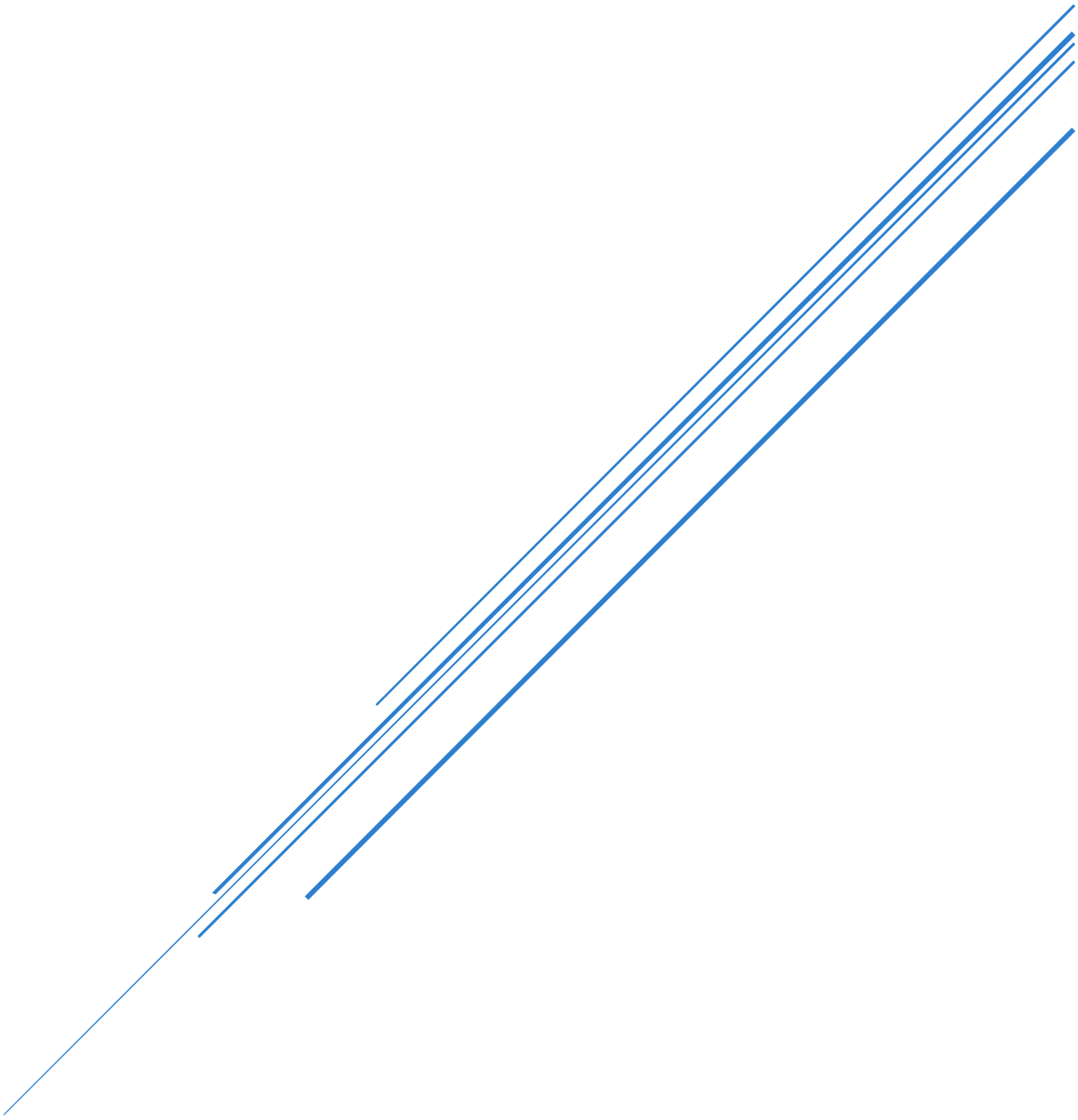
AUTOMATING NETWORK MANAGEMENT: INTEGRATING ANSIBLE WITH OCNOS

V 1.0



IP Infusion OCNOS

Table of Contents

1. Introduction: 2

2. Ansible Requirements: 3

1. Ansible Connection: 3

2. Ansible Network OS: 3

3. Privilege Escalation: 3

4. Ansible OcNOS Modules: 3

3. **Building the network inventory:** 4

4. Customizing Ansible's settings 5

1. How it works... 5

5. Variables: 5

1. Defining Variables: 5

2. Host Variables and Group Variables: 6

3. Important: 6

6. Building Ansible's playbook 7

1. How it works... 8

7. Ansible's conditionals: 10

1. How it works... 10

8. Ansible's loops: 12

1. How it works... 12

9. Jinja2 with Ansible: 14

1. How it works... 15

10. Ansible's filters: 16

1. How it works... 16

11. Deploy BGP on OCNOS routers using Ansible: 18

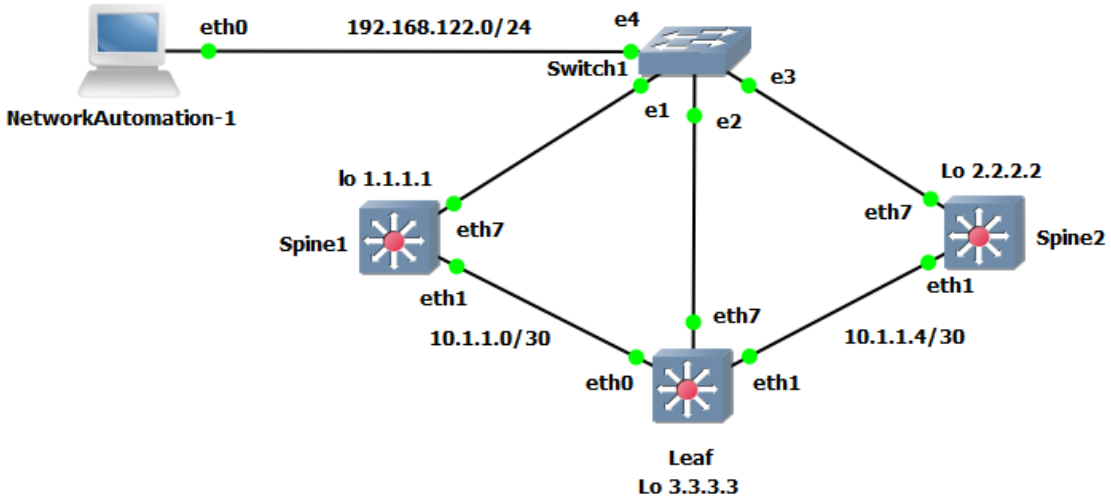
