

Ansible Ad-Hoc command w/ RAW module Basic lab use case

In this lab activity, the focal point is on leveraging Ansible ad-hoc commands, with an emphasis on the RAW Module, to establish connections to devices that do not support or have Python enabled. Ansible, an open-source automation tool, is adept at managing network device configurations. Unlike executing playbooks, which entail a series of tasks, Ansible ad-hoc commands are standalone commands designed for executing quick, immediate tasks.

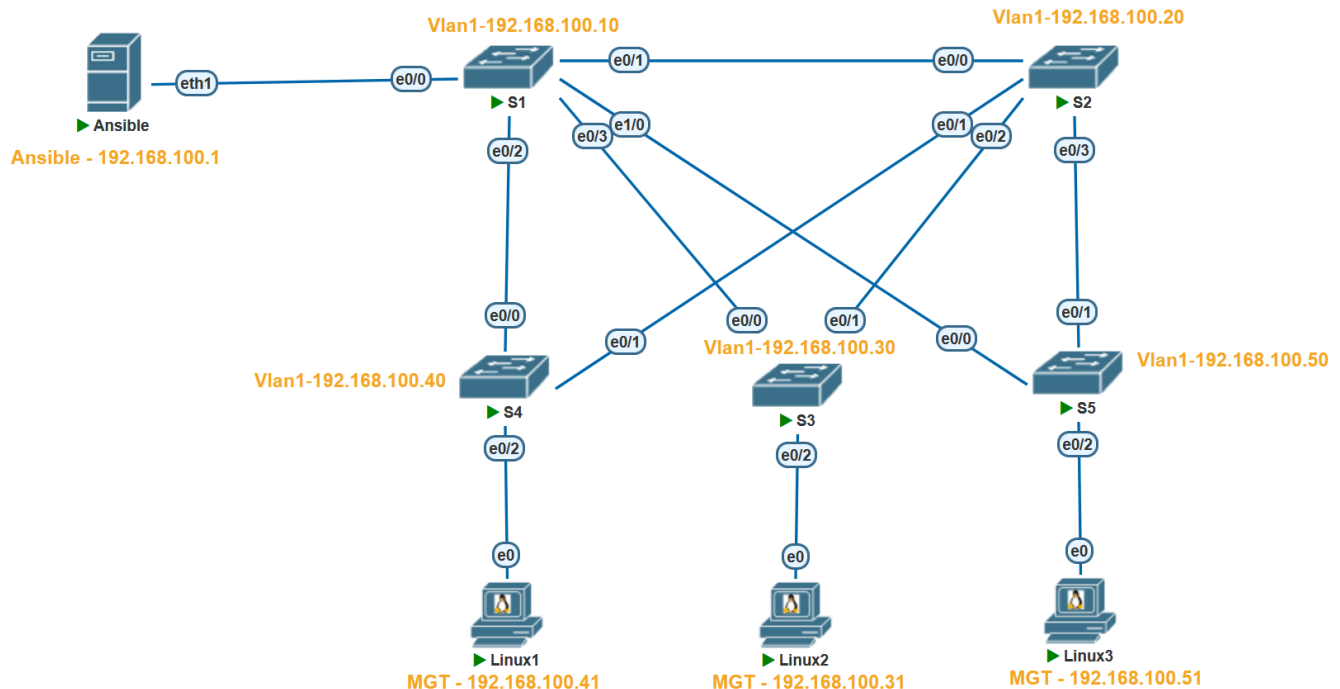
The RAW Module in Ansible is particularly useful in environments where devices do not support Python. This module allows for the execution of commands in their native form, without relying on the Ansible's module facility. Therefore, even in the absence of Python, the RAW Module facilitates seamless interactions with the devices.

See Ansible official documentation for more information:

https://docs.ansible.com/ansible/latest/command_guide/intro_adhoc.html

Through this lab, you will garner hands-on experience on the versatility of Ansible ad-hoc commands, especially when employing the RAW Module for devices devoid of Python support or Ansible playbooks.

We will use the following topology:



Here are some of the information on lab devices

Hostname	
Ansible	EVE-NG/GNS3 Network appliance @192.168.100.1
S1	Cisco Switch running ios_12 @192.168.100.10
S2	Cisco Switch running ios_12 @192.168.100.20
S3	Cisco Switch running ios_12 @192.168.100.30
S4	Cisco Switch running ios_12 @192.168.100.40
S5	Cisco Switch running ios_12 @192.168.100.50

Prerequisites:

- EVE-NG/GNS3
- Ansible appliance
- Cisco images for EVE-NG/GNS3

To execute Ansible ad-hoc commands, certain prerequisites are required:

1. **Ansible Installation on a Control Node:** This could be your personal computer or a dedicated server that serves as the communication bridge between Ansible and your network devices. In this lab, we utilize an EVE-NG/GNS3 provided automation appliance. The EVE-NG/GNS3 network container is equipped with widely utilized network automation tools such as Netmiko, NAPALM, Pyntc, and Ansible.
2. **Python:** Given that Ansible is developed in Python, it's imperative to have Python installed on the control node. However, as we are utilizing the Network appliance, this requirement is already satisfied.
3. **Network Modules:** Ansible employs network modules (e.g., ios_command for Cisco IOS devices) to foster interactions with network devices.

Proceeding to create a sample lab with Ansible ad-hoc commands targeting Cisco devices, the primary steps encompass:

1. **Ansible Installation on the Control Node:** This can be achieved via a package manager such as apt or yum, with the assumption that the control node setup is already in place.
2. **Host File Setup:** Ansible necessitates an inventory file to keep track of the hosts constituting your network.
3. **Router/Switch Connectivity Test:** Employing Ansible's ping module, you can ascertain the connectivity to the Cisco router.
4. **Command Execution on Router:** Utilize ad-hoc commands to execute commands on the router, particularly focusing on the raw module for devices without Ansible playbooks.
5. **Output Filtering:** Ansible facilitates output filtering to extract the desired information.
6. **Output Saving:** The ad-hoc commands also enable saving the output to a file for future reference or analysis.
7. **Ansible use case:** The ad-hoc commands enable troubleshooting and saving of output to a file for future reference or analysis.

