NetConfig Documentation

Release 1.0

Matt Vitale

Contents

NetConfig

Master / Development Branch

1.1 What Is NetConfig?

NetConfig started out as a graphical overlay for my existing Python scripts, and I've been expanding it's features ever since. It was originally built specifically for Cisco switches, routers, and firewalls, using IOS, IOS-XE, NX-OS, and ASA operating systems. All device data is pulled in real-time via SSH and Netmiko.

NetConfig can retrieve a list of devices in one of two ways:

- Stored in a local SQLAlchemy database file
- Retrieved via API calls on an existing NetBox installation

In version 1.1, vendor neutral support was added using individual device files.

1.2 Features

NetConfig was originally built as a graphical overlay for common CLI based interactions with non-API supported Cisco networking equipment. At the core of the program is a need to access accurate, real-time information about any SSH enabled network device. NetConfig accomplishes this by refreshing all page contents each time the page is refreshed, by pulling the information via SSH at the time of the page refresh.

NetConfig provides:

- Real-time information into your network devices
- Graphical overlay for existing Network devices without support for API's or other web-based interfaces

1.3 Installation

Reference the Installation Guide section for instructions how on how to install NetConfig at readthedocs.io install guide Install instructions were written for an Ubuntu 16.04 64-bit server. NetConfig has not been tested with other OS's.
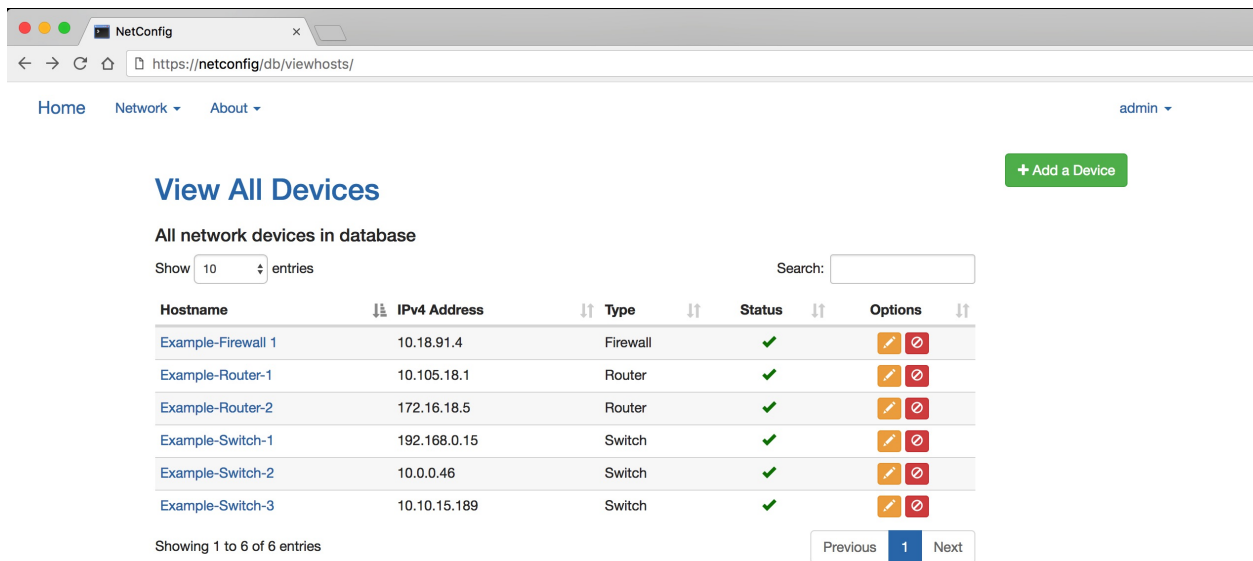
1.4 Upgrade

Reference the Upgrading secion for instructions on upgrading the software at the readthedocs.io upgrade guide. See the latest release page to download the most recent NetConfig version.

1.5 NetBox Integration

Reference the Netbox-Integration secion for instructions on pulling device inventory from an existing Netbox installation readthedocs.io Netbox integration guide. Netbox can be found at their GitHub repository.

1.6 Screenshots

1.7 Important Caveats

For all devices, Netconfig expects the hostname configured to match the actual hostname of the device (case-sensitive). If not, some features may not work properly.

