Network Automation with Ansible

(Cisco ios network devices)

By Ershad RAMEZANI

Table of Contents

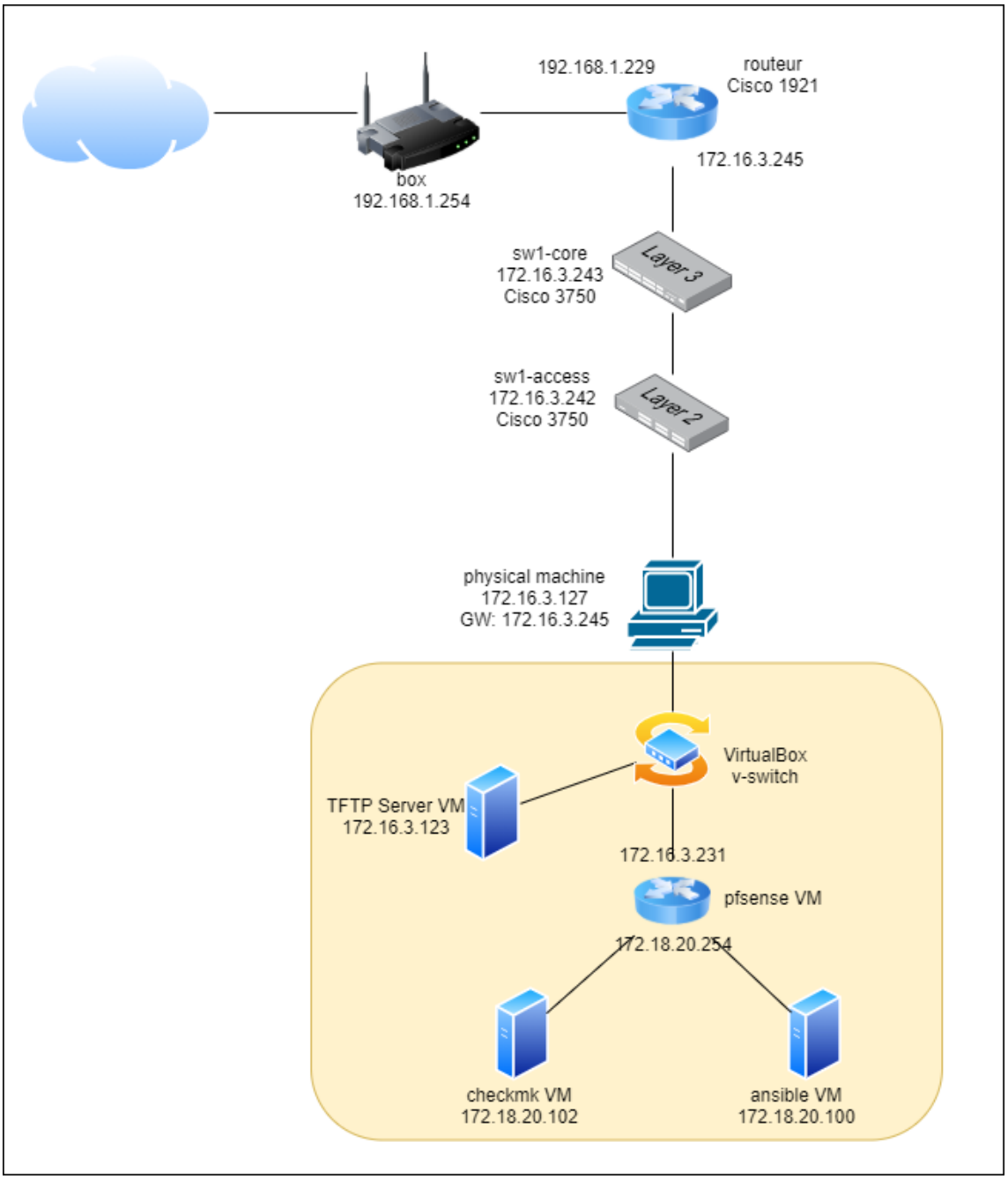
Preface

In this article we are going to learn how to automate configuration of Cisco network devices using Ansible. The main goal of this article is to give you a general overview of how to create your Ansible inventory, introduce you to the different modules that are used for network automation by giving some practical examples, discovering the *collections* that are used for network automation and different *connections* plugins required by these collections.

But first thing first, we will start by explaining the network topology that we have implemented in this article.

Topology

This topology consists of three Cisco devices. Two switches are connecting in line to a router and all three are configured in 172.16.3.0/24 network. We also have three VMs on a VirtualBox hypervisor with host-only network configuration in 172.18.20.0/24 network. We will use ansible server VM to configure the network devices such as backing up running configuration and the image files into TFTP server and configuring SNMP protocol on network devices in order to monitor them using *CheckMK* monitoring server.



Ansible installation

I'm going to install the ansible core version 2.3 on an ubuntu machine. Ansible installation is as simple as these three commands (plus apt update and apt upgrade). Please refer to this link for Ansible documentation:

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

```
sudo apt update & sudo apt upgrade
sudo apt install software-properties-common
sudo apt-add-repository --yes --update ppa:ansible/ansible
sudo apt update
sudo apt install ansible -y
```

The ansible root directory will be created in */etc/ansible* which by default contains the *ansible.cfg* file, the *hosts* file and the empty *roles* folder. Ansible.cfg is the configuration file where ansible will look for its configuration parameters. Hosts file is the default inventory for all the hosts that can be managed by Ansible.

We can create our project in the directory of our choice. I am going to create it in my home directory.

```
cd /home/ansible
mkdir cisco_ios_project
```

inventory

The inventory hosts file is in .ini format and contains all the hosts that can be managed by ansible. Hosts can be written as a single host name (aliases) or as a group. In addition to theirs names, we can add different variables for each or a group of hosts. For example adding *ansible_host* variable for each host to define the FQND or IP address of the host to guaranty its reachability within the ansible infrastructure. Or the *ansible_user* variable for a group to define the user with whom Ansible can connect to a group of hosts.

```
[router]
R1 ansible_host=172.16.3.254

[core]
sw1-core ansible_host=172.16.3.243

[access]
sw1-access ansible_host=172.16.3.242

[lan:children]
core
access

[network:children]
router
lan

[network:vars]
ansible_user=cisco
ansible_password= cisco
```

Note: All codes in this article are available in my github at https://github.com/ershad-ra/ansible_ios_automation

But in a large network infrastructure, this kind of inventory files can get soon complicated to manage. In order to facilitate the inventory management, we can use another method based on inventory directories and files. In this method, beside the hosts file which contains hosts and groups, we will create two folders named *group_vars* and *host_vars*. The group_vars contains the files for group variables and host_vars contain the hosts specific variables. These files are in *yaml* format (unlike the hosts file) and are named according to their corresponding host or group name in the hosts file.

Tree command on inventory folder	Hosts file	Group_vars/Network.yml	Host_vars/R1.yml
inventory ├── group_vars │ └── network.yml ├── hosts └── host_vars ├── R1.yml ├── sw1-access.yml └── sw1-core.yml	[router] R1 [core] sw1-core [access] sw1-access [lan:children] core access [network:children] router lan	ansible_user: cisco ansible_password: cisco	ansible_host: 172.16.3.245

Ansible.cfg file

If we execute the ansible all --list-hosts command in the CLI, we will get the following message:

```
ansible@master:~/cisco_ios_project$ ansible all --list-hosts
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'
  hosts (0):
```

This warning message means that there is no hosts configured in the default hosts file. And only localhost is available (Ansible by default adds localhost to this list). That is because the default location for inventory hosts file is */etc/ansible/hosts*. To change this default behavior, we will create a new ansible.cfg file in our project directory and reconfigure this default parameter.

```
[defaults]
inventory=/home/ansible/cisco_ios_project/inventory/hosts
```

```
ansible@master:~/cisco_ios_project$ ansible all --list-hosts
  hosts (3):
    R1
    sw1-core
    sw1-access
```

Note that according to Ansible precedence rules, the ansible.cfg file located in the current directory where we are executing the ansible command has the highest priority, and the home directory sits in second place. So although there is an ansible.cfg file located in */etc/ansible* directory, our project specific ansible.cfg file will be considered.

Changes can be made and used in a configuration file which will be searched for in the following order:

- `ANSIBLE_CONFIG` (environment variable if set)
- `ansible.cfg` (in the current directory)
- `~/.ansible.cfg` (in the home directory)
- `/etc/ansible/ansible.cfg`

Ansible will process the above list and use the first file found, all others are ignored.

Modules, plugins and collections

Ansible modules are units of code that can control system resources or execute system commands. Modules can be executed directly on remote hosts with ad-hoc commands or through playbooks. almost all the ansible modules are written in python languages (some of Windows modules are written in PowerShell). That is why python installation and its version on ansible controller and the nodes is important.

Similar to modules are plugins, which are pieces of code that extend core Ansible functionality.

A list of modules and plugins can come together and make a collection. Collections are a distribution format for Ansible content that can include playbooks, roles, modules and plugins. We can install collections through a distribution server such as Ansible Galaxy. The command for installing collections is *ansible-galaxy*.

