# Enhanced Ansible ad-hoc and Cisco IOS playbook Configuration Guide

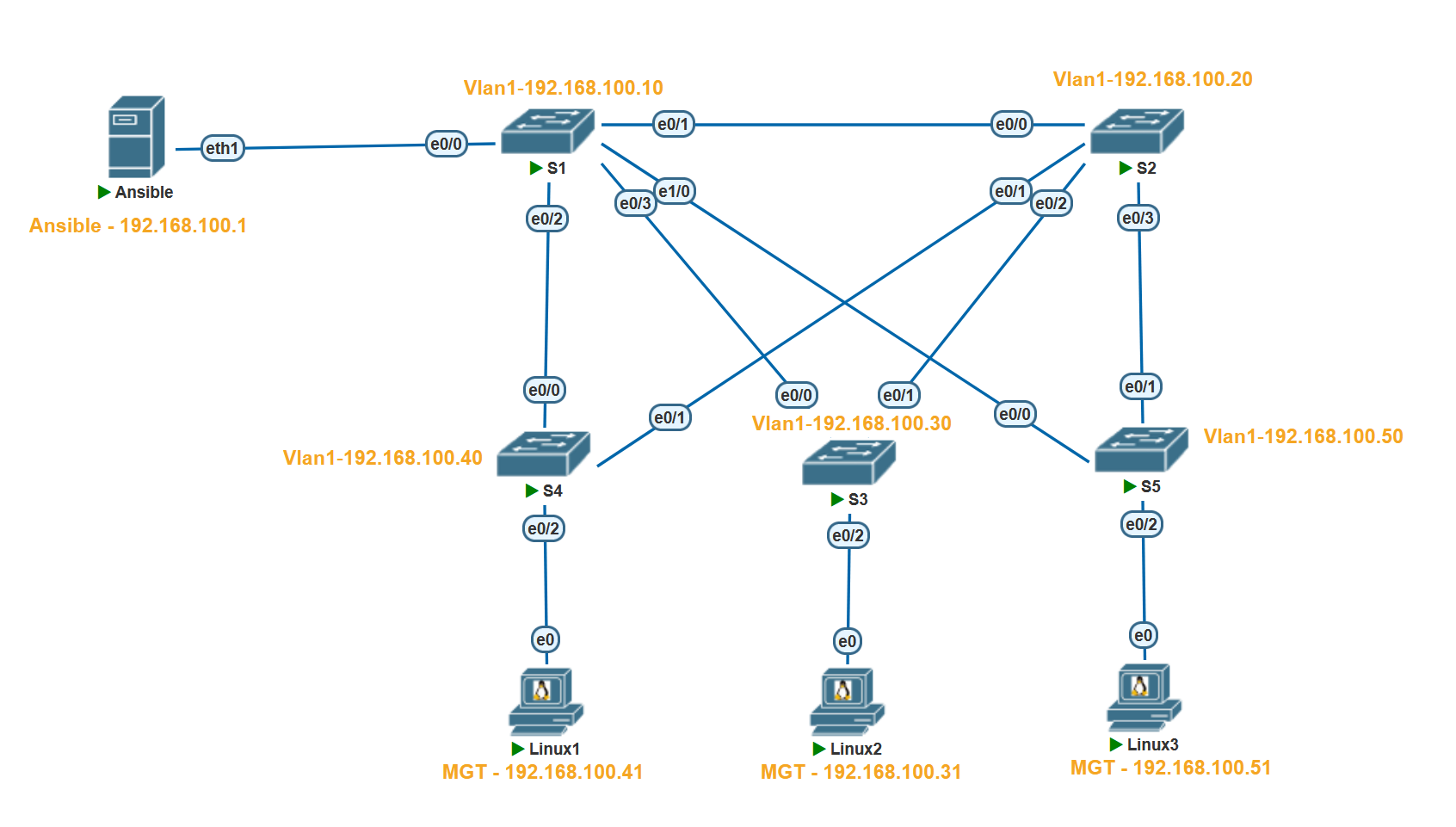
This guide **builds** upon the initial Ansible and Cisco Switch configuration detailed in the previously provided guide (https://github.com/ WiredWizardry/Network/tree/master/04\_Ansible\_Basic\_AdHoc). It aims to further enhance your network automation capabilities using Ansible playbooks. These playbooks are instrumental in retrieving vital network information such as ARP tables, MAC address tables, and device versions, thus simplifying network management and troubleshooting.

