Getting Device Information with Ansible

Table of Contents

1. AIM . . . 1
1.1. Introduction to Ansible . . . 1
1.2. Ansible Context . . . 1
2. Related and background material . . . 2
3. Installation of Ansible . . . 3
3.1. Ansible Installation . . . 3
3.1.1. Ansible Versions and Naming . . . 3
3.1.1.1. Ansible Engine . . . 3
3.1.1.2. Ansible Inventory . . . 3
3.1.1.3. Ansible-vault . . . 4
3.1.1.4. Ansible-vault Configuration . . . 4
3.2. Modules . . . 4
3.2.1. Raw module . . . 5
3.2.1.1. Raw module output from basic command line tests . . . 5
3.2.2. Rest API calls from Ansible . . . 7
3.2.3. Modules Maintained by the Ansible Core Team . . . 7
3.2.4. Core Network Modules . . . 9
3.2.5. Further Modules . . . 10
4. Example platforms to be queried . . . 11
4.1. Juniper . . . 12
4.1.1. Pre-requisites . . . 13
4.1.2. Playbooks . . . 13
4.1.2.1. Configuration retrieval with juniper_junos_config . . . 13
4.1.2.2. Inventory retrieval with juniper_junos_facts . . . 14
4.1.2.3. MAC table retrieval with juniper_junos_table . . . 15
4.1.2.4. ARP table retrieval with juniper_junos_table . . . 17
4.1.3. Debugging . . . 17

Author: Sean Donnellan

"The way you deal with automation is by upgrading people's skills so they can get the jobs of the future."

1. AIM

This document/section is designed to be a starting point for retrieval of general network device configuration and information. This is seen in the context of feeding larger DBs or information systems so that the overall aim is to show how to use Ansible to populate a DB. Ansible can also be used to manage operating systems and applications and this is not discussed here.

1.1. Introduction to Ansible

The document describes an Ansible environment which is primarily used to manage components, such as network components, that can be reached through SSH. There are a number of tools for software defined networking or network automation and Ansible is one that is primarily used to bring network devices into the fold. Higher level integration with orchestration engines is explicitly allowed. This document explains how this is to be done in the context of configuration retrieval and it also explains the internals of Ansible and its workings in relation to this environment as Ansible allows for embedded logic and is not just an adapter.

1.2. Ansible Context

Ansible is to be seen in the following context.

