# ndmtk Documentation

***Release 0.1.8***

**Paul Greenberg**

**Apr 24, 2017**

# Contents

Table of Contents:

CHAPTER 1

# Introduction

**Table of Contents:**



## Network Discovery and Management Toolkit

Network Discovery and Management Toolkit (*ndmtk*) makes Ansible *work* for both Traditional and Software-Defined Network (SDN) network management.

### Purpose

The future of network management lies in the area of Artificial Intelligence. Any network-enabled device will be able to build connectivity to a remote peer on-demand, without human intervention. The restraint on that ability are the AI-enabled systems acting as gatekeepers. AI is impossible without ongoing data collection, data analysis, probing, and modeling. As such, networks of the future need tools to perform the above tasks.

This toolkit is designed to accomplish the data collection piece of the AI puzzle. Specifically, the toolkit is designed to:

- discover data on network devices and capture the entirety of available data
- configure network devices via SSH, telnet, console, or terminal server

- collect, analyze, and store the data via command-line interactions; it performs data analysisn and, if necessary, it performs additional data collection and/or device configuration tasks.

## Audience

The intended audience of this toolkit are system and network engineers and designers, as well as the researchers dealing with AI.

## Artificial Intelligence (AI)

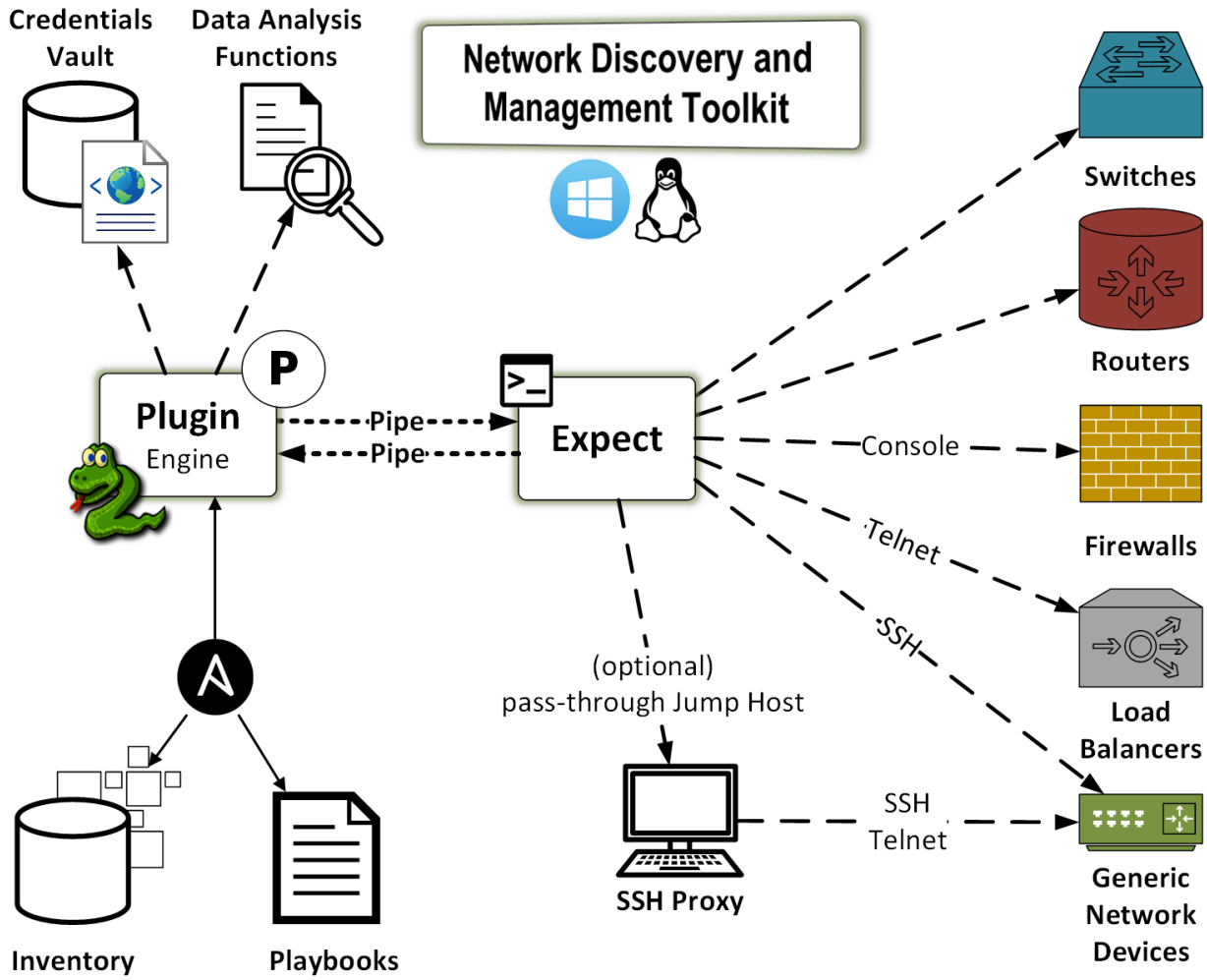
The toolkit is delivered in a form of an Ansible plugin. However, it could work well with Chef, or any other orchestration tool. The reason Ansible became a framework of choice is its modularity. The toolkit itself is modular. It allows extended existing functionality. For example, the plugin does not blindly run pre-defined commands. Rather, it first collects all of the commands forming the understanding of the function of a particular device in a network. Once the plugin receives the data, it runs it through its algorithms and determines whether there are any additional command required to further gather data. That process continues until the algorithms determine that the collection is complete.