Step 1: Configure Switch

First, we have to configure the S1 for connectivity and to allow ssh. Use the below configs for S1

```

no ip domain name lookup
no logging console
cdp run
host S1
int Vlan1
ip address 192.168.100.10 255.255.255.0
no shut
username user privilege 15 secret password
line con 0
login local
line aux 0
line vty 0 4
login local
transport input all
ip domain-name www.Test.home
crypto key generate rsa
1024

```

```

Switch>en
Switch#config t
Enter configuration commands, one per line. End with CNTL/Z.
Switch(config)#no ip domain name lookup
Switch(config)#no logging console
Switch(config)#cdp run
Switch(config)#host S1
S1(config)#int Vlan1
S1(config-if)# ip address 192.168.100.10 255.255.255.0
S1(config-if)# no shut
S1(config-if)#username user privilege 15 secret password
S1(config)#line con 0
S1(config-line)# login local
S1(config-line)#line aux 0
S1(config-line)#line vty 0 4
S1(config-line)# login local
S1(config-line)# transport input all
S1(config-line)# ip domain-name www.Test.home
S1(config)#crypto key generate rsa
The name for the keys will be: S1.www.Test.home
Choose the size of the key modulus in the range of 360 to 4096 for your
General Purpose Keys. Choosing a key modulus greater than 512 may take
a few minutes.

```

```

How many bits in the modulus [512]: 1024
% Generating 1024 bit RSA keys, keys will be non-exportable...
[OK] (elapsed time was 0 seconds)

```

```

S1(config)#end
S1#
S1#show vlan brief

```

VLAN	Name	Status	Ports
1	default	active	Et0/0, Et0/1, Et0/2, Et0/3
1002	fddi-default	act/unsup	
1003	token-ring-default	act/unsup	
1004	fddinet-default	act/unsup	
1005	trnet-default	act/unsup	

Let's go through each of the commands listed:

no ip domain name lookup: This command disables DNS lookup. Without this, any mistyped command in the console is interpreted as a hostname by the router, and it will attempt to resolve it via DNS, which can cause a delay.

no logging console: This command disables logging to the console. By default, the router sends all log messages to its console port. Therefore, disabling this can be helpful in not interrupting CLI access with log messages.

cdp run: This command enables the Cisco Discovery Protocol (CDP). CDP is a Cisco proprietary protocol used to discover Cisco devices in your network.

host S1: This command changes the hostname of the device to "S1".

int vlan 1: This command enters the configuration mode for VLAN 1.

ip address 192.168.100.10 255.255.255.0: This command assigns the IP address 192.168.100.10 with a subnet mask of 255.255.255.0 to VLAN 1.

no shut: This command brings up the VLAN interface if it's administratively down. It's equivalent to "enable this interface".

username user privilege 15 secret password: This command creates a user with the username "user", assigns it a privilege level of 15 (the highest level, equivalent to root or admin), and sets the password to "password". The keyword "secret" indicates that the password will be stored in a hashed format.

line con 0: This command enters line configuration mode for the console port.

login local: This command sets the login method to use the local user database for authentication. It's used here for the console and VTY lines, meaning that the username and password set earlier will be used for console and remote logins.

line vty 0 4: This command enters line configuration mode for the first 5 VTY lines (0-4). VTY lines are used for Telnet and SSH access to the device.

transport input all: This command is also used under vty configuration mode. It allows all types of protocols (telnet, SSH, etc.) for remote access. However, for security purposes, it is recommended to allow only SSH (i.e., "transport input ssh").

ip domain-name www.Test.home: This command sets the domain name of the device to www.Test.home. This is required for generating the RSA keys which are used by SSH for encryption and decryption.

crypto key generate rsa: This command initiates the process of generating RSA keys which are required for SSH. After entering this command, you will be prompted to enter the modulus size.

1024: This is the modulus size for the RSA keys. It represents the key length of 1024 bits. The larger the key size, the more secure the SSH connection, but at the cost of more processor overhead.

These commands together configure your Cisco IOS device for secure remote access, enabling you to manage your device without needing to be physically connected to it. It also disables some default settings like DNS lookup on mistyped commands and console logging.

Once configured Vlan1 should have a IP in place

```
S1#show run int vlan1
Building configuration...

Current configuration : 64 bytes
!
interface Vlan1
ip address 192.168.100.10 255.255.255.0
end
S1#
```

```
S2#show run int vlan1
Building configuration...

Current configuration : 64 bytes
!
interface Vlan1
ip address 192.168.100.20 255.255.255.0
end
S2#
```

```
S3#show run int vlan1
Building configuration...

Current configuration : 64 bytes
!
interface Vlan1
ip address 192.168.100.30 255.255.255.0
end
```

```
S4#show run int vlan1
Building configuration...

Current configuration : 64 bytes
!
interface Vlan1
ip address 192.168.100.40 255.255.255.0
end
```

```
S5#show run int vlan1
Building configuration...

Current configuration : 64 bytes
!
interface Vlan1
ip address 192.168.100.50 255.255.255.0
end
S5#
```

- Repeat this step for all of S2-5 assigning them different IP's and hostname according to the IP's listed above

Step 2: Configure Network connectivity to Ansible and verify connectivity.

-Next, we have to configure the Ansible control node and verify that we can reach our router/switch.-

1. To statically configure a network right click the docker container and set the static ip: **192.168.100.1/24** in our case

Note: If you downloaded ansible and installed it on ansible you make changes in /etc/network/interfaces to set the IP on a interface by opening the “/etc/network/interfaces” file using a text editor

The screenshot shows the configuration page for a Docker container named 'Ansible'. The 'Name' field is 'Ansible' and the 'Description' is 'Docker.io'. The 'Icon' is 'Server.png'. Under 'CPU (Core)', 'RAM (MB)', and 'Delay', the values are 1, 256, and 0 respectively. In the 'Ethernet' section, 'Eth1 DHCP' is unchecked, and 'Eth1 Static IP (x.x.x.x/y)' is set to '192.168.100.1/24'.