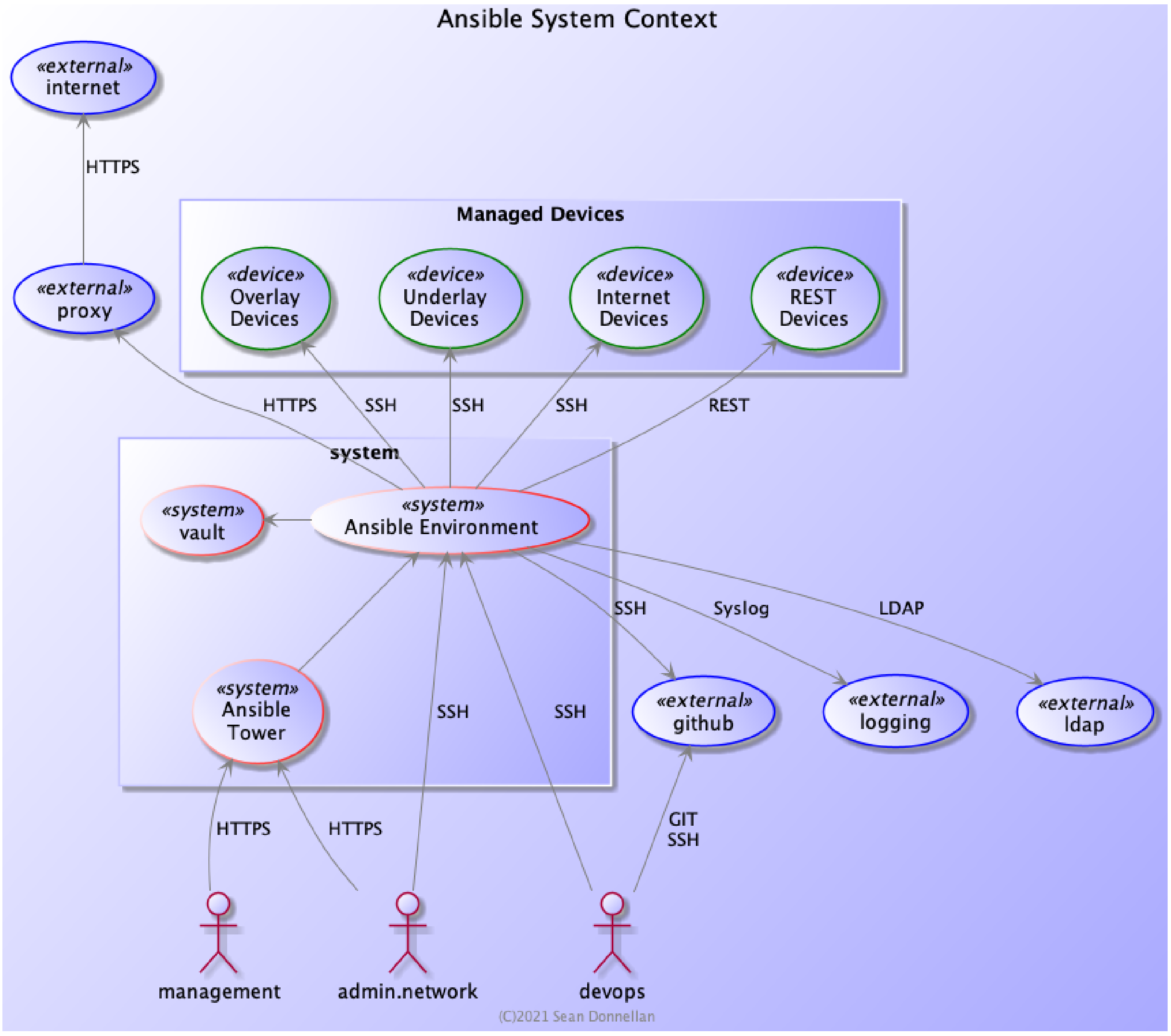

Figure 1. High level context of Ansible integration

2. Related and background material

Related information

How to build your inventory

https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#adding-variables-to-inventory

Some Ansible best practices

https://blog.theodo.com/2015/10/best-practices-to-build-great-ansible-playbooks/

3. Installation of Ansible

The section deals with the installation and initial setup of the solution. Items included are the integration and set-up of dependencies.

3.1. Ansible Installation

This section deals primarily with the initial set-up of Ansible and another section deals with the integration and logic that is involved with operating the environment through Ansible.

Related information

Recommended section - Installing Ansible with pip Ansible installation recommended procedure

https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html

3.1.1. Ansible Versions and Naming

The Ansible version being used is the open source version. Ansible is available as a command line tool and also with a GUI called Tower.

3.1.1.1. Ansible Engine

☑ One (Ansible) is provided and maintained by the open source community

3.1.1.2. Ansible Inventory

Related information

How to build your inventory

https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#adding-variables-to-inventory

Listing 1. Example INI format inventory

```
[JunipersHAint]
gw1-vSRX ansible_ssh_host=1.2.3.4
gw2-vSRX ansible_ssh_host=4.5.6.7
gw3-vSRX ansible_ssh_host=8.9.10.11
[JunipersHA]
junip02Node0 ansible_ssh_host=12.13.14.15
[JunipersHAint:vars]
ansible_python_interpreter=/usr/bin/python3.6
[JunipersLU:vars]
ansible_python_interpreter=/usr/bin/python3.6
```

Listing 2. Example YAML format inventory

```
all:
  hosts:
    10.251.2.10:
  vars:
    ansible_python_interpreter: /usr/bin/python3
    enable_plugins: yaml
  children:
    junos:
      hosts:
        10.251.2.2:
      vars:
        user: marvin
        ansible_user: marvin
        ansible_connection: netconf
        ansible_network_os: junos
```

3.1.1.3. Ansible-vault

https://docs.ansible.com/ansible/latest/user_guide/vault.html

Ansible vault is to be used locally on the Ansible system for ephemeral data such as the username and password that are secret. As this data is more easily locally stored in order to keep the setup simple Ansible vault is used locally for this to make debugging and independent use possible during initial set-up and integration.