The plugin has no required arguments and parameters, because there are a number of default commands available for various operating systems, e.g. Cisco Nexus OS, Arista EOS, Linux, etc.

## Structured Data

Importantly, once the plugin completes its tasks it produces a number of reports in JSON, YAML, and JUnit formats. These reports provide a map of what was done, where the collected data reside, and what that data is.

## Workflow Diagram



CHAPTER 2

# User Guide

**Table of Contents:**



Please note that the plugin always collects version and configuration information when mining for network data.

## Prerequisites

The plugin requires the presence of two binaries:

- `ssh`
- `expect`

The users of the plugin should know YAML., because it is the toolkit's abstraction format.

The toolkit works on Linux. However, it could work on Windows too. If you are interested in Windows support, please open an issue.

 *Back to Top*

## Builtin Commands

This plugin uses the following approach when determining which commands are availble to run on a remote device.

- First, each device must carry `ndmtk_os` or `os` attribute. Based on the value ofthe attribute, the plugin performs a lookup in `files/cli/os/` directory the plugin's directory inside Python's `site-packages` directory. For example, Cisco ASA firewall must have either `ndmtk_os=cisco_asa` or `os-cisco_asa` attribute.
- Then, the plugin will try to locate `files/cli/os/cisco_asa.yml` file. Once located, the plugin will read it and collect all of the cli commands associated with Cisco ASA operating system.

Please note, based on the information, the plugin will also record which commands show configuration and version information, and which commands should be used to disable paging or switch to automation mode.

The `disable_defaults` option isables default pre-packages commands for various operating systems. It is commonly used when configuring a device, as opposed to gathering data of a device.

 *Back to Top*

## User-Defined Commands

### Per Operating System

The `cliset_os_dir` points to the path to YAML files containing user-defined commands on per operating system basis. The plugin will run the commands only if the plugin is able to locate a file matching a remote host's operating system in this directory.

For example, if a host's operating system is `cisco_nxos`, the plugin will look for `cisco_nxos.yml` file in this directory. If the file is found, then the plugin will run the commands it found in the file.

Please note that the plugin runs the commands in addition to any default commands, unless they are disabled with `disable_defaults`.

The default commands for various operating systems are located in `<python_site_packages>/ndmtk/files/cli/os` directory.

 *Back to Top*

### Per Individual Host

The `cliset_host_dir` points to the path to YAML files containing user-defined commands on per host basis. The plugin will run the commands only if the plugin is able to locate a file matching a remote host's hostname in this directory.

For example, if a host's hostname is `ny-fw01`, the plugin will look for `ny-fw01.yml` file in this directory. If the file is present and readable, then the plugin will run the commands it finds in the file.

Please note that the plugin runs the commands in addition to any default commands, unless they are disabled with `disable_defaults`.

 *Back to Top*

### Specific Tasks

Frequently, there is a need to run a specific set of commands for non-data collection purposes, e.g. device configuration. The `cliset_spec` points to the path to a single YAML file containing user-defined commands.

As with the previously discussed user-defined commands, the plugin runs the in addition to any default commands, unless they are disabled with `disable_defaults`.

 *Back to Top*

## Exceptions

The `cliset_exc` points the path to a single YAML file containing exceptions to both default and user-defined commands. The root element of the YAML data structure is `exceptions`. The structure is a list of dictionaries/items. Each dictionary item must have at least one of the keys: `cli`, `host`, and/or `os`. The keys are strings containing regular expressions.