Ansible core already comes with some preinstalled collections. We can list these collections with *ansible-galaxy collection list* command. This picture is just a snippet of the collections prompt in my CLI:

Modules are gathered into collections such as *netcommon*, *windows*, *ios*, etc. and collections are gathered into namespaces such as *cisco*,

community, *amazon*, etc. In the link below you can see an index list of the existing collections and also the namespaces these collections belong to:

https://docs.ansible.com/ansible/latest/collections/index.html

In order to refer to a specific module, you can mention its *FQCN* name which consist of three parts:

Namespace.collection.module	**Such as: cisco.ios.ios_config**

In this article we are going to use modules from these collections that are preinstalled in Ansible core:

- **Ansible.builtin**
- **Ansible.netcommon**
- **Cisco.ios**

Let's review some modules in these collections. We will use some them in this article:

- **Ansible.builtin:**
  - Copy – Copy files to remote locations
  - File – Manage files and file properties
  - gather_facts – Gathers facts about remote hosts
  - raw– Executes a low-down and dirty command
  - command– Execute commands on targets
- **ansible.netcommon:**
  - cli_command: Run a cli command on cli-based network devices
  - cli_config – Push text based configuration to network devices over network_cli
  - net_get – Copy a file from a network device to Ansible Controller
  - net_ping – Tests reachability using ping from a network device
  - net_put – Copy a file from Ansible Controller to a network device
  - netconf_ module – netconf device configuration
  - netconf_get – Fetch configuration/state data from NETCONF enabled network devices
- **Cisco.ios:**
  - ios_banner – Module to configure multiline banners
  - Ios_command – Module to run commands on remote devices
  - Ios_config – Module to manage configuration sections
  - Ios_facts – Module to collect facts from remote devices
  - Ios_hostname – Resource module to configure hostname
  - Ios_interface_– Resource module to configure interfaces
  - Ios_ping – Tests reachability using ping from IOS switch
  - Ios_snmp_server – Resource module to configure snmp server
  - Ios_vlans – Resource module to configure VLANs

Parameters and attributes

For each module to execute its task on the targets there are some parameters and attributes that we need to define. To know more about these elements we can visit the ansible documentation website. There are also some examples how to use the module:

```
- name: configure interface settings
  cisco.ios.ios_config:
    lines:
    - description test interface
    - ip address 172.31.1.1 255.255.255.0
    parents: interface Ethernet1
```

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_config_module.html

Connections plugins

Connection plugins allow Ansible to connect to the target hosts so it can execute tasks on them. Ansible ships with many connection plugins, but only one can be used per host at a time. The most commonly used are the *paramiko SSH*, *native ssh* (just called ssh), and *local* connection types. There are also *network_cli* and *winrm* connections:

Ansible.builtin.paramiko_ssh: It uses *Paramiko* which is a python implementation of SSH to connect to and run tasks on the targets.

Ansible.builtin.local: This connection plugin allows ansible to execute tasks on the Ansible controller instead of on a remote host.

Ansible.builtin.ssh: This connection plugin allows Ansible to communicate to the target machines through normal SSH command line.

Ansible.builtin.winrm: Runs tasks over Microsoft's *WinRM*. Windows Remote Management is a Windows-native built-in remote management protocol that lets network administrators to access, edit and update data from local and remote computers.

Ansible.netcommon.network_cli: This connection plugin provides a connection to remote devices over the SSH and implements a CLI shell. This connection plugin is typically used by network devices for sending and receiving CLi commands to network devices. *ansible-pylibssh* is needed on the local controller which executes this connection.

ansible.netcommon.netconf: Provides a persistent connection using the *netconf* protocol. NETCONF is a network management protocol allowing an NMS (network management system) to deliver, modify, and delete configurations of network devices through standard APIs on devices.

https://info.support.huawei.com/info-finder/encyclopedia/en/NETCONF.html

ansible.netcommon.libssh: Runs tasks using *libssh* for ssh connection. *libssh* is a multiplatform C library implementing the SSHv2 protocol on client and server side. With libssh, you can remotely execute programs, transfer files, use a secure and transparent tunnel, manage public keys and much more.

Connection plugins pour each collection are mentioned at the end of the documentation page:

https://docs.ansible.com/ansible/latest/plugins/connection.html

https://docs.ansible.com/ansible/latest/collections/ansible/netcommon/index.html

You can check the list of connections plugins installed on ansible core with this command:

```
ansible-doc -t connection -l
```

```
ansible@master:~/cisco_ios_project$ ansible-doc -t connection -l
ansible.netcommon.grpc          Provides a persistent connection using the gRPC protocol
ansible.netcommon.httpapi       Use httpapi to run command on network appliances
ansible.netcommon.libssh        Run tasks using libssh for ssh connection
ansible.netcommon.napalm        Provides persistent connection using NAPALM
ansible.netcommon.netconf       Provides a persistent connection using the netconf protocol
ansible.netcommon.network_cli   Use network_cli to run command on network appliances
ansible.netcommon.persistent    Use a persistent unix socket for connection
community.aws.aws_ssm           execute via AWS Systems Manager
community.docker.docker         Run tasks in docker containers
community.docker.docker_api     Run tasks in docker containers
community.docker.nsenter        execute on host running controller container
community.general.chroot        Interact with local chroot
```

Ad-hoc commands

Now that we know some basics about Ansibles different features, let's make our first communication with the hosts.

An Ansible ad hoc command uses the */usr/bin/ansible* command-line tool to automate a single task on one or more managed nodes.

Ansible.builtin.raw module

Here is an ad-hoc command to execute *ansible.builtin.raw* module on the *sw1-core* device:

```
ansible sw1-core -m raw -a "show version"
```

- Show version is a command that does not need the privileges rights.
- Note that the username and password for ssh connection are defined in *group_vars/network.yml* file.

```
ansible@master:~/cisco_ios_project$ ansible router -m raw -a "show version"
R1 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: Unable to negotiate with 172.16.3.245 port 22: no m
atching key exchange method found. Their offer: diffie-hellman-group-exchange-sha1,diffie-hellman-grou
p14-sha1,diffie-hellman-group1-sha1",
    "unreachable": true
}
```

The message we receive telling us that ansible tried to communicate with the switch but the key exchange method offering by the switch's ssh is not define for the SSH service that ansible is using.

The ssh server on ansible server is OpenSSH, so we need the define the method in OpenSSH. To do so, open */etc/ssh/ssh_config* file, add these three lines at the end of the file and restart the OpenSSH service.

```
Ciphers aes256-ctr,aes128-cbc,3des-cbc,aes192-cbc,aes256-cbc
KexAlgorithms +diffie-hellman-group1-sha1
HostKeyAlgorithms=ssh-rsa
```

Run again the ad-hoc command and we get another error message:

```
ansible@master:~/cisco_ios_project$ ansible router -m raw -a "show version"
R1 | 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 ho
st.
```

We need to disable *host key checking* to avoid ansible form checking the devices public key. Host key checking is implemented in order to verify the remote node's public key with the public keys already registered in system's *known_hosts* file to prevent man-in-the-middle attacks.

https://www.ibm.com/docs/en/zos/2.2.0?topic=program-host-key-checking

Disable host key checking by adding this line in ansible.cfg file:

```
[defaults]
inventory=/home/ansible/cisco_ios_project/inventory/hosts

host_key_checking=false
~
~
```

And run again the command. This time we are successfully getting the results:

```
ansible@master:~/cisco_ios_project$ ansible router -m raw -a "show version"
R1 | CHANGED | rc=0 >>
Cisco IOS Software, C1900 Software (C1900-UNIVERSALK9-M), Version 15.4(3)M3, RELEASE SOFTWARE (fc2)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2015 by Cisco Systems, Inc.
Compiled Fri 05-Jun-15 12:31 by prod_rel_team

ROM: System Bootstrap, Version 15.0(1r)M16, RELEASE SOFTWARE (fc1)

routeur uptime is 26 minutes
System returned to ROM by power-on
System image file is "usbflash0:c1900-universalk9-mz.SPA.154-3.M3.bin"
Last reload type: Normal Reload
Last reload reason: power-on
```

If we want to get only some parts of the show version command such as device and image name, we can filter them with grep:

```
ansible network -m raw -a "show version" | grep "CHANGED\|image"
```

```
ansible@master:~/cisco_ios_project$ ansible network -m raw -a "show version" | grep "CHANGED\|image"
R1 | CHANGED | rc=0 >>
System image file is "usbflash0:c1900-universalk9-mz.SPA.154-3.M3.bin"
sw1-core | CHANGED | rc=0 >>
System image file is "flash:/c3750-ipbasek9-mz.122-52.SE.bin"
sw1-access | CHANGED | rc=0 >>
System image file is "flash:/c2950-i6k2l2q4-mz.121-22.EA13.bin"
```

Ansible.netcommon collection

Cli_command module

In this example we are going to use another module named *cli_command* from *netcommon* collection:

```
ansible all -m cli_command -a "command='show version | inc image'"
```

And the result is:

```
ansible@master:~/cisco_ios_project$ ansible all -m cli_command -a "command='show version | inc image'"
sw1-core | FAILED! => {
    "changed": false,
    "msg": "Connection type ssh is not valid for this module"
}
R1 | FAILED! => {
    "changed": false,
    "msg": "Connection type ssh is not valid for this module"
}
sw1-access | FAILED! => {
    "changed": false,
    "msg": "Connection type ssh is not valid for this module"
}
```

In order to use *netcommon* collection, we need to define the appropriate connection plugin for this collection. As we saw earlier, one of these connections is *network_cli*. We should also define which kind of operation system the network device has, with *ansible_network_os* variable (a variable belong to *network_cli* connection plugin).

