salt-nornir

Release 0.10.0

Denis Mulyalin

May 15, 2022

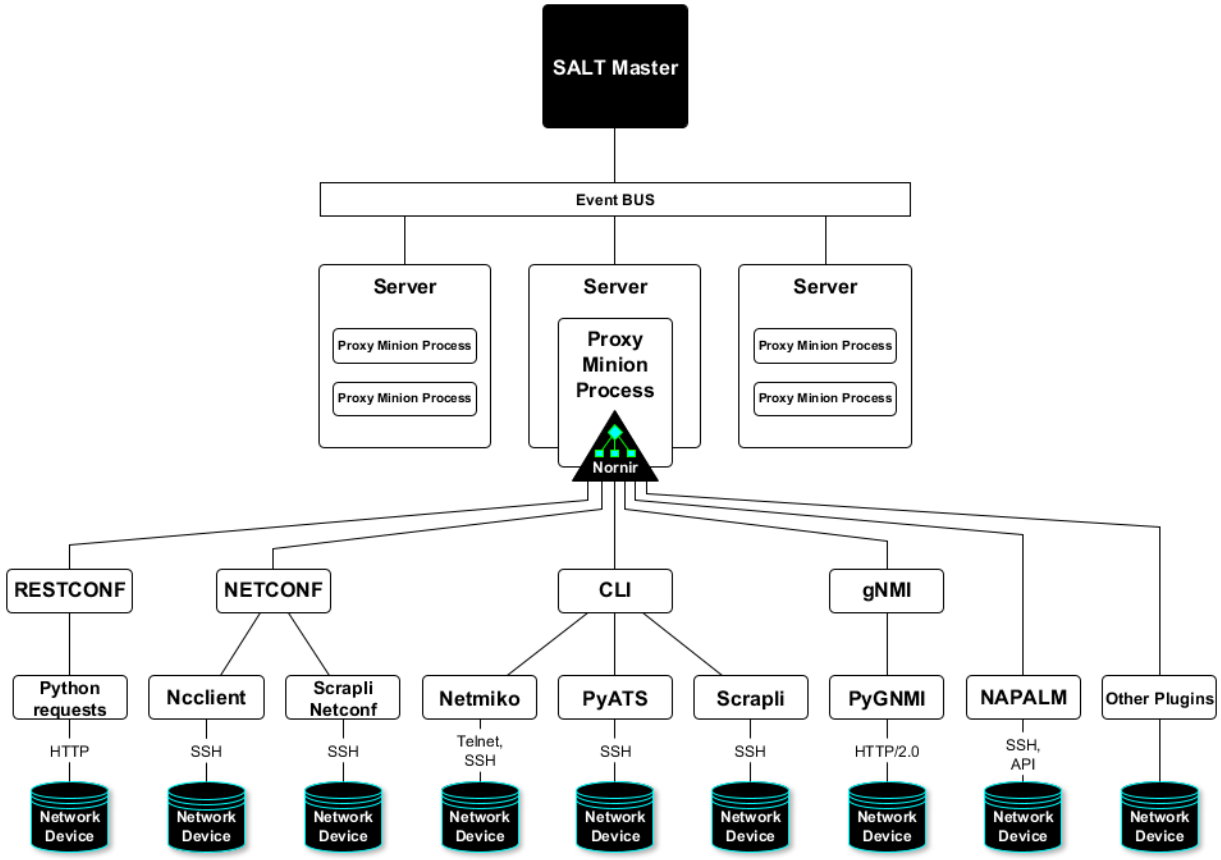
CONTENTS:

CHAPTER

ONE

OVERVIEW

Salt-Nornir is a SaltStack Proxy Minion built for Network Automation using Nornir framework.



1.1 Why Salt-Nornir?

SCALING

Salt-Nornir helps to address scaling issues of interacting with devices at high numbers, efficiently using resources without sacrificing execution speed.

Normally, for each network device dedicated proxy-minion process configured and once started, each consuming about 100 MByte of RAM.

Contrary, single instance of Salt-Nornir Nornir Proxy Minion capable of managing multiple network devices using Nornir.

As a result, the more devices single Salt-Nornir Proxy Minion manages, the more system resources saved.

At one extreme, single Salt-Nornir proxy can manage one device only, giving the fastest execution time but consuming the most of resources.

At another extreme, Salt-Nornir proxy minion can manage 1000 devices, slowly crawling them over time but saving on resource usage.

Optimal number of devices managed by Salt-Nornir proxy depends on the environment it operates in and decided on a case by case basis.

EXTENDABILITY

Nornir and SALT both use Python, both are pluggable frameworks and both open-source.

Given that both frameworks has Python API and native support for modules/plugins, extendability is endless.

Examples:

(1) Usually Proxy Minion locked in a certain way of interacting with network devices using single library of choice. Salt-Nornir, on the other hand, handles interactions using plugins.

As a result, same Salt-Nornir Proxy Minion can manage network devices using several libraries. Out of the box Salt-Nornir Proxy Minion comes with support for NAPALAM, Netmiko, Scrapli, Scrapli Netconf, Ncclient, PyGNMI, PyATS and requests libraries to interact with network infrastructure.

(2) SaltStack has CLI utilities, schedulers, returners, REST API, pillar, beacons, event bus, mine, templating engines and many other pluggable systems. All of them available for use with Salt-Nornir Proxy Minion.

(3) In case none of the existing plugins suite the use case, Nornir Tasks plugins together with SaltStack Python API systems can be used to address the problem.

1.2 How it fits together

SaltStack software, Nornir framework, nornir_salt and salt_nornir packages - how it fits together?

```
We can solve any problem by introducing an extra level of indirection.

    David J. Wheeler

...except for the problem of too many levels of indirection
```

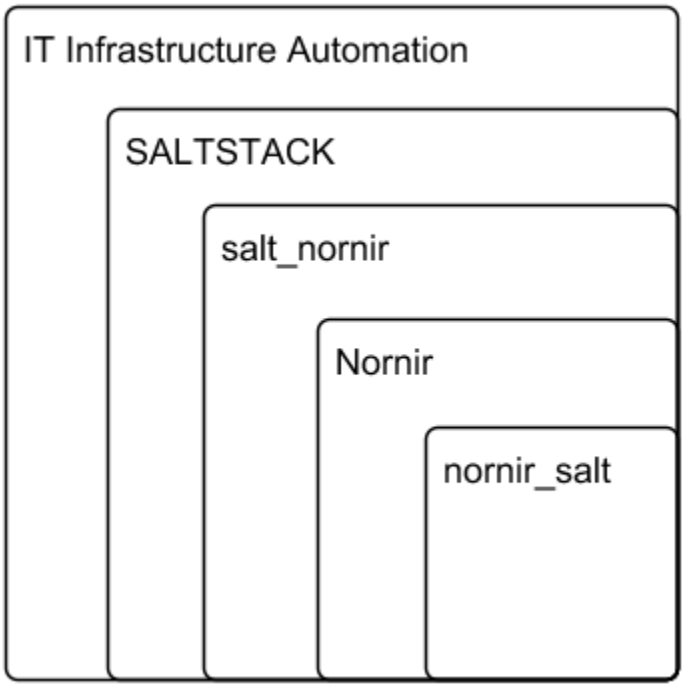
From more specific to more abstract:

nornir_salt - is a collection of Nornir plugins, born in the process of creating Nornir Proxy Minion

Nornir - pluggable automation framework to interact with network devices using pure Python

salt_nornir - collection of SaltStack modules that use Nornir to manage network devices

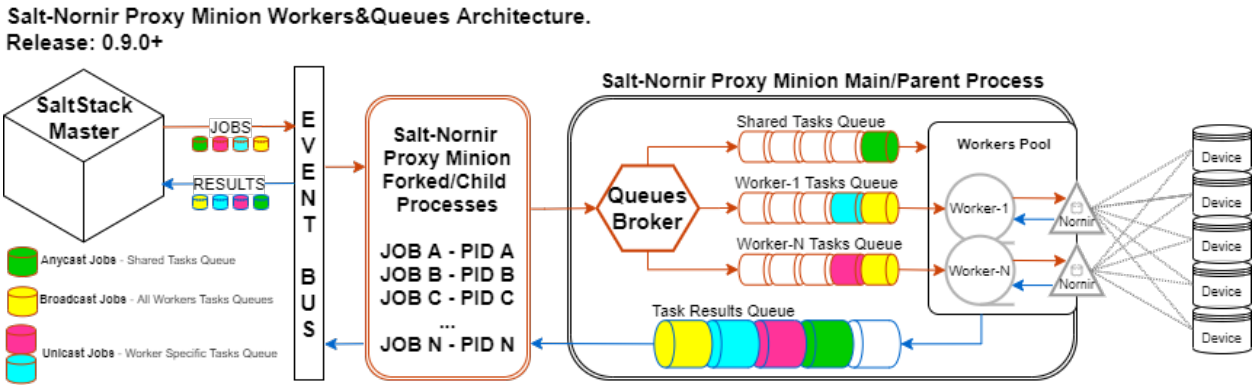
SaltStack - Python-based, open-source software for event-driven IT automation, remote task execution and configuration management (Wikipedia)



1.3 How it works

Wrapping Nornir in Salt Proxy Minion allows to run jobs against multiple devices. Single proxy process can apply configuration or retrieve devices' state using Nornir plugins.

Salt-Nornir Proxy Minion controls and enables shared access to network devices resources using queues for child and main processes communication.



Above architecture assumes:

- Double targeting required to narrow down tasks execution to a subset of hosts

- In addition to knowing how pillar works, one need to know how Nornir inventory structured, as Nornir inventory integrated with proxy-minion pillar

To address double targeting, Salt-Nornir comes with filtering functions, refer to Nornir-Salt FFun functions for details. Filter functions use `Fx` Salt CLI arguments to filter hosts.

For example:

```
# target only IOL1 and IOL2 hosts:
salt nrp1 nr.cli "show clock" FB="IOL[12]"
```

1.4 Nornir process watchdog or known issues

Slowly crashlooping system is usually preferable to a system that simply stops working.

To address various issues that can happen during lifespan of Nornir Proxy minion process each such a process has watchdog thread running. Watchdog constantly execute checks on a predefined intervals controlled by `watchdog_interval` parameter (default 30s).

Problems watchdog should be capable of handling:

1. **Memory overconsumption**. `memory_threshold_mbyte` and `memory_threshold_action` proxy minion settings can help to prevent proxy minion process from running out of memory. Normally, because Nornir Proxy minion uses multiprocessing to run tasks instead of threading it is not prone to memory leak issues, however, having capability to log or restart process in response to consuming too much memory can be helpful in extreme cases like bugs in new software releases.

2. **Stale child processes**. During Nornir proxy minion testing was detected that some child processes started to execute tasks might stuck for unknown reason. Probably bug of some sort. That usually leads to child process running indefinitely, consuming system resources and task never been completed. To mitigate that problem, watchdog runs lifespan detection for all child process by measuring their age, if age grows beyond `child_process_max_age` parameter (default 660s), watchdog kills such a process.

3. **Stale connections to devices**. Sometime connections to devices might become unusable. For instance device rebooted or network connectivity issue. Nornir plugins usually not capable of recovering from such a problems, as a result watchdog runs connection checks to confirm they are alive, clearing them otherwise.

3.1. **Connections keepalives**. Common connections liveness detection mechanism usually requires sending some data down the connection channel, receiving some data from device in response. Because of that, connections effectively kept alive, preventing them from timing out on device end due to inactivity.

4. **Running out of file descriptors (fd) problem**. On Unix systems each process can have limited number of file descriptors created, usually around 1000, because Nornir proxy minion uses multiprocessing queues for inter-process communications, effectively creating pipes on a lower level, each such a pipe consume file descriptor. But after child processes destroyed, not all file descriptors deleted, fd leaking, after reaching OS limit, prevents proxy minion process from running tasks. Watchdog on each run creates and destroys test pipes, restarting Nornir proxy minion process on failure to do so. Nornir proxy minion process restart leads to clearing of all previously created pipes and release of file descriptors. Future Nornir proxy releases might include a fix for this problem, but other reasons might lead to fd leaks, having mechanism in place to detect and recover from such a problem could be of great benefit regardless.

5. **Worker thread stops for some reason**. Some tasks might lead to worker thread exit on error, that wold stop execution of further submitted tasks. To solve that problem watchdog thread calls worker thread's `is_alive` method verify its status, restarting it if it stopped.

1.5 Working with Large number of devices

SaltStack is an asynchronous framework, that allows it to scale fairly well. However, for managing several thousands network devices certain details need to be taken into account.

Single salt-nornir proxy minion can manages several thousands of devices, things to take care of:

- SaltStack Event bus has a limit on amount of results data it can transmit for single job, if salt-nornir runs a task for say 1000 devices, it is likely that this limit will be exceeded. To address this one can adjust SaltStack event bus `max_event_size` limit in master's configuration, default is `1048576` bytes. Another approach could be to save task results to minion's local file system and retrieve then using SaltStack `cp` module or via out of band, salt-nornir execution module supports `dump` and `tf` arguments for that.
- OS ulimit FD (file descriptor) maximum count value should be increased, as each TCP connections salt-nornir establishes with devices consumes file descriptors, default FD limit usually is 1024.
- SSH requires encryption, encryption consumes CPU resources, having many SSH connections open off the same salt-nornir instance can become non practical. Two options exist, (1) set proxy minion setting `proxy_always_alive=False`, to close connection to device as soon as task execution completes or (2) adjust `connections_idle_timeout` value to suit your need to close connection to devices only after idle timeout expires.
- AAA Servers (TACACS, RADIUS) overload with requests from devices establishing connections with salt-nornir proxy minion, adjust RetryRunner `num_connectors` threads value to limit the number of simultaneous connections initializations and `num_workers` to limit the amount of devices task executed on simultaneously

CHAPTER

TWO

INSTALLATION

SaltStack Nornir Proxy Minion tested only for Linux, it is never tested with and unlikely will work on Windows. It is recommended to use Red Hat/CentOS or Ubuntu Linux distributions.

Python 2 is not supported, recommended version is Python 3.6 and higher.

Install from PyPi:

```
pip install salt-nornir[prodmin]
```

Or explicitly specifying Python version:

```
python3 -m pip install salt-nornir[prodmin]
```

Or install GIT and run installation of latest source code from GitHub master brunch:

```
python3 -m pip install git+https://github.com/dmulyalin/salt-nornir
```

Salt Nornir need to be installed on proxy minion machine. If planning to use runner module, `salt-nornir` should be installed on Salt Master machine as well.

For installation of SaltStack master and minion/proxy-minion modules refer to official documentation.

2.1 Installation extras

Salt-Nornir comes with these installation extras.

Table 1: Salt-Nornir extras packages

Name	Description
`dev`	Installs libraries required for development e.g. pytest, black, pre-commit etc.
`prodmin`	Production ready minimum set. Installs Netmiko, Ncclient and requests libraries to provide support for managing devices over SSH, NETCONF and RESTCONF. In addition, installs libraries to extended Salt-Nornir functionality such as Tabulate, Rich, TTP etc. All libraries have versions fixed to produce tested and working environment.
`prodmax`	Production ready maximum set. Installs all `prodmin` libraries together with additional modules required to support complete Salt-Nornir feature set such as PyGNMI, PyATS, Scrapli, NAPALM etc. All libraries have versions fixed to produce tested and working environment.

To install Salt-Nornir only, without any additional plugins:

```
pip install salt-nornir
```

To install minimum production set:

```
pip install salt-nornir[prodmin]
```

To install maximum production set:

```
pip install salt-nornir[prodmax]
```

2.2 SaltStack versions tested

Nornir Proxy minion was well tested and confirmed working with these versions of SaltStack:

- salt 3002
- salt 3003
- salt 3004

Other SaltStack versions might work too, but not tested.

2.3 Nornir Salt Dependency

Main dependency is nornir-salt package, it is must be of the same major and minor versions as `salt-nornir` package.

Compatible versions

salt-nornir	0.10.*	0.9.*	0.8.*	0.7.*	0.6.*	0.5.*	0.4.*	0.3.*
nornir-salt	0.10.*	0.9.*	0.8.*	0.7.*	0.6.*	0.5.*	0.4.*	0.3.*

2.4 Upgrade Procedure

Install updated packages:

```
python3 -m pip install nornir-salt --upgrade
python3 -m pip install salt-nornir --upgrade
```

Restart your proxy minions to pick up updated version.

Optionally run to verify software versions after Proxy Minions Started:

```
salt nrp1 nr.nornir version
```

2.5 Common installation issues

Issues mainly arise around installing all required dependencies. General rule of thumb - try Googling errors you getting or search StackOverflow.

1 `PyYAML` dependency - if getting error while doing `pip install salt-nornir`:

```
ERROR: Cannot uninstall 'PyYAML'. It is a distutils installed project and thus we cannot␣
↪accurately
determine which files belong to it which would lead to only a partial uninstall.
```

try:

```
python3 -m pip install salt-nornir --ignore-installed
```

2 `setuptools` dependency - if getting error while doing `pip install salt-nornir`:

```
ModuleNotFoundError: No module named 'setuptools_rust'
```

try:

```
python3 -m pip install -U pip setuptools
```

2.6 Using docker containers

Refer to Salt Nornir Docker Repository on how to setup SaltStack Master and Nornir Proxy Minion using docker containers.

CHAPTER

THREE

GETTING STARTED

3.1 Install SALTSTACK

For installation of SALTSTACK master and minion (proxy-minion is part of minion) modules refer to official documentation.

For example, CentOS7 Python3 latest SaltStack installation boils down to these commands:

- Import SaltStack repository key - `sudo rpm --import https://repo.saltproject.io/py3/redhat/7/x86_64/latest/SALTSTACK-GPG-KEY.pub`
- Add SaltStack repository - `curl -fsSL https://repo.saltproject.io/py3/redhat/7/x86_64/latest.repo | sudo tee /etc/yum.repos.d/salt.repo`
- Clear yum cache - `sudo yum clean expire-cache`
- Install master and minion - `sudo yum install salt-master salt-minion`
- Start salt-master - `sudo systemctl enable salt-master && sudo systemctl start salt-master`

3.2 Install Nornir

From PyPi:

```
python3 -m pip install salt_nornir[prodmax]
```

If it fails due to PyYAML incompatibility try running this command:

```
python3 -m pip install salt_nornir[prodmax] --ignore-installed PyYAML
```

3.3 Configure SALT Master

Master configuration file located on SALT Master machine - machine where you installed `salt-master` package.

Backup original master config file - `mv /etc/salt/master /etc/salt/master.old` and edit file `/etc/salt/master`:

```
interface: 0.0.0.0 # indicates IP address to listen/use for connections
master_id: lab_salt_master
pki_dir: /etc/salt/pki/master
timeout: 120

file_roots:
  base:
    - /etc/salt

pillar_roots:
  base:
    - /etc/salt/pillar
```

Create pillar directory if required - `mkdir /etc/salt/pillar/`.

3.4 Define Proxy Minion pillar inventory

Pillar files located on Salt Master machine.