**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 vios\_l2 @192.168.100.10 |
| **S2** | Cisco Switch running vios\_l2 @192.168.100.20 |
| **S3** | Cisco Switch running vios\_l2 @192.168.100.30 |
| **S4** | Cisco Switch running vios\_l2 @192.168.100.40 |
| **S5** | Cisco Switch running vios\_l2 @192.168.100.50 |

Prerequires:

* 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 0: Configure Switch and Ansible

Refer to the initial guide for configuring the Cisco switch (S1 - S5) for basic connectivity and SSH access.

Refer to the initial guide for configuring Ansible

## Step 1: Configure Switch and Ansible

The first step involves creating an Ansible playbook to retrieve the ARP (Address Resolution Protocol) table from Cisco switches. The ARP table is crucial as it maps IP addresses to MAC addresses, helping in network troubleshooting and understanding device connections on the network.

* 1. **Creating the Playbook (getarp1.yml)**

Open a text editor (nano, in this case) on the control node to create the playbook:

nano getarp1.yml

The content of the playbook should be as follows:

---

- name: Retrieve ARP table

hosts: all

gather\_facts: false

tasks:

- name: show arp

raw: "show arp"

register: print\_output

- debug: var=print\_output.stdout\_lines

**Explanation**:

**hosts: all:** Targets all hosts specified in the Ansible inventory.

**gather\_facts: false:** Disables gathering system facts to speed up the execution.

**tasks**: Defines the operations to be performed.

**raw: "show arp":** Sends the show arp command directly to the Cisco switch CLI. This command displays the ARP table.

**register: print\_output:** Stores the output of the command in the variable print\_output.

**debug: var=print\_output.stdout\_lines:** Prints the output to the console for inspection.

* 1. **Executing the Playbook**

Run the playbook using the following command:

ansible-playbook getarp1.yml -u [username] -k

Replace [username] with the appropriate SSH username for the switches. The -k option prompts for the SSH password.

* 1. **Optional: Filtering Output**

In scenarios where specific entries are needed, use grep to filter the output:

**To filter by MAC address:**

ansible-playbook getarp1.yml -u [username] -k | grep 'ok:\|4700'

To filter by IP address:

ansible-playbook getarp1.yml -u [username] -k | grep 'ok:\|100.51'

These commands extract lines containing the 'ok:' status or the specified MAC/IP addresses.

This playbook is a fundamental tool for network engineers, allowing quick retrieval of ARP tables from multiple devices in a network. It not only saves time but also ensures accuracy by automating a task that would otherwise be manual and prone to human error. By integrating such playbooks into regular network operations, network engineers can significantly enhance efficiency and responsiveness in managing and troubleshooting network issues.

## Step 2: Getting MAC Address Table with Ansible

The second step involves using an Ansible playbook to extract the MAC (Media Access Control) address table from Cisco switches. This MAC address table is vital for network management as it associates each MAC address with its corresponding port on the switch, aiding in network monitoring and security enforcement.

**1.** **Creating the Playbook (getmacaddress1.yml)**

Start by opening a text editor on the control node:

nano getmacaddress1.yml

The content of the playbook should be as follows:

---

- name: Get MAC address table information

hosts: all

gather\_facts: false

tasks:

- name: show MAC address table

raw: "show mac address-table"

register: print\_output

- debug: var=print\_output.stdout\_lines

**Explanation:**

**hosts: all**: This specifies that the playbook should run on all hosts listed in the Ansible inventory.

**gather\_facts: false:** This disables the default gathering of system facts to expedite the playbook execution.

**tasks:** Defines the sequence of tasks to be executed.

**raw: "show mac address-table":** Executes the show mac address-table command on the switch. This command is used to display the MAC address table of the switch.

**register: print\_output:** Captures the output of the command and stores it in the print\_output variable.

**debug: var=print\_output.stdout\_lines:** Outputs the contents of print\_output to the console for review.

**2. Executing the Playbook**

To run the playbook, use the following command:

ansible-playbook getmacaddress1.yml -u [username] -k

Replace [username] with the actual SSH username for the switches. The -k flag prompts for the SSH password.

1. **Optional: Filtering Output**

If you need to focus on specific MAC addresses, you can filter the output:

**Filter by a specific MAC address:**

ansible-playbook getmacaddress1.yml -u [username] -k | grep 'ok:\|4700'

This command filters the playbook's output, showing only lines containing 'ok:' (indicating successful task execution) or the specified MAC address.