3.1.1.4. Ansible-vault Configuration

TBD

3.2. Modules

Ansible can be used with modules that expand its use and control certain elements of devices or complete device groups. A basic module that is described below is the raw module which is effectively a vanilla SSH access and other modules are, for example, a row of netscaler modules that control individual parts of a netscaler configuration. This section describes a number of initial or key modules as used in the environment.

3.2.1. Raw module

https://docs.ansible.com/ansible/2.5/modules/raw_module.html

Executes a low-down and dirty SSH command, not going through the module subsystem.

This is useful and should only be done in two cases.

1. The first case is installing python-simplejson on older (Python 2.4 and before) hosts that need it as a dependency to run modules, since nearly all core modules require it.
2. Another is speaking to any devices such as routers that do not have any Python installed.

In any other case, using the shell or command module is much more appropriate.

- Arguments given to raw are run directly through the configured remote shell.
- Standard output, error output and return code are returned when available.
- There is no change handler support for this module.
- This module does not require python on the remote system, much like the script module.
- As noted above from the Ansible documentation this module is especially useful when access devices that are either not directly supported by other modules or do not have the pre-requisites to run the command or shell modules.
- This module is used when boarding new devices that require initial install steps before using other modules or for devices that are not supported by modules yet.

This module does not have a change handler and is therefore only for initial boarding or unsupported devices which need to have additional care taken due to the absent change handler.

3.2.1.1. Raw module output from basic command line tests

Notice in the following example that one device is in the .ssh/known_hosts and one is not which causes an error as shown. The other device which has been accessed by ssh manually prior to calling the raw module works fine. The inventory file called has passwords and two test IPs and is not shown here.

Listing 3. Raw module output

```
ansible -m raw -a "help" -i inventory-test netscalers-test
10.1.2.3 | FAILED | rc=-1 >>
Using a SSH password instead of a key is not possible because Host Key checking
```

```
is enabled and sshpass does not support this.  Please add this host's
fingerprint to your known_hosts file to manage this host.

10.1.2.3 | SUCCESS | rc=0 >>
Done

NetScaler command-line interface

Try :
'help <commandName>' for full usage of a specific command
'help <groupName>'   for brief usage of a group of commands
'help -all'          for brief usage of all CLI commands
'man <commandName>'  for a complete command description

'?' will show possible completions in the current context

The command groups are:
basic                app                  aaa
appflow              appfw                appqoe
audit                authentication       authorization
autoscale            ca                   cache
cli                  cluster              cmp
feo                  cr                   cs
db                   dns                  dos
event                filter               gslb
HA                   ica                  ipsec
lb                   lsn                  network
ns                   ntp                  policy
pq                   protocol             qos
rdp                  responder            rewrite
rise                 router               snmp
spillover            sc                   ssl
stream               system               subscriber
tm                   transform            tunnel
utility              vpn                  wi
wf                   smpp                 lldp
mediaclassifica      ulfd                 reputation
pcp
Done
################################################################################
#                                                                              #
#         WARNING: Access to this system is for authorized users only          #
#          Disconnect IMMEDIATELY if you are not an authorized user!           #
#                                                                              #
################################################################################

Shared connection to 10.1.2.3 closed.
```