Add these two variables into *group_vars/network.yml*:

```
ansible_user: cisco
ansible_password: cisco
ansible_connection: network_cli
ansible_network_os: ios
```

Run again the command:

```
ansible all -m cli_command -a "command='show version | inc image'"
```

```
ansible@master:~/cisco_ios_project$ ansible all -m cli_command -a "command='show version | inc image'"
[WARNING]: ansible-pylibssh not installed, falling back to paramiko
[WARNING]: ansible-pylibssh not installed, falling back to paramiko
[WARNING]: ansible-pylibssh not installed, falling back to paramiko
sw1-core | SUCCESS => {
    "changed": false,
    "stdout": "System image file is \"flash:/c3750-ipbasek9-mz.122-52.SE.bin\"",
    "stdout_lines": [
        "System image file is \"flash:/c3750-ipbasek9-mz.122-52.SE.bin\""
    ]
}
R1 | SUCCESS => {
    "changed": false,
    "stdout": "System image file is \"usbflash0:c1900-universalk9-mz.SPA.154-3.M3.bin\"",
    "stdout_lines": [
        "System image file is \"usbflash0:c1900-universalk9-mz.SPA.154-3.M3.bin\""
    ]
}
sw1-access | SUCCESS => {
    "changed": false,
    "stdout": "System image file is \"flash:/c2950-i6k2l2q4-mz.121-22.EA13.bin\"",
    "stdout_lines": [
        "System image file is \"flash:/c2950-i6k2l2q4-mz.121-22.EA13.bin\""
    ]
}
ansible@master:~/cisco_ios_project$
```

As we saw earlier, *ansible-pylibssh* is required for *network_cli* connection. We can install it with these commands:

```
sudo apt install python3-pip
sudo pip3 install ansible-pylibssh
```

let's see another example of *cli_command*:

Net_ping module

This module tests reachability using ping from network device to a remote destination.

```
ansible R1 -m net_ping -a "dest='8.8.8.8'"
```

```
ansible@master:~/cisco_ios_project$ ansible R1 -m net_ping -a "dest='8.8.8.8'"
R1 | SUCCESS => {
    "changed": false,
    "commands": "ping ip 8.8.8.8",
    "packet_loss": "0%",
    "packets_rx": 5,
    "packets_tx": 5,
    "rtt": {
        "avg": 15,
        "max": 16,
        "min": 12
    }
}
```

Cisco.ios collection

Ios_facts module

An example of *cisco.ios* modules is the *ios_facts* module. it captures some facts about the ios device as in the picture:

```
ansible R1 -m ios_facts
```

```
ansible@master:~/cisco_ios_project$ ansible R1 -m ios_facts
R1 | SUCCESS => {
    "ansible_facts": {
        "ansible_net_api": "cliconf",
        "ansible_net_gather_network_resources": [],
        "ansible_net_gather_subset": [
            "default"
        ],
        "ansible_net_hostname": "routeur",
        "ansible_net_image": "usbflash0:c1900-universalk9-mz.SPA.154-3.M3.bin",
        "ansible_net_iostype": "IOS",
        "ansible_net_model": "CISCO1921/K9",
        "ansible_net_python_version": "3.10.6",
        "ansible_net_serialnum": "FCZ2109400W",
        "ansible_net_system": "ios",
        "ansible_net_version": "15.4(3)M3",
        "ansible_network_resources": {}
    },
    "changed": false
}
```

We can change the amount and type of these captured facts by adding the *gather_subset* parameter:

```
ansible R1 -m ios_facts -a "gather_subset=all"
```

```
ansible@master:~/cisco_ios_project$ ansible R1 -m ios_facts -a "gather_subset=all"
R1 | FAILED! => {
    "changed": false,
    "msg": "command timeout triggered, timeout value is 30 secs.\nSee the timeout setting options in the Network Debug and Troubleshooting Guide."
}
```

We can see that the connection timeouts. The raison is that there are some facts that the module is trying to capture that needs the enable level privilege which we have not provided yet. To do so, we can add *-bK* parameters at the end of our ad-hoc command. *-b* forces ansible to use enable mode (become mode in ansible) and *-K* forces ansible to ask for enable password which is *cisco* in our case.

```
ansible@master:~/cisco_ios_project$ ansible R1 -m ios_facts -a "gather_subset=all" -bK
BECOME password:
R1 | SUCCESS => {
    "ansible_facts": {
        "ansible_net_all_ipv4_addresses": [
            "192.168.1.229",
            "172.16.3.245"
        ],
        "ansible_net_all_ipv6_addresses": [],
        "ansible_net_api": "cliconf",
        "ansible_net_config": "Building configuration...\n\nCurrent configuration : 1578 bytes\n!\n! Last configu
stamps debug datetime msec\nservice timestamps log datetime msec\nno service password-encryption\n!\nhostname rou
xGYAPZHXmCs0\n!\nno aaa new-model\n!\n!\n!\n!\n!\n!\n!\n!\n!\n!\n!\n!\n!\n!\nip domain name ershad\nip cef\nno ipv
i pid CISCO1921/K9 sn FCZ2109400W\n!\n!\nusername ershad password 0 ershad\nusername admin password 0 admin\nuser
```

This time it captured lots of facts including interfaces' details.

We can also add these two parameters into *group_vars/network.yml* to avoid adding them into ad-hoc commands:

```
ansible_user: cisco
ansible_password: cisco
ansible_connection: network_cli
ansible_network_os: ios
ansible_become: yes
ansible_become_password: cisco
ansible_become_method: enable
```

Playbooks

Ansible Playbooks are lists of tasks that automatically execute against hosts. They are written into *yaml* format. We are going to create a folder named playbooks in our project directory and place all playbooks into that folder.

Show commands with ios_command module in playbook

To execute a playbook we use *ansible-playbook* program. Here is an example of a playbook for show version and clock command we saw earlier:

```
---
- name: show run for all
  hosts: network
  tasks:
    - name: run show version commands
      ios_command:
        commands:
          - show version | incl Version
          - show clock
      register: output

    - name: print output
      debug:
        var: output.stdout_lines
```

The first task executing *ios_command* module from *cisco.ios* collection with *commands* parameter to execute the *show version* command and *register* the result in a variable named *output*. The second task simply printing the *output* into CLI with the help on *debug* module from *ansible.builtin* collection.

The result of executing this playbook is the following:

```
PLAY [show run for all] ********************************************************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [R1]
ok: [sw1-core]
ok: [sw1-access]

TASK [run show version commands] ***********************************************************************************
ok: [R1]
ok: [sw1-core]
ok: [sw1-access]

TASK [print output] ************************************************************************************************
ok: [R1] => {
    "output.stdout_lines": [
        [
            "Cisco IOS Software, C1900 Software (C1900-UNIVERSALK9-M), Version 15.4(3)M3, RELEASE SOFTWARE (fc2)",
            "ROM: System Bootstrap, Version 15.0(1r)M16, RELEASE SOFTWARE (fc1)"
        ],
        [
            "*20:37:37.311 UTC Sat Nov 26 2022"
        ]
    ]
}
ok: [sw1-core] => {
    "output.stdout_lines": [
        [
            "Cisco IOS Software, C3750 Software (C3750-IPBASEK9-M), Version 12.2(52)SE, RELEASE SOFTWARE (fc3)",
            "BOOTLDR: C3750 Boot Loader (C3750-HBOOT-M) Version 12.2(44)SE5, RELEASE SOFTWARE (fc1)",
            "Switch Ports Model              SW Version            SW Image"
        ],
        [
            "*03:33:28.321 UTC Mon Mar 1 1993"
        ]
    ]
}
ok: [sw1-access] => {
    "output.stdout_lines": [
        [
            "IOS (tm) C2950 Software (C2950-I6K2L2Q4-M), Version 12.1(22)EA13, RELEASE SOFTWARE (fc2)"
        ],
        [
            "*03:34:10.739 UTC Mon Mar 1 1993"
        ]
    ]
}

PLAY RECAP *********************************************************************************************************
R1                         : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw1-access                 : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw1-core                   : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Copy output to file

We can add the *copy* module as another task into this playbook to *copy* the specific content into a file:

```
---
- name: show run for all
  hosts: network
  tasks:
    - name: run show version commands
      ios_command:
        commands:
          - show version | incl Version
          - show clock
      register: output

    - name:
      file:
        path: ./output
        state: directory
        mode: '0755'

    - name: copy output to file
      copy:
        content: "{{ output.stdout_lines | to_nice_yaml }}"
        dest: "./output/{{ inventory_hostname }}.show_commands.txt"
~
~
~
```

At first it creates a folder in the current directory (the directory where we are executing the command). And then it copy the *stdout_lines* list of each device into a seperate file thanks to *inventory_hostname* defined in dest parameter.

Note that we used another parameter named *to_nice_yaml* here which parse the content into human readable format. These kind of parameters are called *fiters*. We can use *filter* to manipulate data. Here is a link to more information about filters:

https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_filters.html

here is the content of output folder:

```
ansible@master:~/cisco_ios_project$ ls -l output/
total 12
-rw-rw-r-- 1 ansible ansible 231 nov.  27 12:37 R1.show_commands.txt
-rw-rw-r-- 1 ansible ansible 144 nov.  27 12:37 sw1-access.show_commands.txt
-rw-rw-r-- 1 ansible ansible 325 nov.  27 12:37 sw1-core.show_commands.txt
ansible@master:~/cisco_ios_project$
```

If we check the content of *R1.show_commands.txt* file:

```
- - Cisco IOS Software, C1900 Software (C1900-UNIVERSALK9-M), Version 15.4(3)M3,
    RELEASE SOFTWARE (fc2)
  - 'ROM: System Bootstrap, Version 15.0(1r)M16, RELEASE SOFTWARE (fc1)'
-   - '*11:34:57.847 UTC Sun Nov 27 2022'
~
~
~
~
```

DICTIONARY AND LIST

It is important to know that ansible uses dictionaries and lists to exchange data within processes such as gathering facts. So it is crucial to learn how to analyse and use data in lists and dictionaries.

Lists are the equivalent of an array in many programming languages and they are presented in two ways:

mylist: **- Item1** **- Item2** **- Item3**	**mylist: [item1, item2, item3]**

Lists are indexed by numbers starting with zero. So if i want to use the first entry (item1) in mylist, I use *mylist[0]* and for the second entry (item2) I use *mylist[1].*

Dictionaries are a collection of *key: value* data sets.

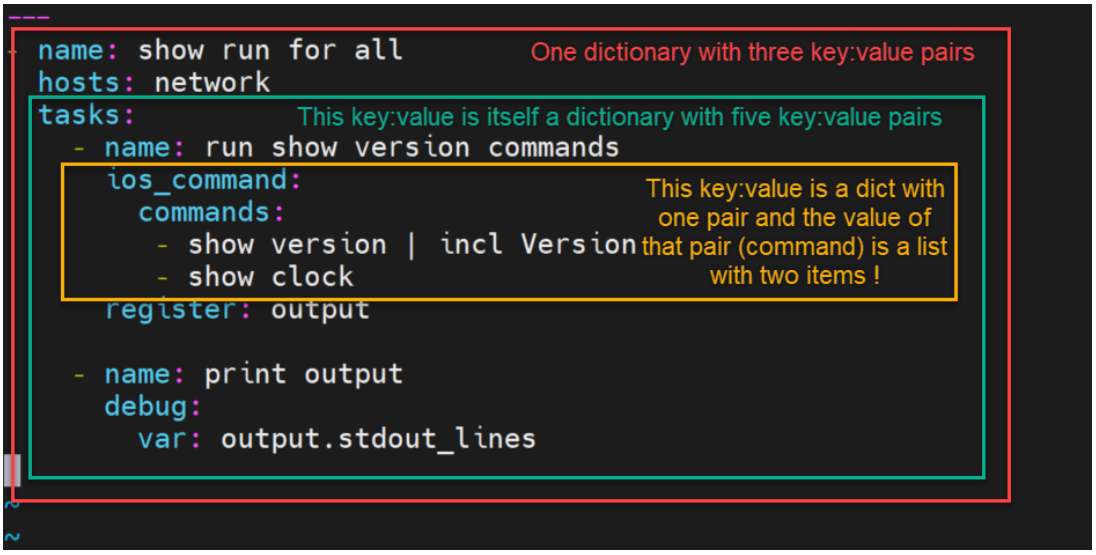
The difference between a list and a dictionary is that in the dictionary every value has a name (key) and not indexed by their positional values like 0,1,2 etc. Here is an example how dictionaries are written:

mydict: **key1: value1** **key2: value2** **key3: value3**	**Mydict: {key1:value1, key3:value2, key3:value3}**

If we want to point to a specific entry, we can use the bracket notation *mydict['key1']* to get *"value1"* string. we can also use *dot notation* like *mydict.key3* to get the *"value3"* string.

In real life, lists and dictionaries commonly appear combined.

Our previous playbook was consisted of dictionaries and lists. Let's look back into it:



The same rule is true for the output of this playbook. the result for each network device is a dictionary which can be identified with curly brackets {}. It has one *key:value* pair and the value for this key is itself a list consist of two lists as the items. The lists are identifiable with square brackets []. (Check it in playbook result in page 11).

```
Output (R1) = { key1: [ [ item1 ], [ item2 ] ] }
```

Note that *output.stdout_lines* is the dot notation method I mentioned earlier and is used to point the a key (stdout_lines) of the a dictionary (output).

Note: *stdout*, *stderr* and *stdin* respectively stands for standard output, standard error and standard input. They are used as containers for output, errors and input for any command we type in the terminal.

Gather_facts module in a playbook

Here we will create a playbook to extract *hostname*, *ios type*, *IP address* and *system version,* and prompt them in the CLI. To do that, we first execute an ad-hoc command with *gather_facts* module to verify all the outputs we could get:

```
ansible core -m gather_facts -a "gather_subset=all"
```

As you can see, this command has lots of output:

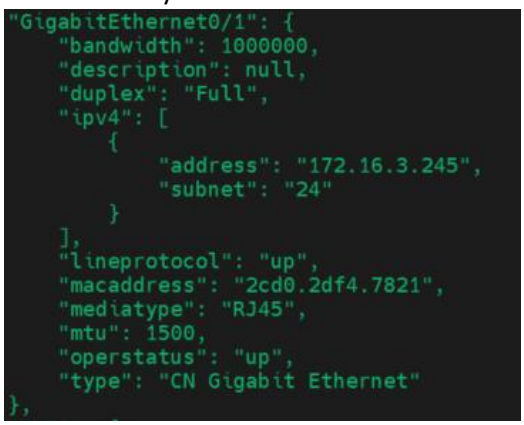
The entire output for R1 (*router* group) is a dictionary consisting of following key:value pairs:

```
{
“ansible_facts”: { . . . }
“changed”:false,
“deprecations”: [],
“warnings”: []
}
```

Ansible_facts is itself another dictionary consisting of several key:value pairs such as:

```
Ansible_net_all_ipv4_addresses:
Ansible_net_filesystems_info:
Ansible_net_hostname:
Ansible_net_image:
Ansible_net_interfaces:
And more.
```

Note that inside the *GigabitEthernet0/*1 we have the key ipv4 which its value is a list consisting of one item. That item itself is a dictionary with address and subnet as keys.



Now if I want to point to that address I would say:

```
ansible_facts.net_interfaces['GigabitEthernet0/1'].ipv4[0].address
```

```
---
- hosts: R1
  gather_facts: true
  gather_subset: all
  tasks:
    - name: print ipv4 address
      debug:
        msg:
            - "{{ ansible_facts.net_interfaces['GigabitEthernet0/1'].ipv4[0].address }}"
```

And the result:

```
ansible@master:~/cisco_ios_project$ ansible-playbook playbooks/gather_facts.yml

PLAY [R1] ***********************************************************************************************************

TASK [Gathering Facts] **********************************************************************************************
ok: [R1]

TASK [print ipv4 address] *******************************************************************************************
ok: [R1] => {
    "msg": [
        "172.16.3.245"
    ]
}

PLAY RECAP **********************************************************************************************************
R1                         : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Note that if we run this playbook against all the devices, we won't get the output for the switches as the variable has the meaning only for the router:

```
ansible@master:~/cisco_ios_project$ ansible-playbook playbooks/gather_facts.yml

PLAY [network] ******************************************************************************************************

TASK [Gathering Facts] **********************************************************************************************
ok: [R1]
ok: [sw1-core]
ok: [sw1-access]

TASK [print ipv4 address] *******************************************************************************************
fatal: [sw1-core]: FAILED! => {"msg": "The task includes an option with an undefined variable. The error was: 'dict
rs to be in '/home/ansible/cisco_ios_project/playbooks/gather_facts.yml': line 6, column 7, but may\nbe elsewhere
g line appears to be:\n\n  tasks:\n    - name: print ipv4 address\n      ^ here\n"}
fatal: [sw1-access]: FAILED! => {"msg": "The task includes an option with an undefined variable. The error was: 'di
ears to be in '/home/ansible/cisco_ios_project/playbooks/gather_facts.yml': line 6, column 7, but may\nbe elsewhere
ing line appears to be:\n\n  tasks:\n    - name: print ipv4 address\n      ^ here\n"}
ok: [R1] => {
    "msg": [
        "172.16.3.245"
    ]
}

PLAY RECAP **********************************************************************************************************
R1                         : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw1-access                 : ok=1    changed=0    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
sw1-core                   : ok=1    changed=0    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
```

We can capture more facts for R1 by adding more variables. But this time I am going to add some definition for each variable:

```
---
- hosts: R1
  gather_facts: true
  gather_subset: all
  connection: local

  tasks:
    - name: print ansible facts
      debug:
        msg:
          - "ios type: {{ ansible_facts.net_iostype }}"
          - "Hostname: {{ ansible_facts.net_hostname }}"
          - "IPv4 Addresses: {{ ansible_facts.net_all_ipv4_addresses }}"
          - "GigabitEthernet0/1 Interface: {{ ansible_facts.net_interfaces['GigabitEthernet0/1'].ipv4[0].address }}"
          - "Image name and Location: {{ ansible_facts.net_image }}"
          - "API: {{ ansible_facts.net_api }}"
          - "Python Version: {{ ansible_facts.net_python_version }}"
          - "Serial Number: {{ ansible_facts.net_serialnum }}"
          - "OS Name: {{ ansible_facts.net_system }}"
          - "Version Number: {{ ansible_facts.net_version }}"
~
```

And the result is more understandable:

```
TASK [print ansible facts] ***********************************************************
ok: [R1] => {
    "msg": [
        "ios type: IOS",
        "Hostname: routeur",
        "IPv4 Addresses: ['192.168.1.229', '172.16.3.245']",
        "GigabitEthernet0/1 Interface: 172.16.3.245",
        "Image name and Location: usbflash0:c1900-universalk9-mz.SPA.154-3.M3.bin",
        "API: cliconf",
        "Python Version: 3.10.6",
        "Serial Number: FCZ2109400W",
        "OS Name: ios",
        "Version Number: 15.4(3)M3"
    ]
}
```

Ios_facts module in a playbook

This module belongs to *cisco.ios* collection and we will use this module in a new playbook to gather facts about R1 and sw1-core and print them out into CLI command line:

```
---
- name: "collect device facts"
  hosts: R1, core
  tasks:
    - name: "gather device facts"
      ios_facts:
      register: device_facts
    - debug:
        var: device_facts
~
~
~
```

The result is:

```
TASK [debug] *******************************************************************************
ok: [R1] => {
    "device_facts": {
        "ansible_facts": {
            "ansible_net_api": "cliconf",
            "ansible_net_gather_network_resources": [],
            "ansible_net_gather_subset": [
                "default"
            ],
            "ansible_net_hostname": "routeur",
            "ansible_net_image": "usbflash0:c1900-universalk9-mz.SPA.154-3.M3.bin",
            "ansible_net_iostype": "IOS",
            "ansible_net_model": "CISCO1921/K9",
            "ansible_net_python_version": "3.10.6",
            "ansible_net_serialnum": "FCZ2109400W",
            "ansible_net_system": "ios",
            "ansible_net_version": "15.4(3)M3",
            "ansible_network_resources": {}
        },
        "changed": false,
        "failed": false
    }
}
ok: [sw1-core] => {
    "device_facts": {
        "ansible_facts": {
            "ansible_net_api": "cliconf",
            "ansible_net_gather_network_resources": [],
            "ansible_net_gather_subset": [
                "default"
            ],
            "ansible_net_hostname": "sw1-core",
            "ansible_net_image": "flash:/c3750-ipbasek9-mz.122-52.SE.bin",
            "ansible_net_iostype": "IOS",
            "ansible_net_model": "WS-C3750G-24T",
            "ansible_net_python_version": "3.10.6",
            "ansible_net_serialnum": "CAT0805N157",
            "ansible_net_stacked_models": [
                "WS-C3750G-24T-E"
```