Add Nornir Inventory and proxy settings to file `/etc/salt/pillar/nrp1.sls`:

```
proxy:
  proxytype: nornir

hosts:
  IOL1:
    hostname: 192.168.217.10
    platform: ios
    groups: [lab]
  IOL2:
    hostname: 192.168.217.7
    platform: ios
    groups: [lab]
```

(continues on next page)

(continued from previous page)

```
groups:
  lab:
    username: nornir
    password: nornir

defaults: {}
```

Create top file `/etc/salt/pillar/top.sls` and add Proxy Minion pillar to base environment:

```
base:
  nrp1:
    - nrp1
```

3.5 Start SALT Master

Start or restart salt-master process to pick up updated configurations:

```
systemctl restart salt-master
systemctl status salt-master
```

3.6 Configure Proxy Minion

Proxy Minion configuration files located on SALT Minion machine - machine where you installed `salt-minion` software. You only need to configure it once and all proxy-minion processes will use it.

Backup original proxy configuration file - `mv /etc/salt/proxy /etc/salt/proxy.old`.

Edit file `/etc/salt/proxy` to look like:

```
master: 192.168.1.1 # IP address or FQDN of master machine e.g. localhost or master.lab
multiprocessing: false # default, but overridden in Nornir proxy minion pillar
mine_enabled: true # not required, but nice to have
pki_dir: /etc/salt/pki/proxy # not required - this separates the proxy keys into a␣
↪different directory
log_level: debug # default is warning, adjust as required
```

Define Proxy Minion service in file `/etc/systemd/system/salt-proxy@.service`:

```
[Unit]
Description=Salt proxy minion
After=network.target

[Service]
Type=simple
ExecStart=/usr/bin/salt-proxy -l debug --proxyid=%i
User=root
Group=root
Restart=always
```

(continues on next page)

(continued from previous page)

```
RestartPreventExitStatus=SIGHUP
RestartSec=5

[Install]
WantedBy=default.target
```

Warning: beware that log level in above configuration set to `debug` that can log and expose sensitive data like device credentials and can consume significant amount of disk space over time.

3.7 Start Nornir Proxy Minion process

Run command to start Nornir Proxy Minion process:

```
systemctl start salt-proxy@nrp1.service
systemctl enable salt-proxy@nrp1.service
systemctl status salt-proxy@nrp1.service
```

Or run in debug mode:

```
salt-proxy --proxyid=nrp1 -l debug
```

To check proxy logs:

```
tail -f /var/log/salt/proxy
```

3.8 Accept Proxy Minion key on master

Run command on salt master machine to view pending keys:

```
[root@localhost /]# salt-key
Accepted Keys:
Denied Keys:
Unaccepted Keys:
nrp1
Rejected Keys:
```

Accept `nrp1` proxy minion key:

```
[root@localhost /]# salt-key -a nrp1
```

3.9 Start using Nornir Proxy Minion

Run commands to test nornir proxy minion operations:

```
salt nrp1 test.ping # verify that process is running
salt nrp1 nr.nornir stats # check statistics for Nornir proxy minion
salt nrp1 nr.nornir test # test task to verify module operation
salt nrp1 nr.nornir inventory # to check Nornir inventory content
salt nrp1 nr.task nr_test # test task to verify Nornir operation
```

Test connectivity to devices:

```
[root@localhost /]# salt nrp1 nr.tping
nrp1:
    ----------
    IOL1:
        ----------
        nornir_salt.plugins.tasks.tcp_ping:
            ----------
            22:
                True
    IOL2:
        ----------
        nornir_salt.plugins.tasks.tcp_ping:
            ----------
            22:
                True
```

Get show commands output from devices:

```
[root@localhost /]# salt nrp1 nr.cli "show clock"
nrp1:
    ----------
    IOL1:
        ----------
        show clock:
            *03:03:04.566 EET Sat Feb 13 2021
    IOL2:
        ----------
        show clock:
            *03:03:04.699 EET Sat Feb 13 2021
```

Check documentation for Nornir execution module `nr.cfg` function:

```
[root@salt-master /]# salt nrp1 sys.doc nr.cfg
nr.cfg:

    Function to push configuration to devices using ``napalm_configure`` or
    ``netmiko_send_config`` or Scrapli ``send_config`` task plugin.

    :param commands: (list) list of commands or multiline string to send to device
    :param filename: (str) path to file with configuration
    :param template_engine: (str) template engine to render configuration, default is␣
↪jinja
```

(continues on next page)

(continued from previous page)

```
    :param saltenv: (str) name of SALT environment
    :param context: Overrides default context variables passed to the template.
    :param defaults: Default context passed to the template.
    :param plugin: (str) name of configuration task plugin to use - ``napalm`` (default)␣
↪or ``netmiko`` or ``scrapli``
    :param dry_run: (bool) default False, controls whether to apply changes to device or␣
↪simulate them
    :param commit: (bool or dict) by default commit is ``True``. With ``netmiko`` plugin
        dictionary ``commit`` argument supplied to commit call using ``**commit``

    Warning: ``dry_run`` not supported by ``netmiko`` plugin

    Warning: ``commit`` not supported by ``scrapli`` plugin. To commit need to send␣
↪commit
        command as part of configuration, moreover, scrapli will not exit configuration␣
↪mode,
        need to send exit command as part of configuration mode as well.

    For configuration rendering purposes, in addition to normal `context variables
    <https://docs.saltstack.com/en/latest/ref/states/vars.html>`_
    template engine loaded with additional context variable `host`, to access Nornir host
    inventory data.

    Sample usage::

        salt nrp1 nr.cfg "logging host 1.1.1.1" "ntp server 1.1.1.2" FB="R[12]" dry_
↪run=True
        salt nrp1 nr.cfg commands='["logging host 1.1.1.1", "ntp server 1.1.1.2"]' FB=
↪"R[12]"
        salt nrp1 nr.cfg "logging host 1.1.1.1" "ntp server 1.1.1.2" plugin="netmiko"
        salt nrp1 nr.cfg filename=salt://template/template_cfg.j2 FB="R[12]"
        salt nrp1 nr.cfg filename=salt://template/cfg.j2 FB="XR-1" commit='{"confirm":␣
↪True}'

    Filename argument can be a template string, for instance::

        salt nrp1 nr.cfg filename=salt://templates/{{ host.name }}_cfg.txt

    In that case filename rendered to form path string, after that, path string used to␣
↪download file
    from master, downloaded file further rendered using specified template engine␣
↪(Jinja2 by default).
    That behavior supported only for filenames that start with ``salt://``. This feature␣
↪allows to
    specify per-host configuration files for applying to devices.

    Sample Python API usage from Salt-Master::

        import salt.client
        client = salt.client.LocalClient()

        task_result = client.cmd(
```

(continues on next page)

(continued from previous page)

```
            tgt="nrp1",
            fun="nr.cfg",
            arg=["logging host 1.1.1.1", "ntp server 1.1.1.2"],
            kwarg={"plugin": "netmiko"},
        )
```

Configure syslog server using `nr.cfg` with Netmiko:

```
[root@localhost /]# salt nrp1 nr.cfg "logging host 1.1.1.1" "logging host 1.1.1.2"␣
↪plugin=netmiko
nrp1:
    ----------
    IOL1:
        ----------
        netmiko_send_config:
            ----------
            changed:
                True
            diff:
            exception:
                None
            failed:
                False
            result:
                configure terminal
                Enter configuration commands, one per line.  End with CNTL/Z.
                IOL1(config)#logging host 1.1.1.1
                IOL1(config)#logging host 1.1.1.2
                IOL1(config)#end
                IOL1#
    IOL2:
        ----------
        netmiko_send_config:
            ----------
            changed:
                True
            diff:
            exception:
                None
            failed:
                False
            result:
                configure terminal
                Enter configuration commands, one per line.  End with CNTL/Z.
                IOL2(config)#logging host 1.1.1.1
                IOL2(config)#logging host 1.1.1.2
                IOL2(config)#end
                IOL2#
```

3.10 Additional Resources

Reference *Use Cases* section for more information on how to use Nornir Proxy Minion.

SALTSTACK official documentation

Collection of useful SALTSTACK resource awesome-saltstack

CHAPTER

FOUR

NORNIR PROXY MODULE

Nornir Proxy Module is a core component of Nornir Proxy Minion. However, users rarely have to interact with it directly unless they writing their own Execution or Runner or State or whatever modules for SaltStack.

What is important to understand are configuration parameters you can use with Nornir Proxy Minion, as they can help to alter default behavior or control various aspects of Nornir Proxy Minion life cycle.

4.1 Introduction

Single Nornir proxy-minion can work with hundreds of devices as opposed to conventional proxy-minion that normally dedicated to managing one device/system only.

As a result, Nornir proxy-minion requires less resources to run tasks against same number of devices. During idle state only one proxy minion process is active, that significantly reduces amount of memory required to manage device endpoints.

Proxy-module required way of operating is `multiprocessing` set to `True` - default value, that way each task executed in dedicated process.

4.2 Dependencies

Nornir 3.x uses modular approach for plugins. As a result required plugins need to be installed separately from Nornir Core library. Main collection of plugins to install is nornir-salt. Nornir Salt repository contains many function used by Salt Nornir Proxy Minion module and is mandatory to have on the system where proxy minion process runs.

4.3 Nornir Proxy Configuration Parameters

Below parameters can be specified in Proxy Minion Pillar.

- `proxytype` - string of value `nornir`
- `multiprocessing` - boolean, `True` by default, multiprocessing is a recommended way to run this proxy, threading mode also works, but might be prone to memory consumption issues
- `process_count_max` - int, default is `-1` no limit, maximum number of processes to use to limit a number of tasks waiting to execute
- `nornir_filter_required` - boolean, default is `False`, to indicate if Nornir filter is mandatory for tasks executed by this proxy-minion. Nornir has access to multiple devices, by default, if no filter provided, task will run for all devices, `nornir_filter_required` allows to change behavior to opposite, if no filter provided, task

will not run at all. It is a safety measure against running task for all devices accidentally, instead, filter `FB="*"` can be used to run task for all devices.

- `runner` - dictionary, Nornir Runner plugin parameters, default is RetryRunner
- `inventory` - dictionary, Nornir Inventory plugin parameters, default is DictInventory populated with data from proxy-minion pillar, pillar data ignored by any other inventory plugins
- `child_process_max_age` - int, default is 660s, seconds to wait before forcefully kill child process
- `watchdog_interval` - int, default is 30s, interval in seconds between watchdog runs
- `proxy_always_alive` - boolean, default is True, keep connections with devices alive or tear them down immediately after each job
- `connections_idle_timeout` - int, seconds, default is 1 equivalent to `proxy_always_alive` set to True, if value equals 0 renders same behavior as if `proxy_always_alive` is False, if value above 1 - all host's device connections torn down after it was not in use for longer then idle timeout value even if `proxy_always_alive` set to True
- `job_wait_timeout` - int, default is 600s, seconds to wait for job return until give up
- `memory_threshold_mbyte` - int, default is 300, value in MBytes above each to trigger `memory_threshold_action`
- `memory_threshold_action` - str, default is `log`, action to implement if `memory_threshold_mbyte` exceeded, possible actions: `log` - send syslog message, `restart` - shuts down proxy minion process.
- `nornir_workers` - number of Nornir instances to create, each instance has worker thread associated with it allowing to run multiple tasks against hosts, as each worker dequeue tasks from jobs queue, default is 3
- `files_base_path` - str, default is `/var/salt-nornir/{proxy_id}/files/`, OS path to folder where to save files on a per-host basis using *ToFileProcessor <https://nornir-salt.readthedocs.io/en/latest/Processors/ToFileProcessor.html>_*,
- `files_max_count` - int, default is 5, maximum number of file version for `tf` argument used by ToFileProcessor
- `nr_cli` - dictionary of default arguments to use with `nr.cli` execution module function, default is none
- `nr_cfg` - dictionary of default arguments to use with `nr.cfg` execution module function, default is none
- `nr_nc` - dictionary of default arguments to use with `nr.nc` execution module function, default is none
- `event_progress_all` - boolean, default is False, if True emits progress events for all tasks using *SaltEventProcessor <https://nornir-salt.readthedocs.io/en/latest/Processors/SaltEventProcessor.html>_*, per-task `event_progress` argument overrides `event_progress_all` parameter.

Nornir uses inventory to store information about devices to interact with. Inventory can contain information about `hosts`, `groups` and `defaults`. Nornir inventory defined in proxy-minion pillar.

Nornir proxy-minion pillar example:

```
proxy:
  proxytype: nornir
  process_count_max: 3
  multiprocessing: True
  nornir_filter_required: True
  proxy_always_alive: True
  connections_idle_timeout: 1
  watchdog_interval: 30
  child_process_max_age: 660
  job_wait_timeout: 600
```

(continues on next page)

(continued from previous page)

```
  memory_threshold_mbyte: 300
  memory_threshold_action: log
  files_base_path: "/var/salt-nornir/{proxy_id}/files/"
  files_max_count: 5
  event_progress_all: True
  nr_cli: {}
  nr_cfg: {}
  nr_nc: {}
  runner:
     plugin: RetryRunner
     options:
        num_workers: 100
        num_connectors: 10
        connect_retry: 3
        connect_backoff: 1000
        connect_splay: 100
        task_retry: 3
        task_backoff: 1000
        task_splay: 100
        reconnect_on_fail: True
        task_timeout: 600
        creds_retry: ["local_creds"]
  inventory:
     plugin: DictInventory

hosts:
  IOL1:
    hostname: 192.168.217.10
    platform: ios
    location: B1
    groups: [lab]
  IOL2:
    hostname: 192.168.217.7
    platform: ios
    location: B2
    groups: [lab]

groups:
  lab:
    username: nornir
    password: nornir
    connection_options:
      napalm:
        optional_args: {dest_file_system: "system:"}

defaults:
  data:
    credentials:
      local_creds:
        username: admin
        password: admin
```

To use other type of inventory or runner plugins define their settings in pillar configuration:

```
proxy:
  runner:
     plugin: threaded
     options:
         num_workers: 100
  inventory:
     plugin: SimpleInventory
     options:
       host_file: "/var/salt-nonir/proxy-id-1/hosts.yaml"
       group_file: "/var/salt-nonir/proxy-id-1/groups.yaml"
       defaults_file: "/var/salt-nonir/proxy-id-1/defaults.yaml"
```

4.4 Nornir runners

Nornir Runner plugins define how to run tasks against hosts. If no `runner` dictionary provided in proxy-minion pillar, Nornir initialized using Nornir Salt RetryRunner plugin with these default settings:

```
runner = {
    "plugin": "RetryRunner",
    "options": {
        "num_workers": 100,
        "num_connectors": 10,
        "connect_retry": 3,
        "connect_backoff": 1000,
        "connect_splay": 100,
        "task_retry": 3,
        "task_backoff": 1000,
        "task_splay": 100,
        "reconnect_on_fail": True,
        "task_timeout": 600,
        "creds_retry": [{"username": "admin", "password": "admin", "port": 2022}]
    },
}
```

4.5 Nornir Proxy Module Functions

Table 1: Common CLI Arguments Summary

Name	Description
refresh_nornir	Function to re-instantiate Nornir object instance refreshing pillar
execute_job	Function to place job in worker thread jobs queue
grains	Retrieve Nornir Proxy Minion grains
grains_refresh	Refresh Nornir Proxy Minion grains
init	Initiate Nornir Proxy-module
kill_nornir	Un-gracefully shutdown Nornir Proxy Minion process
list_hosts	Produces a list of hosts' names managed by this Proxy
nr_data	To retrieve values from `nornir_data` Nornir Proxy Minion dictionary
ping	To test Nornir Proxy Minion process
queues_utils	Utility function to manage Proxy Minion queues
run	Used to run Nornir Task
shutdown	Gracefully shutdown Nornir Instance
stats	Produces a dictionary of Nornir Proxy Minion statistics
workers_utils	Utility function to manage Proxy Minion workers

4.5.1 refresh_nornir

salt_nornir.proxy.nornir_proxy_module.**_refresh_nornir**(**args*, ***kwargs*)

4.5.2 execute_job

salt_nornir.proxy.nornir_proxy_module.**execute_job**(*task_fun*, *kwargs*, *identity*)

Function to submit job request to Nornir Proxy minion jobs queue, wait for job to be completed and return results.

Parameters

- **task_fun** – (str) name of nornir task function/plugin to import and run
- **kwargs** – (dict) any arguments to submit to Nornir task `**kwargs`
- **identity** – (dict) dictionary of uuid4, jid, function_name keys

`identity` parameter used to identify job results in results queue and must be unique for each submitted job.

4.5.3 grains

salt_nornir.proxy.nornir_proxy_module.**grains**()

Populate grains

4.5.4 grains_refresh

salt_nornir.proxy.nornir_proxy_module.**grains_refresh**()
Does nothing, returns empty dictionary

4.5.5 init

salt_nornir.proxy.nornir_proxy_module.**init**(*opts*, *loader=None*)
Initiate Nornir by calling InitNornir()

4.5.6 kill_nornir

salt_nornir.proxy.nornir_proxy_module.**kill_nornir**(**args*, ***kwargs*)
This function kills Nornir process and its child process as fast as possible.

> **Warning:** this function kills main Nornir process and does not recover it

4.5.7 list_hosts

salt_nornir.proxy.nornir_proxy_module.**list_hosts**(***kwargs*)
Return a list of hosts managed by this Nornir instance

Parameters **Fx** – filters to filter hosts

4.5.8 nr_data

salt_nornir.proxy.nornir_proxy_module.**nr_data**(*key*)
Helper function to return values from nornir_data dictionary, used by `nr.cli`, `nr.cfg` and `nr.nc` execution module functions to retrieve default kwargs values from respective proxy settings' attributes.

Parameters **key** – (str or list) if string return value for single key, if list return a dictionary keyed by items in given key list.

4.5.9 nr_version

salt_nornir.proxy.nornir_proxy_module.**nr_version**()
Function to return a report of installed packages and their versions, useful for troubleshooting dependencies.

4.5.10 ping

salt_nornir.proxy.nornir_proxy_module.**ping**()
Return Nornir proxy status

4.5.11 queues_utils

salt_nornir.proxy.nornir_proxy_module.**queues_utils**(*call*)
Function to retrieve operational data for job queues

Parameters **call** – (str) utility to invoke - `results_queue_dump`

Supported calls:

- **`results_queue_dump` - drain items from result queue and return their content,** put items copies back into the queue afterwards

4.5.12 run

salt_nornir.proxy.nornir_proxy_module.**run**(*task*, *loader*, *identity*, *name*, *nr*, *wkr_data*, ***kwargs*)
Function for worker Thread to run Nornir tasks.

Parameters

- **task** – (obj) callable task function
- **loader** – (obj) `__salt__.loader` object instance
- **identity** – (dict) Task results queue identity for SaltEventProcessor
- **kwargs** – (dict) passed to `task.run` after extracting CLI arguments
- **name** – (str) Nornir task name to run
- **nr** – (obj) Worker instance Nornir object

4.5.13 shutdown

salt_nornir.proxy.nornir_proxy_module.**shutdown**()
This function implements this protocol to perform Nornir graceful shutdown:

1. Signal worker and watchdog threads to stop
2. Close all connections to devices
3. Close jobs and results queues
4. Kill all child processes
5. Delete Nornir object

Proxy Minion process keeps running afterwards, but cannot do anything.