1- Create Playbook 19

2. How it works... 20

Reference: 21

1. Introduction:

We will outline how to automate **OCNOS** devices. We will explore how to interact with OCNOS devices using Ansible, and how to provision different services and protocols on OCNOS devices using various Ansible modules. We will base our illustration on the following sample network diagram of a network:



The following table outlines the devices in our sample topology and their respective management **Internet Protocols** (**IPs**):

Device	MGMT Port	MGMT IP
Spine 1	eth7	192.168.122.145
Spine 2	eth7	192.168.122.146
Leaf 1	eth7	192.168.122.12

2. Ansible Requirements:

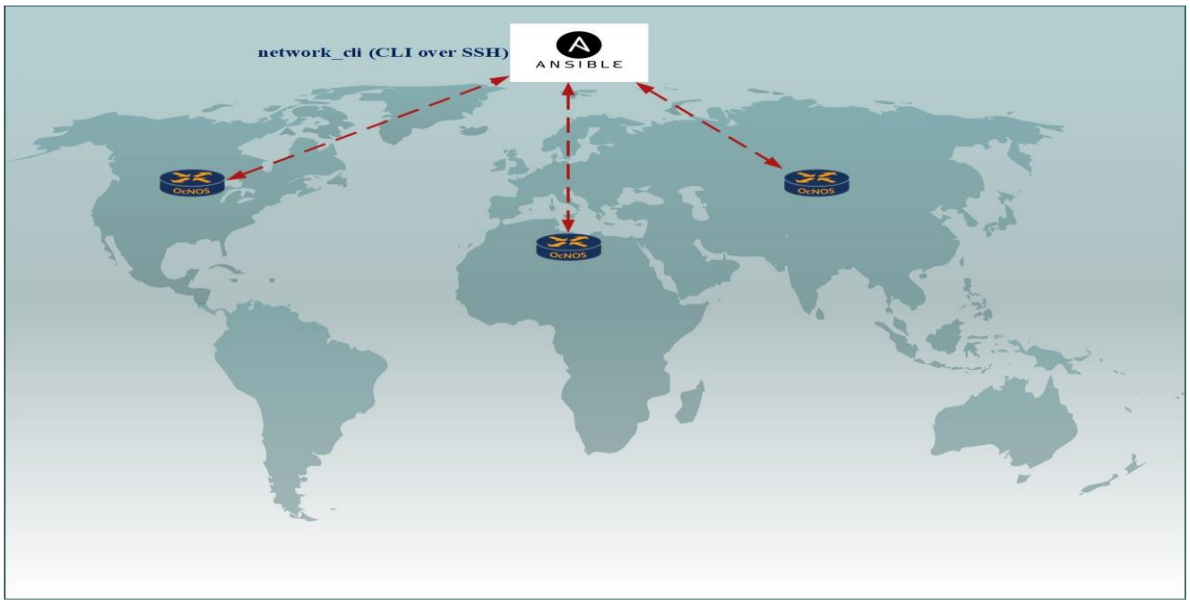
We should install OCNOS module to control or attractive with OCNOS Routers and setup up the following parameters as describe below and show in picture.