3.2.2. Rest API calls from Ansible

https://docs.ansible.com/ansible/latest/modules/uri_module.html

3.2.3. Modules Maintained by the Ansible Core Team

Listing 4. https://docs.ansible.com/ansible/2.4/core_maintained.html

```
    acl - Sets and retrieves file ACL information.
    add_host - add a host (and alternatively a group) to the ansible-playbook
in-memory inventory
    apt - Manages apt-packages
    apt_key - Add or remove an apt key
    apt_repository - Add and remove APT repositories
    assemble - Assembles a configuration file from fragments
    assert - Asserts given expressions are true
    async_status - Obtain status of asynchronous task
    at - Schedule the execution of a command or script file via the at command.
    authorized_key - Adds or removes an SSH authorized key
    aws_s3 - manage objects in S3.
    blockinfile - Insert/update/remove a text block surrounded by marker lines.
    cloudformation - Create or delete an AWS CloudFormation stack
    command - Executes a command on a remote node
    copy - Copies files to remote locations
    debconf - Configure a .deb package
    debug - Print statements during execution
    dnf - Manages packages with the *dnf* package manager
    ec2 - create, terminate, start or stop an instance in ec2
    ec2_group - maintain an ec2 VPC security group.
    ec2_metadata_facts - Gathers facts (instance metadata) about remote hosts
within ec2
    ec2_snapshot - creates a snapshot from an existing volume
    ec2_vol - create and attach a volume, return volume id and device map
    ec2_vpc_net - Configure AWS virtual private clouds
    ec2_vpc_net_facts - Gather facts about ec2 VPCs in AWS
    ec2_vpc_subnet - Manage subnets in AWS virtual private clouds
    ec2_vpc_subnet_facts - Gather facts about ec2 VPC subnets in AWS
    fail - Fail with custom message
    fetch - Fetches a file from remote nodes
    file - Sets attributes of files
    find - Return a list of files based on specific criteria
    get_url - Downloads files from HTTP, HTTPS, or FTP to node
    getent - a wrapper to the unix getent utility
    git - Deploy software (or files) from git checkouts
    group - Add or remove groups
    group_by - Create Ansible groups based on facts
    import_playbook - Import a playbook
    import_role - Import a role into a play
    import_tasks - Import a task list
    include **(D)** - Include a play or task list
    include_role - Load and execute a role
    include_tasks - Dynamically include a task list
    include_vars - Load variables from files, dynamically within a task
    iptables - Modify the systems iptables
    lineinfile - Ensure a particular line is in a file, or replace an existing
line using a back-referenced regular expression.
    meta - Execute Ansible 'actions'
    mount - Control active and configured mount points
    package - Generic OS package manager
    pause - Pause playbook execution
    ping - Try to connect to host, verify a usable python and return ``pong`` on
success
    pip - Manages Python library dependencies.
    raw - Executes a low-down and dirty SSH command
    rhn_channel - Adds or removes Red Hat software channels
    rpm_key - Adds or removes a gpg key from the rpm db
    s3_bucket - Manage S3 buckets in AWS, Ceph, Walrus and FakeS3
    script - Runs a local script on a remote node after transferring it
    seboolean - Toggles SELinux booleans.
    selinux - Change policy and state of SELinux
    service - Manage services.
    set_fact - Set host facts from a task
    setup - Gathers facts about remote hosts
    shell - Execute commands in nodes.
    slurp - Slurps a file from remote nodes
    stat - Retrieve file or file system status
    subversion - Deploys a subversion repository.
    synchronize - A wrapper around rsync to make common tasks in your playbooks
quick and easy.
    sysctl - Manage entries in sysctl.conf.
    systemd - Manage services.
    template - Templates a file out to a remote server
    unarchive - Unpacks an archive after (optionally) copying it from the local
machine.
    uri - Interacts with webservices
    user - Manage user accounts
    wait_for - Waits for a condition before continuing
    wait_for_connection - Waits until remote system is reachable/usable
    win_acl - Set file/directory/registry permissions for a system user or group
    win_acl_inheritance - Change ACL inheritance
    win_command - Executes a command on a remote Windows node
    win_copy - Copies files to remote locations on windows hosts
    win_disk_image - Manage ISO/VHD/VHDX mounts on Windows hosts
    win_dns_client - Configures DNS lookup on Windows hosts
    win_domain - Ensures the existence of a Windows domain.
    win_domain_controller - Manage domain controller/member server state for a
Windows host
    win_domain_membership - Manage domain/workgroup membership for a Windows
host
```

```
     win_file - Creates, touches or removes files or directories.
     win_get_url - Fetches a file from a given URL
     win_group - Add and remove local groups
     win_owner - Set owner
     win_package - Installs/uninstalls an installable package
     win_path - Manage Windows path environment variables
     win_ping - A windows version of the classic ping module
     win_reboot - Reboot a windows machine
     win_regedit - Add, change, or remove registry keys and values
     win_service - Manages Windows services
     win_share - Manage Windows shares
     win_shell - Execute shell commands on target hosts.
     win_stat - returns information about a Windows file
     win_template - Templates a file out to a remote server.
     win_updates - Download and install Windows updates
     win_user - Manages local Windows user accounts
     yum - Manages packages with the *yum* package manager
     yum_repository - Add or remove YUM repositories
```

3.2.4. Core Network Modules

The following list is a list of lists of modules. For the complete list please follow the link. Each item in the list has at least one or more modules that are relevant to the item.