4.5.14 stats

salt_nornir.proxy.nornir_proxy_module.**stats**(**args*, ***kwargs*)

Function to gather and return stats about Nornir proxy process.

Parameters **stat** – name of stat to return, returns all by default

Returns dictionary with these parameters:

- proxy_minion_id - if of this proxy minion
- main_process_is_running - set to 0 if not running and to 1 otherwise
- main_process_start_time - time.time() function to indicate process start time in epoch
- main_process_start_date - time.ctime() function date to indicate process start time
- main_process_uptime_seconds - int, main proxy minion process uptime
- main_process_ram_usage_mbyte - int, RAM usage
- main_process_pid - main process ID i.e. PID
- main_process_host - hostname of machine where proxy minion process is running
- jobs_started - int, overall number of jobs started
- jobs_completed - int, overall number of jobs completed
- jobs_failed - int, overall number of jobs failed
- jobs_job_queue_size - int, size of jobs queue, indicating number of jobs waiting to start
- jobs_res_queue_size - int, size of results queue, indicating number of results waiting to be collected by child process
- tasks_completed - overall number of completed Nornir tasks (including subtasks)
- tasks_failed - overall number of failed Nornir tasks (including subtasks)
- hosts_count - int, number of hosts/devices managed by this proxy minion
- hosts_connections_active - int, overall number of connection active to devices
- hosts_tasks_failed - overall number of hosts that failed all tasks within single job
- timestamp - time.ctime() timestamp of stats function run
- watchdog_runs - int, overall number of watchdog thread runs
- watchdog_child_processes_killed - int, number of stale child processes killed by watchdog
- watchdog_dead_connections_cleaned - int, number of stale hosts' connections cleaned by watchdog
- child_processes_count - int, number of child processes currently running
- main_process_fd_count - int, number of file descriptors in use by main proxy minion process
- main_process_fd_limit - int, fd count limit imposed by Operating System for minion process

4.5.15 workers_utils

salt_nornir.proxy.nornir_proxy_module.**workers_utils**(*call*)

Function to retrieve operational data for Nornir Worker instances

Parameters **call** – (str) utility to invoke

Supported calls:

- stats - return worker statistics keyed by worker id with these parameters:
  - is_busy - boolean, indicates if worker doing the work
  - worker_jobs_completed - counter of completed jobs
  - worker_jobs_failed - counter of completely failed jobs
  - worker_connections - hosts' connections info
  - worker_jobs_queue - size of the worker specific jobs queue
  - worker_hosts_tasks_failed - counter of overall host failed tasks
  - worker_jobs_started - counter of started jobs

CHAPTER

FIVE

NORNIR EXECUTION MODULE

SaltStack Nornir Execution Module exposes functionally of Nornir Proxy Minion to work with devices and systems. Users can invoke Execution Modules functionality using SALT CLI or one of SALT API - Python, REST. Usually, execution modules are the modules that users work with a lot because they directly mapped to managed system functionality such as CLI or NETCONF server.

5.1 Introduction

Nornir Execution Module complements Nornir Proxy Minion Module to interact with devices over SSH, Telnet, NETCONF or any other methods supported by Nornir connection plugins.

Things to keep in mind:

- execution module functions executed on same machine where proxy-minion process runs
- `multiprocessing` set to `True` is recommended way of running Nornir proxy-minion
- with multiprocessing on, dedicated process starts for each job
- tasks executed one after another, but task execution against hosts happening in order controlled by logic of Nornir Runner in use, usually in parallel using threading.

5.1.1 Commands timeout

It is recommended to increase salt command timeout or use `--timeout=60` option to wait for minion return, as all together it might take more than 5 seconds for task to complete. Alternatively, use `--async` option and query results afterwards:

```
[root@localhost /]# salt nrp1 nr.cli "show clock" --async

Executed command with job ID: 20210211120453972915
[root@localhost /]# salt-run jobs.lookup_jid 20210211120453972915
nrp1:
    ----------
    IOL1:
        ----------
        show clock:
            *08:17:22.691 EET Sat Feb 13 2021
    IOL2:
        ----------
        show clock:
```

(continues on next page)

(continued from previous page)

```
           *08:17:22.632 EET Sat Feb 13 2021
[root@localhost /]#
```

5.1.2 AAA considerations

Quiet often AAA servers (Radius, Tacacs) might get overloaded with authentication and authorization requests coming from devices due to Nornir establishing connections with them, that effectively results in jobs failures.

To overcome that problem Nornir Proxy Module uses Nornir Salt RetryRunner plugin by default. `RetryRunner` developed to address aforementioned issue in addition to implementing retry logic.

5.1.3 Targeting Nornir Hosts

Nornir can manage many devices and uses it's own inventory, additional filtering `Fx` functions introduced in Nornir Salt library to narrow down tasks execution to certain hosts/devices.

Sample command to demonstrate targeting capabilities:

```
salt nrp1 nr.cli "show clock" FB="R*" FG="lab" FP="192.168.1.0/24" FO='{"role": "core"}'
```

5.1.4 Jumphosts or Bastions

`RetryRunner` included in Nornir Salt library supports `nr.cli` and `nr.cfg` with `plugin="netmiko"` and `nr.nc` with `plugin="ncclient"` functions to interact with devices behind SSH Jumphosts.

Sample Jumphost definition in host's inventory data of proxy-minion pillar:

```
hosts:
  LAB-R1:
    hostname: 192.168.1.10
    platform: ios
    password: user
    username: user
    data:
      jumphost:
        hostname: 172.16.0.10
        port: 22
        password: admin
        username: admin
```

`RetryRunner` on first task execution will initiate single connection to Jumphost, and will use it to proxy connections to actual devices.

5.2 Common CLI Arguments

A number of Command Line Interface arguments can be supplied to Nornir Proxy Module Execution Module functions to influence various aspects of task execution process.

Some of the command line options use Nornir Processor plugins. This plugins tap into task execution flow to perform additional actions or process task results.

To invoke processor plugin need to supply execution module functions with processors arguments providing required parameters to control processor plugin behavior.

All supported processors executed in this order:

```
event_progress ->
-> DataProcessor ->
-> iplkp ->
-> xml_flake ->
-> xpath ->
-> jmespath ->
-> match ->
-> run_ttp ->
-> ntfsm ->
-> TestsProcessor ->
-> DiffProcessor ->
-> ToFileProcessor
```

Table 1: Common CLI Arguments Summary

Name	Description
Fx	Filters to target subset of devices using FFun Nornir-Salt function
context	Overrides context variables passed by *render* to `file.apply_template_on_contents` exec mod function
dcache	Saves full task execution results to Nornir in-memory (RAM) Inventory `defaults` data
defaults	Default template context passed by *render* to `file.apply_template_on_contents` exec mod function
diff	Calls Nornir-Salt DiffProcessor to produce results difference
dp	Allows to call any function supported by Nornir-Salt DataProcessor
download	Renders arguments content using Salt cp module
dump	Saves complete task results to local file system using Nornir-Salt DumpResults function
event_failed	If True, emit events on Salt Events Bus for failed tasks
event_progress	If True, emit events on Salt Events Bus for tasks execution progress
hcache	Saves host's task execution results to host's in-memory (RAM) Inventory data
iplkp	Performs in CSV file or DNS lookup of IPv4 and IPv6 addresses to replace them in output
jmespath	uses JMESPath library to run query against structured results data
match	Filters text output using Nornir-Salt DataProcessor match function
ntfsm	Parse nr.cli output using TextFSM ntc-templates
RetryRunner parameters	Task parameters to influence RetryRunner execution logic
render	Renders arguments content using Salt renderer system
run_ttp	Calls Nornir-Salt DataProcessor run_ttp function to parse results using TTP
saltenv	Salt Environment name to use with *render* and *download* to source files, default is `base`
table	Formats results to text table using Nornir-Salt TabulateFormatter
template_engine	Template Engine name to use with *render* to render files, default is `jinja`
tests	Run tests for task results using Nornir-Salt TestsProcessor
tf	Saves results to local file system using Nornir-Salt ToFileProcessor
to_dict	Transforms results to structured data using Nornir-Salt ResultSerializer
xml_flake	Uses Nornir-Salt DataProcessor `xml_flake` function to filter XML output
xpath	Uses Nornir-Salt DataProcessor `xpath` function to filter XML output
worker	Worker to use for task, supported values `all` or number from 1 to `nornir_workers` Proxy Minion parameter of default value 3

5.2.1 Fx

Uses Nornir-Salt `FFun` function to form a subset of hosts to run this task for.

Supported functions: `nr.task`, `nr.cli`, `nr.cfg`, `nr.cfg_gen`, `nr.test`, `nr.nc`, `nr.do`, `nr.http`, `nr.tping`, `nr.inventory`

CLI Arguments:

- `Fx` - any of Nornir Salt FFun function filters

Sample usage:

```
salt nrp1 nr.cli "show clock" FB="R*" FG="lab" FP="192.168.1.0/24" FO='{"role": "core"}'
```

5.2.2 dcache

Saves full task execution results to Nornir `defaults` in-memory (RAM) inventory data. Saved information non-persistent across Proxy Minion reboots.

Primary usecase is to share task results between tasks for rendering, targeting or processing.

Supported functions: `nr.task`, `nr.cli`, `nr.cfg`, `nr.cfg_gen`, `nr.nc`, `nr.do`, `nr.http`, `nr.tping`, `nr.gnmi`

CLI Arguments:

- `dcache` - nornir inventory `defaults` data dictionary key name to save results under or if set to boolean True, uses `dcache` as a key name

Sample usage:

```
salt nrp1 nr.cli "show clock" dcache="show_clock_output"
salt nrp1 nr.cli "show clock" dcache=True
```

To view in-memory `defaults` inventory can use utility function:

```
salt npr1 nr.nornir inventory
```

To clean up cached data can either restart Proxy Minion or use utility function:

```
salt npr1 nr.nornir clear_dcache cache_keys='["key1", "key2"]'
```

5.2.3 diff

Uses Nornir Salt DiffProcessor to produce difference between current task results and previous results saved by `ToFileProcessor`.

Supported functions: `nr.task`, `nr.cli`, `nr.nc`, `nr.do`, `nr.http`, `nr.gnmi`

CLI Arguments:

- `diff` - `ToFileProcessor` file group name to run difference with
- `last` - `ToFileProcessor` file version number, default is 1

Sample usage:

```
salt nrp1 nr.cli "show ip route" diff="show_route" last=1
```

5.2.4 dp

Uses Nornir-Salt DataProcessor plugin designed to help with processing Nornir task results.

Supported functions: `nr.task`, `nr.cli`, `nr.cfg`, `nr.cfg_gen`, `nr.nc`, `nr.do`, `nr.http`, `nr.gnmi`

CLI Arguments:

- `dp` - data processor functions list to process task results

CLI argument `dp` can be comma-separated string or list of `DataProcessor` function names or dictionary keyed by `DataProcessor` function name with values set to dictionary which contains arguments for `DataProcessor` function.

Sample usage:

```
salt nrp1 nr.nc get_config dp="xml_to_json"
salt nrp1 nr.nc get_config dp="load_xml, flatten"
salt nrp1 nr.nc get_config dp='["load_xml", "flatten"]'
salt nrp1 nr.cli "show version" dp='[{"fun": "match", "pattern": "Version"}]'
salt nrp1 nr.nc get_config source=running dp='[{"fun": "xml_flatten"}, {"fun": "key_
↪filter", "pattern": "*bgp*, *BGP*"}]'
```

Last example will call `xml_flatten` function first following with `key_filter` with `{"pattern": "*bgp*, *BGP*"}` dictionary arguments.

5.2.5 download

SaltStack has cp module, allowing to download files from Salt Master, `donwload` keyword can be used to indicate arguments that should download content for.

Supported URL schemes are: salt://, http://, https://, ftp://, s3://, swift:// and file:// (local filesystem).

Keys listed in `download` argument ignored by *render* argument even if same key contained with `render` argument. Arguments names listed in `donwload` are not rendered, only loaded from Salt Master.

Supported functions: `nr.task, nr.cli, nr.cfg, nr.cfg_gen, nr.nc, nr.do, nr.http`

CLI Arguments:

- `download` - list of arguments to download content for, default is `["run_ttp", "iplkp"]`

For example, to render content for filename argument:

```
salt nrp1 nr.cfg filename="salt://templates/logging_config.txt" download='["filename"]'
```

Primary use cases for this keyword is revolving around enabling or disabling downloading and rendering for certain arguments. Execution Module Functions adjust `download` keyword list content by themselves and usually do not require manual modifications.

5.2.6 dump

Salt Event bus has limit on the amount of data it can transfer from Proxy Minion to Master, because of that, results produced by Proxy minion might get trimmed beyond certain threshold.

This can be addressed in several ways:

- increase event bus data transmission threshold
- use returner to return results to external database or other system

In addition to above option, Nornir Proxy Minion can make use of Nornir Salt DumpResults function to save complete results of task execution to local file system. That data can be later retrieved from proxy Minion machine.

Another usecase that `DumpResults` function can help to solve is results logging for audit, review or historical data purposes.

Supported functions: `nr.task, nr.cli, nr.cfg, nr.cfg_gen, nr.test, nr.nc, nr.do, nr.http`

CLI Arguments:

- `dump` - `ToFileProcessor` file group name where to save results

Sample usage:

```
salt nrp1 nr.cli "show run" dump="show_run_output"
```

Results saved to proxy minion local file system under `files_base_path`, default is:

```
/var/salt-nornir/{proxy_id}/files/{filegroup}__{timestamp}__{rand}__{proxy_id}.txt
```

Where:

- `proxy_id` - Nornir Proxy Minion ID
- `filegroup` - `ToFileProcessor` file group name where to save results
- `timestamp` - date timestamp
- `rand` - random integer between 1 and 1000

5.2.7 event_failed

Salt Event bus allows Proxy Minion processes to emit events, so that salt-master reactor system can act upon them and trigger execution of various actions.

`event_failed` CLI argument instructs Nornir Proxy minion to emit events for failed tasks.

Event's tag formed using this formatter:

```
nornir-proxy/{proxy_id}/{host}/task/failed/{name}
```

Where:

- `proxy_id` - Nornir Proxy Minion ID
- `host` - hostname of device that failed this task
- `name` - name of the failed task

Event body contains task execution results dictionary.

Failed tasks determined using results `failed` or `success` attributes, if `failed` is True or `success` is False task considered as failed.

Combining `event_failed` with `nr.test` function allows to implement event driven automation in response to certain tests failure. Each test translated to a separate task result and `event_failed` emit events on a per-test basis enabling to construct very granular react actions on Salt Master.

Sample event content:

```
nornir-proxy/nrp1/ceos1/task/failed/nornir_salt.plugins.tasks.nr_test   {
    "_stamp": "2022-02-11T11:14:16.081755",
    "cmd": "_minion_event",
    "data": {
        "changed": false,
        "connection_retry": 3,
        "diff": "",
        "exception": "",
        "failed": true,
        "host": "ceos1",
        "name": "nornir_salt.plugins.tasks.nr_test",
        "result": "Traceback (most recent call last):\n  File "/usr/local/lib/python3.6/
↪site-packages/nornir/core/task.py", line 99, in start\n    r = self.task(self, **self.
↪params)\n  File "/usr/local/lib/python3.6/site-packages/nornir_salt/plugins/tasks/nr_
↪test.py", line 65, in nr_test\n    raise RuntimeError(excpt_msg)\nRuntimeError\n",
```

(continues on next page)

(continued from previous page)

```
        "task_retry": 3
    },
    "id": "nrp1",
    "pretag": null,
    "tag": "nornir-proxy/nrp1/ceos1/task/failed/nornir_salt.plugins.tasks.nr_test"
}
```

Supported functions: `nr.task`, `nr.cli`, `nr.cfg`, `nr.cfg_gen`, `nr.test`, `nr.nc`, `nr.do`, `nr.http`, `nr.gnmi`, `nr.file`, `nr.diff`, `nr.find`, `nr.learn`

CLI Arguments:

- `event_failed` - boolean, default is False, if True will emit events for failed tasks.

Sample usage:

```
salt nrp1 nr.test suite="salt://tests/suite.txt" event_failed=True
```

5.2.8 event_progress

Argument to use Nornir-Salt `SaltEventProcessor` plugin to emit task execution progress events to SaltStack Events Bus. This is mainly useful for tracking tasks' flow, debugging and general assurance.

For example, `event_progress` used by `nr.call` Runner Module function to capture and print messages to terminal informing user about task execution progress.

Supported functions: `nr.task`, `nr.cli`, `nr.cfg`, `nr.cfg_gen`, `nr.test`, `nr.nc`, `nr.do`, `nr.http`, `nr.gnmi`, `nr.file`, `nr.diff`, `nr.find`, `nr.learn`

CLI Arguments:

- `event_progress` - boolean, default is False, if True will emit events for tasks progress.

Sample usage:

```
salt nrp1 nr.cli "show clock" event_progress=True
```

To listen to events generated by Proxy Minion when `event_progress=True` can open additional session to master server and run `salt-run nr.event` runner function.

Nornir Proxy Minion pillar parameter `event_progress_all` can be used to control default behavior, `event_progress` overrides `event_progress_all` parameter.

5.2.9 hcache

Saves individual host's task execution results in host's in-memory (RAM) inventory data. Saved information non-persistent across Proxy Minion reboots.

Primary usecase is to share task results data between tasks for rendering, targeting or processing.

Supported functions: `nr.task`, `nr.cli`, `nr.cfg`, `nr.cfg_gen`, `nr.nc`, `nr.do`, `nr.http`, `nr.tping`, `nr.gnmi`

CLI Arguments:

- `hcache` - host's data dictionary key name to save results under or if set to boolean True, uses `hcache` as a key name

Sample usage:

```
salt nrp1 nr.cli "show clock" hcache="show_clock_output"
salt nrp1 nr.cli "show clock" hcache=True
```

To view in-memory inventory can use utility function:

```
salt npr1 nr.nornir inventory FB="hosname-1"
```

To clean up cached data can either restart Proxy Minion or use utility function:

```
salt npr1 nr.nornir clear_hcache FB="hosname-1"
salt npr1 nr.nornir clear_hcache cache_keys='["key1", "key2"]'
```

5.2.10 iplkp

Uses Nornir-Salt `DataProcessor` iplkp function function to lookup IPv4 and IPv6 addresses using DNS or CSV file and replace them in device output with lookup results.

Supported functions: `nr.cli`

CLI Arguments:

- `iplkp` - value can be `dns` to indicate that need to use DNS or reference to a CSV file on Salt Master in a format `salt://path/to/file.txt`

First column in CSV file must be IPv4 or IPv6 address, second column should contain replacement value.

`iplkp` uses this formatter to replace IP addresses in results: `{ip}({lookup})` - where `ip` is the original IP address string and `lookup` is the lookup result value.

