Netbox$_{N}etdev_{I}nventory$

Release 0.2.3

Oct 10, 2019

Contents:

1 Quickstart **3**
- 1.1 Installation . . . 3
- 1.2 Configuration . . . 4
- 1.3 Device list . . . 5
- 1.4 Filter . . . 5
- 1.5 Import and interconnect . . . 7

2 Import devices data **9**
- 2.1 Create a device in Netbox . . . 9
- 2.2 Usage . . . 9
- 2.3 Configuration . . . 12
- 2.4 Data imported . . . 12

3 Interconnect devices **15**
- 3.1 Usage . . . 15
- 3.2 Configuration . . . 17
- 3.3 Neighbours finding . . . 18

4 Compatibility and specific custom parsers **21**
- 4.1 Compatibility . . . 21
- 4.2 List of specific parsers . . . 21
- 4.3 Napalm only features . . . 21

5 Indices and tables **23**

netbox-netprod-importer is a tool dedicated to help reflecting your production in Netbox as an IPAM/DCIM, independently of your information system. It connects to a given list of network devices and parse their status and configuration to import them into Netbox like they are currently configured.

It is thought to be generic and infrastructure agnostic. It means that imported data will probably need to be adapted by some custom scripts, like the specification of roles, tennant and other properties on objects.

To be the most platform agnostic as possible, data are fetched through Napalm, with some custom parsers when more info are needed.

CHAPTER 1

Quickstart

Netbox should reflect the status of your production. Its philosophy is that your production should be configured related to Netbox, but Netbox should not be synced from what is currently running.

However, moving to Netbox can be complicated depending on the current knowledge base. For this case, if you trust how your production is configured, Netbox can be populated the 1st time from what is currently running, to then make Netbox the single source of truth and base the production around it.

netbox-netdev-inventory has 2 main functions:

- *import devices data*
- *interconnect*

Import will fetch the current status of a list of devices. Interconnect will build a graph of neighbours to create connections between each other inside Netbox.

Table of Contents

- *Quickstart*
  - *Installation*
  - *Configuration*
  - *Device list*
  - *Filter*
    - *Example*
  - *Import and interconnect*

1.1 Installation

Run:

```
pip3 install netbox_netdev_inventory
```

Or by using setuptools:

```
python3 ./setup.py install
```

netbox-netdev-inventory is tested under python 3.4 to 3.7

1.2 Configuration

The configuration is quite minimal yaml file:

```
########################
#### Global options ####
########################

## Be more verbose ##
verbose: None

## Disable ssl warnings in urllib3 ##
disable_ssl_warnings: False


################
#### Netbox ####
################

netbox:
  # Netbox API URL
  url: "https://netbox.tld/api"
  # username: "user"
  # password: "password"
  # or to use a token instead
  token: "CHANGEME"


#########################
#### Interconnections ####
#########################

# On some devices, LLDP will expose the host FQDN. If devices are stored on
# Netbox only by their hostname, the interconnection process will not be able
# to find them. Fill this list to strip the domain name from exposed names.
remove_domains:
  - "foo.tld"
  - "bar.tld"

# vim: set ts=2 sw=2:
```

Adapt it and save it either as:

- *~/.config/netbox-netdev-inventory/config.yml*
- */etc/netbox-netdev-inventory/config.yml*

Or can be set with the environment variable `CONFIG_PATH`. Example: `CONFIG_PATH=./config.yml netbox-netdev-inventory ...`

To turn off unverified HTTPS warning messages request: *InsecureRequestWarning: Unverified HTTPS request is being made. Advised verification verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings InsecureRequestWarning)* In the configuration file, set the *disable_ssl_warnings* option to *True* Default *False*

1.3 Device list

To import the state of some devices, netbox-netdev-inventory takes a yaml that lists which hosts to target. One device is declared like the following:

```
switch-fqdn:
  # Napalm driver name to use
  driver: napalm_driver_name
  # optional. Will be used instead of the switch fqdn to init the connection
  target: some_ip
  # optional. Only needed for interconnect
  discovery_protocol: lldp, cdp or multiple
```

Read the documentation of each subparser to use it in netbox-netdev-inventory.

discovery_protocol can take the values “lldp”, “cdp” or “multiple”. Since the CDP protocol is proprietary, it is only supported by CISSCO equipment. CDP detection only works with nxos, nxos_ssh and ios drivers.