As you can see, the outputs are presented into predefined *key:value* pairs. You can see the list of these return values through this link:
https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_facts_module.html

Now what if we want to have this output into a report ? We can use the *blockinfile* module from *ansible.builtin* collection to build a yaml file called *facts.yml*. we use *Jinja2* expressions within the blockinfile module to customize and select the information we want to capture from the ansible facts that were captured from the ios_facts task.

```
---
- name: "collect device facts"
  hosts: network
  tasks:
    - name: "gather device facts"
      ios_facts:
      register: device_facts

    - name: "write Device Facts"
      blockinfile:
        path: ./facts.yml
        create: yes
        block: |
          device_facts:
          {% for host in play_hosts %}
          {% set node = hostvars[host] %}
            {{ node.ansible_net_hostname }}:
              serial_number: {{ node.ansible_net_serialnum }}
              ios_version: {{ node.ansible_net_version }}
              device_model: {{ node.ansible_net_model }}
              image_name: {{ node.ansible_net_image }}
          {% endfor %}
      run_once: yes
      delegate_to: localhost
```

When a text is surrounded by *{ % % }* delimiters, it means that there is some special function or code running. It is used when the test inside is not passed to the template, but rather a function or feature of the template language (like a for loop our an if conditional).
Here is the result of this playbook:

```
ansible@master:~/cisco_ios_project$ ansible-playbook playbooks/ios_facts.yml

PLAY [collect device facts] ****************************************************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [R1]
ok: [sw1-core]
ok: [sw1-access]

TASK [gather device facts] ******************************************************************************************
ok: [R1]
ok: [sw1-core]
ok: [sw1-access]

TASK [write Device Facts] *******************************************************************************************
changed: [R1 → localhost]

PLAY RECAP **********************************************************************************************************
R1                         : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw1-access                 : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw1-core                   : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

And let's check the content of the *facts.yml* file created in our project directory:

```
# BEGIN ANSIBLE MANAGED BLOCK
device_facts:
  router:
    serial_number: FCZ2109400W
    ios_version: 15.4(3)M3
    device_model: CISCO1921/K9
    image_name: usbflash0:c1900-universalk9-mz.SPA.154-3.M3.bin
  sw1-core:
    serial_number: CAT0805N157
    ios_version: 12.2(52)SE
    device_model: WS-C3750G-24T
    image_name: flash:/c3750-ipbasek9-mz.122-52.SE.bin
  sw1-access:
    serial_number: FOC0616X1Q0
    ios_version: 12.1(22)EA13
    device_model: WS-C2950-24
    image_name: flash:/c2950-i6k2l2q4-mz.121-22.EA13.bin
# END ANSIBLE MANAGED BLOCK
~
~
~
~
~
```

Special variables

Magic variables

You can access information about ansible operations, including the python version being used, the hosts and groups in inventory, and the directories for playbooks and roles, using magic variables. *Magic variable* names are reserved and should not be set *user variables* with these names.
The most commonly used magic variables are *hostvars*, *groups*, *group_names* and *inventoy_hostname*. With hostvars, we can access variables defined for any host in the play, at any point of the playbook. We can access ansible facts using the hostvars variable too, but only after we have gathered facts (our example above).
In our example above, we are using two magic variables: hostvars and play_hosts. Here is the list of magic variables:
https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html

connection variables

note that *connection variables* such as ansible_host and ansible_user are also *special variables*. You have their list in the same link above.

Back up running configuration with ios_command playbook

We can use ios_command module in a playbook with five tasks to backup running config of network devices:

```yaml
---
- name: backup configuration
  hosts: network
  gather_facts: false
  connection: network_cli

  vars:
    backup_root: ./backups

  tasks:
    - name: run show running-config on remote devices
      ios_command:
        commands: "show running-config"
      register: config

    - name: ensure backup folder is created
      file:
        path: "{{ backup_root }}"
        state: directory
      run_once: yes

    - name: ensure device folder is created
      file:
        path: "{{ backup_root }}/{{ inventory_hostname }}"
        state: directory

    - name: get timestamp
      command: date +%Y%m%d
      register: timestamp

    - name: get/copy backup
      copy:
        content: "{{ config.stdout[0] }}"
        dest: "{{ backup_root }}/{{ inventory_hostname }}/running-config_{{ timestamp.stdout }}"
~
```

1. It execute the running-config command on the device and register the result into config variable.
2. It ensures that backup folder is created in the current directory.
3. It ensures that device folder inside the backup folder.
4. It creates another variable named timestamp to register the current time and date to be able to use it in backup file naming.
5. It uses the copy module to copy the first item of the stdout list (*stdout[0]*) of config file content into the specified destination.

The result of running the playbook:

```
ansible@master:~/cisco_ios_project$ ansible-playbook playbooks/ios_command_backup.yml

PLAY [backup configuration] ****************************************************************************************

TASK [run show running-config on remote devices] *******************************************************************
ok: [R1]
ok: [sw1-core]
ok: [sw1-access]

TASK [ensure backup folder is created] *****************************************************************************
changed: [R1]

TASK [ensure device rolder is created] *****************************************************************************
changed: [R1]
changed: [sw1-core]
changed: [sw1-access]

TASK [get timestamp] ***********************************************************************************************
changed: [sw1-core]
changed: [sw1-access]
changed: [R1]

TASK [get/copy backup] *********************************************************************************************
changed: [sw1-core]
changed: [R1]
changed: [sw1-access]

PLAY RECAP *********************************************************************************************************
R1                         : ok=5    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw1-access                 : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw1-core                   : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Note that as we have several hosts in the playbook (network group), Ansible will perform all the tasks for every single host. Except the task with *run_once = yes* option.

Here is the content of backup folder:

```
ansible@master:~/cisco_ios_project$ tree backups/
backups/
├── R1
│   └── running-config_20221127
├── sw1-access
│   └── running-config_20221127
└── sw1-core
    └── running-config_20221127

3 directories, 3 files
ansible@master:~/cisco_ios_project$
```

And the content of *R1/running-config_20221127*:

```
Building configuration...

Current configuration : 1506 bytes
!
version 15.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname router
!
boot-start-marker
boot-end-marker
!
!
enable secret 5 $1$JwZm$TdMm6a4L8VxGYAPZHXmCs0
!
no aaa new-model
```

Back up running configuration with ios_config playbook

In this playbook we are going to use *ios_config* module to create a playbook for automating running configuration backup.

```
---
- hosts: network
  gather_facts: false
  connection: network_cli
  tasks:
    - name: backup config
      ios_config:
        backup: yes
~
~
```

This module by default will create a backup folder in the playbook's current directory and backup the running config of each device into a separate file and name it will device hostname and the timestamp at the time of execution. Here is the result:

```
ansible@master:~/cisco_ios_project$ ansible-playbook playbooks/ios_config_backup.yml

PLAY [network] *****************************************************************************************************

TASK [backup config] ***********************************************************************************************
changed: [R1]
changed: [sw1-core]
changed: [sw1-access]

PLAY RECAP *********************************************************************************************************
R1                         : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw1-access                 : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw1-core                   : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

```
ansible@master:~/cisco_ios_project$ ls -l playbooks/backup/
total 12
-rw-rw-r-- 1 ansible ansible 1637 nov.  27 00:58 R1_config.2022-11-27@00:58:09
-rw-rw-r-- 1 ansible ansible 1396 nov.  27 00:58 sw1-access_config.2022-11-27@00:58:11
-rw-rw-r-- 1 ansible ansible 3405 nov.  27 00:58 sw1-core_config.2022-11-27@00:58:09
```

```
Building configuration...

Current configuration : 1577 bytes
!
! Last configuration change at 22:48:10 UTC Sat Nov 26 2022 by cisco
!
version 15.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname router
!
boot-start-marker
boot-end-marker
!
!
enable secret 5 $1$JwZm$TdMm6a4L8VxGYAPZHXmCs0
!
no aaa new-model
!
!
!
"playbooks/backup/R1_config.2022-11-27@00:58:09" [noeol] 111L, 1637B                    1,1           Haut
```

As you can see this method is much more simpler.

Backup ios images to TFTP server

If we want to upgrade the ios image of our network devices, it is always a good practice to back up the existing images before the upgrade. We can back up the ios images with the help of a TFTP server.

First let's see how the backup works manually through CLI on a device:

With *dir flash:* command in enable mode, we can verify the files existing in the device's flash:

```
Directory of flash:/

    2  -rwx         108  Mar 01 1993 00:05:07 +00:00  info
    3  -rwx          66  Jan 01 1970 00:03:44 +00:00  env_vars
    4  drwx         640  Mar 01 1993 00:08:37 +00:00  html
   15  -rwx         108  Mar 01 1993 00:08:37 +00:00  info.ver
   16  -rwx          25  Mar 01 1993 00:09:24 +00:00  snmpengineid
   17  -rwx        1336  Mar 01 1993 00:05:52 +00:00  config.text
   18  -rwx        3159  Mar 01 1993 11:06:08 +00:00  config.old
   19  -rwx         736  Mar 01 1993 00:09:56 +00:00  vlan.dat
   20  -rwx     3721984  Jan 01 1970 00:35:34 +00:00  c2950-i6k2l2q4-mz.121-22.EA13.bin
   22  -rwx        1930  Mar 01 1993 00:05:52 +00:00  private-config.text

7741440 bytes total (2337792 bytes free)
```

The image has *.bin* extension. To copy it to TFTP server, type the command below and the image uploads to TFTP server. As you can see switch prompt some questions which are not suitable for automation. We can disable is with file prompt quiet command in configuration terminal mode:

```
sw1-access#copy flash:c2950-i6k2l2q4-mz.121-22.EA13.bin tftp://172.16.3.123/
Address or name of remote host [172.16.3.123]?
Destination filename [c2950-i6k2l2q4-mz.121-22.EA13.bin]?
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!
3721984 bytes copied in 28.496 secs (130614 bytes/sec)
sw1-access#
```