Sample usage:

```
salt nrp1 nr.cli "show ip int brief" iplkp="salt://lookup/ip.txt"
salt nrp1 nr.cli "show ip int brief" iplkp="dns"
```

Where `salt://lookup/ip.txt` content is:

```
ip,hostname
10.0.1.4,ceos1:Eth1
10.0.1.5,ceos2:Eth1
```

And this would be the results produced:

```
nrp1:
    ----------
    ceos1:
        ----------
        show ip int brief:
                                                                                ␣
↪Address
            Interface       IP Address        Status       Protocol            MTU   ␣
↪Owner
            --------------- ----------------- ------------ -------------- ----------- ---
↪----
```

(continues on next page)

(continued from previous page)

```
            Ethernet1       10.0.1.4(ceos1:Eth1)/24       up            up                 ␣
↪ 1500
            Loopback1       1.1.1.1/24         up             up                 65535
```

`iplkp` replaced `10.0.1.4` with lookup results `10.0.1.4(ceos1:Eth1)` in device output.

5.2.11 jmespath

Uses Nornir-Salt `DataProcessor` jmespath function to run JMESPath query against structured data results or JSON string.

Supported functions: `nr.task, nr.cfg, nr.cfg_gen, nr.nc, nr.do, nr.http, nr.cli, nr.gnmi`

CLI Arguments:

- `jmespath` - JMESPath query expression string

Sample usage:

```
salt nrp1 nr.task nornir_napalm.plugins.tasks.napalm_get getters='["get_interfaces"]'␣
↪jmespath='interfaces'
```

5.2.12 match

Uses Nornir-Salt `DataProcessor` match function to filter text results using regular expression pattern.

Supported functions: `nr.task, nr.cli, nr.cfg, nr.cfg_gen, nr.nc, nr.do, nr.http`

CLI Arguments:

- `match` - regex pattern to search for
- `before` - integer indicating how many lines before match to include in results

Sample usage:

```
salt nrp1 nr.cli "show version" match="Version.*"
salt nrp1 nr.cli "show version" match="Version.*" before=1
```

5.2.13 ntfsm

Uses Nornir-Salt `DataProcessor` ntfsm function to parse show commands output using TextFSM ntc-templates.

Supported functions: `nr.cli`

CLI Arguments:

- `ntfsm` - bool, if True uses TextFSM ntc-templates to parse output

Sample usage:

```
salt nrp1 nr.cli "show version" ntfsm=True
salt nrp1 nr.cli "show version" dp=ntfsm
```

5.2.14 RetryRunner parameters

A number of Nornir-Salt Proxy Minion execution module functions support RetryRunner parameters to influence task execution logic.

Supported functions: `nr.task`, `nr.cli`, `nr.cfg`, `nr.cfg_gen`, `nr.nc`, `nr.do`, `nr.http`, `nr.gnmi`, `nr.test`

CLI Arguments:

- `run_connect_retry` - number of connection attempts
- `run_task_retry` - number of attempts to run task
- `run_creds_retry` - list of connection credentials and parameters to retry while connecting to device
- `run_num_workers` - number of threads for tasks execution
- `run_num_connectors` - number of threads for device connections

Sample usage - retry various connection parameters:

```
salt nrp1 nr.cfg filename="salt://templates/logging_config.txt" run_creds_retry='["local_
↪creds", "dev_creds"]'
```

Sample usage - disable task and connection retries:

```
salt nrp1 nr.cfg filename="salt://templates/logging_config.txt" run_connect_retry=0 run_
↪task_retry=0
```

Sample usage - run tasks sequentially on hosts one by one:

```
salt nrp1 nr.cfg filename="salt://templates/logging_config.txt" run_num_workers=1
```

Sample usage - set rate of device's connections to 5 per-second:

```
salt nrp1 nr.cfg filename="salt://templates/logging_config.txt" run_num_connectors=5
```

5.2.15 render

SaltStack has renderers system, that system allows to render text files content while having access to all Salt Execution Module Functions and inventory data.

If render argument value points to one of supported URL schemes are: salt://, http://, https://, ftp://, s3://, swift:// and file:// (local filesystem). File content downloaded from specified URL prior to rendering.

Supported functions: `nr.task`, `nr.cli`, `nr.cfg`, `nr.cfg_gen`, `nr.nc`, `nr.do`, `nr.http`, `nr.gnmi`

CLI Arguments:

- `render` - list of argument to render content for, default is `["config", "data", "filter", "filter_", "filters", "filename"]`

For example, to render content for filename argument:

```
salt nrp1 nr.cfg filename="salt://templates/logging_config.txt" render='["filename"]'
```

Primary use cases for this keyword is revolving around enabling or disabling rendering for certain arguments. Execution Module Functions adjust `render` keyword list content by themselves and usually do not require any modifications.

5.2.16 run_ttp

Uses Nornir-Salt `DataProcessor` run_ttp function to parse text results using TTP library and produce structured data.

Supported functions: `nr.task, nr.cli, nr.do`

CLI Arguments:

- `run_ttp` - TTP template reference
- `ttp_structure` - TTP results structure, supported values: `flat_list` (default), `list` or `dictionary`

Sample usage:

```
salt nrp1 nr.cli "show version" run_ttp="Version: {{ version }}"
salt nrp1 nr.cli "show version" run_ttp="salt://ttp/parse_version.txt"
salt nrp1 nr.cli "show ip arp" run_ttp="ttp://platform/cisco_ios_show_ip_arp.txt"
salt nrp1 nr.cli run_ttp="salt://ttp/parse_commands.txt" ttp_structure=list
```

TTP templates can be specified inline, sourced from salt-master using `salt://path` or from TTP Templates collection repository using `ttp://path` providing that it is installed on proxy minion machine.

`run_ttp` with `nr.cli` function also supports sourcing commands to collect from devices from within TTP template input tags using `commands` argument. For example:

```
<input name="version">
commands = ["show version"]
</input>

<input name="interfaces">
commands = ["show run"]
</input>

<group name="facts" input="version">
cEOS tools version: {{ tools_version }}
Kernel version: {{ kernel_version }}
Total memory: {{ total_memory}} kB
Free memory: {{ total_memory}} kB
</group>

<group name="interfaces" input="interfaces">
interface {{ interface }}
   description {{ description | re(".*") }}
   ip address {{ ip }}/{{ mask }}
</group>
```

Supplying above template to `nr.cli` function with `run_ttp` argument will result in running `show version` and `show run` commands, placing output in appropriate inputs and parsing it with dedicated groups, returning parsing results.

5.2.17 table

Uses Nornir Salt TabulateFormatter function to transform task results in a text table representation.

Supported functions: `nr.task, nr.cli, nr.cfg, nr.cfg_gen, nr.test, nr.nc, nr.do`

CLI Arguments:

- `table` - boolean or table type indicator, supported values: True, `brief`, `extend`
- `headers` - list of table headers to form table for
- `headers_exclude` - list of table headers to exclude from final table results
- `sortby` - name of the header to sort table by, default is `host`
- `reverse` - if True, sorts table in reverse order, False by default

Sample usage:

```
salt nrp1 nr.cli "show clock" table=brief
salt nrp1 nr.cli "show clock" table=True
salt nrp1 nr.cli "show clock" table=True headers="host, results"
salt nrp1 nr.cli "show clock" table=True headers="host, results" sortby="host"␣
↪reverse=True
```

5.2.18 tests

Uses Nornir Salt TestsProcessor plugin to test task results.

Tests can be specified inline as a list of lists or can reference tests suite file on salt-master using `salt://path` format.

Supported functions: `nr.task, nr.cli, nr.cfg, nr.cfg_gen, nr.nc, nr.do, nr.http`

CLI Arguments:

- `tests` - reference to a list of tests to run
- `failed_only` - boolean, default is False, to indicate if to return results for failed tests only
- `remove_tasks` - boolean, default is True, to indicate if need to remove tested task results

Sample usage:

```
salt nrp1 nr.cli "show version" "show clock" tests='[["show version", "contains", "5.2.9b
↪"], ["show clock", "contains", "Source: NTP"]]'
salt nrp1 nr.cli "show version" "show clock" tests="salt://tests/suite.txt"
```

5.2.19 tf

Uses Nornir Salt ToFileProcessor plugin to save task execution results to proxy minion local file system under `files_base_path`, default is `/var/salt-nornir/{proxy_id}/files/`

Supported functions: `nr.task, nr.cli, nr.cfg, nr.cfg_gen, nr.nc, nr.do, nr.http, nr.gnmi`

CLI Arguments:

- `tf` - `ToFileProcessor` file group name where to save results
- `tf_skip_failed` - boolean, default is False, if True, will not save results for failed tasks

Sample usage:

```
salt nrp1 nr.cfg "logging host 1.1.1.1" tf="logging_config"
salt nrp1 nr.cfg "logging host 1.1.1.1" tf="logging_config" tf_skip_failed=True
```

`tf_skip_failed` can be useful when only want to save results to file for non failed tasks. For example, if `RetryRunner` runs task and it fails on first attempt but succeed on second, it might not make sense to store failed task results, which is the case by default, `tf_skip_failed` help to alter that behavior.

5.2.20 to_dict

Uses Nornir Salt ResultSerializer function to transform task results in a structured data - dictionary or list.

This function used by default for all task results unless `TabulateFormatter` `table` argument provided.

Supported functions: `nr.task, nr.cli, nr.cfg, nr.cfg_gen, nr.nc, nr.do, nr.http, nr.gnmi`

CLI Arguments:

- `add_details` - boolean, default is False, if True will add task execution details to the results
- `to_dict` - boolean, default is True, if False will produce results list structure

Sample usage:

```
salt nrp1 nr.cli "show clock" add_detals=True
salt nrp1 nr.cli "show clock" add_detals=True to_dict=False
```

5.2.21 xml_flake

Uses Nornir-Salt `DataProcessor` xml_flake function to flatten XML results structure to dictionary and filter dictionary keys using glob pattern.

Supported functions: `nr.task, nr.nc, nr.do, nr.http`

CLI Arguments:

- `xml_flake` - glob pattern to filter keys

Sample usage:

```
salt nrp1 nr.nc get_config xml_flake="*bgp:config*"
```

5.2.22 xpath

Uses Nornir-Salt `DataProcessor` xpath function to run xpath query against XML results.

Supported functions: `nr.task, nr.nc, nr.do, nr.http`

CLI Arguments:

- `xpath` - LXML library supported xpath expression

Sample usage:

```
salt nrp1 nr.nc get_config xpath='//config/address[text()="1.1.1.11"]'
```

Beware that XML namespaces removed from XML results before running xpath on them. If this behavior is not desirable, need to use `dp` keyword instead with required arguments for `xpath` function including namespaces map dictionary.

`xpath` function processes results received from device and executed locally on the minion machine, if you need to filter results returned from device, for `nr.nc` function consider using filter arguments. The complication is that if, for example, you running `get_config` NETCONF operation, full device config retrieved from device and passed via `xpath` function on proxy minion, this could be processing intensive especially for big configurations combined with significant number of devices simultaneously returning results.

5.2.23 worker

Starting with Nornir 0.9.0 support added for several Nornir Instances with dedicated worker threads, allowing to greatly increase Proxy Minion task execution throughput. If no `worker` argument provided task can be executed by any of the workers.

Supported functions: `nr.task`, `nr.cli`, `nr.cfg`, `nr.cfg_gen`, `nr.test`, `nr.nc`, `nr.do`, `nr.http`, `nr.gnmi`, `nr.file`, `nr.diff`, `nr.find`, `nr.learn`, `nr.nornir`

CLI Arguments:

- `worker` - Worker to use for task, supported values `all` or number from 1 to `nornir_workers` Proxy Minion parameter of default value 3

Sample usage:

```
salt nrp1 nr.cli "show clock" worker=3
salt nrp1 nr.nornir connections worker=2
salt nrp1 nr.nornir disconnect worker=all
```

5.3 Execution Module Functions

Table to summarize functions available in Nornir Proxy Execution Module and their purpose.

nr.function	description	supported plugins
nr.cfg	Function to modify devices configuration over ssh or telnet connections	napalm (default), netmiko, scrapli
nr.cfg_gen	Function to generate devices configuration using SALT templating system with Nornir inventory, mainly for testing purposes	
nr.cli	Function for show commands output collection over ssh or telnet	netmiko (default), scrapli
nr.diff	To diff content of files or network with files saved by `ToFileProcessor`	
nr.do	Function to execute actions with a set of steps calling other execution functions. Allows to construct simple work flows.	
nr.file	Function to work with files saved by `ToFileProcessor` - read, delete, list etc.	
nr.find	To search for various information in files saved by `ToFileProcessor`	
nr.gnmi	Interact with devices using gNMI protocol	pygnmi
nr.http	To run HTTP requests against API endpoints	requests
nr.learn	This function is to save task results in files using `ToFileProcessor` and `nr.do` actions	
nr.nc	Function to work with devices using NETCONF	ncclient (default), scrapli_netconf
nr.nornir	Function to call Nornir Utility Functions	
nr.task	Function to run any Nornir task plugin	
nr.test	Function to test show commands output produced by nr.cli function	netmiko (default), scrapli
nr.tping	Function to run TCP ping to devices' hostnames	

5.3.1 nr.cfg

`salt_nornir.modules.nornir_proxy_execution_module.`**`cfg`**`(`*commands*, ***kwargs*`)`

Function to push configuration to devices using NAPALM, Netmiko, Scrapli or or PyATS task plugin.

Parameters

- **`commands`** – (list) list of commands or multiline string to send to device
- **`filename`** – (str) path to file with configuration or template
- **`template_engine`** – (str) template engine to render configuration, default is jinja2
- **`saltenv`** – (str) name of SALT environment
- **`context`** – Overrides default context variables passed to the template.
- **`defaults`** – Default context passed to the template.
- **`plugin`** – (str) name of configuration task plugin to use - `napalm` (default) or `netmiko` or `scrapli` or `pyats`
- **`dry_run`** – (bool) default False, controls whether to apply changes to device or simulate them
- **`commit`** – (bool or dict) by default commit is `True`. With `netmiko` plugin if `commit` argument is a dictionary it is supplied to commit call as arguments
- **`config`** – (str) configuration string or template to send to device

> **Warning:** `dry_run` not supported by `netmiko` and `pyats` plugins

Warning: `commit` not supported by `scrapli` and `pyats` plugins. To commit need to send commit command as part of configuration, moreover, scrapli will not exit configuration mode, need to send exit command as part of configuration commands as well.

For configuration rendering purposes, in addition to normal context variables template engine loaded with additional context variable *host*, to access Nornir host inventory data.

Sample usage:

```
salt nrp1 nr.cfg "logging host 1.1.1.1" "ntp server 1.1.1.2" FB="R[12]" dry_run=True
salt nrp1 nr.cfg commands='["logging host 1.1.1.1", "ntp server 1.1.1.2"]' FB="R[12]
↪"
salt nrp1 nr.cfg "logging host 1.1.1.1" "ntp server 1.1.1.2" plugin="netmiko"
salt nrp1 nr.cfg filename=salt://template/template_cfg.j2 FB="R[12]"
salt nrp1 nr.cfg filename=salt://template/cfg.j2 FB="XR-1" commit='{"confirm": True,
↪ "confirm_delay": 60}'
salt nrp1 nr.cfg config="snmp-server location {{ host.location }}"
salt nrp1 nr.cfg "snmp-server location {{ host.location }}"
```

Filename argument can be a template string, for instance:

```
salt nrp1 nr.cfg filename=salt://templates/{{ host.name }}_cfg.txt
```

In that case filename rendered to form path string, after that, path string used to download file from master, downloaded file further rendered using specified template engine (Jinja2 by default). That behavior supported for URL schemes: salt://, http://, https://, ftp://, s3://, swift:// and file:// (local filesystem). This feature allows to specify per-host configuration files for applying to devices.

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()

task_result = client.cmd(
    tgt="nrp1",
    fun="nr.cfg",
    arg=["logging host 1.1.1.1", "ntp server 1.1.1.2"],
    kwarg={"plugin": "netmiko"},
)
```

5.3.2 nr.cfg_gen

salt_nornir.modules.nornir_proxy_execution_module.**cfg_gen**(**commands*, ***kwargs*)

Function to render configuration from template file. No configuration pushed to devices.

This function can be useful to stage/test templates or to generate configuration without pushing it to devices.

Parameters

- **filename** – (str) path to template
- **template_engine** – (str) template engine to render configuration, default is jinja2
- **saltenv** – (str) name of SALT environment
- **context** – Overrides default context variables passed to the template.

- **defaults** – Default context passed to the template.
- **config** – (str) configuration string or template to send to device

For configuration rendering purposes, in addition to normal context variables template engine loaded with additional context variable *host*, to access Nornir host inventory data.

Sample usage:

```
salt nrp1 nr.cfg_gen filename=salt://templates/template.j2 FB="R[12]"
salt nrp1 nr.cfg_gen config="snmp-server location {{ host.location }}"
salt nrp1 nr.cfg_gen "snmp-server location {{ host.location }}"
```

Sample template.j2 content:

```
proxy data: {{ pillar.proxy }}
jumphost_data: {{ host["jumphost"] }}
hostname: {{ host.name }}
platform: {{ host.platform }}
```

Filename argument can be a template string, for instance:

```
salt nrp1 nr.cfg_gen filename="salt://template/{{ host.name }}_cfg.txt"
```

In that case filename rendered to form path string, after that, path string used to download file from master, downloaded file further rendered using specified template engine (Jinja2 by default). That behavior supported for URL schemes: salt://, http://, https://, ftp://, s3://, swift:// and file:// (local filesystem). This feature allows to specify per-host configuration files for applying to devices.

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()

task_result = client.cmd(
    tgt="nrp1",
    fun="nr.cfg_gen",
    kwarg={"filename": "salt://template/{{ host.name }}_cfg.txt"},
)
```

5.3.3 nr.cli

salt_nornir.modules.nornir_proxy_execution_module.**cli**(**args*, ***kwargs*)

Method to retrieve commands output from devices using `send_command` task plugin from either Netmiko or Scrapli library.

Parameters

- **args** – (list or str) list of cli commands as arguments
- **commands** – (list or str) list of cli commands or single command as key-word argument
- **filename** – (str) path to file with multiline commands string
- **plugin** – (str) name of send command task plugin to use - `netmiko` (default), `scrapli`, `napalm` or `pyats`

- **render** – (list) list of arguments to pass through SaltStack rendering system, by default renders content of `["filename", "commands"]` arguments.
- **use_ps** – (bool) if True, uses Netmiko with promptless mode to send commands
- **kwargs** – (dict) any additional arguments to use with specified `plugin` send command method

Sample Usage:

```
salt nrp1 nr.cli "show clock" "show run" FB="IOL[12]" use_timing=True delay_factor=4
salt nrp1 nr.cli commands='["show clock", "show run"]' FB="IOL[12]"
salt nrp1 nr.cli "show clock" FO='{"platform__any": ["ios", "nxos_ssh", "cisco_xr"]}
↪'
salt nrp1 nr.cli commands='["show clock", "show run"]' FB="IOL[12]" plugin=napalm
salt nrp1 nr.cli "show clock" use_ps=True cutoff=60 initial_sleep=10
```

`use_ps` enables to use promptless mode of interaction with device's cli, refer to netmiko_send_commands_ps Nornir-Salt task plugin for details.