1.4 Filter

To import the status of some devices, netbox-netdev-inventory accepts yaml, which lists the criteria for selecting devices to target. It looks like this:

```
#Mandatory section, but may be empty.
#Used with interconnect and inventory.
discovery_protocol:
    #[driver]: [discovery protocol]
    ios: cdp
    nxos: multiple
    nxos_ssh: multiple
    junos: lldp

#Filter section, device selection criteria are prescribed.
filter:
    q:
    region:
        - england
    site:
        - london
        - birmingham
    rack:
    status: 1
    role:
    tenant_group:
    tenant:
        - it
    manufacturer:
        - cisco
```

(continues on next page)

(continued from previous page)

```
    device_type:
    mac_address:
    has_primary_ip: True
    platform:
    virtual_chassis_member:
    console_ports:
    console_server_ports:
    power_ports:
    power_outlets:
    interfaces:
    pass_through_ports:
```

Full online documentation on filter keys is available on a running NetBox instance in /api/docs/, section GET /dcim/devices/ Most filter keys accept slug input

Mandatory in the platform you need to specify the NAPALM driver

1.4.1 Example

3 switches are wanted to be imported:

- *switch-1.foo.tld*, which is a Cisco Nexus. The IP to target will be deduced by resolving the fqdn/hostname.
- *switch-2.bar.tld*, which is a Juniper. *switch-2.bar.tld* does not resolve, so an IPv4 will be specified as target.
- *switch-3.foo.tld*, which is a Cisco Nexus. The IP to target will be deduced by resolving the fqdn/hostname. And also determine the interconnect via cdp. The cdp protocol works so far with nxos, nxos_ssh and ios
- *switch-4.foo.tld*, which is a Cisco Nexus. The IP to target will be deduced by resolving the fqdn/hostname. And also determine the interconnect via cdp and lldp. The *multiple* option only works for nxos, nxos_ssh and ios.

To declare 2 switches, define a yaml named *devices.yaml*:

```
switch-1.foo.tld:
  driver: "nxos_ssh"

switch-2.bar.tld:
  driver: "junos"
  target: "192.0.2.3"

switch-3.foo.tld:
  driver: "nxos"
  discovery_protocol: "cdp"

switch-4.foo.tld:
  driver: "nxos"
  discovery_protocol: "multiple"
```

Then to use it:

```
$ netbox-netdev-inventory import -f devices.yaml
```

1.5 Import and interconnect

Import is meant to import the state of some devices, like creating their interfaces, attaching their IP, etc. The complete documentation and list of feature can be found *here*.

Import a list of devices:

```
$ netbox-netdev-inventory import -f devices.yaml
```

Once all devices interfaces are created, with the previous command, neighbours can be discovered and interconnected between each other:

```
$ netbox-netdev-inventory interconnect -f devices.yaml
```

Full documentation for the interconnect feature can be found *here*.

You can also run an inventory, which first starts the import and then the interconnect:

```
$ netbox-netdev-inventory inventory -F filter.yaml
```

CHAPTER 2

Import devices data

The importer goal is to import the state and data of a network device. It is not meant to create a device or magically rack it, but to populate it as it is currently configured. It is based on Napalm to be platform agnostic, when possible, but uses some custom specific parsers when needed.