URL: https://github.com/IPInfusion/ocnos-ansible-collection/tree/master/ipinfusion/ocnos

1. Ansible Connection:

Type of connection between control Node and Routers is network_cli (CLI over SSH)

```
ansible  connection = network  cli
```



2. Ansible Network OS:

A network connection pluggin would be created by the name ‘ocnos.py’
To set the OS to ocnos:

```
ansible_network_os = ocnos
```

3. Privilege Escalation:

Ansible provides the option of enable, become and authorize. For OcNOS implementation, choose the ‘become’ mode. This is a top-level ansible parameter and allows escalated privileges in any network platform. It will be equivalent to the ‘enable’ mode inside OcNOS which is used to apply configurations on higher privilege level.

4. Ansible OcNOS Modules:

Modules are the smallest unit of execution in Ansible. Modules are written in Python. Modules can be directly used by a ansible playbook. Also, it can be organized into roles, where roles in turn can have tasks which executes certain modules.
Playbook → roles → Tasks → modules.

3. Building the network inventory:

we will outline how to build and structure the Ansible inventory to describe the sample network setup outlined previously. The Ansible inventory is a pivotal part in Ansible, as it defines and groups devices that should be managed by Ansible.

- we create a **hosts** file with the following content:

```
[spine]
spine1   ansible_host=192.168.122.145
spine2   ansible_host=192.168.122.146
[leaf]
leaf1    ansible_host=192.168.122.12
[core]
spine[1:2]
[access]
leaf1
```

1- This section defines a group named "**spine** "and **"leaf"**
2- This line defines a host named like "spine1" within the "spine" group. It also specifies the device's IP address.
3- To check list of host or devices
$ ansible {name_group} --list-hosts

4. Customizing Ansible's settings

Ansible has many settings that can be adjusted and controlled using a configuration file called **ansible.cfg**. This file has multiple options that control many aspects of Ansible, including how Ansible looks and how it connects to managed devices.

- Create an **Ansible.cfg** file, as shown in the following code:

```
[defaults]
inventory=./hosts
retry_files_enabled=False
ansible_connection=network_cli
gathering=explicit
host_key_checking = False
```

- This section first defines a inventory director, Ansible connection The‘Control Node’ can connect to ‘Managed Node’ via multiple modes
  A- network_cli (CLI over SSH)
  B- Netconf
  C- Httpapi
  D- Local

1. How it works...

We build the Ansible inventory using the hosts file and we define multiple groups in order to group the different devices in our network infrastructure.

Finally, we create the Ansible.cfg file and configure it to point to our hosts file, to be used as the Ansible inventory file. We set the gathering to explicit in order to disable the setup module, which runs by default to discover facts for the managed hosts

5. Variables:

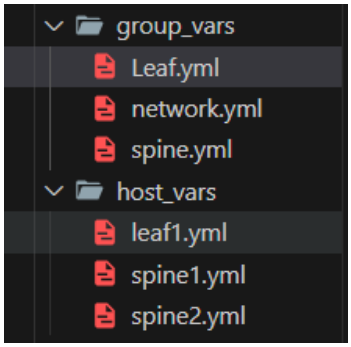
Ansible supports variables that can be used to store values that can be reused throughout files in an entire Ansible project. This can help simplify creation and maintenance of a project and reduce the incidence of errors. Variables provide a convenient way to manage dynamic values for a given environment in your Ansible project.