```
sw1-access#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
sw1-access(config)#file prompt quiet
sw1-access(config)#
```

Now let's create a playbook to automate this action for all our devices:

```
---
- name: backup images with tftp
  hosts: network
  gather_facts: no
  vars:
    ansible_command_timeout: 300
    tftp_server: 172.16.3.123
  tasks:
    - name: collecting device facts
      ios_facts:

    - name: enable file prompt quiet
      ios_config:
        lines: file prompt quiet

    - name: copy image to tftp server
      ios_command:
        commands:
          - command: "copy {{ ansible_net_image }} tftp://{{ tftp_server }}/"
~
~
```

This playbook has three tasks. The first one will gather facts which includes magic variables such as *ansible_net_image* (the location of the image in flash:). The second task will execute file prompt quiet. And the last one will execute the copy command on the device. The playbook perform the tasks in order and each task in parallel on all the devices in network group.

Note that the default command timeout which is the amount of time ansible should wait for response from the remote device, is set to 30 seconds. So we need to increase this amount to ensure that ansible will wait enough till all the images have been copied into TFTP server. To change this default for ansible we need to add this section in ansible file:

```
[persistent_connection]
command_timeout=200
```

here is the result of executing this playbook:

```
ansible@master:~/cisco_ios_project$ ansible-playbook playbooks/copy_image.yml

PLAY [backup images with tftp] ************************************************************

TASK [collecting device facts] ************************************************************
ok: [R1]
ok: [sw1-core]
ok: [sw1-access]

TASK [enable file prompt quiet] ***********************************************************
[WARNING]: To ensure idempotency and correct diff the input configuration lines should be similar
to how they appear if present in the running configuration on device
changed: [R1]
changed: [sw1-core]
changed: [sw1-access]

TASK [copy image to tftp server] **********************************************************
ok: [sw1-access]
ok: [sw1-core]
ok: [R1]

PLAY RECAP ********************************************************************************
R1                         : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued
=0    ignored=0
sw1-access                 : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued
=0    ignored=0
sw1-core                   : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued
=0    ignored=0
```

We can ignore this warning as the playbook execution was successful.

In TFTP Server log:

```
TFTP connected from 172.16.3.245:61356 on 11/27/2022 2:10:28 PM, binary, PUT. Completed, file name: C:\TFTP-Root\c1900-universalk9-mz.SPA.154-3.M3.bin.
TFTP connected from 172.16.3.243:51780 on 11/27/2022 2:07:31 PM, binary, PUT. Completed, file name: C:\TFTP-Root\c3750-ipbasek9-mz.122-52.SE.bin.
TFTP connected from 172.16.3.242:52989 on 11/27/2022 2:07:01 PM, binary, PUT. Completed, file name: C:\TFTP-Root\c2950-i6k2l2q4-mz.121-22.EA13.bin.
TFTP connected from 172.16.3.242:52989 on 11/27/2022 2:06:38 PM, binary, PUT. Started, file name: C:\TFTP-Root\c2950-i6k2l2q4-mz.121-22.EA13.bin.
TFTP connected from 172.16.3.243:51780 on 11/27/2022 2:06:38 PM, binary, PUT. Started, file name: C:\TFTP-Root\c3750-ipbasek9-mz.122-52.SE.bin.
TFTP connected from 172.16.3.245:61356 on 11/27/2022 2:06:38 PM, binary, PUT. Started, file name: C:\TFTP-Root\c1900-universalk9-mz.SPA.154-3.M3.bin.
```

Files backed up into server:

Nom	Modifié le	Type	Taille
c1900-universalk9-mz.SPA.154-3.M3.bin	27/11/2022 14:10	Fichier BIN	73 837 Ko
c2950-i6k2l2q4-mz.121-22.EA13.bin	27/11/2022 14:23	Fichier BIN	3 635 Ko
c3750-ipbasek9-mz.122-52.SE.bin	27/11/2022 14:07	Fichier BIN	10 850 Ko

Upgrading cisco ios image playbook

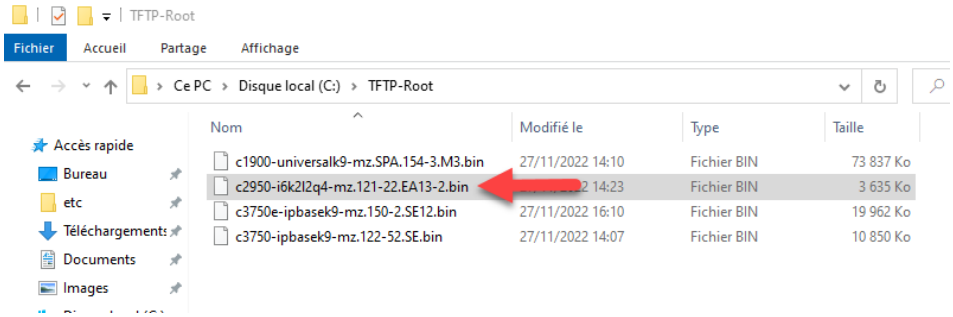
After backing up the current ios image, we would like to upgrade the image to a recent version. In order to do that we will create a playbook consist of three different plays.

This playbook is based on the work of Roger Perkin: https://www.youtube.com/watch?v=hLhHZ_uju2Q

I am going to execute this playbook on the sw1-access. If we check the ios_fact result of this switch we can find out the ios version and the image name:

```
ansible@master:~/cisco_ios_project$ ansible sw1-access -m ios_facts
sw1-access | SUCCESS => {
    "ansible_facts": {
        "ansible_net_api": "cliconf",
        "ansible_net_gather_network_resources": [],
        "ansible_net_gather_subset": [
            "default"
        ],
        "ansible_net_hostname": "sw1-access",
        "ansible_net_image": "flash:c2950-i6k2l2q4-mz.121-22.EA13-2.bin",
        "ansible_net_iostype": "IOS",
        "ansible_net_model": "WS-C2950-24",
        "ansible_net_python_version": "3.10.6",
        "ansible_net_serialnum": "FOC0616X1Q0",
        "ansible_net_system": "ios",
        "ansible_net_version": "12.1(22)EA13",
        "ansible_network_resources": {}
    },
    "changed": false
}
```

We have already backed up this image in the previous section and I will upload the same image into TFTP server:



I've also deleted the current *.bin* file from the flash to make space for the new upload (that could be done in playbook as well):

```
Directory of flash:/

    2  -rwx         108  Mar 01 1993 01:05:07 +01:00  info
    3  -rwx         114  Mar 01 1993 01:12:59 +01:00  env_vars
    4  drwx         640  Mar 01 1993 01:08:37 +01:00  html
   15  -rwx         108  Mar 01 1993 01:08:37 +01:00  info.ver
   16  -rwx          25  Mar 01 1993 01:09:24 +01:00  snmpengineid
   18  -rwx        3159  Mar 01 1993 12:06:08 +01:00  config.old
   19  -rwx         736  Mar 01 1993 01:09:56 +01:00  vlan.dat
   20  -rwx        1930  Mar 01 1993 01:13:01 +01:00  private-config.text
   22  -rwx        1354  Mar 01 1993 01:13:01 +01:00  config.text
```

The first play of the playbook verifies if the *ansible_net_version* is matching the *upgrade_ios_version*:

As my switch already has the latest version of the ios image and this lab is just for a test, I am just going to add a different name than the *ansible_net_version* to simulate the need for an upgrade:

```
---
- name: PLAY1 - Upgrade CISCO IOS
  hosts: sw1-access

  vars:
    upgrade_ios_version: 12.1(22)EA13-new

  tasks:
    - name: P1T1 - Check current version
      ios_facts:

    - debug:
        msg:
        - "Current version is {{ ansible_net_version }}"
        - "Upgrade image is {{ upgrade_ios_version }}"

    - debug:
        msg:
        - "Image is not compliant and will be upgraded"

      when: ansible_net_version != upgrade_ios_version
```

The second play has three tasks. First it will gather the *ansible_date_time* fact at the time of execution. Then it sets a variable for time and date. In the third task it will create a folder in our backups directory. It uses local system's hostvars for this purpose. Note that this play has ran only once and not as each remote device:

```
- name: PLAY2 - Create backup folder for today's date
  hosts: localhost
  tasks:
    - name: P2T1 - Get ansible date/time facts
      setup:
        filter: "ansible_date_time"
        gather_subset: "!all"

    - name: P2T2 - Store datetime as fact
      set_fact:
        datetime: "{{ ansible_date_time.date }}"

    - name: P2T3 - Create Directory {{hostvars.localhost.datetime}}
      file:
        path: ~/cisco_ios_project/backups/{{hostvars.localhost.datetime}}
        state: directory
  run_once: true
```

In the third play we have 9 tasks. From task 1 to 3 we backup running configuration and save the current configuration:

```
- name: PLAY3 - backup running configuration
  hosts: sw1-access
  vars:
    upgrade_ios_image: c2950-i6k2l2q4-mz.121-22.EA13-2.bin
    upgrade_ios_version: 12.1(22)EA13
    ansible_command_timeout: 600
    tftp_ip: 172.16.3.123

  tasks:
   - name: P3T1 - Backup Running Config
     ios_command:
       commands: show run
     register: config

   - name: P3T2 - Save output to ~/cisco_ios_project/backups/
     copy:
       content: "{{config.stdout[0]}}"
       dest: "~/cisco_ios_project/backups/{{hostvars.localhost.datetime}}/{{ inventory_hostname }}-{{hostvars.localhost.datetime}}-config.txt"

   - name: P3T3 - Save running config
     ios_config:
       save_when: always
```

In task 4 we upload the new image into the switch. In task 5 we set the new image as the boot image, in task 6 we reboot the device and in task 7 we ask ansible to wait for 90 seconds until the switch is came back up and then try to connect and continue the tasks flow. Note that Ansible server will need to be able to resolve the DNS hostname of the switch for this to work as written. To do so, I added the switch hostname and IP address into */etc/hosts* file.