Table of Contents

- *Import devices data*
  - *Create a device in Netbox*
  - *Usage*
    - *Example*
  - *Configuration*
  - *Data imported*
    - *Interface form factor*
    - *IP addresses and VRF*

2.1 Create a device in Netbox

Before importing the data of a device, it should be created in Netbox. netbox-netdev-inventory will not create a device for the user, as it is difficult to do so by staying infrastructure agnostic. It just needs a hostname and all fields required by Netbox, the rest being part of the listed features will be populated by netbox-netdev-inventory.

2.2 Usage

An import can be started through the subcommand `import`:

```
usage: netbox-netdev-inventory import [-h] [-u user] [-p] [-t THREADS] [--overwrite]␣
↪[-v LEVEL] [ -f DEVICES | -F FILTER ]

arguments:
  -f devices, --file devices
                        Yaml file containing a definition of devices to poll
  or
  -F FILTER, --filter FILTER
                        Yaml file containing the device definition filter for polling␣
↪from NetBox

optional arguments:
  -h, --help            show this help message and exit
  --overwrite           overwrite devices already pushed
  -u user, --user user  user to use for connections to the devices
  -p, --password        ask for credentials for connections to the devices
  -P PASSWORD, --Password PASSWORD
                        credentials for connections to the devices
  -t THREADS, --threads THREADS
                        number of threads to run
  -v LEVEL, --verbose LEVEL
                        verbose output debug, info, warning, error and
                        critical, default: error
```

By default, connecting to the devices will use the default authentication mechanism of the napalm driver, which is normally the current user and no password/authentication by key. To change this behavior, the `-u/--user` and `-p/--password|-P/--Password` options can be used to specify the user to use, and tells the importer to asklset for the password to use.

The import is multithreaded, and split by device. The default number of threads is 10, but can be changed with the `-t/--threads` option.

Importing a device will replace the current data in Netbox, but not clean (by default) what has not been found by fetching the device state. If a device is already populated in Netbox, network interfaces already added but not found during the import will not be cleaned, same as the IP addresses that do not seem to be configured anymore. This behavior can be changed by enabling the `--overwrite` option, which will clean all interfaces and IP that have not been found during the import.

Toggle the verbose mode with the `-v/--verbose LEVEL` option to get a more verbose output. Default error.

The `devices` parameter is a yaml file, representing the devices list to import, as detailed *here*.

2.2.1 Example

Considering a yaml file `~/importer/devices.yml` containing these devices:

```
switch-1.foo.tld:
  driver: "nxos_ssh"

switch-2.bar.tld:
  driver: "junos"
  target: "192.0.2.3"
```

To simply apply the import on these devices, do:

```
$ netbox-netdev-inventory import -f ~/importer/devices.yml
```

Considering that the current user is named `foo`, if a password is needed for this user to connect to these devices, do:

```
$ netbox-netdev-inventory import -p -f ~/importer/devices.yml
```

To use a different user, for example *bar* do:

```
$ netbox-netdev-inventory import -u bar -p -f ~/importer/devices.yml
```

And to use more threads and enable the overwrite mode to get a clean clone of a device state:

```
$ netbox-netdev-inventory import -u bar -p -t 30 --overwrite -f ~/importer/devices.yml
```

Considering a yaml file `~/importer/filter.yml` containing this filter:

```
discovery_protocol:
    ios: cdp
    nxos: multiple
    nxos_ssh: multiple
    junos: lldp

filter:
    q:
    region:
        - england
    site:
        - london
        - birmingham
    rack:
    status: 1
    role:
    tenant_group:
    tenant:
        - it
    manufacturer:
        - cisco
    device_type:
    mac_address:
    has_primary_ip: True
    platform:
    virtual_chassis_member:
    console_ports:
    console_server_ports:
    power_ports:
    power_outlets:
    interfaces:
    pass_through_ports:
```

Full online documentation on filter keys is available on a running NetBox instance in /api/docs/, section GET /dcim/devices/

We will choose London and birmingham sites in England, the equipment is active, the owner is it, the manufacturer is cisco and has a primary ip:

```
$ netbox-netdev-inventory import -u bar -p -t 30 --overwrite -F ~/importer/filter.yml
```

2.3 Configuration

For the import part, the only configuration needed in your *config file* is the following one:

```
netbox:
  # Netbox API URL
  url: "https://netbox.tld/api"
  # username: "user"
  # password: "password"
  # or to use a token instead
  token: "CHANGEME"
```

It is used to get and push the fetched data from and to Netbox. This block is self documented, and is used to get the Netbox API URL and credentials.