Save the changes

2. Restart/Start the network appliance and validate that your configs took by running the **ifconfig** command.

```
root@Ansible:/Docker-images# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.177.0.2 netmask 255.255.0.0 broadcast 10.177.255.255
    ether 02:42:0a:b1:00:02 txqueuelen 0 (Ethernet)
    RX packets 11 bytes 906 (906.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.100.1 netmask 255.255.255.0 broadcast 192.168.100.255
    ether 50:00:00:3c:00:01 txqueuelen 1000 (Ethernet)
    RX packets 35 bytes 2606 (2.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@Ansible:/Docker-images#
```

You should see the ip address you configured on the network adapter on the output.

3. Validate network connectivity by pinging all the devices you want to reach

The command **ping 192.168.100.10** is used to test the network connectivity from the ansible control node to SW1 with the IP address 192.168.100.10 and so forth for the other switches/destination

```
root@Ansible:/Docker-images# ping 192.168.100.10
PING 192.168.100.10 (192.168.100.10) 56(84) bytes of data.
64 bytes from 192.168.100.10: icmp_seq=1 ttl=255 time=0.327 ms
64 bytes from 192.168.100.10: icmp_seq=2 ttl=255 time=0.286 ms
64 bytes from 192.168.100.10: icmp_seq=3 ttl=255 time=0.482 ms
64 bytes from 192.168.100.10: icmp_seq=4 ttl=255 time=0.325 ms
64 bytes from 192.168.100.10: icmp_seq=5 ttl=255 time=0.356 ms
^C
--- 192.168.100.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4091ms
rtt min/avg/max/mdev = 0.286/0.355/0.482/0.067 ms
root@Ansible:/Docker-images# ping 192.168.100.20
PING 192.168.100.20 (192.168.100.20) 56(84) bytes of data.
64 bytes from 192.168.100.20: icmp_seq=1 ttl=255 time=0.878 ms
64 bytes from 192.168.100.20: icmp_seq=2 ttl=255 time=0.749 ms
64 bytes from 192.168.100.20: icmp_seq=3 ttl=255 time=0.692 ms
64 bytes from 192.168.100.20: icmp_seq=4 ttl=255 time=0.742 ms
64 bytes from 192.168.100.20: icmp_seq=5 ttl=255 time=0.838 ms
^C
--- 192.168.100.20 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4091ms
rtt min/avg/max/mdev = 0.692/0.779/0.878/0.068 ms
root@Ansible:/Docker-images# ping 192.168.100.30
PING 192.168.100.30 (192.168.100.30) 56(84) bytes of data.
64 bytes from 192.168.100.30: icmp_seq=1 ttl=255 time=0.818 ms
64 bytes from 192.168.100.30: icmp_seq=2 ttl=255 time=0.820 ms
64 bytes from 192.168.100.30: icmp_seq=3 ttl=255 time=0.609 ms
64 bytes from 192.168.100.30: icmp_seq=4 ttl=255 time=0.656 ms
64 bytes from 192.168.100.30: icmp_seq=5 ttl=255 time=0.756 ms
^C
--- 192.168.100.30 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4076ms
rtt min/avg/max/mdev = 0.609/0.731/0.820/0.085 ms
root@Ansible:/Docker-images# ping 192.168.100.40
PING 192.168.100.40 (192.168.100.40) 56(84) bytes of data.
64 bytes from 192.168.100.40: icmp_seq=1 ttl=255 time=0.870 ms
64 bytes from 192.168.100.40: icmp_seq=2 ttl=255 time=0.695 ms
64 bytes from 192.168.100.40: icmp_seq=3 ttl=255 time=0.440 ms
64 bytes from 192.168.100.40: icmp_seq=4 ttl=255 time=0.822 ms
64 bytes from 192.168.100.40: icmp_seq=5 ttl=255 time=0.668 ms
^C
--- 192.168.100.40 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4084ms
rtt min/avg/max/mdev = 0.440/0.699/0.870/0.149 ms
root@Ansible:/Docker-images# ping 192.168.100.50
PING 192.168.100.50 (192.168.100.50) 56(84) bytes of data.
64 bytes from 192.168.100.50: icmp_seq=1 ttl=255 time=0.844 ms
64 bytes from 192.168.100.50: icmp_seq=2 ttl=255 time=0.677 ms
64 bytes from 192.168.100.50: icmp_seq=3 ttl=255 time=0.793 ms
64 bytes from 192.168.100.50: icmp_seq=4 ttl=255 time=0.800 ms
64 bytes from 192.168.100.50: icmp_seq=5 ttl=255 time=0.790 ms
64 bytes from 192.168.100.50: icmp_seq=6 ttl=255 time=0.700 ms
^C
--- 192.168.100.50 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5106ms
rtt min/avg/max/mdev = 0.677/0.767/0.844/0.058 ms
root@Ansible:/Docker-images#
```

Step 3: Configure Ansible

1. Validate that you can ssh into S1

```
ssh user@192.168.100.10
```

```
root@Ansible:/Docker-images# ssh user@192.168.100.10
The authenticity of host '192.168.100.10 (192.168.100.10)' can't be established.
RSA key fingerprint is SHA256:XdUP5/MLFvTFbxyxqWumuimdrI4S88PygkUSb4e2uU.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.100.10' (RSA) to the list of known hosts.
Password:
S1#
S1#
S1#
```

Configure host resolution.