1.8 Contribute

- Source Code: NetConfig on GitHub
- Issue Tracker: NetConfig Issue Tracker
- Documentation: NetConfig on ReadTheDocs
- Subreddit: NetConfig on Reddit

1.9 Support

If you are having issues, please let us know Please file an issue in the GitHub issue tracker

1.10 License

NetConfig is licensed under the GPL v3.0 license. A copy of the license is provided in the root NetConfig directory, or you can view it online here

Installation Guides

2.1 Ubuntu 16.04 Server

Installation guide for Ubuntu 16 04 Server

2.2 CentOS 7 Server

Installation guide for CentOS 7 Server

Upgrading

3.1 Upgrading NetConfig to Latest Version

3.1.1 Before Starting

If running NetConfig on a VM, it is *highly* recommended to take a snapshot prior to upgrading. In case of any issues, you can roll back any changes by reverting to the previous snapshot.

3.1.2 Upgrade Process

Version 1.3.0 (beta) or newer

As of Version 1.3.0 (beta), the Upgrade process has been fully automated into the ~/netconfig/upgrade.sh script.

If you are on v1.3.0 (beta) or higher, simply run the following commands to run the script as the local 'netconfig' user:

```
su - netconfig
/home/netconfig/netconfig/upgrade.sh
```

Version 1.2.3 (beta) or earlier

Change to NetConfig user

```
su - netconfig
```

Change to NetConfig directory

```
cd /home/netconfig/netconfig
```

Checkout master branch

```
git checkout master
```

Pull new files

```
git pull origin master
```

Verify git status

```
git status
```

Run Upgrade script. If upgrade script is not executable, run the 'chmod' command below first

```
chmod +x upgrade.sh
./upgrade.sh
```

Restart NetConfig service

```
sudo supervisorctl restart netconfig
```

3.1.3 Verifying Upgrade

In your web browser, navigate to the home NetConfig page. In the Top Menu, under About, you should see the latest software version displayed.
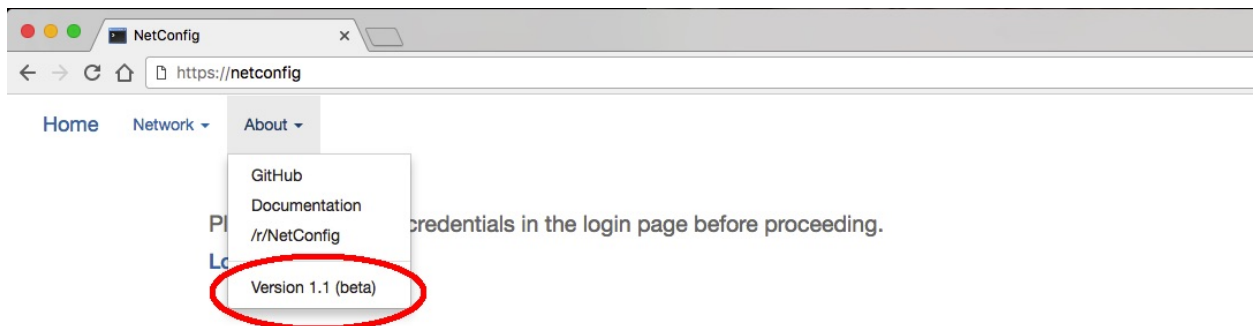
3.1.4 Potential Caveats

If any manual changes are made to any NetConfig files (except for the settings and log files), the command 'git pull origin master' may fail or throw an error. If so, you can stash (delete) any manual changes made, then repull from NetConfig's GitHub respository. This will replace any custom changes made in files with the standard NetConfig system files, so be careful if any custom changes are critical to your environment.

The command is:

Upgrade Script doesn't run:

If the upgrade script doesn't run, make sure it is executable first.

```
ls -lah
# -rw-r--r--   1 netconfig  staff   1.2K Jan  2 14:30 upgrade.sh
```

If it is missing an 'x' in the above output, run this command:

```
chmod +x upgrade.sh
```

The 'ls -lah' output should now read as follows:

```
ls -lah
# -rwxr-xr-x   1 netconfig  staff   1.2K Jan  2 14:30 upgrade.sh
```

Netbox Integration

4.1 Configuring NetConfig

Netconfig supports two methods of tracking network device inventory:

1. Local database using SQLAlchemy

2. API call through Netbox, an open source DCIM solution found here: https://github.com/digitalocean/netbox

By default, NetConfig is set to use a local SQLAlchemy database

4.1.1 Configure Netconfig for Netbox API Access

In the root Netconfig directory, open file 'instance/settings.py' (this should be /home/netconfig/netconfig/instance/settings.py)

Navigate to the line that reads:

    DATALOCATION = 'local'

Change it to read:

    DATALOCATION = 'netbox'

Navigate to the line that reads:

    NETBOXSERVER = ''

Set the URL for your Netbox server using single quotes. Example:

    NETBOXSERVER = 'http://netbox.domain.com'

Save and close the file

Custom Field #2

Click the Add Custom Field + button, and use the below settings. The Description field and the checkbox for 'Required' are both optional and up to you. Additonally the fields with "weight" in the name can be any number from 1-100, and are up to you.

```
Object:                dcim > device type
Type:                  Selection
Name:                  Netconfig_OS
Label:                 [blank]
Is Filterable:         Checked
Weight:                100
Custom Field Choices
    1st Value field:   IOS
    1st Value weight:  97
    2nd Value field:   IOS-XE
    2nd Value weight:  98
    3rd Value field:   NX-OS
    3rd Value weight:  99
    4th Value field:   ASA
    4th Value weight:  100
```

Click Save

Screenshot example:

4.3 Configuring Devices in Netbox to be used by NetConfig

For any new or existing devices you want to use with Netconfig, when creating or editing a device, the following conditions must be met:

- For each device, set the field Netconfig under the Custom Fields section to Yes

- A Primary IPv4 address must be configured for each device (IPv6 is not supported at this time)

- The IP address must be reachable by Netconfig, and Netconfig must be able to SSH into the device

- The Device Type (assigned to a device you want pulled) must have one of the 4 currently supported OS's set in the Custom Fields: IOS, IOS-XE, NX-OS, or ASA

Add Vendor Support

5.1 Getting Started

Currently as of version 1.0b, NetConfig only support Cisco routers, switches, and firewalls, running IOS, IOS-XE, NX-OS, or ASA's. However in version 1.1b, support was added for different vendors and different vendor model network devices.

This guide intends to document how to create a base device class for a vendor, as well as individual device model classes, for support with integrating into NetConfig. Any new device classes that work successfully and are tested thoroughly may be submitted as a Pull Request for official integration into the overall project.

5.2 Create Base Devive Class

The base device class is used for different vendors. Currently, the only existing base device class is for Cisco, titled "cisco_base_device.py". In this example, we will create a new base device class for vendor "Acme".

Before creating any files, make sure to SSH in to the NetConfig server as user 'netconfig', or switch over to the user 'netconfig' once logging in.

```
su - netconfig
```

5.2.1 File Location

In this example, the file is located in ~/netconfig/app/device_classes/device_definitions/acme_base_device.py.

```
cd ~/netconfig/app/device_classes/device_definitions
touch acme_base_device.py
vi acme_base_device.py
```

The basic structure of this file is as follows:

```python
from base_device import BaseDevice


class AcmeBaseDevice(BaseDevice):
    # functions go here
```

The file has a few base functions that are required by the overall NetConfig program. As NetConfig grows its feature set, more required functions may be added, and will need to be added/updated in here as necessary.

All functions in here are expected to work with any/all network devices made by this vendor. Model specific functions will go in the individual device model classes, explained at the end of this file.