1. Defining Variables:

Variables can be defined in a bewildering variety of places in an Ansible project. However, this can be simplified to three basic scope levels:
• **Global scope:** Variables set from the command line or Ansible configuration
• **Play scope:** Variables set in the play and related structures
• **Host scope:** Variables set on host groups and individual hosts by the inventory, fact gathering, or registered tasks.

2. Host Variables and Group Variables:

Inventory variables that apply directly to hosts fall into two broad categories: **host variables** that apply to a specific host, and **group variables** that apply to all hosts in a host group or in a group of host groups. Host variables take precedence over group variables, but variables defined by a playbook take precedence over both.



3. Important:

The recommended practice is to define inventory variables using **host_vars** and **group_vars** directories, and not to define them directly in the inventory file or files.

- Example of group_var file :

```
---
ansible_network_os: "ipinfusion.ocnos.ocnos"
ansible_user: "ocnos"
ansible_password: "ocnos"
ansible_connection: "network_cli"
ansible_become: "yes"
ansible_become_method: "enable"
```

- Ansible uses existing privilege escalation systems to execute tasks with root privileges or with another user's permissions and method which privilege escalation method should be used
- **Important**
- We need to install the OcNOS Ansible module from Ansible Galaxy by using the command:

ansible-galaxy collection install ipinfusion.ocnos

- Example of hosts_var file :

```
---
hostname: leaf1
profile: leaf
loopback: 3.3.3.3
local_ASN: 65412
links:
 - port: eth0
   neighbor_switch: spine1
   port_local_ip: 10.1.1.2
   neighbor_port_ip: 10.1.1.1
   neighbor_ASN: 65501
 - port: eth1
   neighbor_switch: spine2
   port_local_ip: 10.1.1.6
   neighbor_port_ip: 10.1.1.5
   neighbor_ASN: 65502
```

We define variables of host in our example we define loopback , AS , interfaces and name interface

It will be helpful for generate configuration ….

6. Building Ansible's playbook

An Ansible playbook is the fundamental element in Ansible that declares what actions we would like to perform on our managed hosts (specified in the inventory). An Ansible playbook is a YAML-formatted file that defines a list of tasks that will be executed on our managed devices, we will outline how to write an Ansible playbook and how to define the hosts that will be targeted by this playbook

```
- name: Playbook name   # Descriptive name for the playbook
  hosts: all          # Target host(s) or group(s) - can be specific or use variables
  become: true        # Optional, run tasks with elevated privileges (e.g., root)
  tasks:              # List of tasks to be executed on the target hosts
    - name: Task name     # Descriptive name for the task
      module_name:        # Name of the Ansible module to be used
        arguments: ...    # Arguments specific to the chosen module (key-value pairs)
```

Module name: module is a program that performs actions on a local machine, or remote host. Modules are expressed as code, usually in Python, and contain metadata that defines when and where a specific automation task is executed, and which users can execute it.

When you download module of Ocnos from galaxy the following modules are installed

- Example of playbook:

```
---

- name: "PLAY 1: Capture and store config"  ####### name of playbook
  hosts: all              ########target host
  connection: network_cli  ###### type of connection
  gather_facts: no ###### by default ansible gathering facts of target hosts

  tasks: ###### Tasks
    - name: Display Hostname ##### name of task
      ipinfusion.ocnos.ocnos_command: ###### using module ocnos_command to execute
command show hostname
        commands: "show hostname"
      register: hostname_result ##### registered variables in any later tasks in your playbook
    - name: Display Hostname debug
      debug: ###### Prints or display  statements during execution it is helpful for troubleshooting
        msg: "Router name is {{ hostname_result.stdout }}"

    - name: Display OS
      ipinfusion.ocnos.ocnos_command:
        commands: "show version"
      register: os_result

    - name: Display OS debug
      debug:
        msg: "{{ hostname_result.stdout }} is running {{ os_result.stdout }}"

    - name: "TASK 1: Get config from routers"
      ipinfusion.ocnos.ocnos_command:
        commands: "show running-config"
      register: cli_result

    - name: "TASK 2: Print Config output"
      debug:
        msg: "{{ cli_result.stdout }}"

    - name: "TASK 3: Create run_config/ folder"
      file: ######This module in Ansible is used to manage files and directories.
        path: "run_config"
        state: directory
      run_once: true #########This directive tells Ansible to execute this task only once, even if it
is running the playbook against multiple hosts.

    - name: "TASK 4: Write run_config to file"
      copy:
        content: "{{ cli_result.stdout }}"
        dest: "run_config/{{ inventory_hostname }}.txt"
```