2. Add entries into the default host file for your devices using the below command: **nano /etc/hosts**

```
GNU nano 4.8 /etc/hosts
127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
192.168.100.10 S1
192.168.100.20 S2
192.168.100.30 S3
192.168.100.40 S4
192.168.100.50 S5
```

3. Add in entries for your devices.

```
192.168.100.10 S1
192.168.100.20 S2
192.168.100.30 S3
192.168.100.40 S4
```

1. 5

4. Validate that host resolution is working by pinging device host entire name.

```
root@Ansible:~# ping S1
PING S1 (192.168.100.10) 56(84) bytes of data:
64 bytes from S1 (192.168.100.10): icmp_seq=1 ttl=255 time=0.394 ms
64 bytes from S1 (192.168.100.10): icmp_seq=2 ttl=255 time=0.293 ms
64 bytes from S1 (192.168.100.10): icmp_seq=3 ttl=255 time=0.327 ms
^C
--- S1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 203ms
rtt min/avg/max/mdev = 0.293/0.338/0.394/0.041 ms
root@Ansible:~# ping S2
PING S2 (192.168.100.20) 56(84) bytes of data:
64 bytes from S2 (192.168.100.20): icmp_seq=1 ttl=255 time=0.950 ms
64 bytes from S2 (192.168.100.20): icmp_seq=2 ttl=255 time=0.552 ms
64 bytes from S2 (192.168.100.20): icmp_seq=3 ttl=255 time=0.500 ms
^C
--- S2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2010ms
rtt min/avg/max/mdev = 0.500/0.667/0.950/0.201 ms
root@Ansible:~# ping S3
PING S3 (192.168.100.30) 56(84) bytes of data:
64 bytes from S3 (192.168.100.30): icmp_seq=1 ttl=255 time=0.962 ms
64 bytes from S3 (192.168.100.30): icmp_seq=2 ttl=255 time=0.586 ms
64 bytes from S3 (192.168.100.30): icmp_seq=3 ttl=255 time=0.524 ms
^C
--- S3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2018ms
rtt min/avg/max/mdev = 0.524/0.690/0.962/0.193 ms
root@Ansible:~# ping S4
PING S4 (192.168.100.40) 56(84) bytes of data:
64 bytes from S4 (192.168.100.40): icmp_seq=1 ttl=255 time=0.775 ms
64 bytes from S4 (192.168.100.40): icmp_seq=2 ttl=255 time=0.589 ms
64 bytes from S4 (192.168.100.40): icmp_seq=3 ttl=255 time=0.582 ms
^C
--- S4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2051ms
rtt min/avg/max/mdev = 0.582/0.648/0.775/0.089 ms
root@Ansible:~# ping S5
PING S5 (192.168.100.50) 56(84) bytes of data:
64 bytes from S5 (192.168.100.50): icmp_seq=1 ttl=255 time=0.821 ms
64 bytes from S5 (192.168.100.50): icmp_seq=2 ttl=255 time=0.458 ms
64 bytes from S5 (192.168.100.50): icmp_seq=3 ttl=255 time=0.482 ms
^C
--- S5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2038ms
rtt min/avg/max/mdev = 0.458/0.587/0.821/0.165 ms
root@Ansible:~#
```


5. Create inventory file for ansible. This will be used to tell Ansible what hosts to use.

1. Navigate to the parent directory with `cd` and verify your path with `pwd`

Use commands below;

```
cd  
pwd
```

```
root@Ansible:/etc/ansible# cd  
root@Ansible:~# pwd  
/root  
root@Ansible:~#
```

6. Create new inventory file that will be used for hosts. An Ansible Inventory File is a cornerstone in Ansible's architecture, functioning as a manifest that defines the nodes or hosts upon which tasks and playbooks will be executed.

Use commands below;

```
nano hosts
```

1. Configure your hostname in the file with group names

```
[ios]  
S1  
S2  
S3  
S4  
S5
```

```
GNU nano 4.8 hosts  
[ios]  
S1  
S2  
S3  
S4  
S5
```

7. Configure ansible configuration file to use your newly created host file.

- Checking the ansible inventory you can see that although you configured hosts ansible is not able to see your host file

○ Use the below command to check ansible host file: **ansible --list-hosts all**

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

- Create new local ansible configuration file with below command: **nano ansible.cfg**

Use below configs for the ansible configuration file and then save the file.

```
[defaults]
inventory = ./hosts
host_key_checking = false
timeout = 5
```

```
GNU nano 4.8 ansible.cfg Modified
[defaults]
hostfile = ./hosts
host_key_checking = false
timeout = 5
```

Let's go through each of the commands listed:

- **[defaults]**: This is the primary default section which includes a variety of settings you can adjust.
- **hostfile = ./hosts** – This specifies the inventory file where Ansible will look to find the hosts that it can connect to. The “./hosts” suggests that the inventory file is named “hosts” and is located in the same directory as the “ansible.cfg” file.
- **host_key_checking = false**: By default, Ansible checks the SSH key of the remote hosts during the first connection. This configuration disables that check. This is often used in environments where host keys aren't yet known or can change, like in cloud or testing environments.
- **timeout = 5**: This controls the number of seconds Ansible will wait for connections to hosts to complete. This is not the time limit for the entire task, but for the initial connection attempt. The default value is usually 10, but here it's set to 5 seconds.

8. Verify ansible is using your created host file.

Checking the ansible host file you can see that ansible is now able to see the hosts in your host file due to using the local ansible configuration file.