The plugin pre-checks each of the commands it has in its queue against the exceptions. If the plugin matches a command with the `cli` regular expression, it performs additional `host` and `os` regular expression searches, if any. If the plugin is able to match all regular expressions within a single exception, it marks the command as `skipped` and never runs it on the actual device. By default, the plugin search for `<ansible_inventory_dir>/files/ndmtk/exceptions.yml` file.

 *Back to Top*

## Limits

A user may limit a scope of the commands from any command line set it supplies to the plugin. The user could use `sections` option.

For example, if a user wants to run only BGP-related commands on a device, the user would add the following to a task `sections="bgp"`. This way, the plugin will only execute the commands that have `bgp` tag attached to them.

 *Back to Top*

## Data Repository

The plugin uses the value supplied with `output` option to determine where to store the data produced by the plugin. If a path contains `%` sign in it, then the plugin performs pre-defined conversions. For example, `%h` is converted to a host's hostname, `%H` to a host's FQDN, and `%E` to epoch timestamp. Please search the plugin's source code for the full list of converted characters.

```
'h': 'hostname',
'p': 'unique_process_id',
'U': os.path.split(os.path.expanduser('~'))[-1],
'Y': str(ts.tm_year).zfill(4),
'm': str(ts.tm_mon).zfill(2),
'd': str(ts.tm_mday).zfill(2),
'H': str(ts.tm_hour).zfill(2),
'M': str(ts.tm_min).zfill(2),
'S': str(ts.tm_sec).zfill(2),
'E': str(int(epoch)),
```

 *Back to Top*

## Security

### SSH Fingerprints

If the `no_host_key_check` option is set to `yes`, it instructs the plugin to accept SSH fingerprints without validation, i.e. trust any SSH fingerprint.

 *Back to Top*

### Jumphosts

The `jumphosts` instructs the plugin to access devices via a chain of jumphosts. In enterprise networks, access to network devices is allowed only from restricted management stations/hosts. This option allows users to run tasks through these hosts.

 *Back to Top*

### Multi-Factor Authentication (MFA)

Recently, enterprise technology users have been moving to multi-factor authentication (MFA). It presents a challenge to network automation. However, with `token_bypass` option pointing to the socket of the process with the knowledge iof what that second (multi) factor is, it is no longer an issue.

 *Back to Top*

## Examples

The following command instructs Ansible to login to `ny-fw01` and collect running configuration from it.

```
ansible-playbook playbooks/collect_configuration.yml
```

Alternatively, a user may collect the output of all relevant operating system commands:

```
ansible-playbook playbooks/collect_all.yml
```

Additionally, this plugin supports Check Mode ("Dry Mode"). In this mode, the plugin will not attempt to login to network devices. This mode is used to test for the existence of access credentials.

```
ansible-playbook playbooks/collect_configuration.yml --check
```

Another way to use the plugin is to configure network devices. The below Ansible playbook configures ACL on a Cisco ASA firewall.

```
ansible-playbook playbooks/configure_acl.yml --check -vvv
ansible-playbook playbooks/configure_acl.yml --vvv
```

This playbook shows how to collecte data via the chaing of devices, i.e `controller` => `10.1.1.1`, `10.1.1.1` => `10.1.2.3`, `10.1.2.3` => `10.2.3.4` => `managed node`.

```
- name: data collection via jumphosts
  action: |
     ndmtk output="/tmp/jump-data-%Y%m%d%H%M%S"
     jumphosts="10.1.1.1,10.1.2.3,10.2.3.4"
     no_host_key_check=yes
```

*Back to Top*

## Common Errors

A user may receive the following error:

```
fatal: [ny-fw01]: FAILED! => {
    "failed": true,
    "msg": "The module ndmtk was not found in configured module paths.
            Additionally, core modules are missing.
            If this is a checkout, run 'git submodule update --init --recursive'
            to correct this problem."
}
```

This is the indication that something is broken with *setup.py*. The issue maybe caused by the lack of permissions. Please open an issue.

*Back to Top*

CHAPTER 3

# Rules Engine

**Table of Contents:**



This page describes the toolkit's rules engine.

## Data Collection Rules

The decision how to collect data from a network device is governed by the rules engine. It is abstructed in the form of YAML.