```
- name: P3T4 - Copy Image // This could take up to 4 minutes
  ios_command:
    commands:
      - command: "copy tftp://{{ tftp_ip }}/{{ upgrade_ios_image }} flash:/{{ upgrade_ios_image }}"

- name: P3T5 - Change Boot Variable to new image
  ios_config:
    commands:
      - "boot system flash:{{ upgrade_ios_image }}"
    save_when: always

- name: P3T6 - Reload the Device
  cli_command:
    command: reload
    prompt:
      - confirm
    answer:
      - 'y'

- name: P3T7 - Wait for device to come back online
  wait_for:
    host: "{{ inventory_hostname }}"
    port: 22
    delay: 90
  delegate_to: localhost
```

We can see the switch is back on:

```
Réponse de 172.16.3.123 : Impossible de joindre l'hôte de destination.
Réponse de 172.16.3.123 : Impossible de joindre l'hôte de destination.
Réponse de 172.16.3.123 : Impossible de joindre l'hôte de destination.
Délai d'attente de la demande dépassé.
Réponse de 172.16.3.242 : octets=32 temps=1 ms TTL=255
Réponse de 172.16.3.242 : octets=32 temps=2 ms TTL=255
Réponse de 172.16.3.242 : octets=32 temps=2 ms TTL=255
Réponse de 172.16.3.242 : octets=32 temps=2 ms TTL=255
```

and the rest of the playbook will check if the version is the upgraded:

```
  - name: P3T8 - Check Image Version
    ios_facts:

  - debug:
      msg:
      - "Current version is {{ ansible_net_version }}"

  - name: P3T9 - ASSERT THAT THE IOS VERSION IS CORRECT
    assert:
      that:
        - upgrade_ios_version == ansible_net_version
  - debug:
      msg:
      - "Software Upgrade has been completed"
```

Here is some part of the playbook execution prompt:

```
TASK [P1T1 - Check current version] ********************************************************************************
ok: [sw1-access]

TASK [debug] ******************************************************************************************************
ok: [sw1-access] => {
    "msg": [
        "Current version is 12.1(22)EA13",
        "Upgrade image is 12.1(22)EA13-new"
    ]
}

TASK [debug] ******************************************************************************************************
ok: [sw1-access] => {
    "msg": [
        "Image is not compliant and will be upgraded"
    ]
}
```

```
PLAY [PLAY3 - backup running configuration] ************************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************************************************
ok: [sw1-access]

TASK [P3T1 - Backup Running Config] ********************************************************************************************************************
ok: [sw1-access]

TASK [P3T2 - Save output to ~/cisco_ios_project/backups/] **********************************************************************************************
changed: [sw1-access]

TASK [P3T3 - Save running config] **********************************************************************************************************************
changed: [sw1-access]

TASK [P3T4 - Copy Image // This could take up to 4 minutes] ********************************************************************************************
ok: [sw1-access]

TASK [P3T5 - Change Boot Variable to new image] ********************************************************************************************************
[WARNING]: To ensure idempotency and correct diff the input configuration lines should be similar to how they appear if present in the running
changed: [sw1-access]

TASK [P3T6 - Reload the Device] ************************************************************************************************************************
ok: [sw1-access]

TASK [P3T7 - Wait for device to come back online] *******************************************************************************************************
ok: [sw1-access -> localhost]

TASK [P3T8 - Check Image Version] **********************************************************************************************************************
ok: [sw1-access]

TASK [debug] *******************************************************************************************************************************************
ok: [sw1-access] => {
    "msg": [
        "Current version is 12.1(22)EA13"
    ]
}

TASK [P3T9 - ASSERT THAT THE IOS VERSION IS CORRECT] ***************************************************************************************************
ok: [sw1-access] => {
    "changed": false,
    "msg": "All assertions passed"
}

TASK [debug] *******************************************************************************************************************************************
ok: [sw1-access] => {
    "msg": [
        "Software Upgrade has been completed"
    ]
}

PLAY RECAP *********************************************************************************************************************************************
localhost                  : ok=4    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw1-access                 : ok=16   changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

And the new situation of the flash:

```
Directory of flash:/

    2  -rwx         108  Mar 01 1993 01:05:07 +01:00  info
    3  -rwx         114  Mar 01 1993 01:58:26 +01:00  env_vars
    4  drwx         640  Mar 01 1993 01:08:37 +01:00  html
   15  -rwx         108  Mar 01 1993 01:08:37 +01:00  info.ver
   16  -rwx          25  Mar 01 1993 01:09:24 +01:00  snmpengineid
   17  -rwx        1930  Mar 01 1993 01:58:28 +01:00  private-config.text
   18  -rwx        3159  Mar 01 1993 12:06:08 +01:00  config.old
   19  -rwx         736  Mar 01 1993 01:09:56 +01:00  vlan.dat
   20  -rwx     3721984  Mar 01 1993 01:58:23 +01:00  c2950-i6k2l2q4-mz.121-22.EA13-2.bin
   22  -rwx        1377  Mar 01 1993 01:58:28 +01:00  config.text
```

Configure SNMP on Cisco IOS with ios_config module

To configure the SNMPv3 on a cisco ios device, we need to first configure a group for SNMP server, SNMP version and the authpriv method with authentication and data encryption with the help of this command:

```
snmp-server group group123 v3 priv
```

Then we define the user who belongs to this group and will connect to the switch with his password (by sha hashing) and the encryption method for priv (by AES 128 encryption):

```
snmp-server user admin group123 v3 auth sha pass123 priv aes 128 pass123
```

This playbook will do the job:

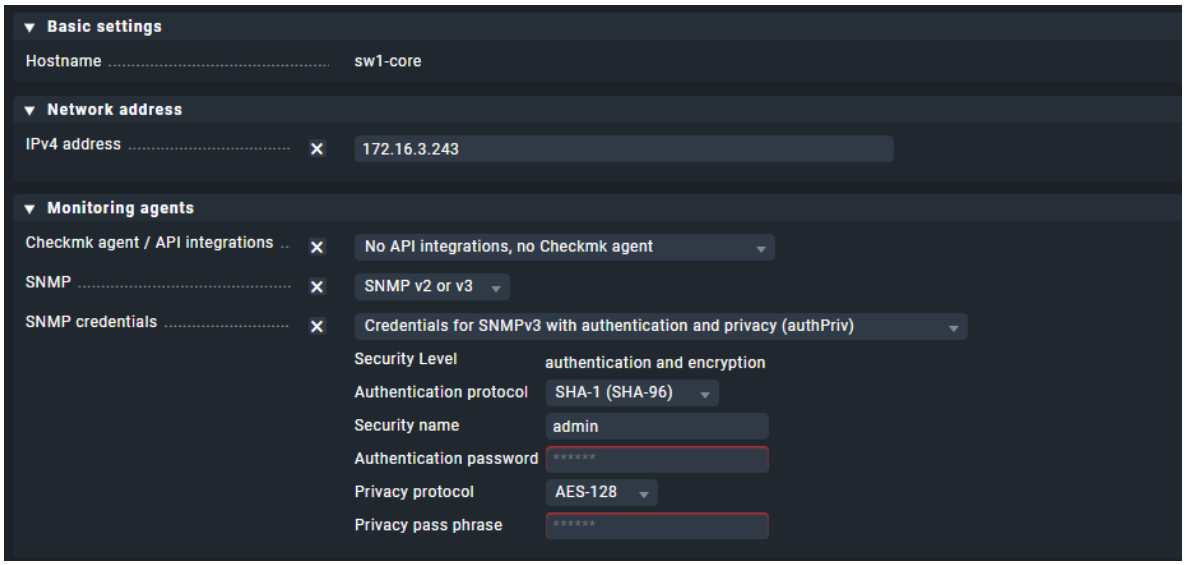
```
---
- name: Configure SNMPv3 on Cisco IOS Switches
  hosts: sw1-core
  gather_facts: no
  vars:
    snmp_auth_pass: pass1234
    snmp_priv_pass: pass5678
    snmp_group: mygroup
    snmp_user: admin

  tasks:
    - name: Create a group and configure SNMPv3
      ios_config:
        lines:
          - "snmp-server group {{ snmp_group }} v3 priv"
          - "snmp-server user {{ snmp_user }} {{ snmp_group }} v3 auth sha {{ snmp_auth_pass }} priv aes 128 {{ snmp_priv_pass }}"
~
```

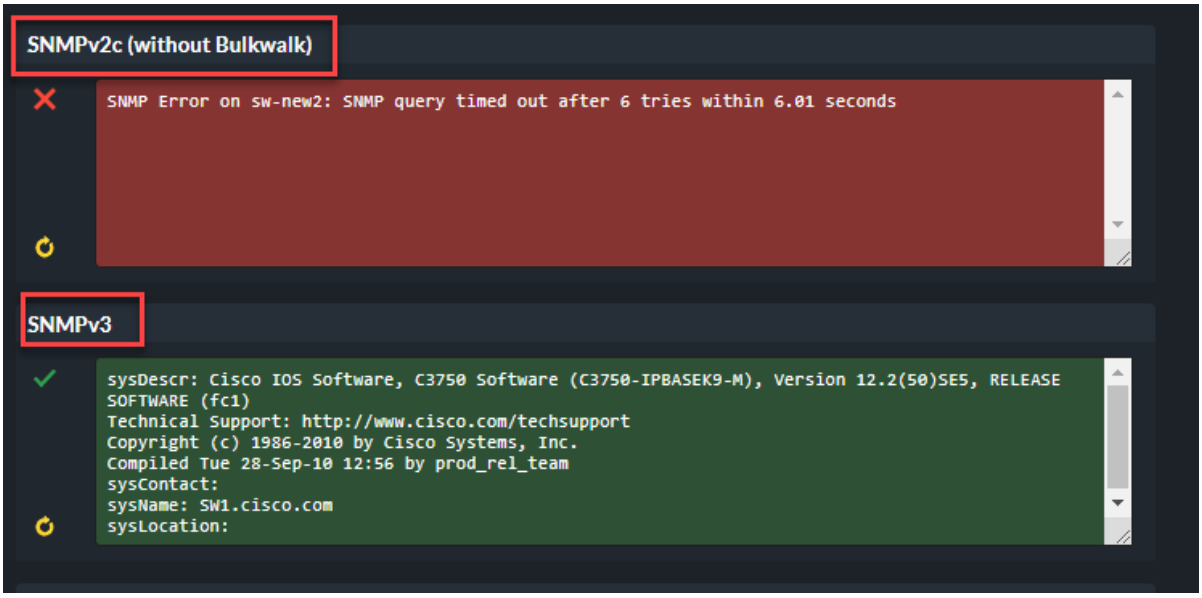
```
ansible@master:~/cisco_ios_project$ ansible-playbook playbooks/snmp_config.yml

PLAY [Configure SNMPv3 on Cisco IOS Switches] ***********************************************************************

TASK [Create a group and configure SNMPv3] **************************************************************************
[WARNING]: To ensure idempotency and correct diff the input configuration lines should be similar to how they appear if pres
changed: [sw1-core]

PLAY RECAP **********************************************************************************************************
sw1-core                   : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

ansible@master:~/cisco_ios_project$
```