Use the command below: **ansible --list-hosts all**

```
root@Ansible:~# ansible --list-hosts all
hosts (5):
  s1
  s2
  s3
  s4
  s5
root@Ansible:~#
```

Step 4: Use Ansible Ad hoc commands (single device)

1. Input the below command to retrieve the output of the S1 switch using ansible.

```
ansible S1 -m raw -a "show ver" -u user -k
```

```
root@Ansible:~# ansible S1 -m raw -a "show ver" -u user -k
SSH password:
S1 | CHANGED | rc=0 >>
Cisco IOS Software, Linux Software (I86BI_LINUXL2-IPBASEK9-M), Experimental Version 15.2(20170809:194209) [dstivers-aug9_2017-high_iron-cts 101]
Copyright (c) 1986-2017 by Cisco Systems, Inc.
Compiled Wed 09-Aug-17 13:49 by xxxxxxxx

ROM: Bootstrap program is Linux

S1 uptime is 1 hour, 18 minutes
System returned to ROM by reload at 0
System image file is "unix:/opt/unetlab/tmp/6/64/i86bi_linux_l2-ipbasek9-ms.high_iron_aug9_"
Last reload reason: Unknown reason

This product contains cryptographic features and is subject to United
States and local country laws governing import, export, transfer and
use. Delivery of Cisco cryptographic products does not imply
third-party authority to import, export, distribute or use encryption.
Importers, exporters, distributors and users are responsible for
compliance with U.S. and local country laws. By using this product you
agree to comply with applicable laws and regulations. If you are unable
to comply with U.S. and local laws, return this product immediately.

A summary of U.S. laws governing Cisco cryptographic products may be found at:
http://www.cisco.com/wwl/export/crypto/tool/stqrg.html

If you require further assistance please contact us by sending email to
export@cisco.com.

Linux Unix (Intel-x86) processor with 943604K bytes of memory.
Processor board ID 67239937
8 Ethernet interfaces
1 Virtual Ethernet interface
1024K bytes of NVRAM.

Configuration register is 0x0
Shared connection to s1 closed.

root@Ansible:~#
```

Let's go through the command:

ansible: This is the Ansible command line tool, used to run tasks and playbooks.

1. **S1:** This represents the target host or group as defined in the Ansible inventory. In this case, it refers to a host or a group named "S1".
2. **-m raw:** The **-m** option specifies the module to use. The **raw** module is used here, which is useful for running commands directly on the remote machine without requiring Python on the remote system. This is particularly handy for network devices or initial setup scenarios.
3. **-a "show ver":** The **-a** option is used to pass arguments to the module. Here, **"show ver"** is the command that will be executed on the remote host. This command is typically used in networking devices, like Cisco routers or switches, to show version information.
4. **-u user:** Specifies the username (**user**) to use for connecting to the remote host. This is part of Ansible's mechanism to authenticate with the host.
5. **-k:** This prompts the user to enter a password for the SSH connection. It's a way of providing SSH credentials interactively.

Function:

- This command connects to the host "S1" using the username "user" and the password provided interactively.
- Executes the **show ver** command on the host "S1" without requiring Python on the remote host.
- Primarily used for gathering version information from a network device.

Context:

- This is particularly useful in network environments, especially when dealing with devices that might not have Python installed.
- It's a quick way to gather information or execute a simple task on a remote host without the overhead of writing a full playbook.

2. Input the below command to retrieve the output of the S1 switch using ansible.

```
ansible S1 -m raw -a "show ip int brief" -u user -k
```

```
root@Ansible:~# ansible S1 -m raw -a "show ip int brief" -u user -k
SSH password:
S1 | CHANGED | rc=0 >>

Interface      IP-Address      OK? Method Status      Protocol
Ethernet0/0    unassigned      YES unset  up          up
Ethernet0/1    unassigned      YES unset  up          up
Ethernet0/2    unassigned      YES unset  up          up
Ethernet0/3    unassigned      YES unset  up          up
Ethernet1/0    unassigned      YES unset  up          up
Ethernet1/1    unassigned      YES unset  up          up
Ethernet1/2    unassigned      YES unset  up          up
Ethernet1/3    unassigned      YES unset  up          up
Vlan1          192.168.100.10 YES NVRAM   up          up
Shared connection to s1 closed.

root@Ansible:~#
```

Step 5: Use Ansible Ad hoc commands (Multiple device)

1. Input the below command to retrieve the output of the S1 switch using ansible.

```
ansible ios -i ./hosts -m raw -a "show ver" -u user -k
```

```
root@Ansible:~#
root@Ansible:~# ansible ios -i ./hosts -m raw -a "show ver" -u user -k
SSH password:
S4 | CHANGED | rc=0 >>
Cisco IOS Software, Linux Software (I86BI_LINUXL2-IPBASEK9-M), Experimental Version 15
Copyright (c) 1986-2017 by Cisco Systems, Inc.
Compiled Wed 09-Aug-17 13:49 by xxxxxxxx

ROM: Bootstrap program is Linux

S4 uptime is 1 hour, 26 minutes
System returned to ROM by reload at 0
System image file is "unix:/opt/unetlab/tmp/6/66/i86bi_linux_l2-ipbasek9-ms.high_iron.
Last reload reason: Unknown reason

This product contains cryptographic features and is subject to United
States and local country laws governing import, export, transfer and
use. Delivery of Cisco cryptographic products does not imply
third-party authority to import, export, distribute or use encryption.
Importers, exporters, distributors and users are responsible for
compliance with U.S. and local country laws. By using this product you
agree to comply with applicable laws and regulations. If you are unable
to comply with U.S. and local laws, return this product immediately.

A summary of U.S. laws governing Cisco cryptographic products may be found at:
http://www.cisco.com/wwl/export/crypto/tool/stqrg.html

If you require further assistance please contact us by sending email to
export@cisco.com.

Linux Unix (Intel-x86) processor with 943604K bytes of memory.
Processor board ID 67244037
4 Ethernet interfaces
1 Virtual Ethernet interface
1024K bytes of NVRAM.

Configuration register is 0x0
Warning: Permanently added 's4,192.168.100.40' (RSA) to the list of known hosts.
Shared connection to s4 closed.

S2 | CHANGED | rc=0 >>
Cisco IOS Software, Linux Software (I86BI_LINUXL2-IPBASEK9-M), Experimental Version 15
Copyright (c) 1986-2017 by Cisco Systems, Inc.
Compiled Wed 09-Aug-17 13:49 by xxxxxxxx

ROM: Bootstrap program is Linux

S2 uptime is 1 hour, 26 minutes
System returned to ROM by reload at 0
```

Let's go through the command:

ansible: The Ansible command line tool for executing tasks or playbooks.

ios: Refers to the target host or group in the Ansible inventory. "ios" suggests that the target devices are running Cisco IOS.