### Basic Conditions

The following rule applies to any Linux distribution. The purpose of the rule is to discover the paths to all binaries in the user's `PATH` environment variable. The information collected is stored in a reference database with tags `binaries` and `configuration`.

```
- description: 'collect the file listing of binaries in PATH'
  cli: 'find $(env | grep "^PATH=" | sed "s/PATH=//;s/:/ /g") -maxdepth 10 -type f -
↪print | sed "s/\/\//\//"'
  tags: ['ref:binaries', 'configuration']
  saveas: '%h.files.bin.txt'
```

Later, we could utilize the references in other rules.

For example, there will be no collection of IP addressing information unless `ifconfig` binary is present.

```
- description: 'collect ip addressing information via sysctl'
  cli: 'ifconfig -a'
  tags: ['network']
  conditions_match_any:
  - 'tag:binaries:.*bin/ifconfig$'
```

At the same time, the plugin will collect information if `ip` binary is present:

```
- description: 'collect ip routing information'
  cli: 'ip route'
  tags: ['network']
  conditions_match_any:
  - 'tag:binaries:.*bin/ip$'
```

 *Back to Top*

## Derivatives

In some cases, it is necessary to run follow up commands to discover more data.

The below rule instructs the plugin to read kernel network interface table `/proc/net/dev` and run follow up `ethtool` commands if `ethtool` is available.

```
- description: 'collect kernel network interface statistics'
  cli: 'cat /proc/net/dev'
  tags: ['network', 'test']
  saveas: '%h.ifstats.txt'
  derivatives:
  - os:
    - generic_linux
    regex:
    - pattern: '^\s*(?P<IF_NAME>\S+):'
      flags: ['add_cli']
    actions:
    - description: 'collect network interface driver and hardware settings from <IF_
↪NAME>'
      cli:
      - 'ethtool <IF_NAME>'
      - 'ethtool --show-pause <IF_NAME>'
      - 'ethtool --show-coalesce <IF_NAME>'
      - 'ethtool --show-ring <IF_NAME>'
      - 'ethtool --driver <IF_NAME>'
      - 'ethtool --show-features <IF_NAME>'
      - 'ethtool --statistics <IF_NAME>'
      - 'ethtool --show-nfc <IF_NAME>'
      - 'ethtool --show-ntuple <IF_NAME>'
```

```
      - 'ethtool --show-eee <IF_NAME>'
      - 'ethtool --show-priv-flags <IF_NAME>'
      - 'ethtool --show-channels <IF_NAME>'
      - 'ethtool --show-time-stamping <IF_NAME>'
      - 'ethtool --show-permaddr <IF_NAME>'
      - 'ethtool --module-info <IF_NAME>'
      - 'ethtool --show-eee <IF_NAME>'
      saveas: '%h.ethtool.<IF_NAME>.txt'
      append: yes
      required: ['IF_NAME']
      conditions_match_all:
      - 'tag:binaries:.*in/ethtool$'
      allow_empty_response: no
```


*Back to Top*

Similarly, the below rule applies to Cisco NX-OS devices. The plugin run `show ip bgp summary vrf all` on a device only if `router bgp` process is configured. Once collected, the plugin collects BGP neighborship information from the output of the command. If the plugin finds a BGP neighbor, it will collect the `advertised-routes` and `received-routes` from that neighbor.

```
- cli: show ip bgp summary vrf all
  tags: ['routing', 'bgp']
  conditions_match_any:
  - '^router bgp'
  derivatives:
  - description: 'BGP neighbor details'
    os:
    - cisco_nxos
    regex:
    - pattern: 'BGP summary information for VRF (?P<VRF>\S+), address family'
      flags: ['purge']
    - pattern: '\s*(?P<IP_ADDRESS>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\s+\d\s+'
      flags: ['add_cli']
    actions:
    - cli: 'show ip bgp neighbors <IP_ADDRESS> vrf <VRF>'
      required: ['IP_ADDRESS', 'VRF']
      format: 'txt'
    - cli: 'show ip bgp neighbors <IP_ADDRESS> advertised-routes vrf <VRF>'
      required: ['IP_ADDRESS', 'VRF']
      format: 'txt'
    - cli: 'show ip bgp neighbors <IP_ADDRESS> received-routes vrf <VRF>'
      required: ['IP_ADDRESS', 'VRF']
      format: 'txt'
```


*Back to Top*

## No Newline

In some cases, there is a need to check the command line options available to a user. Traditionally, a user would do it with question mark at the end of the user's request.