Required Functions

These functions and their specific names are required for NetConfig, including the required inputs and expected outputs. How they process the inputs in order to return the required output varies for each specific network vendor.

```python
# Input: None required
# Purpose: Provide the command used to enter a configuration mode in the device
# Output:
#    command (string) - Outputs the command used to enter configuration mode for all
→vendor devices

def get_cmd_enter_configuration_mode(self):
    # Function logic goes here
    return commandStr

# Input: None required
# Purpose: Provide the command used to exit the configuration mode in the device
# Output:
#    command (string) - Outputs the command used to exit configuration mode for all
→vendor devices

def get_cmd_exit_configuration_mode(self):
    # Function logic goes here
    return commandStr

# Input: None required
# Purpose: Provide the command used to enable a specific interface
# Output:
#    commandStr (string) - Outputs the command used to enable / activate / unshut /
→bring online an interface for all vendor devices
# Output Type: String

def get_cmd_enable_interface(self):
    # Function logic goes here
    return commandStr

# Input: None required
# Purpose: Provide the command used to disable a specific interface
# Output:
#    commandStr (string) - Outputs the command used to disable / deactivate /
→shutdown / bring offline an interface for all vendor devices

def get_cmd_disable_interface(self):
    # Function logic goes here
    return commandStr
```

```python
# Input:
#    interface (string) - The name of the interface that is to be enabled
#    activeSession (Netmiko class) - The active, existing SSH session for a device,
→stored as a Netmiko class
# Purpose: Enable a specific interface
# Output:
#    resultsList (list) - Outputs all command results as displayed the client network
→device when enabling an interface

def run_enable_interface_cmd(self, interface, activeSession):
    # Function logic goes here
    return resultsList


# Input:
#    interface (string) - The name of the interface that is to be disabled
#    activeSession (Netmiko class) - The active, existing SSH session for a device,
→stored as a Netmiko class
# Purpose: Disable a specific interface
# Output:
#    resultsList (list) - Outputs all command results as displayed the client network
→device when disabling an interface

def run_disable_interface_cmd(self, interface, activeSession):
    # Function logic goes here
    return resultsList


# Input:
#    activeSession (Netmiko class) - The active, existing SSH session for a device,
→stored as a Netmiko class
# Purpose: Saves the running-configuration settings on the device into memory
# Output:
#    resultsList (list) - Outputs all command results as displayed the client network
→device when enabling an interface, with each new line (separated by carriage
→return) in its own line in the returned list

def save_config_on_device(self, activeSession):
    # Function logic goes here
    return resultsList


# Input:
#    interface (string) - The name of the interface to edit the configuration settings
#    datavlan (string) - The data vlan ID to set on the interface.  Note: This is an
→optional variable, and may submitted as an empty string instead
#    voicevlan (string) - The voice vlan ID to set on the interface.  Note: This is
→an optional variable, and may submitted as an empty string instead
#    other (list) - A list (separated by carriage returns) of any additional commands,
→ manually entered by the user, needing to be configured for the specified interface.
→  Note: This is an optional variable, and may submitted as an empty string instead
#    activeSession (Netmiko class) - The active, existing SSH session for a device,
→stored as a Netmiko class
# Purpose: Edits the configuration settings for a specific interface on a device
# Output:
#    resultsList (list) - Outputs all command results as displayed the client network
→device when edit an interface, with each new line (separated by carriage return) in
→its own line in the returned list
```

```python
def run_edit_interface_cmd(self, interface, datavlan, voicevlan, other,
→activeSession):
    # Function logic goes here
    return resultsList


# Input:
#    activeSession (Netmiko class) - The active, existing SSH session for a device,
→stored as a Netmiko class
# Purpose: Pulls any inventory information about the device (Cisco equivalent: "show
→inventory")
# Output:
#    resultsList (list) - Outputs all command results as displayed by the client
→network device as returned once executing the command, with each new line
→(separated by carriage return) in its own line in the returned list

def pull_inventory(self, activeSession):
    # Function logic goes here
    return resultsList


# Input:
#    activeSession (Netmiko class) - The active, existing SSH session for a device,
→stored as a Netmiko class
# Purpose: Pulls any version information about the device (Cisco equivalent: "show
→version")
# Output:
#    resultsList (list) - Outputs all command results as displayed by the client
→network device as returned once executing the command.  The list is formatted where
→each new line of output (as determined by \n [carriage-return]) is separated in the
→returned list.

def pull_version(self, activeSession):
    # Function logic goes here
    return resultsList
```

5.3 Create Individual Devive Type Class

The specific device type class is used for the same vendor (as created above). However a different device type file needs to be created for each type of device that uses different commands, unique commands, or returns output differently than other models by the same vendor.Currently, the only existing device type classeses are for Cisco, which are "cisco_ios.py", "cisco_asa.py", and "cisco_nxos.py". Note that NetConfig support both IOS and IOS-XE, however their commands and outputs are identical, so they both use "cisco_ios.py". In this example, we will create a new base device class for vendor "Acme".

