables](#)

3.5 Run custom scripts for a project using NSO-TAF

- If any custom scripts need to be run before or after the NSO-TAF runs the actual test cases.
- Those scripts can be uploaded under nso-taf/develop/scripts/

```
root@vm-pchinta2-002:~/Tail-f-NSO/nso-taf/develop# tree scripts/
scripts/
├── post
├── post_main.py
├── pre
│   └── test.sh
└── pre_main.py
```

- `pre` directory supports all the scripts to be uploaded which needs to be run before the testing phase gets started by NSO-TAF.
- `post` directory support all the scripts to be uploaded which needs to be run after the testing phase is completed by NSO-TAF.
- NSO-TAF supports both `bash` and `python` scripts as custom scripts

3.6 Update and run test cases using NSO-TAF

- If the project environment is already existing and running, upload the test cases to /var/opt/{project name}/test-scripts/
 - If the scripts are JSON payloads, an `expected` directory with all the responses for the JSON payloads needs to be uploaded in the same directory
 - `endpoint` is the file which needs to be updated manually for the curl URL which is the service endpoint
 - Run the nso-taf.py in `run-mode` as explained above
- If the project environment is not already existing, upload the test cases to nso-taf/develop/test-scripts/
 - Run nso-taf.py in either `system-mode` and `github-mode` and the test cases will be as part of taf's execution

3.7 Write a test script or test case

- NSO-TAF supports test scripts written in Python, Lux or JSON
- NSO-TAF is flexible to run a test script in any format

Below is a sample Lux script format:

```
[doc Test for validating for basement switch config]
[global ncs_dir=/home/prashant/work/ncs-4.2.1-sep]
[global network_element_id=bsm0-b-025]
[global device_model=ME-3400G-2CS-A]
[global serial=serial0]
[global uplink_domain_type=WIP]
[global vtp_domain=vtp01]
[global location=Sydney]
[global interface_local=0/0/3]
[global device_remote=pop-s-01]
[global interface_remote=0/0/1]
[global ip_address=127.0.0.1]
[global gateway_ip=192.168.0.1]
[global protocol=ssh]
[global port=10022]
[shell main]
!/bin/bash
!source $ncs_dir/ncsrc
!ncs_cli -u admin -C
?admin connected from.*
```

```

!config
!network infrastructure access basements basement
$network_element_id device-model $device_model location
$location serial $serial vtp-domain $vtp_domain uplink-domain-
type $uplink_domain_type service-reconciliation no management-
network ip-address $ip_address gateway-address $gateway_ip
port $port protocol $protocol
!top
!network infrastructure access basements basement
$network_element_id uplinks $interface_local device-remote
$device_remote interface-remote $interface_remote
"""?
.*
syntax[\\s]error.*
""
!end
!exit
!exit

```

Below is a sample JSON payload:

```

{
  "input": {
    "mme-name": "ericsson-mme-01",
    "device": "NX-7k1",
    "request-id": "2",
    "request-action": "CreateRequest",
    "mme-type": "ericsson",
    "nexus-config": {
      "device-role": "primary",
      "l2-vlan": [{
        "l2-vlan-id": 920,
        "l2-vlan-name": "192.168.25.0/24"
      }],
      "vrf-wsn-community" : "6167:6804",
      "vrf-edn-community" : "65004:11300",
      "vrf": [{
        "vrf-name": "RAN",
        "sgs-lite": "true",
        "vip": [{
          "dns-vip": "2001:4888:200:1005:528:28a:0:42/128",
          "backup-egress-svi": 215
        }
      ]
    }
  },
  "vrrp-v6": {
    "vrrp-number": 203,
    "priority": 110,
    "vrrp-primary-ip": "fe80:4888:a52:3105:528:23::",
  }
}

```

```
    "vrrp-secondary-ip": "2001:4888:a52:3105:528:23::"  
  },  
  {  
    "dns-vip": "2001:4888:a52:10:528:28a:0:10/128",  
    "next-hop-ip": "2001:4888:a52:3105:528:28a::",  
    "egress-svi": 2805,  
    "isbfd": "true",  
    "backup-nexthop": "2001:4888:a5f:2016:528:23:0:1",  
    "backup-egress-svi": 15  
  }  
}]  
}  
}  
}
```