This playbook provides a streamlined approach to gather MAC address table information across multiple network devices. It's particularly useful in environments where regular monitoring of MAC address entries is essential for security, compliance, and network management. Automating this process with Ansible enhances efficiency, reduces the likelihood of human error, and contributes to more effective network administration practices.

## Step 3: Displaying Version and Interface Information with Ansible

The objective of this step is to create and execute an Ansible playbook, named shver1.yml, that retrieves both the version and interface information from Cisco switches. This information is critical for verifying the software version for compliance and security purposes, as well as for checking the status of network interfaces.

1.Creating the Playbook (shver1.yml)

Start by opening a text editor on the control node:

nano shver1.yml

The content of the playbook should be as follows:

---

- name: Show version and interface information

hosts: all

gather\_facts: false

connection: local

tasks:

- name: Run multiple commands on remote nodes

ios\_command:

commands:

- show version

- show ip int brief

register: print\_output

- debug: var=print\_output.stdout\_lines

**Explanation**:

**hosts: all:** Targets all hosts defined in the Ansible inventory.

**gather\_facts: false:** Disables automatic fact gathering to speed up execution.

**connection: local:** Runs the playbook on the local machine.

**tasks:** Lists the operations to be performed.

**ios\_command:** A module tailored for Cisco IOS devices, used to send commands to them.

**commands:** A list of commands to be executed on the switches. Here, show version displays software version information, and show ip int brief provides a summary of the interface statuses.

**register: print\_output**: Captures the output of the commands in the variable print\_output.

**debug: var=print\_output.stdout\_lines:** Displays the captured output for verification and analysis.

2. Executing the Playbook

To run the playbook, use the following command:

ansible-playbook shver1.yml -u [username] -k

Replace [username] with the appropriate SSH username for the switches. The -k option will prompt for the SSH password.

This playbook serves as an essential tool for network administrators, allowing for quick and efficient retrieval of version and interface information from Cisco switches. The ability to execute multiple commands in a single playbook and view their outputs can greatly aid in routine network checks, compliance audits, and troubleshooting tasks. Automation via Ansible in this manner not only saves time but also ensures consistency and accuracy in network management operations.

## Step 4: Backup and Save Output to File with Ansible

The purpose of this step is to develop an Ansible playbook (getver2.yml) that executes the show version command on Cisco switches, retrieves the output, and then saves this information to a file. This task is vital for maintaining records of the software version of network devices, which can be crucial for compliance, auditing, and troubleshooting. Input the below command to retrieve the output of the S1 switch using ansible.

1. **Creating the Playbook (getver2.yml)**

Open a text editor on the control node:

nano getver2.yml

Then, input the following content into the playbook:

---

- name: Backup show version command output

hosts: all

gather\_facts: false

connection: local

tasks:

- name: Run show version command

ios\_command:

commands:

- show version

register: print\_output

- debug: var=print\_output.stdout\_lines

- name: Save output to a file

copy: content="{{ print\_output.stdout[0] }}" dest="./output/{{ inventory\_hostname }}.txt"

**Explanation**:

**hosts: all:** Indicates that the playbook will be executed on all hosts in the Ansible inventory.

**gather\_facts: false:** Disables the default fact-gathering to speed up the playbook.

**connection: local:** Specifies that the playbook runs on the local machine.

**tasks**: Lists the tasks to be executed.

**ios\_command:** An Ansible module designed for Cisco IOS devices. It sends specified commands to the devices.

**commands**: The actual commands to be executed on the switches. show version provides detailed version information of the Cisco IOS software.

**register: print\_output:** Saves the output of the command in the print\_output variable.

**debug: var=print\_output.stdout\_lines:** Displays the output for review.

**copy:** An Ansible module used to save the output to a file.

**content:** Specifies the content to be saved, in this case, the first line of the stdout from print\_output.

**dest:** The destination path and filename for the saved file, dynamically using the host name for organization.

1. **Preparing Output Directory**

Before running the playbook, create a directory to store the output files:

mkdir output

1. Executing the Playbook

Run the playbook with the following command:

ansible-playbook getver2.yml -u [username] -k

Replace [username] with the SSH username for the switches. The -k flag is used to prompt for the SSH password.

This playbook is an efficient tool for automated backup and documentation of Cisco switch software versions. By saving the output of show version to individual files named after each switch, it facilitates easy tracking of software versions and aids in historical record-keeping. This approach is especially useful for compliance with IT policies, auditing purposes, and maintaining a robust change management process in network environments.