Before creating any files, make sure to SSH in to the NetConfig server as user 'netconfig', or switch over to the user 'netconfig' once logging in.

```
su - netconfig
```

5.3.1 File Location

Create a new directory for the vendor.

```
mkdir ~/netconfig/app/device_classes/device_definitions/acme
cd ~/netconfig/app/device_classes/device_definitions/acme
```

Create a new 'init' file

```
touch __init__.py
vi __init__.py
```

Add the following lines into the file:

```python
from acme_os import AcmeOS


__all__ = ['AcmeOS']
```

Now create the new device file for Acme OS type devices:

```
touch acme_os.py
vi acme_os.py
```

The basic structure of this file is as follows:

```python
from ..acme_base_device import AcmeBaseDevice


class AcmeOS(AcmeBaseDevice):
    # functions go here
    return x
```

The file has a few functions that are required by the overall NetConfig program. As NetConfig grows its feature set, more required functions may be added, and will need to be added/updated in here as necessary.

All functions in here are expected to work with only this specific network device type, by this specific vendor. Any functions that function identically, and are supported by this vendor across all of their device models/types, may go in the acme_base_device.py file instead.

Required Functions

These functions and their specific names are required for NetConfig, including the required inputs and expected outputs. How they process the inputs in order to return the required output varies for each specific network vendor.

```python
# Input:
#     activeSession (Netmiko class) - The active, existing SSH session for a device,
→stored as a Netmiko class
# Purpose: Pulls any version information about the device (Cisco equivalent: "show
→version")
# Output:
#     resultsList (list) - Outputs all command results as displayed by the client
→network device as returned once executing the command.  The list is formatted where
→each new line of output (as determined by \n [carriage-return]) is separated in the
→returned list.


def pull_version(self, activeSession):
    # Function logic goes here
    return resultsList


# Input: None required
# Purpose: Provide the command used to display the active/running configuration
→settings
```

```
#    commandStr (string) - Outputs the command used to display the active/running
→configuration settings

def cmd_run_config(self):
    # Function logic goes here
    return commandStr

# Input: None required
# Purpose: Provide the command used to display the saved/startup configuration
→settings
# Output:
#    commandStr (string) - Outputs the command used to display the saved/startup
→configuration settings

def cmd_start_config(self):
    # Function logic goes here
    return commandStr

# Input: None required
# Purpose: Provide the command used to display the the CDP/LLDP neighbors, with each
→new line (separated by carriage return) in its own line in the returned list
# Output:
#    commandStr (string) - Outputs the command used to display the CDP/LLDP neighbors

def cmd_cdp_neighbor(self):
    # Function logic goes here
    return commandStr

# Input:
#    activeSession (Netmiko class) - The active, existing SSH session for a device,
→stored as a Netmiko class
# Purpose: Pulls the active/running configuration settings for the device
# Output:
#    resultsList (list) - Outputs the active/running configuration settings, with
→each new line (separated by carriage return) in its own line in the returned list

def pull_run_config(self, activeSession):
    # Function logic goes here
    return resultsList

# Input:
#    activeSession (Netmiko class) - The active, existing SSH session for a device,
→stored as a Netmiko class
# Purpose: Pulls the saved/startup configuration settings for the device
# Output:
#    resultsList (list) - Outputs the saved/startup configuration settings, with each
→new line (separated by carriage return) in its own line in the returned list

def pull_start_config(self, activeSession):
    # Function logic goes here
    return resultsList

# Input:
#    activeSession (Netmiko class) - The active, existing SSH session for a device,
→stored as a Netmiko class
# Purpose: Pulls the CDP/LLDP neighbors for the device
```

```python
# Output:
#    tableHeader (string) - String containing the table header lines, as retrieved
→from (usually) the first line of output, with each category separated by comma.
#        Example: Hostname,Src Port,Model,Dest Port,etc
#    tableBody (list) - List with each line an output row retrieved from the devices
→CDP/LLDP table.  Each column separated by comma.  There should be the same number
→of columns in each row, and the same number of columns as in the tableHeader.
Outputs the CDP/LLDP neighbors, with each new line (separated by carriage return) in
→its own line in the returned list

def pull_cdp_neighbor(self, activeSession):
    # Function logic goes here
    return tableHeader, tableBody


# Input:
#    activeSession (Netmiko class) - The active, existing SSH session for a device,
→stored as a Netmiko class
# Purpose: Pulls different information about a device, stored into 3 separate lists:
#    interfaceConfig (list) - Configuration settings for the interface
#    interfaceMacAddressesHeader (string) - A string containing the table header for
→the MAC Address table output, with each column separated by a comma
#    interfaceMacAddressesBody (list) - A list with each row containing each line of
→data in the interface MAC Address table output, with each column separated by a
→comma.  Note: This should only be run on devices that store MAC addresses
→associated with their interface.  Otherwise simply return an empty string
#    interfaceStatistics (list) - Any relevant interface statistics that should be
→shown for the interface (Cisco example: show interface FastEthernet0/1)
# Output:
#    interfaceConfig, interfaceMacAddressesHeader, interfaceMacAddressesBody,
→interfaceStatistics (lists) - Array specifics detailed above

def pull_interface_info(self, activeSession):
    # Function logic goes here
    return interfaceConfig, interfaceMacAddressesHeader, interfaceMacAddressesBody,
→interfaceStatistics


# Input:
#    activeSession (Netmiko class) - The active, existing SSH session for a device,
→stored as a Netmiko class
# Purpose: Pulls the current device uptime
# Output:
#    resultsStr (string) - Outputs the current uptime of the device as a string

def pull_device_uptime(self, activeSession):
    # Function logic goes here
    return resultsStr


# Input:
#    activeSession (Netmiko class) - The active, existing SSH session for a device,
→stored as a Netmiko class
# Purpose: Pulls the list of interfaces on the device
# Output:
#    tableHeader (string) - String containing the table header lines.
#    resultsList (list) - Outputs a list of interfaces and relevant status settings,
→with each new line (separated by carriage return) in its own line in the returned
→list (Cisco example: "show ip interface brief")
```

```python
def pull_host_interfaces(self, activeSession):
    # Function logic goes here
    return tableHeader, resultsList


# Input:
#    interfaces (list) - Array of strings, returned from the device, where each
→string contains information on if the interface is up/online, down/offline, and
→administratively down/forced offline.  This function does not correctly interface
→status with the interface directly, so tracking the interface names is irrelevant
→here
# Purpose: Returns the number of interfaces online, offline, forced offline, and
→total count
# Output:
#    upCount (int) - Total number of interfaces active/online
#    downCount (int) - Total number of interfaces down/offline
#    disabledCount (int) - Total number of interfaces administratvely down/forced
→offline
#    totalCount (int) - Total number of interfaces


def count_interface_status(self, interfaces):
    # Function logic goes here
    return upCount, downCount, disabledCount, totalCount
```