Commands can be templates and rendered using Jinja2 Templating Engine:

```
salt nrp1 nr.cli "ping 1.1.1.1 source {{ host.lo0 }}"
```

Commands to run on devices can be sourced from text file on a Salt Master or any other location with supported URL schemes: salt://, http://, https://, ftp://, s3://, swift:// and file:// (local filesystem), that text file can also be a template, it is rendered using SaltStack rendering system:

```
salt nrp1 nr.cli filename="salt://device_show_commands.txt"
```

Combining above two features we can supply per-host commands like this:

```
salt nrp1 nr.cli filename="salt://{{ host.name }}_show_commands.txt"
```

Where `{{ host.name }}_show_commands.txt` file can be a template as well.

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()

task_result = client.cmd(
    tgt="nrp1",
    fun="nr.cli",
    arg=["show clock"],
    kwarg={"plugin": "netmiko"},
)
```

5.3.4 nr.diff

salt_nornir.modules.nornir_proxy_execution_module.**diff**(**args*, ***kwargs*)

Provide difference between current and previously learned information or between versions of files stored by ToFileProcessor.

Parameters

- **diff** – (str) ToFileProcessor filegroup name
- **last** – (int or list or str) filegroup file indexes to diff, default is 1
- **kwargs** – (dict) any additional kwargs to use with nr.file diff call or DiffProcessor

diff attribute mandatory.

If last is a single digit e.g. 1, diff uses nr.do function to execute action named same as filegroup attribute and uses results to produce diff with previously saved filegroup files using DiffProcessor.

If last is a list e.g. [2, 5] or string 1, 2- will use nr.file diff call to produce diff for previously saved results without retrieving data from devices.

Sample usage:

```
salt nrp1 nr.diff interface
salt nrp1 nr.diff interface last=1
salt nrp1 nr.diff interface last='[1, 5]'
salt nrp1 nr.diff interface last="1,5"
```

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()

task_result = client.cmd(
    tgt="nrp1",
    fun="nr.diff",
    arg=["interface"],
    kwarg={"last": 1},
)
```

5.3.5 nr.do

salt_nornir.modules.nornir_proxy_execution_module.**do**(**args*, ***kwargs*)

Function to perform steps defined under nornir:actions configuration section at:

- Minion's configuration
- Minion's grains
- Minion's pillar data
- Master configuration (requires pillar_opts to be set to True in Minion config file in order to work)
- File on master file system

To retrieve actions content Salt nr.do uses config.get execution module function with merge key set to True.

Each step definition requires these keywords to be defined:

- function - mandatory, name of any execution module function to run

- `args` - optional, any arguments to use with function
- `kwargs` - optional, any keyword arguments to use with function
- `description` - optional, used by `dir` to list action description

Any other keywords defined inside the step are ignored.

Parameters

- **`stop_on_error`** – (bool) if True (default) stops execution on error in step, continue execution in error if False
- **`filepath`** – (str) path to file with actions steps
- **`default_renderer`** – (str) shebang string to render file using `slsutil.renderer`, default ``jinja|yaml`
- **`describe`** – (bool) if True, returns action content without executing it, default is False
- **`kwargs`** – (any) additional `kwargs` to use with actions steps, `kwargs` override `kwargs` dictionary defined within each step, for example, in command `salt nrp1 nr.do configure_ntp FB="*core*"`, `FB` argument will override `FB` arguments defined within steps.
- **`tf`** – (bool) if True, `ToFileProcessor` saves each step results in file named after step name if no `tf` argument provided within step, default is False
- **`diff`** – (bool) if True, `DiffProcessor` runs diff for each step result using files named after step name if no `diff` argument provided within step, default is False

Returns dictionary with keys: `failed` bool, `result` list; `result` key contains a list of results for steps; If `stop_on_error` set to `True` and error happens, `failed` key set to `True`

Special action names `dir` and `dir_list` used to list all actions available for proxy minion where `dir` returns table and `dir_list` produces a list of actions.

Note: if `filepath` argument provided, actions defined in other places are ignored; file loaded using SaltStack `slsutil.renderer` execution module function, as a result file can contain any of supported SaltStack renderer content and can be located at any url supported by `cp.get_url` execution module function - supported URL schemes are: salt://, http://, https://, ftp://, s3://, swift:// and file:// (local filesystem). File content must render to a dictionary keyed by actions' names.

Sample actions steps definition using proxy minion pillar:

```
nornir:
  actions:
    arista_wr:
      function: nr.cli
      args: ["wr"]
      kwargs: {"FO": {"platform": "arista_eos"}}
      description: "Save Arista devices configuration"
    configure_ntp:
      - function: nr.cfg
        args: ["ntp server 1.1.1.1"]
        kwargs: {"FB": "*"}
      - function: nr.cfg
        args: ["ntp server 1.1.1.2"]
        kwargs: {"FB": "*"}
```

(continues on next page)

(continued from previous page)

```
    - function: nr.cli
      args: ["show run | inc ntp"]
      kwargs: {"FB": "*"}
```

Sample actions steps definition using text file under `filepath`:

```
arista_wr:
  function: nr.cli
  args: ["wr"]
  kwargs: {"FO": {"platform": "arista_eos"}}
  description: "Save Arista devices configuration"
configure_ntp:
  - function: nr.cfg
    args: ["ntp server 1.1.1.1"]
    kwargs: {"FB": "*"}
    description: "1. Configure NTP server 1.1.1.1"
  - function: nr.cfg
    args: ["ntp server 1.1.1.2"]
    kwargs: {"FB": "*"}
    description: "2. Configure NTP server 1.1.1.2"
  - function: nr.cli
    args: ["show run | inc ntp"]
    kwargs: {"FB": "*"}
    description: "3. Collect NTP configuration"
```

Action name `arista_wr` has single step defined, while `configure_ntp` action has multiple steps defined, each executed in order.

Multiple actions names can be supplied to `nr.do` call.

Warning: having column `:` as part of action name not permitted, as `:` used by Salt `config.get` execution module function to split arguments on path items.

Sample usage:

```
salt nrp1 nr.do dir
salt nrp1 nr.do dir_list
salt nrp1 nr.do arista_wr
salt nrp1 nr.do configure_ntp arista_wr stop_on_error=False
salt nrp1 nr.do configure_ntp FB="*core*" add_details=True
salt nrp1 nr.do arista_wr filepath="salt://actions/actions_file.txt"
```

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()

task_result = client.cmd(
    tgt="nrp1",
    fun="nr.do",
    arg=["configure_ntp", "arista_wr"],
```

(continues on next page)

(continued from previous page)

```
    kwarg={"FB": "R[12]"},
)
```

5.3.6 nr.file

salt_nornir.modules.nornir_proxy_execution_module.**file**(**args*, ***kwargs*)

Function to manage Nornir-salt files.

Parameters

- **call** – (str) files task to call - ls, rm, read, diff
- **kwargs** – (dict) any additional kwargs such Fx filters or call function arguments

File tasks description:

- ls - list files of this Proxy Minions, returns list of dictionaries
- rm - removes file with given name and index number
- read - displays content of file with given name and index number
- diff - reads two files and returns diff

ls arguments

Parameters **filegroup** – (str or list) tf or list of tf filegroup names of the files to list, lists all files by default

Returns files list

rm arguments

Parameters **filegroup** – (str or list) tf or list of tf filegroup names of the files to remove, if set to True will remove all files for all filegroups

Returns list of files removed

read arguments

Parameters

- **filegroup** – (str or list) tf or list of tf filegroup names of the files to read
- **last** – (int) version of content to read

Returns results reconstructed out of files content

diff arguments

Parameters

- **filegroup** – (str or list) tf filegroup name to diff
- **last** – (int or list or str) files to diff, default is [1, 2] - last 1 and last 2 files

Returns files unified difference

Sample usage:

```
salt nrp1 nr.file read ip
salt nrp1 nr.file rm ip interface
salt nrp1 nr.file diff routes last='[1,2]'
```

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()

task_result = client.cmd(
    tgt="nrp1",
    fun="nr.file",
    arg=["ls"],
    kwarg={"filegroup": "interfaces"},
)
```

5.3.7 nr.find

salt_nornir.modules.nornir_proxy_execution_module.**find**(**args*, ***kwargs*)

Search for information stored in Proxy Minion files.

This function does not query devices but only uses information stored locally by `ToFileProcessor`.

Parameters

- **headers** – (str or list) table headers, default is `keys`
- **table** – (str) TabulateFormatter table directive, default is `extend`
- **headers_exclude** – (str or list) table headers to exclude, default is `["changed", "diff", "failed", "name", "connection_retry", "task_retry"]`
- **reverse** – (bool) default is False, reverses table order if True
- **sortby** – (str) column header name to sort table by
- **last** – (int) file group version of files to search in
- **Fx** – (str) Nornir host filters
- **args** – (list) list of `ToFileProcessor` filegroup names to search in
- **kwargs** – (dict) key-value pairs where keys are keys to search for, values are criteria to check

Returns list of dictionaries with matched results

Find uses `DataProcessor find` function to do search and supports searching in a list of dictionaries, dictionary and text.

If no `args` provided `nr.find` fails.

Sample usage:

```
salt nrp1 nr.find ip ip="1.1.*"
salt nrp1 nr.find mac arp mac="1b:cd:34:5f:6c"
salt nrp1 nr.find ip ip="1.1.*" last=5 FB="*CORE*"
salt nrp1 nr.find ip mask__ge=23 mask__lt=30 FC="CORE"
salt nrp1 nr.find interfaces description__contains="ID #123321"
```

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()
```

(continues on next page)

(continued from previous page)

```
task_result = client.cmd(
    tgt="nrp1",
    fun="nr.find",
    arg=["ip"],
    kwarg={"ip": "1.1.*"},
)
```

5.3.8 nr.gnmi

salt_nornir.modules.nornir_proxy_execution_module.**gnmi**(*call*, **args*, ***kwargs*)

Function to interact with devices using gNMI protocol utilizing one of supported plugins.

Parameters

- **call** – (str) (str) connection object method to call or name of one of extra methods
- **plugin** – (str) Name of gNMI plugin to use - pygnmi (default)
- **method_name** – (str) name of method to provide docstring for, used only by `help` call
- **path** – (list or str) gNMI path string for `update`, `delete`, `replace` extra methods calls
- **filename** – (str) path to file with call method arguments content
- **kwargs** – (dict) any additional keyword arguments to use with call method

Returns method call results

Available gNMI plugin names:

- `pygnmi` - `nornir-salt` built-in plugin that uses PyGNMI library to interact with devices.

gNMI specification defines several methods to work with devices - `subscribe`, `get` and `set`. `set` further supports `delete`, `update` and `replace` operations.

Warning: `subscribe` is not supported by `nr.gnmi` function.

Sample usage of `pygnmi` plugin:

```
salt nrp1 nr.gnmi capabilities FB="*"
salt nrp1 nr.gnmi get "openconfig-interfaces:interfaces/interface[name=Loopback100]"
salt nrp1 nr.gnmi get path='["openconfig-interfaces:interfaces/
↪interface[name=Loopback100]"]'
salt nrp1 nr.gnmi get path="openconfig-interfaces:interfaces, openconfig-network-
↪instance:network-instances"
salt nrp1 nr.gnmi set update='[["openconfig-interfaces:interfaces/
↪interface[name=Loopback100]/config", {"description": "Done by gNMI"}]]'
salt nrp1 nr.gnmi set replace='[["openconfig-interfaces:interfaces/
↪interface[name=Loopback1234]/config", {"name": "Loopback1234", "description": "New
↪"}]]'
salt nrp1 nr.gnmi set delete="openconfig-interfaces:interfaces/
↪interface[name=Loopback1234]"
```

Extra Call Methods

- `dir` - returns methods supported by `gNMIclient` connection object, including extra methods defined by nornir-salt `pygnmi_call` task plugin:

```
salt nrp1 nr.gnmi dir
```

- `help` - returns `method_name` docstring:

```
salt nrp1 nr.gnmi help method_name=set
```

- `replace` - shortcut method to `set` call with `replace` argument, first argument must be path string, other keyword arguments are configuration items:

```
salt nrp1 nr.gnmi replace "openconfig-interfaces:interfaces/
↪interface[name=Loopback100]/config" name=Loopback100 description=New
```

- `update` - shortcut method to `set` call with `update` argument, first argument must be path string, other keyword arguments are configuration items:

```
salt nrp1 nr.gnmi update "openconfig-interfaces:interfaces/
↪interface[name=Loopback100]/config" description="RID Loop"
```

- `delete` - shortcut method to `set` call with `delete` argument, accepts a list of path items or `path` argument referring to list:

```
salt nrp1 nr.gnmi delete "openconfig-interfaces:interfaces/
↪interface[name=Loopback100]" "openconfig-interfaces:interfaces/
↪interface[name=Loopback123]"
salt nrp1 nr.gnmi delete path='["openconfig-interfaces:interfaces/
↪interface[name=Loopback100]", "openconfig-interfaces:interfaces/
↪interface[name=Loopback123]"]'
```

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()

task_result = client.cmd(
    tgt="nrp1",
    fun="nr.gnmi",
    arg=["get"],
    kwarg={"path": ["openconfig-interfaces:interfaces"]},
)
```

If `filename` argument provided it is rendered using `slsutil.renderer` function and must produce a dictionary with keys being valid arguments supported by call method. For example, `pygnmi` plugin `set` call can look like this:

```
salt nrp1 nr.gnmi set filename="salt://path/to/set_args.txt"
```

Where `salt://path/to/set_args.txt` content is:

```
replace:
  - - "openconfig-interfaces:interfaces/interface[name=Loopback35]/config"
    - {"name": "Loopback35", "description": "RID Loopback"}
  - - "openconfig-interfaces:interfaces/interface[name=Loopback36]/config"
```

(continues on next page)

(continued from previous page)

```
    - {"name": "Loopback36", "description": "MGMT Loopback"}
update:
  - - "openconfig-interfaces:interfaces/interface[name=Loopback35]/config"
    - {"name": "Loopback35", "description": "RID Loopback"}
  - - "openconfig-interfaces:interfaces/interface[name=Loopback36]/config"
    - {"name": "Loopback36", "description": "MGMT Loopback"}
delete:
  - "openconfig-interfaces:interfaces/interface[name=Loopback35]"
  - "openconfig-interfaces:interfaces/interface[name=Loopback36]"
```

`salt://path/to/set_args.txt` content will render to a dictionary supplied to `set` call as a `**kwargs`.

`pygnmi` plugin order of operation for above case is `delete -> replace -> update`

5.3.9 nr.http

salt_nornir.modules.nornir_proxy_execution_module.**http**(**args*, ***kwargs*)

HTTP requests related functions

Parameters

- **method** – (str) HTTP method to use
- **url** – (str) full or partial URL to send request to
- **kwargs** – (dict) any other kwargs to use with requests.<method> call

This function uses nornir_salt http_call task plugin, reference that task plugin documentation for additional details.

Sample usage:

```
salt nrp1 nr.http get "http://1.2.3.4/api/data/"
salt nrp1 nr.http get "https://sandbox-iosxe-latest-1.cisco.com/restconf/data/" verify=False auth='["developer", "C1sco12345"]'
```

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()

task_result = client.cmd(
    tgt="nrp1",
    fun="nr.http",
    arg=["get", "http://1.2.3.4/api/data/"],
)
```

5.3.10 nr.learn

salt_nornir.modules.nornir_proxy_execution_module.**learn**(**args*, ***kwargs*)

Store task execution results to local filesystem on the minion using `tf` (to filename) attribute to form filenames.

Parameters

- **fun** – (str) name of execution module function to call
- **tf** – (str) `ToFileProcessor` filegroup name
- **tf_skip_failed** – (bool) default is True, do not save failed tasks
- **args** – (list) execution module function arguments
- **kwargs** – (dict) execution module function key-word arguments

This task uses `ToFileProcessor` to store results and is a shortcut to calling individual execution module functions with `tf` argument.

Supported execution module functions are `cli, nc, do, http`. By default calls `nr.do` function.

`tf` attribute mandatory except for cases when using `nr.do``function e.g. ``salt nrp1 nr.learn mac interface`, in that case `tf` set equal to file group name - `mac` and `interface` for each action call using `nr.do` function `tf=True` attribute.

Sample usage:

```
salt nrp1 nr.learn mac
salt nrp1 nr.learn mac ip interface FB="CORE-*"
salt nrp1 nr.learn "show version" "show int brief" tf="cli_facts" fun="cli"
```

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()

task_result = client.cmd(
    tgt="nrp1",
    fun="nr.learn",
    arg=["mac", "ip"],
    kwarg={"FB": "CORE-*"},
)
```

5.3.11 nr.nc

salt_nornir.modules.nornir_proxy_execution_module.**nc**(**args*, ***kwargs*)

Function to interact with devices using NETCONF protocol utilizing one of supported plugins.

Available NETCONF plugin names:

- `ncclient` - `nornir-salt` built-in plugin that uses `ncclinet` library to interact with devices
- `scrapli` - uses `scrapli_netconf` connection plugin that is part of `nornir_scrapli` library, it does not use `scrapli_netconf` task plugins, but rather implements a wrapper around `scrapli_netconf` connection plugin connection object.

Parameters

- **call** – (str) ncclient manager or scrapli netconf object method to call

- **plugin** – (str) Name of netconf plugin to use - ncclient (default) or scrapli
- **data** – (str) path to file for `rpc` method call or rpc content
- **method_name** – (str) name of method to provide docstring for, used only by `help` call

Special `call` arguments/methods:

- `dir` - returns methods supported by Ncclient connection manager object:

```
salt nrp1 nr.nc dir
```

- `help` - returns `method_name` docstring:

```
salt nrp1 nr.nc help method_name=edit_config
```

- `transaction` - same as `edit_config`, but runs this (presumably more reliable) work flow:
  1. Lock target configuration datastore
  2. If client and server supports it - Discard previous changes if any
  3. Edit configuration
  4. If client and server supports it - validate configuration if `validate` argument is True
  5. If client and server supports it - do commit confirmed if `confirmed` argument is True
  6. If client and server supports it - do commit operation
  7. Unlock target configuration datastore
  8. If client and server supports it - discard all changes if any of steps 3, 4, 5 or 6 fail
  9. Return results list of dictionaries keyed by step name

  If any of steps 3, 4, 5, 6 fail, all changes discarded.