The Ansible playbook is structured as a list of plays and each play targets a specific group of hosts (defined in the inventory file). Each play can have one or more tasks to execute against the hosts in this play. Each task runs a specific Ansible module that has a number of arguments.we reference the variables that we defined in the previous recipe inside the {{ }} brackets. Ansible reads these variables from either group_vars or host_vars, and the module that we used in this playbook is the debug module, which displays as a custom message specified in the msg parameter to the Terminal output , the ocnos_command module to execute command on Ocnos Devices .

The execute of playbook is shown here:

```
ansible-playbook -i hosts playbook.yml
PLAY [PLAY 1: Capture and store config]
TASK [Display Hostname] ********************************************************************************************************
ok: [R2]
omitted…
TASK [Display Hostname debug] **************************************************************************************************
ok: [R2] => {
    "msg": "Router name is ['spine1']"
}
Omitted …
TASK [Display OS] **************************************************************************************************************
ok: [R2]
TASK [Display OS debug] ********************************************************************************************************
ok: [R2] => {
    "msg": "['spine1'] is running ['Software version: DEMO_VM-OcNOS-SP-PLUS-x86-6.5.2-SANDBOX 06/10/2024 09:57:58\\n Copyright (C) 2024 IP Infusion. All rights reserved\\n\\n Software Product: OcNOS-SP, Version: 6.5.2\\n Build Number: 50\\n Release: SANDBOX\\n Hardware Model: \\n Software Feature Code: PLUS-x86\\n Software Baseline Version: 6.5.0-164']"
}
Omitted …
TASK [TASK 1: Get config from routers]
*********************************************************************************************************************
ok: [R2]
TASK [TASK 2: Print Config output]
*********************************************************************************************************************************
ok: [R2] => {
    "msg": [
        "!\n! Software version: DEMO_VM-OcNOS-SP-PLUS-x86-6.5.2.50-SANDBOX 06/10/2024 09:57:58\n!\n! Last configuration change at 10:23:22 UTC Wed Jul 10 2024 by root\n!\nfeature netconf-ssh \nfeature netconf-tls \nno service password-encryption\n!\nsnmp-server enable traps link linkDown\nsnmp-server enable traps link linkUp\n!\nip vrf management\n!\nkey chain MykeyISIS\n key-id 1\n  key-string encrypted 0x3bfbfa9a6cc900c7\n!\nqos enable\n!\nhostname spine1\nip domain-lookup\nerrdisable cause stp-bpdu-guard\nfeature telnet\nfeature ssh\ 2\n dynamic-hostname\n net 49.0001.0100.1000.0001.00\n passive-interface lo\n!\n!\nend" omitted ..
    ]
}
TASK [TASK 3: Create run_config/ folder]
*********************************************************************************************************************************
ok: [R2]
TASK [TASK 4: Write run_config to file]
*********************************************************************************************************************************
ok: [R2]
PLAY RECAP
*********************************************************************************************************************************
R2                         : ok=8    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

7. Ansible's conditionals:

One of the core features of Ansible is conditional task execution. This provides us with the ability to control which tasks to run on a given host based on a condition/test that we specify. We will outline how to configure conditional task execution.

1. Create a new playbook called ansible_cond.yml.
2. Place the following content in the new playbook as shown here:

```
---
- name: Using conditionals
  hosts: all
  gather_facts: no
  tasks:
    - name: Run for leaf nodes Only
      ipinfusion.ocnos.ocnos_command:
        commands: "show hostname"
      register: hostname_result
      when: "'access' in group_names"

    - name: Run Hostname for leaf nodes
      debug:
        msg: "Router name is {{ hostname_result.stdout }}"
      when: "'access' in group_names"

    - name: Run for Only spine node
      ipinfusion.ocnos.ocnos_command:
        commands: "show version"
      register: os_result
      when: inventory_hostname == 'spine1'
```

1. How it works...

Ansible uses **when** statement to provide conditional execution for the tasks. **When** a statement is applied at the task level and if the condition in the **when** statement evaluates to **true**, the task is executed for the given host. If **false**, the task is skipped for this host. The output as a result of running the preceding playbook is shown here:

```
root@NetworkAutomation-1:/home/OCNOS_ansible# ansible-playbook -i hosts ansible_cond.yml

PLAY [Using conditionals] ******************************************************************************
TASK [Run for leaf nodes Only]
*********************************************************************************************
skipping: [R2]
skipping: [R3]
ok: [R1]
TASK [Run Hostname for leaf nodes]
***************************************************************************************************
skipping: [R2]
skipping: [R3]
ok: [R1] => {
    "msg": "Router name is ['leaf1']"
}
TASK [Run for Only spine node]
*********************************************************************************************************
**********
skipping: [R3]
skipping: [R1]
ok: [R2]

TASK [Run OS for spine node]
*********************************************************************************************************
**********
skipping: [R3]
skipping: [R1]
ok: [R2] => {
    "msg": "R2 is running ['Software version: DEMO_VM-OcNOS-SP-PLUS-x86-6.5.2-SANDBOX 06/10/2024 09:57:58\\n
Copyright (C) 2024 IP Infusion. All rights reserved\\n\\n Software Product: OcNOS-SP, Version: 6.5.2\\n Build Number: 50\\n
Release: SANDBOX\\n Hardware Model: \\n Software Feature Code: PLUS-x86\\n Software Baseline Version: 6.5.0-164']"
}

PLAY RECAP
*********************************************************************************************************
**********
R1                 : ok=2    changed=0    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
R2                 : ok=2    changed=0    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
R3                 : ok=0    changed=0    unreachable=0    failed=0    skipped=4    rescued=0    ignored=0
```

when statement can take a single condition as seen in the first task, or can take a list of conditions as seen in the second task. If when is a list of conditions, all the conditions need to be true for the task to be executed.

8. Ansible's loops:

In some cases, we need to run a task inside an Ansible playbook to loop over some data. Ansible's loops allow us to loop over a variable (a dictionary or a list) multiple times to achieve this behavior. We will outline how to use Ansible's loops

1- Create a new playbook called ansible_loops.yml
2- Inside the group_vars/spine.yml file, incorporate the following content:

users:
admin: admin123
oper: oper123

3- Inside the ansible_loops.yml file, incorporate the following content:

```
---
- name: Ansible Loop over a Dictionary
  hosts: spine
  gather_facts: no
  tasks:
  - name: Loop over Username and Passowrds
    ipinfusion.ocnos.ocnos_command:
    debug:
      msg: "Router {{ hostname }} with user {{ item.key }} password {{ item.value }}"
    with_dict: "{{ users}}"
```

1. How it works...

Ansible supports looping over two main iterable data structures: lists and dictionaries. We use the loops keyword when we need to iterate over lists, and we use **with_dicts** when we loop over a dictionary (users is a dictionary data structure where the username is the key and the passwords are the values). In both cases, we use the item keyword to specify the value in the current iteration. In the case of **with_dicts**, we get the key using **item.key** and we get the value using **item.value**.
The output of the preceding playbook run is shown here:

```
root@NetworkAutomation-1:/home/OCNOS_ansible# ansible-playbook ansible_loops.yml -i hosts

PLAY [Ansible Loop over a Dictionary]
**********************************************************************************************

TASK [Loop over Username and Passowrds]
**********************************************************************************************
ok: [R3] => (item={'key': 'admin', 'value': 'admin123'}) => {
    "msg": "Router R3 with user admin password admin123"
}
ok: [R2] => (item={'key': 'admin', 'value': 'admin123'}) => {
    "msg": "Router R2 with user admin password admin123"
}
ok: [R3] => (item={'key': 'oper', 'value': 'oper123'}) => {
    "msg": "Router R3 with user oper password oper123"
}
ok: [R2] => (item={'key': 'oper', 'value': 'oper123'}) => {
    "msg": "Router R2 with user oper password oper123"
}

PLAY RECAP
**********************************************************************************************
R2                 : ok=1    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
R3                 : ok=1    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

9. Jinja2 with Ansible:

Jinja2 is a powerful templating engine for Python and is supported by Ansible. It is also used to generate any text-based files, such as HTML, CSV, or YAML. We can utilize Jinja2 with Ansible variables to generate custom configuration files for network devices, we will outline how to use Jinja2 templates with Ansible

1- Create a new file inside the group_vars directory called network.yml:

```
ntp_servers:
- 172.20.1.1
- 172.20.2.1
```

2. Create a new templates directory and create a new **Ocnos_basic.j2** file with the following content:

```
hostname {{ hostname }}
!
feature ntp vrf management
ntp enable vrf management
ntp master vrf management
ntp master stratum 1 vrf management
{% for server in ntp_servers %}
ntp allow {{ server }} vrf management
{% endfor %}
!
```

In Jinja, variables are written in double curly braces.