Listing 5. https://docs.ansible.com/ansible/latest/modules/list_of_network_modules.html

```
Network Modules
  A10
  Aci
  Aireos
  Aos
  Aruba
  Asa
  Avi
  Bigswitch
  Citrix
  Cloudengine
  Cloudvision
  Cumulus
  Dellos10
  Dellos6
  Dellos9
  Eos
  F5
  Fortios
  Illumos
  Interface
  Ios
  Iosxr
  Junos
  Layer2
  Layer3
  Lenovo
  Netconf
  Netscaler
  Netvisor
  Nuage
  Nxos
  Ordnance
  Ovs
  Panos
  Protocol
  Radware
  Routing
  Sros
  System
  Vyos
```

3.2.5. Further Modules

The list of Ansible modules is long:

https://docs.ansible.com/ansible/latest/modules/modules_by_category.html#modules-by-category

1. There are github repositories with custom modules
2. There are other modules apart from network:
    a. Cloud Modules
    b. Clustering Modules
    c. Commands Modules
    d. Crypto Modules
    e. Database Modules
    f. Files Modules
    g. Identity Modules
    h. Inventory Modules
    i. Messaging Modules
    j. Monitoring Modules
    k. Net Tools Modules
    l. Network Modules (extract shown above)
    m. Notification Modules
    n. Packaging Modules
    o. Remote Management Modules
    p. Source Control Modules
    q. Storage Modules
    r. System Modules
    s. Utilities Modules
    t. Web Infrastructure Modules
    u. Windows Modules
3. There are vendor maintained lists of custom modules
4. You can also create your own modules.

4. Example platforms to be queried

this section outline a non exclusive list of example devices that can be queried and how to get basic information from them.

4.1. Juniper

Related information

Juniper - Using Ansible to Retrieve Facts from Devices Running Junos OS

https://www.juniper.net/documentation/en_US/junos-ansible/topics/task/operational/junos-ansible-device-facts-retrieving.html

junos_facts – Collect facts from remote devices running Juniper Junos

https://docs.ansible.com/ansible/latest/modules/junos_facts_module.html

Using Ansible to Retrieve or Compare Junos OS Configurations

https://www.juniper.net/documentation/en_US/junos-ansible/topics/topic-map/junos-ansible-configuration-retrieving.html

Using Ansible with Junos PyEZ Tables to Retrieve Operational Information from Devices Running Junos OS

https://www.juniper.net/documentation/en_US/junos-ansible/topics/topic-map/junos-ansible-data-retrieving-using-junos-pyez-tables.html

Juniper PyEZ Cookbook

https://www.juniper.net/documentation/en_US/day-one-books/DO_PyEZ_Cookbook.pdf

Predefined Junos PyEZ Operational Tables (Structured Output)

https://www.juniper.net/documentation/en_US/junos-pyez/topics/reference/general/junos-pyez-tables-op-predefined.html

Netconf enabled Platform Options

https://docs.ansible.com/ansible/latest/network/user_guide/platform_netconf_enabled.html

How to Automate Juniper Devices Using Ansible | How Networks Work

https://medium.com/@gustavsthr/how-to-automate-juniper-devices-using-ansible-how-networks-work-92a0a3eb31bc

Juniper uses Ansible in conjunction with netconf to manage devices. Juniper also provides a number of packages in addition to what is provided by core Ansible. The links above provide details on the different options available and the examples below provide concrete tested examples.

4.1.1. Pre-requisites

1. Python is required
    a. Python 3 is preferred (over python 2 which is deprecated)
2. Ansible is required

Ansible can be installed with pip (python installer) or natively. When installing the open source version of Ansible for one user, for example for automation, it is recommended to do so with pip and potentially also in a separate python environment.

Listing 6. Unix (Linux) installation of Juniper related components

```
ansible-galaxy install Juniper.junos
pip3 install ncclient
pip3 install jxmlease
pip3 install junos-eznc
```

Listing 7. Junos commands to enable netconf

```
[edit system services]
user@host# set netconf ssh
user@host# commit
```

It is advisable to set up a special user for the automation. The above uses port 830 per default and more detailed explanations are available at he following link.

https://www.juniper.net/documentation/en_US/junos-ansible/topics/task/installation/junos-ansible-server-installing.html

4.1.2. Playbooks

The following are tested example playbooks that can be used as examples.