We can check the applied configuration on the cisco device with *show snmp group* and *show run | include snmp* commands:

```
sw1-core(config)#do sh snmp group
groupname: mygroup                        security model:v3 priv
readview : v1default                      writeview: <no writeview specified>
notifyview: <no notifyview specified>
row status: active

sw1-core(config)#do sh run | inclu snmp
snmp-server group mygroup v3 priv
sw1-core(config)#
```

Here is the result for adding the switch to be monitored in CheckMK:





Creating VLAN with playbook

We can use this playbook to create the vlans:

```yaml
---
- name: Create VLAN if do'nt exist on a Cisco switch
  hosts: sw1-core
  gather_facts: no
  connection: local

  vars:
    vlan_id: 10
    vlan_name: admin-sys
    interface: GigabitEthernet1/0/1

  tasks:
    - name: T1 - Collect facts about VLAN "{{ vlan_id }}" on device
      ios_command:
        commands: show vlan id "{{ vlan_id }}" | include enet
      register: sh_vlan_output

    - name: T2 - show if the vlan exist
      debug:
        var: sh_vlan_output

    - name: T3 - Check if VLAN is defined and does not do anything if existing
      set_fact:
        vlan_exists: false
      when: sh_vlan_output.stdout_lines is defined

    - name: T4 - Check if VLAN is undefined and if not then go for below task to Add VLAN
      set_fact:
        vlan_exists: true
      when: sh_vlan_output.stdout_lines is undefined

    - name: T5 - Add VLAN using aggregate
      ios_vlan:
        aggregate:
        - {vlan_id: "{{ vlan_id }}", name: "{{ vlan_name }}", interfaces: "{{ interface }}"}

    - name: T6 - Collect facts about VLAN "{{ vlan_id }}" on device
      ios_command:
        commands: show vlan id "{{ vlan_id }}" | include enet
      register: sh_vlan_output2

    - name: T7 - show if the vlan exist
      debug:
        var: sh_vlan_output2
```

We are using ios_vlan module from cisco.iso collection. Note that this module will be removed from cisco.ios in a release after 2022-06-01.

The playbook will assign GigabitEthernet1/0/1 interface to the vlan 10 named admin-sys that it will create with ios_vlan module.

Note that we have added two conditional tasks with ansible.builtin.set_fact module. If the first condition is met, ansible will continue executing the tasks and creates the vlan. But if the first condition is not met, ansible will not execute vlan creation task.

I added the tasks 2, 6 and 7 to show you vlan situation before and after vlan creation by ios_vlan module:

```
TASK [T2 - show if the vlan exist] ********************************************************************
task path: /home/ansible/cisco_ios_project/playbooks/create_vlan.yml:18
redirecting (type: connection) ansible.builtin.network_cli to ansible.netcommon.network_cli
redirecting (type: terminal) ansible.builtin.ios to cisco.ios.ios
redirecting (type: cliconf) ansible.builtin.ios to cisco.ios.ios
redirecting (type: become) ansible.builtin.enable to ansible.netcommon.enable
ok: [sw1-core] => {
    "sh_vlan_output": {
        "changed": false,
        "failed": false,
        "stdout": [
            ""
        ],
        "stdout_lines": [
            [
                ""
            ]
        ]
    }
}
```

```
TASK [T3 - Check if VLAN is defined and does not do anything if existing] ****************************
task path: /home/ansible/cisco_ios_project/playbooks/create_vlan.yml:22
redirecting (type: connection) ansible.builtin.network_cli to ansible.netcommon.network_cli
redirecting (type: terminal) ansible.builtin.ios to cisco.ios.ios
redirecting (type: cliconf) ansible.builtin.ios to cisco.ios.ios
redirecting (type: become) ansible.builtin.enable to ansible.netcommon.enable
ok: [sw1-core] => {"ansible_facts": {"vlan_exists": false}, "changed": false}

TASK [T4 - Check if VLAN is undefined and if not then go for below task to Add VLAN] *****************
task path: /home/ansible/cisco_ios_project/playbooks/create_vlan.yml:27
skipping: [sw1-core] => {"changed": false, "skip_reason": "Conditional result was False"}

TASK [T5 - Add VLAN using aggregate] ******************************************************************
task path: /home/ansible/cisco_ios_project/playbooks/create_vlan.yml:32
redirecting (type: connection) ansible.builtin.network_cli to ansible.netcommon.network_cli
redirecting (type: terminal) ansible.builtin.ios to cisco.ios.ios
redirecting (type: cliconf) ansible.builtin.ios to cisco.ios.ios
redirecting (type: become) ansible.builtin.enable to ansible.netcommon.enable
redirecting (type: modules) ansible.builtin.ios_vlan to cisco.ios.ios_vlan
redirecting (type: action) ansible.builtin.ios to cisco.ios.ios
redirecting (type: action) ansible.builtin.ios to cisco.ios.ios
redirecting (type: modules) ansible.builtin.ios_vlan to cisco.ios.ios_vlan
changed: [sw1-core] => {"changed": true, "commands": ["vlan 10", "name admin-sys", "interface GigabitE
thernet1/0/1", "switchport mode access", "switchport access vlan 10"]}
```

```
TASK [T7 - show if the vlan exist] ********************************************************************
task path: /home/ansible/cisco_ios_project/playbooks/create_vlan.yml:42
redirecting (type: connection) ansible.builtin.network_cli to ansible.netcommon.network_cli
redirecting (type: terminal) ansible.builtin.ios to cisco.ios.ios
redirecting (type: cliconf) ansible.builtin.ios to cisco.ios.ios
redirecting (type: become) ansible.builtin.enable to ansible.netcommon.enable
ok: [sw1-core] => {
    "sh_vlan_output2": {
        "changed": false,
        "failed": false,
        "stdout": [
            "10   enet  100010     1500  -      -      -        -    -        0      0"
        ],
        "stdout_lines": [
            [
                "10   enet  100010     1500  -      -      -        -    -        0      0"
            ]
        ]
    }
}
META: ran handlers
META: ran handlers

PLAY RECAP ********************************************************************************************
sw1-core                   : ok=6    changed=1    unreachable=0    failed=0    skipped=1    rescued=0
   ignored=0
```

```
VLAN Name                              Status    Ports
---- --------------------------------- --------- -------------------------------
1    default                           active    Gi1/0/2, Gi1/0/3, Gi1/0/4, Gi1/0/5, Gi1/0/6, Gi1/0/7,
                                                 Gi1/0/19, Gi1/0/20, Gi1/0/21, Gi1/0/22, Gi1/0/23, Gi1
10   admin-sys                         active    Gi1/0/1
1002 fddi-default                      act/unsup
1003 token-ring-default                act/unsup
1004 fddinet-default                   act/unsup
1005 trnet-default                     act/unsup
```

In the second run, we will see that the ios_vlan module didn't execute any commands:

```
TASK [T3 - Check if VLAN is defined and does not do anything if existing] *****************************
task path: /home/ansible/cisco_ios_project/playbooks/create_vlan.yml:22
redirecting (type: connection) ansible.builtin.network_cli to ansible.netcommon.network_cli
redirecting (type: terminal) ansible.builtin.ios to cisco.ios.ios
redirecting (type: cliconf) ansible.builtin.ios to cisco.ios.ios
redirecting (type: become) ansible.builtin.enable to ansible.netcommon.enable
ok: [sw1-core] => {"ansible_facts": {"vlan_exists": false}, "changed": false}

TASK [T4 - Check if VLAN is undefined and if not then go for below task to Add VLAN] ****************
task path: /home/ansible/cisco_ios_project/playbooks/create_vlan.yml:27
skipping: [sw1-core] => {"changed": false, "skip_reason": "Conditional result was False"}

TASK [T5 - Add VLAN using aggregate] ******************************************************************
task path: /home/ansible/cisco_ios_project/playbooks/create_vlan.yml:32
redirecting (type: connection) ansible.builtin.network_cli to ansible.netcommon.network_cli
redirecting (type: terminal) ansible.builtin.ios to cisco.ios.ios
redirecting (type: cliconf) ansible.builtin.ios to cisco.ios.ios
redirecting (type: become) ansible.builtin.enable to ansible.netcommon.enable
redirecting (type: modules) ansible.builtin.ios_vlan to cisco.ios.ios_vlan
redirecting (type: action) ansible.builtin.ios to cisco.ios.ios
redirecting (type: action) ansible.builtin.ios to cisco.ios.ios
redirecting (type: modules) ansible.builtin.ios_vlan to cisco.ios.ios_vlan
ok: [sw1-core] => {"changed": false, "commands": []}
```

All codes in this article are available in my github at https://github.com/ershad-ra/ansible_ios_automation

END