CHAPTER 6

Contributing to NetConfig

6.1 How to Contribute to NetConfig

You can contribute to NetConfig in multiple ways.

6.1.1 Bugs/Issues

If you encounter a bug when using NetConfig, you can submit a new issue on GitHub at https://github.com/v1tal3/
netconfig/issues

Please provide as much detail as possible, including:

- Exact steps taken to reproduce issue

- Any error messages you see

- Version of NetConfig you are using

- Any (non-private) information on the affected device (if your issue is with a specific device/model)

6.1.2 Feature Requests

If you have a feature request, you can file it on the issue submission page on GitHub at https://github.com/v1tal3/
netconfig/issues

Please provide as much relevant detail into your request as possible.

6.1.3 Pull Requests

If you have a code contribution you'd like to make, pull requests are encouraged! All pull requests will reviewed, and
feedback will be provided, regardless of if it's accepted, rejected, or changes are requested first.

Please make sure to test any pull requests thoroughly on the latest Development branch changes.

A few requirments for any code contributions:

- Submit all pull requests on the 'development' branch in GitHub

- Test all pull requests as thoroughly as possible

- Note any bugs, outstanding issues, or anything that still needs to be tested (if you were unable to test a part)

- **All submit .py files must adhere to PEP8/Flake8 standards. Currently, all NetConfig .py files adhere to Flake8 standards,**

    - E501 - line too long (82 > 79 characters)

    - N802 - function name should be lowercase

    - N803 - argument name should be lowercase

    - N806 - variable in function should be lowercase

6.1.4 Documentation

Any and all documentation submissions are welcome. If you submit detailed documentation, I will be happy to format it and introduce it properly into the official readthedocs.io documentation.

CHAPTER 7

Index

- genindex
- search