4.1.2.1. Configuration retrieval with juniper_junos_config

An initial use case for all devices is that configuration information be gathered directly after, or during, onboarding of the device. This is done so as to be able to ensure that the system is in a known state. A further reason for this is to enable the archiving of the configurations so as to be able to easily recover from unknown states.

Listing 8. playbooks/junos-get-config.yml

```yaml
---
- name: "Get Junos OS configuration."
  hosts: junos
  roles:
    - Juniper.junos
  connection: local
  gather_facts: no

  tasks:
    - name: "Get committed configuration"
      juniper_junos_config:
        retrieve: "committed"
        dest_dir: "{{ playbook_dir }}"
        format: "xml"
      register: response
    - name: "Print result"
      debug:
        var: response
```

for further details see: https://www.juniper.net/documentation/en_US/junos-ansible/topics/topic-map/junos-ansible-configuration-retrieving.html

4.1.2.2. Inventory retrieval with juniper_junos_facts

Inventory is vital to record the hardware and software that is being used and potentially needs to be replaced if it fails.

Listing 9. playbooks/junos-get-inventory.yml

```
---
- name: Check netconf and get device inventory for juniper device with junos
  hosts: junos
  roles:
    - Juniper.junos
  connection: local
  gather_facts: no

  tasks:
    - name: Checking NETCONF connectivity
      wait_for:
        host: "{{ inventory_hostname }}"
        port: 830
        timeout: 5
    - name: Retrieve device facts and inventory and save to file
      juniper_junos_facts:
        savedir: "{{ playbook_dir }}"
        config_format: "xml"
      register: result
    - name: Print result
      debug:
        var: result.ansible_facts.junos.config
```

4.1.2.3. MAC table retrieval with juniper_junos_table

MAC Table retrieval uses some features of PyEZ which is installed separately to Ansible. In this case the original PyEZ file ethernetswitchingtable.yml has been replaced with a modified version in order to work. the " key: mac-table-entry" line has been added. The PyEZ-mod-ethernetswitchingtable.yml file is the referenced from from the playbook file shown further below.

Listing 10. playbooks/PyEZ-mod-ethernetswitchingtable.yml

```
---
EthernetSwitchingTable:
    rpc: get-ethernet-switching-table-information
    args:
        detail: True
    item: ethernet-switching-table
    key: mac-table-entry
    view: EthernetSwitchingView

EthernetSwitchingView:
    fields:
        count: mac-table-count
        learned: mac-table-learned
        persistent: mac-table-persistent
        entries: _MacTableEntriesTable

_MacTableEntriesTable:
    item: mac-table-entry
    key: mac-vlan
    view: _MacTableEntriesView

_MacTableEntriesView:
    fields:
        vlan: mac-vlan
        vlan_tag: mac-vlan-tag
        mac_address: mac-address
        type: mac-type
        age: mac-age
        learned_time: mac-learned-time
        action: mac-action
        next_hop: mac-nexthop
        interface: mac-interface
        interface-list: _MacTableInterfacesTable

_MacTableInterfacesTable:
    item: mac-interfaces-list
    key: mac-interfaces
    view: _MacTableInterfacesView

_MacTableInterfacesView:
    fields:
        interfaces: mac-interfaces
```

Listing 11. playbooks/junos-get-MAC.yml

```yaml
---
- name: Get MAC information
  hosts: junos
  roles:
    - Juniper.junos
  connection: local
  gather_facts: no

  tasks:
    - name: Get MAC information
      juniper_junos_table:
        file: "PyEZ-mod-ethernetswitchingtable.yml"
        path: "{{ playbook_dir }}"
        table: "EthernetSwitchingTable"
      register: result

    - name: Print response
      debug:
        var: result
```

4.1.2.4. ARP table retrieval with juniper_junos_table

Listing 12. playbooks/junos-get-ARP.yml

```yaml
---
- name: Get ARP information
  hosts: junos
  roles:
    - Juniper.junos
  connection: local
  gather_facts: no

  tasks:
    - name: Get ARP information using Junos PyEZ Table
      juniper_junos_table:
        file: "arp.yml"
      register: result

    - name: Print response
      debug:
        var: result
```

4.1.3. Debugging

In some cases default scripts may not work or custom scripts may be required or an automation may fail to work. For some of these cases it helps to know how to attain answers that clarify the background of the queries.