-i ./hosts: The **-i** option specifies the inventory file. **./hosts** indicates the inventory file is named "hosts" and is located in the current directory.

-m raw: Chooses the **raw** module, which is used for executing commands directly on remote devices, especially useful when the remote host doesn't have Python.

-a "show ver": Passes **"show ver"** as the argument to the module. This command is typically used to display version information on Cisco IOS devices.

-u user: Specifies "user" as the username for SSH authentication with the remote device.

-k: Prompts for the SSH password interactively.

Function:

- Executes the **show ver** command on Cisco IOS devices specified in the "hosts" inventory file.
- Uses SSH for connecting, with the username "user" and a password provided at runtime.
- Ideal for quickly gathering version information from Cisco IOS devices without Python prerequisites.

Christian H

<https://www.linkedin.com/in/christian-h-8a668aa5/>

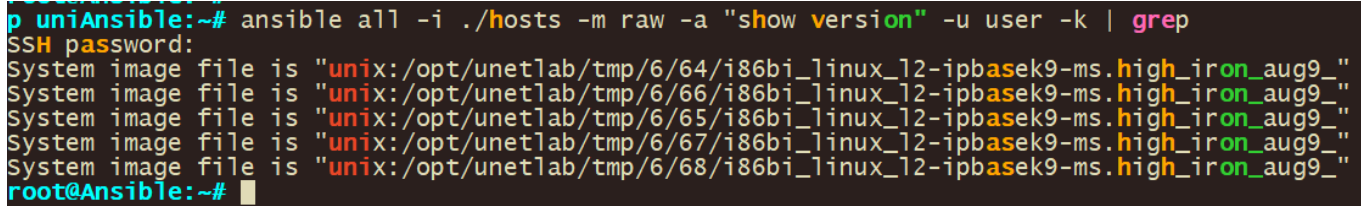
Use Case:

- Suitable for network environments with Cisco IOS devices.
- Provides a swift and direct method to execute a basic command across multiple devices, based on the defined inventory.
- Particularly valuable in scenarios where quick, ad-hoc information retrieval from network devices is needed.

Step 6: Filter on Ansible output

1. You can filter on the output that you get back from ansible using the below command:

```
ansible all -i ./hosts -m raw -a "show version" -u user -k | grep uni
```



```
p uniAnsible:~# ansible all -i ./hosts -m raw -a "show version" -u user -k | grep
SSH password:
System image file is "unix:/opt/unetlab/tmp/6/64/i86bi_linux_l2-ipbasek9-ms.high_iron_aug9_"
System image file is "unix:/opt/unetlab/tmp/6/66/i86bi_linux_l2-ipbbasek9-ms.high_iron_aug9_"
System image file is "unix:/opt/unetlab/tmp/6/65/i86bi_linux_l2-ipbbasek9-ms.high_iron_aug9_"
System image file is "unix:/opt/unetlab/tmp/6/67/i86bi_linux_l2-ipbbasek9-ms.high_iron_aug9_"
System image file is "unix:/opt/unetlab/tmp/6/68/i86bi_linux_l2-ipbbasek9-ms.high_iron_aug9_"
root@Ansible:~#
```

This command is similar to the previous ones, but this time it includes a pipe (“|”) to the “grep” command:

| grep uni: The output of the ansible command is piped (“|”) into the “grep” command. “grep” is a command-line utility for searching plain-text data sets for lines that match a regular expression. In this case, it’s used to filter and display only the lines of the output that contain the string “flash0”.

So, in short, this command will ask for a password, then connect as the “user” to all hosts defined in the “hosts” file, run the “show version” command on each of them, and then filter the output to display only the lines that contain “uni”.

- ```
ansible all -i ./hosts -m raw -a "show version" -u user -k | grep "CHANGED\\|Version"
```

This command is like the previous ones, but this time it includes another pipe (“|”) on the “grep” command:

“grep” is a command-line utility for searching plain-text data sets for lines that match a regular expression.

In this case, it's used to filter and display only the lines of the output that contain the string "CHANGED" or "Version". CHANGED is used because that is the line that contains the switch username in this example.

In this context, the “\” character is used as an escape character. It’s telling the shell to interpret the character that follows it literally, as part of the string, rather than as a special character with its own meaning.

In the “**grep “CHANGED\\|Version”**” portion of the command, “\\|” is used to indicate a logical OR in the “grep” command. It’s saying “match lines that contain “CHANGED” or “Version””.

Without the “\”, the pipe character “|” would be interpreted by the shell as a control operator used for piping the stdout (standard output) of one command into the stdin (standard input) of another. By escaping it with “\”, **you’re telling the shell to pass the “|”** character as part of the argument to “grep”, so “grep” can use it as the OR operator in its regular expression.

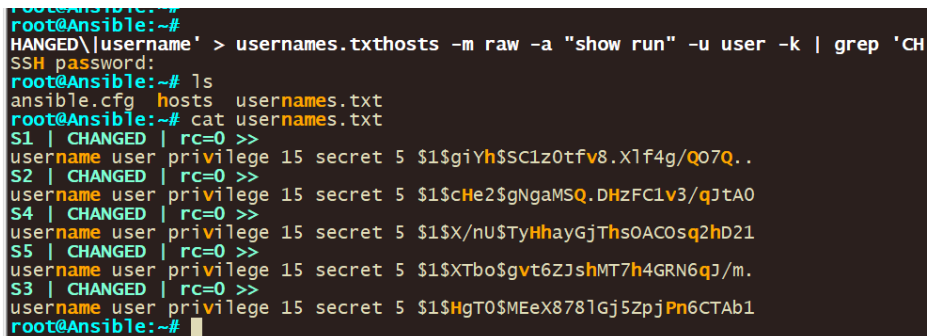
So, in short, this command will ask for a password, then connect as the “user” to all hosts defined in the “Eve-ng/GNS3hosts” file, run the “show version” command on each of them, and then filter the output to display only the lines that contain “CHANGED” or “Version”.