  Sample usage:

```
salt nrp1 nr.nc transaction target="candidate" config="salt://path/to/
↪config_file.xml" FB="*core-1"
```

Warning: beware of difference in keywords required by different plugins, e.g. `filter` for `ncclient` vs `filter_/filters` for `scrapli_netconf`, refer to modules' api documentation for required arguments, using, for instance `help` call: `salt nrp1 nr.nc help method_name=get_config`

Examples of sample usage for `ncclient` plugin:

```
salt nrp1 nr.nc server_capabilities FB="*"
salt nrp1 nr.nc get_config filter='["subtree", "salt://rpc/get_config_data.xml"]'␣
↪source="running"
salt nrp1 nr.nc edit_config target="running" config="salt://rpc/edit_config_data.xml
↪" FB="ceos1"
salt nrp1 nr.nc transaction target="candidate" config="salt://rpc/edit_config_data.
↪xml"
salt nrp1 nr.nc commit
salt nrp1 nr.nc rpc data="salt://rpc/iosxe_rpc_edit_interface.xml"
```

(continues on next page)

(continued from previous page)

```
salt nrp1 nr.nc get_schema identifier="ietf-interfaces"
salt nrp1 nr.nc get filter='<system-time xmlns="http://cisco.com/ns/yang/Cisco-IOS-
↪XR-shellutil-oper"/>'
```

Examples of sample usage for `scrapli_netconf` plugin:

```
salt nrp1 nr.nc get filter_=salt://rpc/get_config_filter_ietf_interfaces.xml␣
↪plugin=scrapli
salt nrp1 nr.nc get_config source=running plugin=scrapli
salt nrp1 nr.nc server_capabilities FB="*" plugin=scrapli
salt nrp1 nr.nc rpc filter_=salt://rpc/get_config_rpc_ietf_interfaces.xml␣
↪plugin=scrapli
salt nrp1 nr.nc transaction target="candidate" config="salt://rpc/edit_config_ietf_
↪interfaces.xml" plugin=scrapli
```

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()

task_result = client.cmd(
    tgt="nrp1",
    fun="nr.nc",
    arg=["get_config"],
    kwarg={"source": "running", "plugin": "ncclient"},
)
```

5.3.12 nr.nornir

salt_nornir.modules.nornir_proxy_execution_module.**nornir_fun**(*fun*, **args*, ***kwargs*)

Function to call various Nornir utility functions.

Parameters

- **fun** – (str) utility function name to call
- **kwargs** – (dict) function arguments

Available utility functions:

- `dir` - return a list of supported functions
- `test` - this method tests proxy minion module worker thread without invoking any Nornir code
- `refresh` - re-instantiates Nornir object after retrieving latest pillar data from Salt Master
- `kill` - executes immediate shutdown of Nornir Proxy Minion process and child processes
- `shutdown` - gracefully shutdowns Nornir Proxy Minion process and child processes
- `inventory` - interact with Nornir Process inventory data, using `InventoryFun` function, by default, for `read_host, read, read_inventory, list_hosts` operations any Nornir worker can respond, for other, non-read operations targets all Nornir workers
- `stats` - returns statistics about Nornir proxy process, accepts `stat` argument of stat name to return
- `version` - returns a report of Nornir related packages installed versions

- `initialized` - returns Nornir Proxy Minion initialized status - True or False
- `hosts` - returns a list of hosts managed by this Nornir Proxy Minion, accepts `Fx` arguments to return only hosts matched by filter
- `connections` - list hosts' active connections for all workers, accepts `Fx` arguments to filter hosts to list, by default returns connections data for all Nornir workers, uses `nornir_salt.plugins.tasks.connections ls` call
- `disconnect` - close host connections, accepts `Fx` arguments to filter hosts and `conn_name` of connection to close, by default closes all connections from all Nornir workers, uses `nornir_salt.plugins.tasks.connections close` call
- `connect` - initiate connection to devices, arguments: `conn_name, hostname, username, password, port, platoform, extras, default_to_host_attributes, close_open`, uses `nornir_salt.plugins.tasks.connections open` call
- `clear_hcache` - clear task results cache from hosts' data, accepts `cache_keys` list argument of key names to remove, if no `cache_keys` argument provided removes all cached data, by default targets all Nornir workers
- `clear_dcache` - clear task results cache from defaults data, accepts `cache_keys` list argument of key names to remove, if no `cache_keys` argument provided removes all cached data, by default targets all Nornir workers
- `workers/worker` - call nornir worker utilities e.g. `stats`
- `results_queue_dump` - return content of results queue

Sample Usage:

```
salt nrp1 nr.nornir inventory FB="R[12]"
salt nrp1 nr.nornir inventory FB="R[12]" worker="all"
salt nrp1 nr.nornir inventory create_host name="R1" hostname="1.2.3.4" platform="ios
↪" groups='["lab"]'
salt nrp1 nr.nornir inventory update_host name="R1" data='{"foo": bar}'
salt nrp1 nr.nornir inventory read_host FB="R1"
salt nrp1 nr.nornir inventory call=delete_host name="R1"
salt nrp1 nr.nornir stats stat="proxy_minion_id"
salt nrp1 nr.nornir version
salt nrp1 nr.nornir shutdown
salt nrp1 nr.nornir clear_hcache cache_keys='["key1", "key2]'
salt nrp1 nr.nornir clear_dcache cache_keys='["key1", "key2]'
salt nrp1 nr.nornir workers stats
salt nrp1 nr.nornir connect conn_name=netmiko username=cisco password=cisco␣
↪platform=cisco_ios
salt nrp1 nr.nornir connect scrapli port=2022 close_open=True
```

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()

task_result = client.cmd(
    tgt="nrp1",
    fun="nr.nornir",
    arg=["stats"],
)
```

5.3.13 nr.task

salt_nornir.modules.nornir_proxy_execution_module.**task**(*plugin*, ***kwargs*)

Function to invoke any of supported Nornir task plugins. This function performs dynamic import of requested plugin function and executes `nr.run` using supplied args and kwargs

Parameters

- **plugin** – (str) `path.to.plugin.task_fun` to run `from path.to.plugin import task_fun`
- **kwargs** – (dict) arguments to use with specified task plugin or common arguments

`plugin` attribute can refer to a file on one of remote locations, supported URL schemes are: salt://, http://, https://, ftp://, s3://, swift:// and file:// (local filesystem). File downloaded, compiled and executed.

File must contain function named `task` accepting Nornir task object as a first positional argument, for example:

```
# define connection name for RetryRunner to properly detect it
CONNECTION_NAME = "netmiko"

# create task function
def task(nornir_task_object, *args, **kwargs):
    pass
```

Note: `CONNECTION_NAME` must be defined within custom task function file if RetryRunner in use, otherwise connection retry logic skipped and connections to all hosts initiated simultaneously up to the number of num_workers.

Sample usage:

```
salt nrp1 nr.task "nornir_napalm.plugins.tasks.napalm_cli" commands='["show ip arp"]
↪' FB="IOL1"
salt nrp1 nr.task "nornir_netmiko.tasks.netmiko_save_config" add_details=False
salt nrp1 nr.task "nornir_netmiko.tasks.netmiko_send_command" command_string="show␣
↪clock"
salt nrp1 nr.task nr_test a=b c=d add_details=False
salt nrp1 nr.task "salt://path/to/task.txt"
salt nrp1 nr.task plugin="salt://path/to/task.py"
```

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()

task_result = client.cmd(
    tgt="nrp1",
    fun="nr.task",
    arg=["nornir_napalm.plugins.tasks.napalm_cli"],
    kwarg={"commands": ["show ip arp"]},
)
```

5.3.14 nr.test

salt_nornir.modules.nornir_proxy_execution_module.**test**(**args*, ***kwargs*)

Function to perform tests for certain criteria against show commands output from devices obtained using `nr.cli` function.

`nr.test` function related arguments

Parameters

- **name** – (str) descriptive name of the test, will be added to results
- **test** – (str) type of test to do e.g.: contains, !contains, equal, custom etc.
- **pattern** – (str) pattern to use for testing, usually string, text or reference a text file on salt master. For instance if `test` is `contains`, `pattern` value used as a pattern for containment check.
- **function_file** – (str) path to text file on salt master with function content to use for `custom` function test
- **saltenv** – (str) name of salt environment to download function_file from
- **suite** – (list or str) list of dictionaries with test items or path to file on salt-master with a list of test item dictionaries
- **subset** – (list or str) list or string with comma separated glob patterns to match tests' names to execute. Patterns are not case-sensitive. Uses `fnmatch.fnmatch` Python built-in function to do matching.
- **dump** – (str) filegroup name to dump results using Nornir-salt `DumpResults`
- **kwargs** – (dict) any additional arguments to use with test function

`nr.cli` function related arguments

Parameters

- **commands** – (str or list) single command or list of commands to get from device
- **plugin** – (str) plugin name to use with `nr.cli` function to gather output from devices - `netmiko` (default) or `scrapli`
- **use_ps** – (bool) default is False, if True use netmiko plugin experimental `PromptlesS` method to collect output from devices
- **cli** – (dict) any additional arguments to pass on to `nr.cli` function

Nornir-Salt `TestsProcessor` plugin related arguments

Parameters

- **failed_only** – (bool) default is False, if True `nr.test` returns result for failed tests only
- **remove_tasks** – (bool) default is True, if False results will include other tasks output as well e.g. show commands output. By default results only contain tests results.

Nornir-Salt `TabulateFormatter` function related arguments

Parameters

- **table** – (bool, str or dict) dictionary of arguments or table type indicator e.g. "brief" or True
- **headers** – (list) list of headers to output table for
- **sortby** – (str) Name of column name to sort table by

- **reverse** – (bool) reverse table on True, default is False

Sample usage with inline arguments:

```
salt np1 nr.test "show run | inc ntp" contains "1.1.1.1" FB="*host-1"
salt np1 nr.test "show run | inc ntp" contains "1.1.1.1" --output=table
salt np1 nr.test "show run | inc ntp" contains "1.1.1.1" table=brief
salt np1 nr.test commands='["show run | inc ntp"]' test=contains pattern="1.1.1.1"
```

Sample usage with a suite of test cases:

```
salt np1 nr.test suite=salt://tests/suite_1.txt
salt np1 nr.test suite=salt://tests/suite_1.txt table=brief
salt np1 nr.test suite=salt://tests/suite_1.txt table=brief subset="config_test*,
↪rib_check*"
```

Where `salt://tests/suite_1.txt` content is:

```
- task: "show run | inc ntp"
  test: contains
  pattern: 1.1.1.1
  name: check NTP cfg
  cli:
    FB: core-*
    plugin: netmiko
- test: contains_lines
  pattern: ["1.1.1.1", "2.2.2.2"]
  task: "show run | inc ntp"
  name: check NTP cfg lines
- test: custom
  function_file: salt://tests/ntp_config.py
  task: "show run | inc ntp"
  name: check NTP cfg pattern from file
- test: custom
  function_file: salt://tests/ntp_config.py
  task:
    - "show ntp status"
    - "show ntp associations"
  name: "Is NTP in sync"
```

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()

task_result = client.cmd(
    tgt="nrp1",
    fun="nr.test",
    kwarg={"suite": "salt://tests/suite_1.txt"},
)
```

Returns a list of dictionaries with check results, each dictionary contains:

```
{
    "host": name of host,
```

(continues on next page)

(continued from previous page)

```
    "name": descriptive name of the test,
    "task": name of task results of which used for test,
    "result": PASS or FAIL,
    "success": True or False,
    "error": None or Error description,
    "test_type": Type of test performed,
    "criteria": Validation criteria used
}
```

Reference Nornir Salt TestsProcessor documentation for more details on using tests suite.

Each item in a test suite executed individually one after another.

In test suite, `task` argument can reference a list of tasks/commands.

Commands output for each item in a suite collected using `nr.cli` function, arguments under `cli` keyword passed on to `nr.cli` function.

List of arguments in a test suite that can refer to a text file to source from one of supported URL schemes: salt://, http://, https://, ftp://, s3://, swift:// and file:// (local filesystem), for example `salt://path/to/file.txt`:

- `pattern` - content of the file rendered and used to run the tests together with `ContainsTest`, `ContainsLinesTest` or `EqualTest` test functions
- `schema` - used with `CerberusTest` test function
- `function_file` - content of the file used with `CustomFunctionTest` as `function_text` argument

Starting with Salt-Nornir 0.7.0 support added for `wait_timeout` and `wait_interval` test item arguments to control the behavior of waiting for test's `success` to evaluate to True following these rules:

1. If `wait_timeout` given, keep executing test until result's `success` evaluates to True or `wait_timeout` expires
2. Between test item execution attempts sleep for `wait_interval`, default is 10 seconds
3. If `wait_timeout` expires, return results for last test execution attempt, after updating `exception`, `failed` and `success` fields accordingly

For example, test below will keep executing for 60 seconds with 20 seconds interval until "show ip route" output contains "1.1.1.1/32" pattern for hosts R1 and R2:

```
- task: "show ip route"
  test: contains
  pattern: "1.1.1.1/32"
  name: "Check if has 1.1.1.1/32 route"
  wait_timeout: 60
  wait_interval: 20
  cli:
    FL: ["R1", "R2"]
```

Warning: for `wait_timeout` feature to work, test result must contain `success` field, otherwise test outcome evaluates to False.

5.3.15 nr.tping

salt_nornir.modules.nornir_proxy_execution_module.**tping**(*ports=None*, *timeout=1*, *host=None*, ***kwargs*)

Tests connection to TCP port(s) by trying to establish a three way handshake. Useful for network discovery or testing.

Parameters

- **int)** (*ports (list of*) – tcp ports to ping, defaults to host's port or 22
- **(int)** (*timeout*) – defaults to 1
- **(str)** (*host*) – defaults to hostname

Sample usage:

```
salt nrp1 nr.tping
salt nrp1 nr.tping FB="LAB-RT[123]"
```

Returns result object with the following attributes set:

- result (dict): Contains port numbers as keys with True/False as values

Sample Python API usage from Salt-Master:

```
import salt.client
client = salt.client.LocalClient()

task_result = client.cmd(
    tgt="nrp1",
    fun="nr.tping",
    kwarg={"FB": "LAB-RT[123]"},
)
```

CHAPTER

SIX

NORNIR RUNNER MODULE

Nornir Runner module reference.

Note: Runner module functions executed on same machine where salt-master process runs.

6.1 Introduction

Nornir-runner module runs on SALT Master and allows to interact with devices behind Nornir proxy minions.

6.2 Nornir Runner module functions

salt_nornir.runners.nornir_proxy_runner_module.**inventory**(**args*, ***kwargs*)

Function to return brief inventory data for certain hosts in a table format.

Parameters

- **FB** – glob pattern matching hostnames of devices behind Nornir
- **Fx** – additional filters to filter hosts, e.g. FG, FP, FL etc.
- **tgt** – nornir proxy minion target, by default targets all - "proxy:proxytype:nornir"
- **tgt_type** – SALT targeting type to use, by default "pillar"
- **verbose** – boolean, returns `nr.cli` output as is if True, flattens to dictionary keyed by devices hostnames if False, default False
- **job_retry** – how many times to retry command if no return from minions, default 0
- **job_timeout** – seconds to wait for return from minions, overrides `--timeout` option, default 30s
- **table** – (str, dict or bool) supplied to TabulateFormatter under `table` keyword to control output table formatting
- **headers** – (list) headers list, default `["minion", "host", "ip", "platform", "groups"]`
- **reverse** – (bool) reverse table order if True, default is False
- **sortby** – (str) header to sort table by, default is `host`

Sample Usage:

```
salt-run nr.inventory host_name_id
salt-run nr.inventory FB="host_name_id" FP="10.1.2.0/24"
```

If it takes too long to get output because of non-responding/unreachable minions, specify `--timeout` or `job_timeout` option to shorten waiting time, `job_timeout` overrides `--timeout`. Alternatively, instead of targeting all nornir based proxy minions, `tgt` and `tgt_type` can be used to target a subset of them:

```
salt-run nr.inventory host_name_id --timeout=10
salt-run nr.inventory host_name_id job_timeout=10 tgt="nornir-proxy-id" tgt_type=
↪"glob"
```

Sample output:

```
[root@localhost /]# salt-run nr.inventory IOL1
+---+--------+----------+---------------+----------+--------+
|   | minion |   host   |       ip      | platform | groups |
+---+--------+----------+---------------+----------+--------+
| 0 |  nrp1  |   IOL1   | 192.168.217.10 |   ios    |  lab   |
+---+--------+----------+---------------+----------+--------+
```

salt_nornir.runners.nornir_proxy_runner_module.**call**(**args*, ***kwargs*)

Method to call any Nornir Proxy Minion Execution Module function against minions. By default this function targets all Nornir Proxy Minions, allowing to simplify targeting hosts managed by them.

Parameters

- **fun** – (str) Nornir Proxy Minion Execution Module function name e.g. `cli, cfg, nc, gnmi` etc.
- **tgt** – (str) SaltStack Nornir Proxy Minions to target, targets all of them by default - `proxy:proxytype:nornir`
- **tgt_type** – (str) SaltStack targeting type to use, default is `pillar`
- **job_retry** – (int) how many times to retry if no results returned from all minions, default 0
- **timeout** – (int) seconds to wait for results from minions before retry, default 300s
- **progress** – progress display type to use - bars, raw, log, if False, no progress displayed
- **ret_struct** – results return structure, default is `dictionary`, also can be `list`
- **args** – (list) any other arguments to use with call function
- **kwargs** – (dict) any other keyword arguments to use with call function

Sample Usage:

```
salt-run nr.call fun="cfg" "logging host 1.2.3.4" FC="CORE"
salt-run nr.call cli "show clock" FB="*" tgt="nr-minion-id*" tgt_type="glob"
```

salt_nornir.runners.nornir_proxy_runner_module.**event**(*jid='all'*, *tag=None*, *progress='log'*, *stop_signal=None*)

Function to listen to events emitted by Nornir Proxy Minions. Matched event printed to terminal.

Parameters

- **tag** – (str) tag regex string, default is `nornir\-proxy/.*`
- **jid** – (int, str) Job ID to listen events for, default is `all`

- **progress** – (str) progress display mode - `log`, `raw`, `bars`, `tree`
- **stop_signal** – (obj) thread Event object, stops listening to events if `stop_signal.is_set()`, if `stop_signal is None`, listens and print events until keyboard interrupt hit - `ctrl+c`

  `bars` and `tree` progress display modes use Rich library, to properly display various symbols and characters need to make sure to use utf-8 encoding for your environment, to ensure that issue these commands in your terminal:

```
[root@salt-master ~]# PYTHONIOENCODING=utf-8
[root@salt-master ~]# export PYTHONIOENCODING
```

CHAPTER

SEVEN

NORNIR STATE MODULE

Nornir State module reference.

7.1 Introduction

This state module uses Nornir proxy execution module to apply configuration to devices.

Warning: This module does not implement idempotent behavior, it is up to Nornir task plugin to handle idempotency or up to user to define work flow steps to achieve desired level of idempotency.

Example

Example using `nr.cfg` and `nr.task` state module functions within SALT state.