Here, the user issues `show service-policy inspect ?` to get available options. However, when doing so, the user does not press `Enter`. The `no_newline` field mimic the describes behavior. If the field does not exist or if it is set to `no`, then the newline character will be appended to the user's request.

```
- description: 'Collect a list of all possible inspection policies.'
  cli: 'show service-policy inspect ?'
  no_newline: yes
  os:
  - cisco_asa
  tags: ['inspect', 'test']
  conditions_match_all:
  - '^policy-map\s'
  - '^\s+class\s'
  - '^\s+inspect\s'
  derivatives:
  - description: 'Collects information about individual inspection policies'
    os:
    - cisco_asa
    regex:
    - pattern: '^\s*(?P<INSPECTION_POLICY_NAME>\S+$)\s+Show'
      flags: ['add_cli']
    actions:
    - description: 'Collects statistics for inspect <INSPECTION_POLICY_NAME> policy'
      cli: 'show service-policy inspect <INSPECTION_POLICY_NAME>'
      required: ['INSPECTION_POLICY_NAME']
      format: 'txt'
```

 *Back to Top*

## Success and Error Overwrites

One of the derivative `show service-policy inspect` commands produces an error:

```
show service-policy inspect h323
ERROR: % Incomplete command
```

In order to avoid an error, a user may add `success_if` field. This causes the plugin to declare the output to be a success, despite that it is failing.

```
- description: 'Collects statistics for inspect <INSPECTION_POLICY_NAME> policy'
  cli: 'show service-policy inspect <INSPECTION_POLICY_NAME>'
  required: ['INSPECTION_POLICY_NAME']
  format: 'txt'
  success_if:
  - '.*'
```

 *Back to Top*

## Facts

The plugin's configuration files has sections dedicated to fact discovery based on the outputs related to software and/or hardware information. For example, CentOS servers have `/etc/os-release` file.

The contents of the file are as follows:

```
NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
```

```
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
REDHAT_SUPPORT_PRODUCT="centos"
REDHAT_SUPPORT_PRODUCT_VERSION="7"
```

The engine has rules to match against that output:

```
- pattern: '^NAME="?(?P<os_name>.*)["]?$'
  add:
  - 'os_class=generic_linux'
  strip_quotes: yes
- pattern: '^VERSION_ID="?(?P<os_version>\d+)"?'
  strip_quotes: yes
- pattern: '^ID=(?P<os_subclass>\S+)$'
  strip_quotes: yes
```

Please note the `strip_quotes` key. When plugin discovers its presence, it strips double quotes from the captured values.

The plugin's analytics engine adds the metadata in the following format:

```
facts:
  os_class: generic_linux
  os_name: CentOS Linux
  os_subclass: centos
  os_version: '7'
```

 *Back to Top*

## Rule Design

Each command executed by the plugin runs in one of the below modes:

- `noop`
- `analytics`
- `configure`
- `pre`
- `post`

 *Back to Top*

## Condition Precedent

The `condition_precedent_all` is a list of conditions. Each condition in the list is a string. The string must conform to the following format: `item predicate value`.

The `item` is what the toolkit will be looking for in `facts` dictionary.

The `predicate` are the type of a predicate used:

- `eq`: equals (both numeric and string, unless there is a type mismatch)
- `ne`: equals (both numeric and string, unless there is a type mismatch)
- `ge`: greater or equal (numeric evaluation)
- `gt`: greater than (numeric evaluation)
- `lt`: less than (numeric evaluation)
- `le`: less or equal (numeric evaluation)
- `rgx`: regular expression evaluation via `match`, as opposed to search

The `value` is variable.

 *Back to Top*

CHAPTER 4

# Access Credentials Management

**Table of Contents:**



This page explains how the toolkit manages network access credentials.

## Ansible Vault

This plugin handles user authentication by way of using user credentials located in Ansible Vault files. By default, the plugin looks up user credentials in `~/.ansible.vault.yml` file. The `safe` option points to the default location of the file.

A user creates the file by running `ansible-vault create ~/.ansible.vault.yml` command. Upon the creation of the file, the Ansible Vault prompts the user of a password. This password is used to decrypt the content of the vault.

The encrypted file is a plain text file. The first line of the file contains a header. The header specifies the version of Ansible Vault, encryption type, and looks like this.

```
$ANSIBLE_VAULT;1.1;AES256
```

A user edits the file with `ansible-vault edit ~/.ansible.vault.yml` command.

A user may save the password to unlock the vault in `~/.ansible.vault.key` file. By default, the plugin uses `lockpick` option to determine the location of the file unlocking the vault.