When playbook is executed, these variables are replaced with desired values.

3. Create a new playbook called ansible_jinja2.yml with the following content:

```
---
- name: Generate OCNOS config from Jinja2
  hosts: all
  gather_facts: no
  tasks:
  - name: Create Configs Directory
    file: path=configs  state=directory

- name: Generate OCNOS config from Jinja2
  hosts: leaf
  gather_facts: no
  tasks:
  - name: Generate OCNOS Basic Config
    template:
      src: "templates/Ocnos_basic.j2"
      dest: "configs/{{inventory_hostname}}.cfg"
```

1. How it works...

We created the **network.yml** file to group all the variables that will apply to all devices under this group. After that, we create Jinja2 files. Inside Jinja2 template, we reference the Ansible variables using {{}}. We also use the for loop construct, {% for server in ntp_servers %} , supported by the Jinja2 templating engine in order to loop over the ntp_servers variable (which is a list) to access each item within this list.

Ansible provides the template module that takes two parameters:
src: This references the Jinja2 template file.
dest: This specifies the output file that will be generated.
In our case, we use the {{inventory_hostname}} variable in order to make the output configuration file unique for each router in our inventory.
The first play in the playbook creates the configs directory to store the configuration files for the network devices. The second play runs the template module on OCNOS devices.

10. Ansible's filters:

Ansible's filters are mainly derived from Jinja2 filters, and all Ansible filters are used to transform and manipulate data (Ansible's variables). In addition to Jinja2 filters, Ansible implements its own filters to augment Jinja2 filters, while also allowing users to define their own custom filters. We will outline how to configure and use Ansible filters to manipulate our input data.

1- Create a new Ansible playbook called ansible_filters.yml, as shown here:

```
---
- name: Ansible Filters
  hosts: spine1
  vars:
    interfaces:
      - { port: eth1, prefix: 10.1.1.0/30 }
  tasks:
    - name: Generate Interface Config
      blockinfile:
        block: |
          hostname {{ hostname | upper }}
          {% for intf in interfaces %}
          !
          interface {{ intf.port }}
            ip address {{intf.prefix | ipv4(1) | ipv4('address') }} {{intf.prefix | ipv4('netmask') }}
          !
          {% endfor %}
        dest: "configs/spine1_interfaces.cfg"
        create: yes
```

1. How it works...

First of all, we are using the **blockinfile** module to create a new configuration file on the Ansible control machine. This module is very similar to the template module. However, we can write the Jinja2 expressions directly in the module in the block option. We define a new variable called interfaces using the vars parameter in the playbook. This variable is a list data structure where each item in the list is a dictionary data structure. This nested data structure specifies the IP prefix used on each interface.
In the Jinja2 expressions, we can see that we have used a number of filters as shown here:

- **{{ hostname | upper}}:** upper is a Jinja2 filter that transforms all the letters of the input string into uppercase. In this way, we pass the value of the hostname variable to this filter and the output will be the uppercase version of this value.
- **{{intf.prefix | ipv4(1) | ipv4('address') }}:** Here, we use the Ansible IP address filter twice. ipv4(1) takes an input IP prefix and outputs the first IP address in this prefix. We then use another IP address filter, ipv4('address'), in order to only get the IP address part

of an IP prefix. So in our case, we take 10.1.1.0/30 and we output 10.1.1.1 for the first interface.

- **{{intf.prefix | ipv4('netmask') }}:** Here, we use the Ansible IP address filter to get the netmask of the IP address prefix, so in our case, we get the /30 subnet and transform it to 255.255.255.252.