2.4 Data imported

The importer fetch the following type of data:

- Network interfaces (physical & virtual):
  - Try to guess the interface form factor
  - MTU
  - MAC Address
  - Description
  - Parent LAG
  - Enabled/Disabled
  - IPv4/IPv6
  - Vlan (only cisco equipment)
  - 802.1Q Mode (only cisco equipment)
- Serial number
- Main IPv4/IPv6

2.4.1 Interface form factor

netbox-netdev-inventory can find the form factor by fetching it from the device and by selecting the matching type on Netbox. A form factor can be for example 1000Base-T, SFP, SFP+, etc.

To correctly detect the interface type, the platform of the targetted device needs to be fully supported by the importer. Some parsers are written to get more info than what napalm allows (read *the documentation about specific parsers* for more details), and are used by the importer.

When an interface type can be fetched from a device, it has then to be translated as a type expected by Netbox. To do so, a list of regexp are written to help for the mapping. This list is certainly incomplete, so someone seeing an unhandled case is welcomed to open an issue about it.

2.4.2 IP addresses and VRF

IP addresses configured on an interface are imported and attached to this interface in Netbox. If an IP already exists in Netbox, it is used it and assigned it to the correct interface. If an IP does not already exist, it is created and assigned to the interface.

> **Warning:** This behavior can be an issue with anycasted ip addresses.

When an IP is part of a VRF, the VRF cannot be guessed from Netbox. As multiple VRF can be declared with the same name but a different route distinguisher, it is not easier to get the correct one and staying infrastructure agnostic. That is the reason why created IP are not assigned to any VRF. Scripts can be use to move them after the import, but the import will let the responsability on the user to do it.

> **Warning:** Be aware that some Napalm drivers do not handle well the notion of VRF. Getting the IP addresses of an interface will sometimes be limited to the default VRF.
>
> **Pull requests are opened on Napalm to fix it:**
>
> - https://github.com/napalm-automation/napalm/pull/815
> - https://github.com/napalm-automation/napalm/pull/819

CHAPTER 3

Interconnect devices

Once all network interfaces are created, the interconnection feature allows to build a graph of some devices neighbours, and create an interconnection between each other in Netbox. It is based on LLDP, CDP and napalm, plus some custom parsers to get more informations that what is fetched by the napalm drivers.

The classic workflow is to start the interconnection after importing the current states of the devices, so all network interfaces exist in Netbox.

Table of Contents

- *Interconnect devices*
  - *Usage*
    - *Example*
  - *Configuration*
  - *Neighbours finding*

3.1 Usage

The interconnections feature can be started through the subcommand `interconnect`:

```
usage: netbox-netdev-inventory interconnect [-h] [-u USER] [-p] [-t THREADS] [-v␣
↪LEVEL] [ -f DEVICES | -F FILTER ]

arguments:
  -f devices, --file devices
                        Yaml file containing a definition of devices to poll
  or
  -F FILTER, --filter FILTER
                        Yaml file containing the device definition filter for polling␣
↪from NetBox
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  -u USER, --user USER  user to use for connections to the devices
  -p, --password        ask for credentials for connections to the devices
  -P PASSWORD, --Password PASSWORD
                        credentials for connections to the devices
  -t THREADS, --threads THREADS
                        number of threads to run
  --overwrite           overwrite data already pushed
  -v LEVEL, --verbose LEVEL
                        verbose output debug, info, warning, error and
                        critical, default: error
```

By default, connecting to the devices will use the default authentication mechanism of the napalm driver, which is normally the current user and no password/authentication by key. To change this behavior, the `-u/--user` and `-p/--password` options can be used to specify the user to use, and tells netbox-netdev-inventory to ask for the password to use.