For example, the below instruction tells the plugin that the password for the vault is located in `/opt/admin/unlock.key`. The authentication credentials for the task are located in `/opt/admin/auth.yml`.

```
- name: collect data from network nodes
  action: ndmtk output="/tmp/data" safe="/opt/admin/auth.yml" lockpick="/opt/admin/
↪unlock.key"
```


*Back to Top*

## Credentials Structure and Format

The expected way to store access credentials is in YAML format. The data structure used is a list of hashes, where each hash represents a single credentials set.

Each hash in the list contains a subset of the following fields:

- `regex` (regular expression): if the regular expression in this field in a hash matches the FQDN or short name of a device, then the hash is preferred over any any other hash having the same or higher priority. However, if there is no match, then the hash is not used.
- `priority` (numeric): the field prioritizes the use of credentials. The entry with lower priority is preferred over the entry with higher priority when multiple entries match a regular expression pattern.
- `default` (boolean): if this field is present and it is set to *yes*, then this credential will be used in the absense of a *regex* match.
- `description` (text, optional): it provides an explanation about an entry.
- `username`
- `password`
- `enable`: this credential is used when prompted to provide enable password. currently, there is no distinction between enable levels.

In the below example a user entered two sets of credentials. The first entry is used for a specific device, i.e. `ny-fw01`. The second entry is used by default when there is no regular expression matching network device host name.

```
---
credentials:
- regex: ny-fw01
  username: admin
  password: 'NX23nKz!'
  password_enable: '3nKz!NX2'
  priority: 1
  description: NY-FW01 password
- default: yes
  username: greenpau
  password: 'My#DefaultPass'
  password_enable: 'Enabled#By$Default'
  priority: 1
  description: my default password
```

Considerations:

- There should be no `default` credential with the same `priority` level.
- There should be no credential with both `regex` and `default` fields present


*Back to Top*

## Multi-factor Authentication Internals

When an Ansible playbook contains tasks related to `ndmtk` plugin, Ansible invokes `ndmtk` callback plugin. The plugin performs lookup the lookup of access credentials in Ansible Vault.

By default, the plugin looks for `safe` and `lockpick` task arguments. If they are not defined, the plugin attempts to read `~/.ansible.vault.yml` (safe) and `~/.ansible.vault.key` (lockpick) files. The looked up access credentials are stored in `task['args']['credentials']` list and passed to `ndmtk` action plugin.

The action plugin invokes `_load_credentials()` function to parse the list. The function returns a list of dictionaries.

```
[
 {u'description': u'SDN Production Cisco Nexus Leaf Switches',
  u'password': u'pin,token',
  u'password_enable': u'pin,token',
  u'pin': u'4526',
  u'priority': 1,
  u'regex': u'^ny-fw02$',
  u'token': u'~/token.bypass',
  u'username': u'greenpau'},
 {u'default': True,
  u'description': u'my default password',
  u'password': u'POC123',
  u'password_enable': u'POC123',
  u'priority': 1,
  u'username': u'admin'}
]
```

When the plugin prepares for the connectivity it takes out one access credentials set (FIFO) and puts it in `self.activekey` variable.

Later, when prompted for the password by a remote device. It fetches the credentials from that variable using `_get_item_from_key()` function.

When a credential set fails, the plugin will lookup additional credentials.

```
fatal: [ny-fw02]: FAILED! => {
    "changed": false,
    "data_dir": "/opt/data/ansible/poc-conf-20170221190959/ny-fw02",
    "failed": true,
    "junit": "/opt/data/ansible/poc-conf-20170221190959/ny-fw02/ny-fw02.junit.xml",
    "msg": "authentication failed",
    "temp_dir": "/Users/greenpau/.ansible/tmp/ndmtk/56cce459-f869-11e6-94e9-
↪f45c89b1bb39/56d9c178-f869-11e6-a3a9-f45c89b1bb39/ny-fw02"
}
```

When dealing with credentials requiring PIN and Soft or Hard Token, a user must provide the path to read tokens via `token` key inside of access credentials hash. For example, the below hash instructs the plugin to user PIN plus Token combination for password. The PIN is `1234` and the token can be found in `~/token.bypass`.

```
- regex: '^ny-fw01$'
  username: 'greenpau'
  password: 'pin,token'
  password_enable: 'pin,token'
  token: '~/.token.bypass'
  pin: '1234'
  priority: 1
  description: 'Token-authenticated device'
```

A user populates the `~/token.bypass` file via CLI command. For example, the below command send Token `4562356` to `~/token.bypass`. Additionally, the user specifies the amount of time the Token will be active, i.e. `10`. The plugin uses the information to determine whether the token is valid or not.

```
date "+%s;572680;10" > ~/.token.bypass
```


*Back to Top*

CHAPTER 5

# Reports and Structured Data

**Table of Contents:**



The important functionality of the toolkit is the ability to produce reports about data collection process in YAML, JSON, and JUnit formats.