The output file for the router after this playbook run is shown here:

```
hostname SPINE1
!
interface eth1
ip address 10.1.1.1 255.255.255.252
!
```

11. Deploy BGP on OCNOS routers using Ansible:

1. Create a new templates directory and create a new BGP_config.j2 file

```
router bgp {{ local_ASN }}
 bgp router-id {{ loopback }}
{% for link in links %}
 neighbor {{ link.neighbor_port_ip }} remote-as {{ link.neighbor_ASN }}
{% endfor %}
address-family ipv4 unicast
{% for link in links %}
 neighbor {{ link.neighbor_port_ip }} activate
{% endfor %}
exit-address-family
commit
```

As we discussed in the previous section (Variable), we will create variables for leaf and spine devices and learned how to work with Jinja. Here, we will create a configuration template to deploy the BGP protocol on OcNOS devices.

```
---
hostname: leaf1
profile: leaf
loopback: 3.3.3.3
local_ASN: 65412
links:
  - port: eth0
    neighbor_switch: spine1
    port_local_ip: 10.1.1.2
    neighbor_port_ip: 10.1.1.1
    neighbor_ASN: 65501
  - port: eth1
    neighbor_switch: spine2
    port_local_ip: 10.1.1.6
    neighbor_port_ip: 10.1.1.5
    neighbor_ASN: 65502
```

Variable of Leaf1

```
---
hostname: spine1
profile: spine
loopback: 1.1.1.1
local_ASN: 65501
links:
  - port: eth0
    neighbor_switch: leaf1
    port_local_ip: 10.1.1.1
    neighbor_port_ip: 10.1.1.2
    neighbor_ASN: 65412
```

Variable of Spine1

1- Create Playbook

2- create BGP configs directory.
3- generate BGP config.
4- push config on Ocnos devices.

```
---
- name: Generate and push OCNOS config from Jinja2
  hosts: all
  gather_facts: no
  tasks:
    - name: Create BGP configs directory
      file:
        path: configs_BGP
        state: directory
    - name: Generate OCNOS BGP Config
      template:
        src: "templates/BGP_config.j2" ######Specifies the source template file located in the templates directory. This file contains the Jinja2 template, which will be processed to generate configuration files.
dest: "configs_BGP/{{ hostname }}.cfg" #### Specifies the destination path where the generated configuration file will be saved.
      loop_control:
        loop_var: all ####The variable all is used in a loop to apply the task multiple times
    - name: Push OCNOS Config to Device
      cli_config:##### The cli_config module is used to send configuration commands to the device.
        config: "{{ lookup('file', 'configs_BGP/{{ hostname }}.cfg') }}" ###This line uses the lookup('file', ...) function to read the contents of the generated configuration file (configs_BGP/{{ hostname }}.cfg) and then sends this content as the configuration to the device. The hostname variable is dynamically substituted, allowing this task to target different devices
      loop_control:
        loop_var: all
      notify: Commit Config ####This line triggers a handler called Commit Config after the configuration is pushed to the device. This ensures that the changes are committed
  handlers: ####The handler is used to commit the configuration changes on the device after the configuration has been successfully pushed
    - name: Commit Config
      ipinfusion.ocnos.ocnos_command:
        commands:
          - "commit"
```

2. How it works...

1. The playbook first creates a directory to store BGP configurations.
2. It then generates these configurations using a Jinja2 template.
3. The generated configurations are pushed to the OCNOS devices.
4. Finally, the configuration changes are committed to make them active.
5. To execute the playbook as shown here:

   $ ansible-playbook -i hosts [name_playbook]
   $ ansible-playbook -i hosts Config_BGP.yml

```
root@NetworkAutomation-1:/home/OCNOS_ansible# ansible-playbook -i hosts config_BGP.yml

PLAY [Generate and push OCNOS config from Jinja2]
**********************************************************************************

TASK [Create BGP configs directory]
**********************************************************************************
ok: [leaf1]
ok: [spine2]
ok: [spine1]

TASK [Generate OCNOS BGP Config]
**********************************************************************************
ok: [spine2]
ok: [leaf1]
ok: [spine1]

TASK [Push OCNOS Config to Device]
**********************************************************************************
changed: [spine2]
changed: [spine1]
changed: [leaf1]

PLAY RECAP
**********************************************************************************
leaf1                : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
spine1               : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
spine2               : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Reference:

- For more information regarding the installation of Ansible, please check the following URL: https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html
- For more information regarding Ansible's conditionals, please check the following URL: https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html
- For more information regarding the different Ansible *looping constructs*, please consult the following URL: https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html
- For more information regarding Jinja2, please check the following URL: https://pyneng.readthedocs.io/en/latest/book/20_jinja2/README.html