The process is multithreaded, and split by device. The default number of threads is 10, but can be changed with the `-t/--threads` option.

Interconnecting devices will not clean old connections in Netbox: if 2 interfaces are marked as connected in Netbox but are not detected as such during the neighbour search, it will be kept as it is. This behavior can be changed by enabling the `--overwrite` option, which will, on each scanned device, clean all connections that have not been found.

Toggle the verbose mode with the `-v/--verbose LEVEL` option to get a more verbose output. Default error.

The `devices` parameter is a yaml file, representing the devices list to import, as detailed *here*.

3.1.1 Example

Considering a yaml file `~/importer/devices.yml` containing these devices:

```
switch-1.foo.tld:
  driver: "nxos_ssh"

switch-2.bar.tld:
  driver: "junos"
  target: "192.0.2.3"

switch-3.foo.tld:
  driver: "nxos"
  discovery_protocol: "cdp"

switch-4.foo.tld:
  driver: "nxos"
  discovery_protocol: "multiple"
```

To simply apply the import on these devices, do:

```
$ netbox-netdev-inventory interco -f ~/importer/devices.yml
```

Considering that the current user is named `foo`, if a password is needed for this user to connect to these devices, do:

```
$ netbox-netdev-inventory interco -p  -f ~/importer/devices.yml
```

To use a different user, for example *bar* do:

```
$ netbox-netdev-inventory interco -u bar -p -f ~/importer/devices.yml
```

And to use more threads:

```
$ netbox-netdev-inventory interco -u bar -p -t 30 -f ~/importer/devices.yml
```

Listing devices from NetBox. Considering a yaml file `~/importer/filter.yml` containing this filter:

```
discovery_protocol:
    ios: cdp
    nxos: multiple
    nxos_ssh: multiple
    junos: lldp

filter:
    q:
    region:
        - england
    site:
        - london
        - birmingham
    rack:
    status: 1
    role:
    tenant_group:
    tenant:
        - it
    manufacturer:
        - cisco
    device_type:
    mac_address:
    has_primary_ip: True
    platform:
    virtual_chassis_member:
    console_ports:
    console_server_ports:
    power_ports:
    power_outlets:
    interfaces:
    pass_through_ports:
```

Full online documentation on filter keys is available on a running NetBox instance in /api/docs/, section GET /dcim/devices/

We will choose london and birmingham sites in England, the equipment is active, the owner is it, the manufacturer is cisco and has a primary ip:

```
$ netbox-netdev-inventory interco -u bar -p -t 30 --overwrite -F ~/importer/filter.yml
```

3.2 Configuration

For the import part, the configuration needed in your *config file* is the following one:

```
netbox:
  # Netbox API URL
  url: "https://netbox.tld/api"
  # username: "user"
  # password: "password"
  # or to use a token instead
  token: "CHANGEME"

# On some devices, LLDP will expose the host FQDN. If devices are stored on
# Netbox only by their hostname, the interconnection process will not be able
# to find them. Fill this list to strip the domain name from exposed names.
remove_domains:
  - "foo.tld"
  - "bar.tld"
```

The `netbox` section is used to get and push the fetched data from and to Netbox. This block is self documented, and is used to get the Netbox API URL and credentials.

As explained in the *LLDP section*, some tweaks are done to maximize the neighbours finding. On some platform, the host property inside LLDP is the fqdn when usually it contains only the hostname. The `remove_domains` option is a list of domain names to workaround it, as the interconnection algorithm will try to find the device in Netbox with and without the domain name, if the host contains it.