## Step 7: Store Ansible file output to a File

1. With ansible you can save your output (filtered or not ) to a file with the below command:

```
ansible all -i ./hosts -m raw -a "show run" -u user -k | grep "CHANGED\\|username" > usernames.txt
```



```

root@Ansible:~#
root@Ansible:~#
root@Ansible:~# ' > usernames.txt
root@Ansible:~# cat usernames.txt
S1 | CHANGED | rc=0 >>
username user privilege 15 secret 5 1giYh$Sc1z0tfv8.X7f4g/Q07Q..
S2 | CHANGED | rc=0 >>
username user privilege 15 secret 5 1CHe2$gNgamsQ.DHzFC1v3/qJtA0
S4 | CHANGED | rc=0 >>
username user privilege 15 secret 5 1X/nU$TyHhayGjThsOACosq2hD21
S5 | CHANGED | rc=0 >>
username user privilege 15 secret 5 1XTbo$gvT6ZJshMT7h4GRN6qJ/m.
S3 | CHANGED | rc=0 >>
username user privilege 15 secret 5 1HgT0$MEeX8781Gj5ZpjPn6CTAb1
root@Ansible:~#

```

This command uses Ansible to execute commands on multiple hosts, filters the output, and then writes the filtered output to a file. Here's what it does:

**“ansible”**: This is the command to run Ansible.

**“all”**: This targets all hosts defined in the inventory file.

**“-i ./hosts”**: Specifies the inventory file that Ansible uses to find the hosts. The “./hosts” implies that the inventory file is named “hosts” and is in the same directory from where the command is being run.

**“-m raw”**: Specifies the module that Ansible uses. The “raw” module is used to run low-level commands, usually when regular Ansible modules are not applicable.

**“-a "show run"”**: Specifies the actual command to be run on the target hosts. In this case, the command is “show run”, which typically displays the running configuration of the system.

**“-u user”**: Specifies the user as which to run the command. Here, the user is named “user”.

**“-k”**: Prompts for a password. Typically used when password-based SSH authentication is used instead of key-based authentication.

**“| grep “CHANGED\\|username””**: The output of the ansible command is piped (“|”) into the “grep” command. “grep” is a command-line utility for searching plain-text data sets for lines that match a regular expression. In this case, it's used to filter and display only the lines of the output that contain the string “CHANGED” or “username”.

**“-> usernames.txt”**: This redirects the output of the command to the file “usernames.txt”. If the file does not exist, it will be created. If it does exist, it will be overwritten.

So, in short, this command will ask for a password, then connect as the “user” to all hosts defined in the “hosts” file, run the “show run” command on each of them, filter the output to display only the lines that contain “CHANGED” or “username”, and then write these filtered lines to “usernames.txt”.

2. You can also grab the entire output for instance grab the running configs of all your devices.

```
ansible all -i ./hosts -m raw -a "show run" -u user -k > shrun.txt
```

```
root@Ansible:~#
xtot@Ansible:~# ansible all -i ./hosts -m raw -a "show run" -u user -k > shrun.tx
SSH password:
root@Ansible:~# ls
ansible.cfg hosts shrun.txt usernames.txt
root@Ansible:~# cat shrun.txt
S1 | CHANGED | rc=0 >>

Building configuration...

Current configuration : 1105 bytes
!
! Last configuration change at 02:32:43 UTC Sat Dec 23 2023
!
version 15.2
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
service compress-config
!
hostname S1
!
boot-start-marker
boot-end-marker
```