Listing 13. testing netconf

```
prompt% ssh root@10.251.2.2 -t -s netconf
Password:
<!-- No zombies were killed during the creation of this user interface -->
<!-- user root, class super-user -->
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:confirmed-
commit:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.0</capability>

<capability>urn:ietf:params:netconf:capability:url:1.0?scheme=http,ftp,file</cap
ability>
    <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>

<capability>urn:ietf:params:xml:ns:netconf:capability:candidate:1.0</capability>
    <capability>urn:ietf:params:xml:ns:netconf:capability:confirmed-
commit:1.0</capability>

<capability>urn:ietf:params:xml:ns:netconf:capability:validate:1.0</capability>

<capability>urn:ietf:params:xml:ns:netconf:capability:url:1.0?protocol=http,ftp,
file</capability>
    <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
    <capability>http://xml.juniper.net/dmi/system/1.0</capability>
  </capabilities>
  <session-id>8291</session-id>
</hello>
]]>]]>
<!-- session end at 2020-04-29 17:51:52 CEST -->
Connection to 10.251.2.2 closed.
```

The above output shows what to expect if connecting to netconf on port 830 via ssh.

Listing 14. Junos - displaying the XML RPC call that will provide the information required to retrieve a MAC table

```
root@core> show ethernet-switching table detail |display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/14.1X53/junos">
    <rpc>
        <get-ethernet-switching-table-information>
                <detail/>
        </get-ethernet-switching-table-information>
    </rpc>
    <cli>
        <banner>{master:0}</banner>
    </cli>
</rpc-reply>

{master:0}
```

Listing 15. Junos - displaying the XML output of a MAC table

```
root@core> show ethernet-switching table detail |display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/14.1X53/junos">
    <ethernet-switching-table-information
xmlns="http://xml.juniper.net/junos/14.1X53/junos-esw" junos:style="extensive">
        <ethernet-switching-table junos:style="extensive">
            <mac-table-count>34</mac-table-count>
            <mac-table-learned>28</mac-table-learned>
            <mac-table-persistent>0</mac-table-persistent>
            <mac-table-entry junos:style="extensive">
                <mac-vlan>CP</mac-vlan>
                <mac-vlan-tag>328</mac-vlan-tag>
                <mac-address>*</mac-address>
                <mac-interface>All-members</mac-interface>
                <mac-interfaces-list junos:style="flood-list">
                    <mac-interfaces>ge-0/0/0.0</mac-interfaces>
                    <mac-interfaces>ge-0/0/3.0</mac-interfaces>
                    <mac-interfaces>ge-0/0/7.0</mac-interfaces>
                    <mac-interfaces>ge-0/0/5.0</mac-interfaces>
                    <mac-interfaces>ge-0/0/9.0</mac-interfaces>
                    <mac-interfaces>ge-0/1/1.0</mac-interfaces>
                    <mac-interfaces>ge-0/0/11.0</mac-interfaces>
                    <mac-interfaces>ge-0/0/2.0</mac-interfaces>
                </mac-interfaces-list>
                <mac-type>Flood</mac-type>
                <mac-action>Forward</mac-action>
                <mac-nexthop>1362</mac-nexthop>
            </mac-table-entry>
```

The above output was used to cross check why the provided PyEZ ethernetswitchingtable.yml file did not provide the results required. In the above case "mac-table-entry" was missing as a key and was added to a modified version of the yaml and then added to the playbooks directory. The modified version is also shown above.

Listing 16. Docker command to connect to devices directly through Python with PyEZ interactively

```
(venv) % docker run -it --rm --name pyez-interactive juniper/pyez python
Python 3.6.8 (default, Apr 22 2019, 10:28:12)
[GCC 6.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from jnpr.junos import Device
>>> import sys
>>> dev = Device('hostnameOrIP', user='username', passwd='password')
>>> dev.open()
Device(...)
>>> print(dev.facts)
{...output here...}
>>> quit()
(venv) %
```

Listing 17. Docker command to connect to PyEZ and run Python scripts from the local directory

```
(venv) % docker run -it --rm --name pyez -v "$(pwd)":/scripts juniper/pyez sh
/scripts # exit
(venv) %
```

see https://github.com/Juniper/py-junos-eznc/blob/master/DOCKER-EXAMPLES.md for further examples.