This functionality enables the plugin's integration with Jenkins, CircleCI, Travis, or any other. Additionally, it provides the ability to Artificial Intelligence (AI) frameworks to understand what the data is, without doing any heavy-lifting. In a sense, the structured data available in the reports becomes an anchor.


 *Back to Top*


## JUnit Reporting

The plugin produces reports in JUnit XML format on per host basis.

Each of the JUnit files has the following testsuites:

- `ndmtk.connect`
- `ndmtk.execute`
- `ndmtk.disconnect`

For example, the `ndmtk.connect` testsuite of `ny-sw01` has the following information. The information is self explanatory. Importanly, the plugin captures terminal output during connection establishment, authentication, and authorization.

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuites>
 <testsuite hostname="ny-sw01" name="ndmtk.connect" errors="0" skipped="0" tests="1"␣
↪failures="0" time="0.41" timestamp="2017-01-15T14:00:20">
  <properties>
   <property name="host" value="ny-sw01"/>
   <property name="os" value="arista_eos"/>
   <property name="output_dir" value="/tmp/test-20170115140020"/>
   <property name="on_error" value="continue"/>
   <property name="on_prompt" value="abort"/>
   <property name="temp_dir" value="/home/greenpau/.ansible/tmp/ndmtk/f3814002-db2a-
↪11e6-87ef-f45c89b1bb39/f3934fe8-db2a-11e6-bffd-f45c89b1bb39/ny-sw01"/>
   <property name="args" value="ssh -o UserKnownHostsFile=/dev/null -o␣
↪StrictHostKeyChecking=no -p 8224 -tt admin@localhost"/>
   <property name="play_uuid" value="f3814002-db2a-11e6-87ef-f45c89b1bb39"/>
   <property name="task_uuid" value="f3934fe8-db2a-11e6-bffd-f45c89b1bb39"/>
   <property name="return_code" value="0"/>
   <property name="return_status" value="ok"/>
   <property name="return_msg" value="ok"/>
   <property name="paging_mode" value="configured"/>
   <property name="scripting_mode" value="disabled"/>
   <property name="prompt_mode" value="disabled"/>
   <property name="clisets" value="/lib/python/site-packages/ndmtk/plugins/action/
↪files/cli/os/arista_eos.yml"/>
  </properties>
  <testcase name="connect" status="ok" time="0.41">
   <system_out><![CDATA[
################################################################################
# connection establishment log :
# /home/greenpau/.ansible/tmp/ndmtk/f3814002-db2a-11e6-87ef-f45c89b1bb39/f3934fe8-
↪db2a-11e6-bffd-f45c89b1bb39/ny-sw01/ny-sw01.log_connect
################################################################################

Warning: Permanently added '[localhost]:8224' (ECDSA) to the list of known hosts.
Password:
Last login: Sat Jan 14 15:43:57 2017 from 10.0.2.2
ny-sw01>
ny-sw01#
terminal length 0
Pagination disabled.

]]>
   </system_out>
   <skipped/>
  </testcase>
 </testsuite>
```

The `ndmtk.execute` contains information about the commands executed by the plugin. Here, the pluging executed `show routing-contex vrf` command. Then, based on the output, the plugin collected additional information about default VRF with `show ip route vrf default detail`.

```
<testcase name="Collects default routing context (VRF)" classname="routing, test"␣
↪status="ok" time="0.371">
 <system_out><![CDATA[
 $ show routing-contex vrf
 |--> $ show ip route vrf default
 |--> $ show ip route vrf default detail
```

```
 ]]>
 </system_out>
 <skipped/>
</testcase>
```


*Back to Top*

## Structure Data

The below are snippets from the output of `ny-sw01.meta.yml` file:

Here, after the `show vrf` was successfully executed, the plugin stored the data in a temporary directory. The output contained six (6) lines. Based on the output, the plugin captured two follow up commands:

- `show ip route vrf management`
- `show ip route vrf management detail`

Next, the command is associated with two tags: `routing`, `vrf`. Based on the `source` field, the source of the commands is pre-packaged operating system based rules, i.e. `os_default`.