3.3 Neighbours finding

To discover neighbours connected to a device, LLDP is used. LLDP is a standard protocol, but is quite permissive, and manufacturers do not all expose the same information in each field. To maximize the information fetched about each neighbour, some custom parsers are done *for fully supported platforms*.

Note: To maximize the neighbours finding, use the import on all devices. This way, if a neighbour cannot be find through a device, there is some chances that the discover from the neighbour will find this same device.

To find a neighbour on Netbox, the interconnect functions will connect to the listed devices, then use LLDP to get the hostname exposed by the neighbour, its network interface name and MAC address. Some platforms will try to interpret the received values: for example, Cisco NXOS will add the domain name setup inside the router to the hostname received by LLDP. So if your device expose its fqdn, for example `switch.bar.tld`, NXOS will transform it as `switch.bar.tld.bar.tld` if `bar.tld` is its domain name. This is why the `remove_domains` option has been written, in the *config file*: if one domain listed in this option is found in the neighbour hostname, it will try to search it in Netbox without this domain name.

On some platforms, the network interface can be exposed via LLDP as aggregated. For example, Cisco can show an interface named `GigabitEthernet0/1` as `Ge0/1`, what can be an issue because netbox-netdev-inventory actually imports the full interface name (`GigabitEthernet0/1`). To help finding them in Netbox, all possible form of interface names are written inside the custom parsers, and are tested in case nothing is found.

When no interface name is exposed nor found, the interface can be searched through the exposed MAC address. It can work in most cases, but be aware that some devices can share the same MAC address on multiple interfaces: Cisco N9000 for example will have the same MAC address for all interfaces configured as layer 2 only. If multiple interfaces are found on Netbox by trying to match on their MAC address, the interconnection will fail, as the correct neighbour interface cannot be determined. This feature is permitted by the specific parsers, and platforms relying only on Napalm will not be able to do that.

Also, if you want to connect switches to servers (linux), and on bond servers or team and in netbox you enter them with MAC addresses, the search will return more than one value, and which is not known. Of course, you can check

the type of interface, but why if you can configure a normal return port_id.

Ansible task to configure:

```
- name: configure lldpd
  lineinfile:
    dest: /etc/lldpd.conf
    line: "configure ports {{ item }} lldp portidsubtype local {{ item }}"
    state: present
    backup: yes
    create: yes
  when: hostvars[inventory_hostname]['ansible_%s' | format(item)]['module'] is defined
  loop: "{{ansible_interfaces }}"
  tags:
    - config_lldp
  notify: restart lldpd
```

Tested on RedHat 6 and 7, lldpd from EPEL repository.

CHAPTER 4

Compatibility and specific custom parsers

4.1 Compatibility

The platforms of the targeted network devices have to be compatible with Napalm. A list of drivers can be found here.

Napalm, however, does not support all features needed by netbox-netdev-inventory. Because of that, some specific parsers have been written to either get more data or enhanced some features to improve the import.

netbox-netdev-inventory has been tested on:

- Cisco IOS (catalyst, 2960)
- Cisco Nexus 9000
- Cisco ASR
- JunOS devices

4.2 List of specific parsers

They can be found in `netbox_netdev_inventory/vendors/`. Fully supported devices are:

- Cisco IOS (catalyst, 2960)
- Cisco Nexus 9000
- JunOS devices

4.3 Napalm only features

When targetting a device which does not have a specific parser, the import is based on Napalm only. In that situation, here is a list of supported features:

4.3.1 Data import

Feature	Supported
Serial number	True
Main IPv4/IPv6	True

Network interfaces

Feature	Supported
Guess the interface form factor:	False
MTU	True
MAC Address	True
Description	True
Parent LAG	False
Enabled/Disabled	True
IPv4/IPv6	True

4.3.2 Interconnect

Specific parsers will fetch the MAC address of each interface, to maximize the finding when the interface name or hostname cannot be found on Netbox. They also yield a list of alternative names for an interface, allowing to deal with aggregated names.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search