File `salt://states/nr_state_test.sls` content located on Master:

```
apply_logging_commands:
  nr.cfg:
    - commands:
      - logging host 1.1.1.1
      - logging host 2.2.2.2
    - plugin: netmiko

apply_ntp_cfg_from_file:
  nr.cfg:
    - filename: "salt://templates/nr_state_test_ntp.j2"
    - plugin: netmiko

use_task_to_save_config:
  nr.task:
    - plugin: "nornir_netmiko.tasks.netmiko_save_config"

use_task_to_configure_logging:
  nr.task:
    - plugin: "nornir_netmiko.tasks.netmiko_send_config"
    - config_commands: "logging host 3.3.3.3"
```

File `salt://templates/nr_state_test_ntp.j2` content located on Master:

```
{%- if host.platform|lower == 'ios' %}
ntp server 1.1.1.1
{%- elif host.platform|lower == 'cisco_xr' %}
ntp peer 1.1.1.1
{%- endif %}
```

Apply state running command on master:

```
salt nr_minion_id state.apply nr_state_test
```

7.2 Nornir State Module Functions

Table 1: State Functions Summary

Name	Description
cfg	Configure devices using Nornir execution module `nr.cfg` function
task	Interact with devices using `nr.task` Execution Module function.
workflow	Executes work flow steps using any SaltStack Execution modules functions

7.2.1 cfg

salt_nornir.states.nornir_proxy_state_module.**cfg**(**args*, ***kwargs*)

Configure devices using Nornir execution module `nr.cfg` function.

Parameters

- **commands** – list of commands to send to device
- **filename** – path to file with configuration
- **template_engine** – template engine to render configuration, default is jinja2
- **saltenv** – name of SALT environment
- **context** – Overrides default context variables passed to the template.
- **defaults** – Default context passed to the template.
- **plugin** – name of configuration task plugin to use - `napalm` (default) or `netmiko` or `scrapli`
- **dry_run** – boolean, default False, controls whether to apply changes to device or simulate them
- **Fx** – filters to filter hosts
- **add_details** – boolean, to include details in result or not

> **Warning:** `dry_run` not supported by `netmiko` plugin

Sample Usage

File `salt://states/nr_state_logging_cfg.sls` content located on Master:

```
apply_logging_commands:
  nr.cfg:
    - commands:
      - logging host 1.1.1.1
      - logging host 2.2.2.2
    - plugin: netmiko
    - FB: "*"
```

Apply state:

```
salt nr_minion_id state.apply nr_state_logging_cfg
```

7.2.2 task

salt_nornir.states.nornir_proxy_state_module.**task**(**args*, ***kwargs*)

Interact with devices using `nr.task` Execution Module function.

Parameters

- **plugin** – `path.to.plugin.task_fun` to use, should form valid python import statement - `from path.to.plugin import task_fun`
- **Fx** – filters to filter hosts
- **add_details** – boolean, to include details in result or not
- **args** – arguments to pass on to task plugin
- **kwargs** – keyword arguments to pass on to task plugin

Sample Usage

File salt://states/nr_state_ntp_cfg.sls content located on Master:

```
use_task_to_configure_logging:
  nr.task:
    - plugin: "nornir_netmiko.tasks.netmiko_send_config"
    - config_commands: "ntp server 1.1.1.1"
```

Apply state:

```
salt nr_minion_id state.apply  nr_state_ntp_cfg
```

7.2.3 workflow

salt_nornir.states.nornir_proxy_state_module.**workflow**(**args*, ***kwargs*)

State function to execute work flow steps using SatlStack Execution modules functions.

State Global Options

State Global Options defined under `options` key.

Parameters

- **report_all** – (bool) if True (default) adds skipped steps in summary report

- **sumtable** – (bool or str) default is False, if True uses text table for summary report, `report_all` always set to True if `sumtable` is True. If `sumtable` is string, it is must correspond to one of the python-tabulate module's table format names e.g. `grid, simple, jira, html`
- **fail_if_any_host_fail_any_step** – (list) steps to decide if state execution failed
- **fail_if_any_host_fail_all_step** – (list) steps to decide if state execution failed
- **fail_if_all_host_fail_any_step** – (list) steps to decide if state execution failed
- **fail_if_all_host_fail_all_step** – (list) steps to decide if state execution failed
- **filters** – (dict) set of `Fx` filters to apply for all steps, per-step filters have higher priority. If no `Fx` filters provided, state steps run without any filters, depending on proxy `nornir_filter_required` setting, steps might fail (if `nornir_filter_required` is True) or run for all hosts (if `nornir_filter_required` is False). If no hosts matched by filters, state execution stops with appropriate comment included in report.
- **hcache** – (bool) if True (default) saves step's per-host results in host's data under step's name key so that other steps can use it
- **dcache** – (bool) if True (default) saves step's full results in defaults data under step's name key so that other steps can use it
- **kwargs** – (dict) common arguments to merge with each step kwargs, step kwargs more specific and overwrite common kwargs

> **Warning:** If proxy minion `nornir_filter_required` parameter set to True, workflow options `filters` must not be empty, but provided to limit overall execution scope.

Individual Step Arguments

Each step in a work flow can have a number of mandatory and optional attributes defined.

Parameters

- **name** – (str) mandatory, name of this step
- **function** – (str) mandatory, name of Nornir Execution Module function to run
- **kwargs** – (dict) `**kwargs` for Execution Module function
- **args** – (list) `*args` for Execution Module function
- **report** – (bool) if True (default) adds step execution results in detailed report
- **run_if_fail_any** – (list) this step will run if `any` of the previous steps in a list failed
- **run_if_pass_any** – (list) this step will run if `any` of the previous steps in a list passed
- **run_if_fail_all** – (list) this step will run if `all` of the previous steps in a list failed
- **run_if_pass_all** – (list) this step will run if `all` of the previous steps in a list passed

While workflow steps can call any execution module function, `run_if_x` properly supported only for Nornir Execution Module functions: `nr.task`, `nr.cli`, `nr.cfg_gen`, `nr.cfg`, `nr.test`, `nr.nc`, `nr.http`, `nr.do` - for all other functions step considered as `PASS` unconditionally.

If function reference `nr.test` with test suite, each test suite test item added to summary report, in addition, step's arguments `run_if_x` conditions **must** reference test suite individual tests' names attribute.

Warning: if you use per host filename feature, e.g. `filename="salt://path/to/{{ host.name }}.cfg"` make sure to either disable state file jinja2 rendering using `#!yaml` shebang at the beginning of the state file or escape double curly braces in filename argument.

Execution of steps done on a per host basis, or, say better, each step determines a set of hosts it needs to run for using `Fx` filters and `run_if_x` conditions. If multiple `run_if_x` conditions specified, host must satisfy all of them - AND logic - for step to be executed for that host.

If no `run_if_x` conditions provided, step executed for all hosts matched by `filters` provided in state global options and/or step `**kwargs`.

If `hcache` or `dcache` set to True in State Global Options in that case for compatible Nornir Execution Module function each step results saved in Nornir in-memory Inventory host's and `defaults` data under step's name key. That way results become part of inventory and available for use by Nornir Execution Module function in other steps. Once workflow execution completed, cached results cleaned. Individual step's `hcache` or `dcache` kwargs can be specified to save step's results under certain key name, in such a case after workflow completed cache not removed for that particular step's results.

Sample state `salt://states/configure_ntp.sls`:

```
main_workflow:
  nr.workflow:
    - options:
        fail_if_any_host_fail_any_step: []
        fail_if_any_host_fail_all_step: []
        fail_if_all_host_fail_any_step: []
        fail_if_all_host_fail_all_step: []
        report_all: False
        filters: {"FB": "*"}
        hcache: True
        dcache: False
        sumtable: False
        kwargs: {"event_progress": True}
    # define pre-check steps
    - pre_check:
        - name: pre_check_if_ntp_ip_is_configured_csr1kv
          function: nr.test
          kwargs: {"FB": "CSR*"}
          args: ["show run | inc ntp", "contains", "8.8.8.8"]
        - name: pre_check_if_ntp_ip_is_configured_xrv
          function: nr.test
          kwargs: {"FB": "XR*"}
          args: ["show run formal ntp", "contains", "8.8.8.8"]
    # here goes definition of change steps
    - change:
        - name: apply_ntp_ip_config
          function: nr.cfg
          args: ["ntp server 8.8.8.8"]
          kwargs: {"plugin": "netmiko"}
          run_if_fail_any: ["pre_check_if_ntp_ip_is_configured_csr1kv", "pre_check_
↪if_ntp_ip_is_configured_xrv"]
          report: True
    # run post check steps
```

(continues on next page)

(continued from previous page)

```
    - post_check:
        - name: check_new_config_applied_csr1kv
          function: nr.test
          args: ["show run | inc ntp", "contains", "8.8.8.8"]
          kwargs: {"FB": "CSR*"}
          run_if_pass_any: ["apply_ntp_ip_config"]
        - name: check_new_config_applied_xrv
          function: nr.test
          args: ["show run ntp", "contains", "8.8.8.8"]
          kwargs: {"FB": "XR*"}
          run_if_pass_any: ["apply_ntp_ip_config"]
    # execute rollback steps if required
    - rollback:
        - name: run_rollback_commands
          function: nr.cfg
          args: ["no ntp server 8.8.8.8"]
          kwargs: {"plugin": "netmiko"}
          run_if_fail_any: ["apply_ntp_ip_config", "check_new_config_applied_csr1kv
↪", "check_new_config_applied_xrv"]
```

Sample usage:

```
salt nrp1 state.sls configure_ntp
```

Executing workflow returns detailed and summary reports. Detailed report contains run details for each step being executed. Summary report contains per-host brief report of all steps statuses, where status can be:

- `PASS` - step passed, Nornir Execution Module task result `failed` attribute is False or `success` attribute is True
- `FAIL` - step failed, Nornir Execution Module task result `failed` attribute is True or `success` attribute is False
- `SKIP` - step skipped and not executed, usually due to `run_if_x` conditions not met for the host
- `ERROR` - State Module encountered exception while running this step

Sample report:

```
nrp1:
----------
          ID: change_step_1
    Function: nr.workflow
      Result: True
     Comment:
     Started: 12:01:58.578925
    Duration: 5457.171 ms
     Changes:
              ----------
              details:
                  |_
                    ----------
                    apply_logging_config:
                        ----------
                        ceos1:
```

(continues on next page)

(continued from previous page)

```
                    ----------
                    netmiko_send_config:
                        ----------
                        changed:
                            True
                        connection_retry:
                            0
                        diff:
                        exception:
                            None
                        failed:
                            False
                        result:
                            configure terminal
                            ceos1(config)#logging host 5.5.5.5
                            ceos1(config)#end
                            ceos1#
                        task_retry:
                            0
                ceos2:
                    ----------
                    netmiko_send_config:
                        ----------
                        changed:
                            True
                        connection_retry:
                            0
                        diff:
                        exception:
                            None
                        failed:
                            False
                        result:
                            configure terminal
                            ceos2(config)#logging host 5.5.5.5
                            ceos2(config)#end
                            ceos2#
                        task_retry:
                            0
summary:
    ----------
    ceos1:
        |_
          ----------
          apply_logging_config:
              PASS
    ceos2:
        |_
          ----------
          apply_logging_config:
              PASS
```

If `sumtable` set to True in workflow's options, summary report section formatted as text table:

```
summary:
    Headers:
    (1) collect_clock
    (2) collect_version
    (3) sleep_1_second
    (4) collect_clock

        host    1     2     3     4
    --  ------  ----  ----  ----  ----
     0  ceos1   PASS  PASS  PASS  PASS
     1  ceos2   PASS  PASS  PASS  PASS
```

Or, if `sumtable` value set to `jira` string:

```
summary:
    Headers:
    (1) collect_clock
    (2) collect_version
    (3) sleep_1_second
    (4) collect_clock

    ||    || host   || 1    || 2    || 3    || 4    ||
    |  0 | ceos1  | PASS | PASS | PASS | PASS |
    |  1 | ceos2  | PASS | PASS | PASS | PASS |
```

CHAPTER

EIGHT

FAQ

Collection of answers to Frequently Asked Question

8.1 How to refresh Nornir Proxy Pillar?

Calling `pillar.refresh` will not update running Nornir instance. Instead, after updating pillar on salt-master, to propagate updates to proxy-minion process either of these will work:

- restart Nornir proxy-minion process e.g. `systemctl restart salt-proxy@nrp1.service`
- run `salt nrp1 nr.nornir refresh` command to re-initialize Nornir

where `nrp1` - Nornir Proxy minion id/name.

8.2 How to target individual hosts behind nornir proxy minion?

To address individual hosts targeting, Nornir filtering capabilities utilized using additional filtering functions, reference nornir-salt module FFun function for more information. But in short have to use `Fx` parameters to filter hosts, for example:

```
# target only IOL1 and IOL2 hosts:
salt nrp1 nr.cli "show clock" FB="IOL[12]"
```

CHAPTER

NINE

USE CASES

9.1 Doing one-liner configuration changes

With NAPALM plugin:

```
salt nrp1 nr.cfg "logging host 1.1.1.1" "ntp server 1.1.1.2" plugin=napalm
```

With Netmiko plugin:

```
salt nrp1 nr.cfg "logging host 1.1.1.1" "ntp server 1.1.1.2" plugin=netmiko
```

With Scrapli plugin:

```
salt nrp1 nr.cfg "logging host 1.1.1.1" "ntp server 1.1.1.2" plugin=scrapli
```

With PyATS plugin:

```
salt nrp1 nr.cfg "logging host 1.1.1.1" "ntp server 1.1.1.2" plugin=pyats
```

Make sure that device configured accordingly and NAPALM or Scrapli or Netmiko can interact with it, e.g. for NAPALM SCP server enabled on Cisco IOS or Scrapli library installed on minion machine or hosts' connection options specified in inventory, e.g.:

```
hosts:
  IOL1:
    hostname: 192.168.217.10
```

(continues on next page)

(continued from previous page)

```
    platform: ios
    groups: [lab]
    connection_options:
      scrapli:
        platform: cisco_iosxe
        port: 22
        extras:
          ssh_config_file: True
          auth_strict_key: False
```

9.2 Using Jinja2 templates to generate and apply configuration

Data defined in pillar file `/etc/salt/pillar/nrp1.sls`:

```
hosts:
  IOL1:
    hostname: 192.168.217.10
    platform: ios
    groups: [lab]
    data:
      syslog: ["1.1.1.1", "2.2.2.2"]
  IOL2:
    hostname: 192.168.217.7
    platform: ios
    groups: [lab]
    data:
      syslog: ["1.1.1.2", "2.2.2.1"]

groups:
  lab:
    username: nornir
    password: nornir
```

Combine data with template file `/etc/salt/template/nr_syslog_cfg.j2`:

```
hostname {{ host.name }}
!
{%- for server in host.syslog %}
logging host {{ server }}
{%- endfor %}
```

Hosts' data injected in templates under `host` variable.

First, optional, generate configuration using `nr.cfg_gen` function without applying it to devices:

```
[root@localhost /]# salt nrp1 nr.cfg_gen filename=salt://templates/nr_syslog_cfg.j2
nrp1:
    ----------
    IOL1:
        ----------
        Rendered salt://templates/nr_syslog_cfg.j2 config:
```

(continues on next page)

(continued from previous page)

```
            hostname IOL1
            !
            logging host 1.1.1.1
            logging host 2.2.2.2
    IOL2:
        ----------
        Rendered salt://templates/nr_syslog_cfg.j2 config:
            hostname IOL2
            !
            logging host 1.1.1.2
            logging host 2.2.2.1
[root@localhost /]#
```

If configuration looks ok, can apply it to devices:

```
[root@localhost /]# salt nrp1 nr.cfg filename=salt://templates/nr_syslog_cfg.j2␣
↪plugin=netmiko
nrp1:
    ----------
    IOL1:
        ----------
        netmiko_send_config:
            ----------
            changed:
                True
            diff:
            exception:
                None
            failed:
                False
            result:
                configure terminal
                Enter configuration commands, one per line.  End with CNTL/Z.
                IOL1(config)#hostname IOL1
                IOL1(config)#!
                IOL1(config)#logging host 1.1.1.1
                IOL1(config)#logging host 2.2.2.2
                IOL1(config)#end
    IOL2:
        ----------
        netmiko_send_config:
            ----------
            changed:
                True
            diff:
            exception:
                None
            failed:
                False
            result:
                IOL2#configure terminal
                IOL2(config)#hostname IOL2
```

(continues on next page)

(continued from previous page)

```
                IOL2(config)#!
                IOL2(config)#logging host 1.1.1.2
                IOL2(config)#logging host 2.2.2.1
                IOL2(config)#end
                IOL2#
```

Verify configuration applied:

```
[root@localhost /]# salt nrp1 nr.cli "show run | inc logging"
nrp1:
    ----------
    IOL1:
        ----------
        show run | inc logging:
            logging host 1.1.1.1
            logging host 2.2.2.2
    IOL2:
        ----------
        show run | inc logging:
            logging host 1.1.1.2
            logging host 2.2.2.1
```

9.3 Using Nornir state module to do configuration changes

Sample Salt Master configuration excerpt defining base environment pillar and states location, file `/etc/salt/master` snippet:

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

pillar_roots:
  base:
    - /etc/salt/pillar
...
```

Define data in pillar file `/etc/salt/pillar/nrp1.sls`:

```
hosts:
  IOL1:
    hostname: 192.168.217.10
    platform: ios
    groups: [lab]
    data:
      syslog: ["1.1.1.1", "2.2.2.2"]
  IOL2:
    hostname: 192.168.217.7
    platform: ios
```

(continues on next page)

(continued from previous page)

```
    groups: [lab]
    data:
      syslog: ["1.1.1.2", "2.2.2.1"]

groups:
  lab:
    username: nornir
    password: nornir
```

Jinja2 template used with state to configure syslog servers, file `salt://templates/nr_syslog_cfg.j2` same as absolute path `/etc/salt/template/nr_syslog_cfg.j2`:

```
hostname {{ host.name }}
!
{%- for server in host.syslog %}
logging host {{ server }}
{%- endfor %}
```

SaltStack State file `/etc/salt/states/nr_cfg_syslog_and_ntp_state.sls` content:

```
# apply logging configuration using jinja2 template
configure_logging:
  nr.cfg:
    - filename: salt://templates/nr_syslog_cfg.j2
    - plugin: netmiko

# apply NTP servers configuration using inline commands
configure_ntp:
  nr.task:
    - plugin: nornir_netmiko.tasks.netmiko_send_config
    - config_commands: ["ntp server 7.7.7.7", "ntp server 7.7.7.8"]

# save configuration using netmiko_save_config task plugin
save_configuration:
  nr.task:
    - plugin: nornir_netmiko.tasks.netmiko_save_config
```

Run `state.apply` command to apply state to devices:

```
[root@localhost /]# salt nrp1 state.apply nr_cfg_syslog_and_ntp_state
nrp1:
----------
          ID: configure_logging
    Function: nr.cfg
      Result: True
     Comment:
     Started: 12:45:41.339857
    Duration: 2066.863 ms
     Changes:
              ----------
              IOL1:
                  ----------
                  netmiko_send_config:
```

(continues on next page)

(continued from previous page)

```
                    ----------
                    changed:
                        True
                    diff:
                    exception:
                        None
                    failed:
                        False
                    result:
                        configure terminal
                        Enter configuration commands, one per line.  End with CNTL/Z.
                        IOL1(config)#hostname IOL1
                        IOL1(config)#!
                        IOL1(config)#logging host 1.1.1.1
                        IOL1(config)#logging host 2.2.2.2
                        IOL1(config)#end
            IOL2:
                ----------
                netmiko_send_config:
                    ----------
                    changed:
                        True
                    diff:
                    exception:
                        None
                    failed:
                        False
                    result:
                        configure terminal
                        Enter configuration commands, one per line.  End with CNTL/Z.
                        IOL2(config)#hostname IOL2
                        IOL2(config)#!
                        IOL2(config)#logging host 1.1.1.2
                        IOL2(config)#logging host 2.2.2.1
                        IOL2(config)#end
                        IOL2#
----------
          ID: configure_ntp
    Function: nr.task
      Result: True
     Comment:
     Started: 12:45:43.407745
    Duration: 717.144 ms
     Changes:
              ----------
              IOL1:
                  ----------
                  nornir_netmiko.tasks.netmiko_send_config:

                      IOL1#configure terminal
                      IOL1(config)#ntp server 7.7.7.7
                      IOL1(config)#ntp server 7.7.7.8
```

(continues on next page)

(continued from previous page)

```
                    IOL1(config)#end
            IOL2:
                ----------
                nornir_netmiko.tasks.netmiko_send_config:
                    configure terminal
                    Enter configuration commands, one per line.  End with CNTL/Z.
                    IOL2(config)#ntp server 7.7.7.7
                    IOL2(config)#ntp server 7.7.7.8
                    IOL2(config)#end
                    IOL2#
----------
          ID: save_configuration
    Function: nr.task
      Result: True
     Comment:
     Started: 12:45:44.126463
    Duration: 573.964 ms
     Changes:
              ----------
              IOL1:
                  ----------
                  nornir_netmiko.tasks.netmiko_save_config:
                      write mem
                      Building configuration...
                      [OK]
                      IOL1#
              IOL2:
                  ----------
                  nornir_netmiko.tasks.netmiko_save_config:
                      write mem
                      Building configuration...
                        [OK]
                      IOL2#

Summary for nrp1
------------
Succeeded: 3 (changed=3)
Failed:    0
------------
Total states run:     3
Total run time:   3.358 s
[root@localhost /]#
```

9.4 Sending Nornir stats to Elasticsearch and visualizing in Grafana

To send stats about Nornir proxy operation using returners need to define scheduler to periodically call `nr.stats` function using returner of choice.

Scheduler configuration in proxy minion pillar file `/etc/salt/pillar/nrp1.sls`:

```
schedule:
  stats_to_elasticsearch:
    function: nr.nornir
    args:
      - stats
    seconds: 60
    return_job: False
    returner: elasticsearch
```

Sample Elasticsearch cluster configuration defined in Nornir Proxy minion pillar, file `/etc/salt/pillar/nrp1.sls`:

```
elasticsearch:
  host: '10.10.10.100:9200'
```

Reference documentation for more details on Elasticsearch returner and module configuration.

If all works well, should see new `salt-nr_nornir-v1` indice created in Elasticsearch database:

```
[root@localhost ~]# curl 'localhost:9200/_cat/indices?v'
health status index                    uuid                   pri rep docs.count docs.
↪deleted store.size pri.store.size
green  open   salt-nr_nornir-v1         p4w66-12345678912345   1   0      14779
↪  0      6.3mb          6.3mb
```

Sample document entry:

```
[root@localhost ~]# curl -XGET 'localhost:9200/salt-nr_nornir-v1/_search?pretty' -H
↪'Content-Type: application/json' -d '
> {
> "size" : 1,
> "query": {
> "match_all": {}
> },
> "sort" : [{"@timestamp":{"order": "desc"}}]
> }'
{
  "took" : 774,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 10000,
```

(continues on next page)

(continued from previous page)


```
      "relation" : "gte"
    },
    "max_score" : null,
    "hits" : [
      {
        "_index" : "salt-nr_nornir-v1",
        "_type" : "default",
        "_id" : "12345678",
        "_score" : null,
        "_source" : {
          "@timestamp" : "2021-02-13T22:56:53.294947+00:00",
          "success" : true,
          "retcode" : 0,
          "minion" : "nrp1",
          "fun" : "nr.stats",
          "jid" : "20210213225653251137",
          "counts" : { },
          "data" : {
            "proxy_minion_id" : "nrp1",
            "main_process_is_running" : 1,
            "main_process_start_time" : 1.6131744901391668E9,
            "main_process_start_date" : "Sat Feb 13 11:01:30 2021",
            "main_process_uptime_seconds" : 82523.12118172646,
            "main_process_ram_usage_mbyte" : 151.26,
            "main_process_pid" : 17031,
            "main_process_host" : "vm1.lab.local",
            "jobs_started" : 1499,
            "jobs_completed" : 1499,
            "jobs_failed" : 0,
            "jobs_job_queue_size" : 0,
            "jobs_res_queue_size" : 0,
            "hosts_count" : 12,
            "hosts_connections_active" : 38,
            "hosts_tasks_failed" : 0,
            "timestamp" : "Sun Feb 14 09:56:53 2021",
            "watchdog_runs" : 2748,
            "watchdog_child_processes_killed" : 6,
            "watchdog_dead_connections_cleaned" : 0,
            "child_processes_count" : 0
          }
        },
        "sort" : [
          1613257013294
        ]
      }
    ]
  }
}
```


Elasticsearch can be polled with Grafana to visualize stats, reference Grafana documentation for details.

9.5 Using runner to work with inventory information and search for hosts

Problem Statement - has 100 Nornir Proxy Minions managing 10000 devices, how do I know which device managed by which proxy.

Solution - Nornir-runner `nr.inventory` function can be used to present brief summary about hosts:

```
# find which Nornir Proxy minion manages IOL1 device
[root@localhost /]# salt-run nr.inventory IOL1
+---+--------+----------+----------------+----------+--------+
|   | minion | hostname |       ip       | platform | groups |
+---+--------+----------+----------------+----------+--------+
| 0 |  nrp1  |   IOL1   | 192.168.217.10 |   ios    |  lab   |
+---+--------+----------+----------------+----------+--------+
```

9.6 Calling task plugins using nr.task

Any task plugin supported by Nornir can be called using `nr.task` execution module function providing that plugins installed and can be imported.

For instance calling task:

```
salt nrp1 nr.task "nornir_netmiko.tasks.netmiko_save_config"
```

internally is equivalent to running this code:

```
from nornir_netmiko.tasks import netmiko_save_config

result = nr.run(task=netmiko_save_config, *args, **kwargs)
```

where `args` and `kwargs` are arguments supplied on cli.

9.7 Targeting devices behind Nornir proxy

Nornir uses `nornir-salt` package to provide targeting capabilities built on top of Nornir module itself. Because of that it is good idea to read FFun function documentation first.

Combining SaltStack and `nornir-salt` targeting capabilities can help to address various usecase.

Examples:

```
# targeting all devices behind Nornir proxies:
salt -I "proxy:proxytype:nornir" nr.cli "show clock" FB="*"

# target all Cisco IOS devices behind all Nornir proxies
salt -I "proxy:proxytype:nornir" nr.cli "show clock" FO='{"platform": "ios"}'

# target all Cisco IOS or NXOS devices behind all Nornir proxies
salt -I "proxy:proxytype:nornir" nr.cli "show clock" FO='{"platform__any": ["ios", "nxos_
↪ssh"]}'
```

(continues on next page)

(continued from previous page)

```
# targeting All Nornir Proxies with ``LON`` in name and all hosts behind them that has␣
↪``core`` in their name
salt "*LON*" nr.cli "show clock" FB="*core*"

# targeting all hosts that has name ending with ``accsw1``
salt -I "proxy:proxytype:nornir" nr.cli "show clock" FB="*accsw1"
```

By default Nornir does not use any filtering and simply runs task against all devices. But Nornir proxy minion configuration `nornir_filter_required` parameter allows to alter default behavior to opposite resulting in exception if no `Fx` filter provided.

9.8 Saving task results to files on a per-host basis

`ToFileProcessor` distributed with `nornir_salt` package can be used to save execution module functions results to the file system of machine where proxy-minion process running.

Sample usage:

```
[root@localhost /]# salt nrp1 nr.cli "show clock" "show ip int brief" tf="show_commands_
↪output"
nrp1:
    ----------
    IOL1:
        ----------
        show clock:
            *12:05:06.633 EET Sun Feb 14 2021
        show ip int brief:
            Interface                  IP-Address      OK? Method Status                ␣
↪Protocol
            Ethernet0/0                unassigned      YES NVRAM  up                    ␣
↪up
            Ethernet0/0.102            10.1.102.10     YES NVRAM  up                    ␣
↪up
            Ethernet0/0.107            10.1.107.10     YES NVRAM  up                    ␣
↪up
            Ethernet0/0.2000           192.168.217.10  YES NVRAM  up                    ␣
↪up
            Ethernet0/1                unassigned      YES NVRAM  up                    ␣
↪up
            Ethernet0/2                unassigned      YES NVRAM  up                    ␣
↪up
            Ethernet0/3                unassigned      YES NVRAM  administratively down␣
↪down
            Loopback0                  10.0.0.10       YES NVRAM  up                    ␣
↪up
            Loopback100                1.1.1.100       YES NVRAM  up                    ␣
↪up
    IOL2:
        ----------
        show clock:
```

(continues on next page)

(continued from previous page)

```
            *12:05:06.605 EET Sun Feb 14 2021
        show ip int brief:
            Interface                  IP-Address      OK? Method Status                ␣
↪Protocol
            Ethernet0/0                unassigned      YES NVRAM  up                    ␣
↪up
            Ethernet0/0.27             10.1.27.7       YES NVRAM  up                    ␣
↪up
            Ethernet0/0.37             10.1.37.7       YES NVRAM  up                    ␣
↪up
            Ethernet0/0.107            10.1.107.7      YES NVRAM  up                    ␣
↪up
            Ethernet0/0.117            10.1.117.7      YES NVRAM  up                    ␣
↪up
            Ethernet0/0.2000           192.168.217.7   YES NVRAM  up                    ␣
↪up
            Ethernet0/1                unassigned      YES NVRAM  administratively down␣
↪down
            Ethernet0/2                unassigned      YES NVRAM  administratively down␣
↪down
            Ethernet0/3                unassigned      YES NVRAM  administratively down␣
↪down
            Loopback0                  10.0.0.7        YES NVRAM  up                    ␣
↪up

[root@localhost /]# tree /var/salt-nornir/nrp1/files/
├── show_commands_output__11_July_2021_07_11_26__IOL1.txt
├── show_commands_output__11_July_2021_07_11_26__IOL2.txt
├── tf_aliases.json

[root@localhost /]# cat /var/salt-nornir/nrp1/files/show_commands_output__11_July_2021_
↪07_11_26__IOL1.txt
*12:05:06.633 EET Sun Feb 14 2021
Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                unassigned      YES NVRAM  up                    up
Ethernet0/0.102            10.1.102.10     YES NVRAM  up                    up
Ethernet0/0.107            10.1.107.10     YES NVRAM  up                    up
Ethernet0/0.2000           192.168.217.10  YES NVRAM  up                    up
Ethernet0/1                unassigned      YES NVRAM  up                    up
Ethernet0/2                unassigned      YES NVRAM  up                    up
Ethernet0/3                unassigned      YES NVRAM  administratively down down
Loopback0                  10.0.0.10       YES NVRAM  up                    up
Loopback100                1.1.1.100       YES NVRAM  up                    up
```

CHAPTER

TEN

PILLAR AND INVENTORY EXAMPLES

This section contains samples that could serve as a starting point to construct pillar for Nornir Proxy Minion on salt-master machine.

10.1 Arista cEOS

Below inventory can be used with Arista cEOS and contains parameters for these connection plugins:

- Netmiko - uses SSH and platform `arista_eos` under base arguments definition
- NAPALM - uses Arista eAPI over HTTP port 80 and platform `eos` as specified in eos_params group's `connection_options`
- Ncclient - as specified in eos_params group's `connection_options` uses port 830 with default device type
- PyATS - for ceos1 has multiple connections defined, including a pool of 3 connections for `vty_1` connection. For ceos2 parameters sourced from base arguments.
- HTTP - uses port 6020 over HTTPS as specified in eos_params group's `connection_options`
- Scrapli - uses SSH without verifying the keys, platform `arista_eos` as specified in eos_params group's `connection_options`
- Scrapli-Netconf - uses SSH paramiko transport with port 830 as specified in eos_params group's `connection_options`
- PyGNMI - uses port 6030 allowing insecure connection as specified in eos_params group's `connection_options`

```
proxy:
  proxytype: nornir
  multiprocessing: True

hosts:
  ceos1:
    hostname: 10.0.1.4
```

(continues on next page)

(continued from previous page)

```
    platform: arista_eos
    groups: [creds, eos_params]
    connection_options:
      pyats:
        extras:
          devices:
            ceos1:
              os: eos
              credentials:
                default:
                  username: nornir
                  password: nornir
              connections:
                default:
                  protocol: ssh
                  ip: 10.0.1.4
                  port: 22
                vty_1:
                  protocol: ssh
                  ip: 10.0.1.4
                  pool: 3

  ceos2:
    hostname: 10.0.1.5
    platform: arista_eos
    groups: [creds, eos_params]
    connection_options:
      pyats:
        platform: eos
        extras:
          devices:
            ceos2: {}

groups:
  creds:
    username: nornir
    password: nornir
  eos_params:
    connection_options:
      scrapli:
        platform: arista_eos
        extras:
          auth_strict_key: False
          ssh_config_file: False
      scrapli_netconf:
        port: 830
        extras:
          ssh_config_file: True
          auth_strict_key: False
          transport: paramiko
          transport_options:
            # refer to https://github.com/saltstack/salt/issues/59962 for details
```

(continues on next page)

(continued from previous page)

```
        # on why need netconf_force_pty False
        netconf_force_pty: False
    napalm:
      platform: eos
      extras:
        optional_args:
          transport: http
          port: 80
    ncclient:
      port: 830
      extras:
        allow_agent: False
        hostkey_verify: False
    http:
      port: 6020
      extras:
        transport: https
        verify: False
        base_url: "restconf/data"
        headers:
          Content-Type: "application/yang-data+json"
          Accept: "application/yang-data+json"
    pygnmi:
      port: 6030
      extras:
        insecure: True
```

10.2 Cisco IOS-XE

Below inventory can be used with Cisco IOSXE based devices and contains parameters for these connection plugins:

- Netmiko - uses SSH and platform `cisco_ios` under base arguments definition
- PyATS - uses `iosxe` platform with SSH protocol on port 22 as specified in `connection_options`
- HTTP - uses HTTPS transport on port 443 with base url "restconf/data" as specified in `connection_options`
- Ncclient - uses port 830 with platform name `iosxe` as specified in `connection_options`
- Scrapli-Netconf - uses port 830 with paramiko transport as specified in `connection_options`
- NAPALM - uses SSH and platform `ios` as specified in `connection_options`
- Scrapli - uses SSH and platform `cisco_iosxe` without verifying SSH keys as specified in `connection_options`

```
proxy:
  proxytype: nornir
  multiprocessing: True

hosts:
  csr1000v-1:
    hostname: sandbox-iosxe-latest-1.cisco.com
```

(continues on next page)

(continued from previous page)

```
  platform: cisco_ios
  username: developer
  password: C1sco12345
  port: 22
  connection_options:
    pyats:
      extras:
        devices:
          csr1000v-1:
            os: iosxe
            connections:
              default:
                ip: 131.226.217.143
                protocol: ssh
                port: 22
    napalm:
      platform: ios
    scrapli:
      platform: cisco_iosxe
      extras:
        auth_strict_key: False
        ssh_config_file: False
    http:
      port: 443
      extras:
        transport: https
        verify: False
        base_url: "restconf/data"
        headers:
          Content-Type: "application/yang-data+json"
          Accept: "application/yang-data+json"
    ncclient:
      port: 830
      extras:
        allow_agent: False
        hostkey_verify: False
        device_params:
          name: iosxe
    scrapli_netconf:
      port: 830
      extras:
        transport: paramiko
        ssh_config_file: True
        auth_strict_key: False
        transport_options:
          netconf_force_pty: False
```

10.3 Cisco IOSXR

Below inventory can be used with Cisco IOSXR based devices and contains parameters for these connection plugins:

- Netmiko - uses SSH and platform `cisco_xr` under base arguments definition
- Ncclient - uses port 830 with platform name `iosxr` as specified in `connection_options`
- Scrapli-Netconf - uses port 830 as specified in `connection_options`
- NAPALM - uses SSH and platform `iosxr` as specified in `connection_options`
- Scrapli - uses SSH and platform `cisco_iosxr` without verifying SSH keys as specified in `connection_options`
- PyATS - uses `iosxr` platform with SSH protocol on port 22 as specified in `connection_options`

```
proxy:
  proxytype: nornir
  multiprocessing: True

hosts:
  iosxr1:
    hostname: sandbox-iosxr-1.cisco.com
    platform: cisco_xr
    username: admin
    password: "C1sco12345"
    port: 22
    connection_options:
      pyats:
        extras:
          devices:
            iosxr1:
              os: iosxr
              connections:
                default:
                  ip: 131.226.217.150
                  protocol: ssh
                  port: 22
      napalm:
        platform: iosxr
      scrapli:
        platform: cisco_iosxr
        extras:
          auth_strict_key: False
          ssh_config_file: False
      ncclient:
        port: 830
        extras:
          allow_agent: False
          hostkey_verify: False
          device_params:
            name: iosxr
      scrapli_netconf:
        port: 830
        extras:
```

(continues on next page)

(continued from previous page)

```
          ssh_config_file: True
          auth_strict_key: False
          transport_options:
            netconf_force_pty: False
```

10.4 Cisco NXOS

Below inventory can be used with Cisco NXOS based devices and contains parameters for these connection plugins:

- Netmiko - uses SSH and platform `nxos_ssh` under base arguments definition
- Ncclient - uses port 830 with platform name `nexus` as specified in `connection_options`
- Scrapli-Netconf - uses port 830 as specified in `connection_options`
- NAPALM - uses SSH and platform `nxos_ssh` as specified in `connection_options`
- Scrapli - uses SSH and platform `cisco_nxos` without verifying SSH keys as specified in `connection_options`
- PyATS - uses `nxos` platform with SSH protocol on port 22 as specified in `connection_options`

```
proxy:
  proxytype: nornir
  multiprocessing: True

hosts:
  sandbox-nxos-1.cisco:
    hostname: sandbox-nxos-1.cisco.com
    platform: nxos_ssh
    username: admin
    password: "Admin_1234!"
    port: 22
    connection_options:
      pyats:
        extras:
          devices:
            sandbox-nxos-1.cisco:
              os: nxos
              connections:
                default:
                  ip: 131.226.217.151
                  protocol: ssh
                  port: 22
      napalm:
        platform: nxos_ssh
      scrapli:
        platform: cisco_nxos
        extras:
          auth_strict_key: False
          ssh_config_file: False
      ncclient:
        port: 830
        extras:
```

(continues on next page)

(continued from previous page)

```
        allow_agent: False
        hostkey_verify: False
        device_params:
          name: nexus
    scrapli_netconf:
      port: 830
      extras:
        ssh_config_file: True
        auth_strict_key: False
        transport_options:
          netconf_force_pty: False
```

PYTHON MODULE INDEX

S

INDEX

Symbols

C

D

E

F

G

H

I

K

L

M

N

P

Q

R

S

T

W