## Step 8: Use ansible for troubleshooting

1. You can use ansible Ad-hoc commands for troubleshooting. It's a matter of what you are looking for.  
You can use the below command as an example:

```
ansible all -i ./hosts -m raw -a "show arp" -u user -k
```

```
SSH password:
root@Ansible:~# ansible all -i ./hosts -m raw -a "show arp" -u user -k
SSH password:
S1 | CHANGED | rc=0 >>

Protocol Address Age (min) Hardware Addr Type Interface

Internet 192.168.100.1 23 5000.003c.0001 ARPA Vlan1
Internet 192.168.100.10 123 aabb.cc80.0100 ARPA Vlan1
Internet 192.168.100.20 123 aabb.cc80.0400 ARPA Vlan1
Internet 192.168.100.30 123 aabb.cc80.0200 ARPA Vlan1
Internet 192.168.100.31 119 50a9.4000.4600 ARPA Vlan1
Internet 192.168.100.40 123 aabb.cc80.0500 ARPA Vlan1
Internet 192.168.100.41 120 50d0.9600.4500 ARPA Vlan1
Internet 192.168.100.50 123 aabb.cc80.0300 ARPA Vlan1
Internet 192.168.100.51 118 50a9.2c00.4700 ARPA Vlan1shared connection to s1 closed.

S4 | CHANGED | rc=0 >>

Protocol Address Age (min) Hardware Addr Type Interface

Internet 192.168.100.1 23 5000.003c.0001 ARPA Vlan1
Internet 192.168.100.10 123 aabb.cc80.0100 ARPA Vlan1
Internet 192.168.100.20 123 aabb.cc80.0400 ARPA Vlan1
Internet 192.168.100.30 123 aabb.cc80.0200 ARPA Vlan1
Internet 192.168.100.31 119 50a9.4000.4600 ARPA Vlan1
Internet 192.168.100.40 123 aabb.cc80.0500 ARPA Vlan1
Internet 192.168.100.41 120 50d0.9600.4500 ARPA Vlan1
Internet 192.168.100.50 123 aabb.cc80.0300 ARPA Vlan1
Internet 192.168.100.51 118 50a9.2c00.4700 ARPA Vlan1shared connection to s4 closed.

S2 | CHANGED | rc=0 >>

Protocol Address Age (min) Hardware Addr Type Interface

Internet 192.168.100.1 23 5000.003c.0001 ARPA Vlan1
Internet 192.168.100.10 123 aabb.cc80.0100 ARPA Vlan1
Internet 192.168.100.20 123 aabb.cc80.0400 ARPA Vlan1
Internet 192.168.100.30 123 aabb.cc80.0200 ARPA Vlan1
Internet 192.168.100.31 119 50a9.4000.4600 ARPA Vlan1
Internet 192.168.100.40 123 aabb.cc80.0500 ARPA Vlan1
Internet 192.168.100.41 120 50d0.9600.4500 ARPA Vlan1
Internet 192.168.100.50 123 aabb.cc80.0300 ARPA Vlan1
Internet 192.168.100.51 118 50a9.2c00.4700 ARPA Vlan1shared connection to s2 closed.

S3 | CHANGED | rc=0 >>

Protocol Address Age (min) Hardware Addr Type Interface

Internet 192.168.100.1 23 5000.003c.0001 ARPA Vlan1
Internet 192.168.100.10 123 aabb.cc80.0100 ARPA Vlan1
Internet 192.168.100.20 123 aabb.cc80.0400 ARPA Vlan1
Internet 192.168.100.30 123 aabb.cc80.0200 ARPA Vlan1
Internet 192.168.100.31 119 50a9.4000.4600 ARPA Vlan1
Internet 192.168.100.40 123 aabb.cc80.0500 ARPA Vlan1
Internet 192.168.100.41 120 50d0.9600.4500 ARPA Vlan1
Internet 192.168.100.50 123 aabb.cc80.0300 ARPA Vlan1
Internet 192.168.100.51 118 50a9.2c00.4700 ARPA Vlan1shared connection to s3 closed.

S5 | CHANGED | rc=0 >>

Protocol Address Age (min) Hardware Addr Type Interface

Internet 192.168.100.1 23 5000.003c.0001 ARPA Vlan1
Internet 192.168.100.10 123 aabb.cc80.0100 ARPA Vlan1
Internet 192.168.100.20 123 aabb.cc80.0400 ARPA Vlan1
Internet 192.168.100.30 123 aabb.cc80.0200 ARPA Vlan1
Internet 192.168.100.31 119 50a9.4000.4600 ARPA Vlan1
Internet 192.168.100.40 123 aabb.cc80.0500 ARPA Vlan1
Internet 192.168.100.41 120 50d0.9600.4500 ARPA Vlan1
Internet 192.168.100.50 123 aabb.cc80.0300 ARPA Vlan1
Internet 192.168.100.51 118 50a9.2c00.4700 ARPA Vlan1shared connection to s5 closed.

root@Ansible:~#
```

### Components:

- ansible**: This is the Ansible command line tool, employed for running tasks or playbooks.
- all**: This signifies the target, which in this case is 'all' hosts or groups defined in the inventory file.
- i ./hosts**: The **-i** option points to the inventory file. Here, **./hosts** indicates the inventory file named 'hosts' located in the current directory.
- m raw**: Specifies the use of the **raw** module. This module is essential for running commands directly on the remote hosts, particularly useful when Python is not installed on these hosts.
- a "show arp"**: The **-a** option provides arguments to the module. **"show arp"** is the command to be executed, typically used to display the ARP table on a network device.
- u user**: This specifies 'user' as the username for SSH connections to the remote hosts.
- k**: Prompts for the SSH password interactively when connecting to the hosts.

### Function:

- Connects to each host listed in the 'hosts' inventory file using SSH with the username 'user' and a password provided upon execution.

- Executes the **show arp** command across all these hosts. The **show arp** command is common in network devices for displaying the Address Resolution Protocol (ARP) table, which maps IP addresses to physical MAC addresses.

#### Context:

- This command is highly efficient for network management tasks, particularly in environments with multiple hosts, where you need to quickly gather ARP table information from each device.
- It's especially handy in network troubleshooting and monitoring scenarios.

2. You can filter using ansible Ad-hoc commands for troubleshooting. It's a matter of what you are looking for and filtering for. You can use the below command as an example:

```
ansible all -i ./hosts -m raw -a "show arp" -u user -k | grep 50a9.2c00.4700
```

\*This is the mac-address for our host

```
root@Ansible:~#
root@Ansible:~#
a9.2c00.4700:~# ansible all -i ./hosts -m raw -a "show arp" -u user -k | grep 50a
SSH password:
Internet 192.168.100.51 125 50a9.2c00.4700 ARPA Vlan1Shared connection to s2 closed.
Internet 192.168.100.51 125 50a9.2c00.4700 ARPA Vlan1Shared connection to s5 closed.
Internet 192.168.100.51 125 50a9.2c00.4700 ARPA Vlan1Shared connection to s3 closed.
Internet 192.168.100.51 125 50a9.2c00.4700 ARPA Vlan1Shared connection to s4 closed.
Internet 192.168.100.51 125 50a9.2c00.4700 ARPA Vlan1Shared connection to s1 closed.
root@Ansible:~#
root@Ansible:~#
root@Ansible:~#
root@Ansible:~#
```

#### Components:

**ansible all**: Executes the command across all hosts in the inventory.

**-i ./hosts**: Specifies the inventory file located at **./hosts**.

**-m raw**: Uses the **raw** module to execute direct commands.

**-a "show arp"**: Runs the **show arp** command on each host, which displays the ARP table.

**-u user**: Connects as the 'user' through SSH.

**-k**: Prompts for the SSH password interactively.

**| grep 50a9.2c00.4700**: Pipes (|) the output of the Ansible command into **grep**, filtering for lines containing '50a9.2c00.4700'.

#### Function:

- This command will connect to every host in the **./hosts** file.
- On each host, it runs **show arp** to display the ARP table.
- The output is then filtered through **grep** for the specific MAC address '50a9.2c00.4700'.
- Useful for finding which hosts have this MAC address in their ARP tables.

#### Use Case:

- Ideal for scenarios where you need to quickly identify the presence and location of a specific device (identified by MAC address) across a network.
- Enhances efficiency in network troubleshooting and monitoring.