```
- _seq: 3
  allow_empty_response: false
  child_cli_id:
  - show ip route vrf management
  - show ip route vrf management detail
  cli: show vrf
  description: Collects VRF information
  format: txt
  lines: '6'
  mode: analytics
  path: /tmp/test-20170115140020/ny-sw01/ny-sw01.show.vrf.txt
  path_tmp: /home/greenpau/.ansible/tmp/ndmtk/f3814002-db2a-11e6-87ef-f45c89b1bb39/
↪f3934fe8-db2a-11e6-bffd-f45c89b1bb39/ny-sw01/ny-sw01.show.vrf.txt
  sha1: 949faac85f41f62566b8609455ad2e67c87e57cb
  source: os_default
  status: ok
  tags:
  - routing
  - vrf
```

Then, there is the `status` field. It provides various information about the data collection task. Importantly, it has `facts` field. It is similar to the data produced by `facter` tool from Puppet labs.

```
status:
  authenticated: 'yes'
  authorized: 'yes'
  clisets:
  - /usr/lib/python/site-packages/ndmtk/plugins/action/files/cli/os/arista_eos.yml
  connect_end: 1484488820999
  connect_end_utc: 2017-01-15T14:00:20 UTC
  connect_start: 1484488820589
  connect_start_utc: 2017-01-15T14:00:20 UTC
  connected: 'yes'
  disconnect_end: 1484488824087
  disconnect_end_utc: 2017-01-15T14:00:24 UTC
```

```
  disconnect_start: 1484488824021
  disconnect_start_utc: 2017-01-15T14:00:24 UTC
  disconnected: 'yes'
  facts:
    hardware_macaddr: 0800.2756.4f61
    memory_free: 2891812 kB
    memory_total: 3887680 kB
    os_arch: i386
    os_class: arista_eos
    os_internal_build_id: c6362f13-ae6d-4c88-b5fd-4678d66018ab
    os_internal_build_version: 4.17.2F-3696283.4172F
    os_name: vEOS
    os_vendor: Arista
    os_version_major: '4'
    os_version_minor: '17'
    os_version_patch: 2F
    uptime: 21 hours and 18 minutes
  paging_mode: configured
  prompt_mode: disabled
  return_code: 0
  return_msg: ok
  return_status: ok
  scripting_mode: disabled
  spawned: 'yes'
task_uuid: f3934fe8-db2a-11e6-bffd-f45c89b1bb39
temp_dir: /home/greenpau/.ansible/tmp/ndmtk/f3814002-db2a-11e6-87ef-f45c89b1bb39/
↪f3934fe8-db2a-11e6-bffd-f45c89b1bb39/ny-sw01
```

The plugin uses the `facts` field when processing output through its Rules Engine.

```
- description: 'Collects routing table'
  cli: 'show ip route vrf all'
  tags: ['routing']
  conditions_precedent_all:
  - 'os_class eq arista_eos'
  - 'os_version_major ge 5'
```

Here, the `show ip route vrf all` will not run on the device, because `facts`'s `os_version_major` is less than the `os_version_major` in the `conditions_precedent_all` for the rule.

If a user wants to run the `show ip route vrf all`, the user should change `conditions_precedent_all` to:

```
conditions_precedent_all:
- 'os_class eq arista_eos'
- 'os_version_major ge 4'
```

 *Back to Top*

## Status Codes

Upon the completion of a particular command, the plugin updates the `status` field of the command. The list of possible values follows:

- `ok`: worked as expected

- `failed`
- `skipped`
- `conditional`: assigned when entered into the database and has `conditions_match` or `conditions_precedent_all` field associated with a command.
- `retry`
- `unknown`: assigned when entered into the database


*Back to Top*

CHAPTER 6

# Frequently Asked Questions

**Table of Contents:**



This page provides answers to commonly asked questions.

## Supported Platforms

The plugin currently supports connectivity to the following target operating systems:

- Arista EOS
- Cisco IOS
- Cisco NX-OS
- Cisco IOS-XE
- Cisco IronPort
- Cisco ASA
- Cisco ACS
- Citrix Netscaler OS
- Juniper SRX
- Juniper QFX
- Any Linux Distribution
- Nuage Networks VSC
- PaloAlto PAN-OS

- Any Linux Distribution

Pending development:

- F5 BIG-IP
- Fortinet FortiGate


*Back to Top*

# Index