10. From start to finish

10. From start to finish

For this last chapter we will go through an example scenario to demonstrate how a user like you could implement Ansible in your own network. The explanation below will follow a simple step by step guide with additional explanation provided by each step, detailing the usage and reasoning behind the taken step.

Before we continue it is recommended that you read through the previous chapters or at least have a basic understanding of both Ansible and Cisco IOS.

Case introduction:
The Company called "Company A" has taken over another company with 1 site. The site of that company was fully implemented with Cisco devices in 2019. Thanks to the fact that the previous company used Cisco devices it was decided to not replace the devices themselves and simply amend the existing configurations to "Company A" standards.
The network administrators decided to streamline the process using Ansible making it as idempotent as possible. After thorough inspection it was decided to amend the following configurations:

- MOTD banner
- ACL's
- VLANs
- SNMP

Requirements to follow along with the walkthrough:

- (Home/Virtual) Lab environment containing: 2 Routers, 2 L3 Switches and 2 L2 switches.
- Working knowledge of virtualization environments (for the Ansible server) or a physical Linux host.
- General understanding of IOS commands and Ansible playbooks (how to save, edit and run).

10.1 Step 1 – Current Network and Init

Initial setup and configurations:
To start off we would strongly advise you to copy our Topology to your lab environment of choice and configure the devices as stated in the code blocks named after the device. This will provide you with the smoothest start as it won't require any additional steps which are not covered by this walkthrough. In case you've been following along with chapter 9 as well you can keep using your current configuration as the ansible playbooks used would lead you to our base configurations. Otherwise just copy the configurations below into your devices CLI.

Topology:

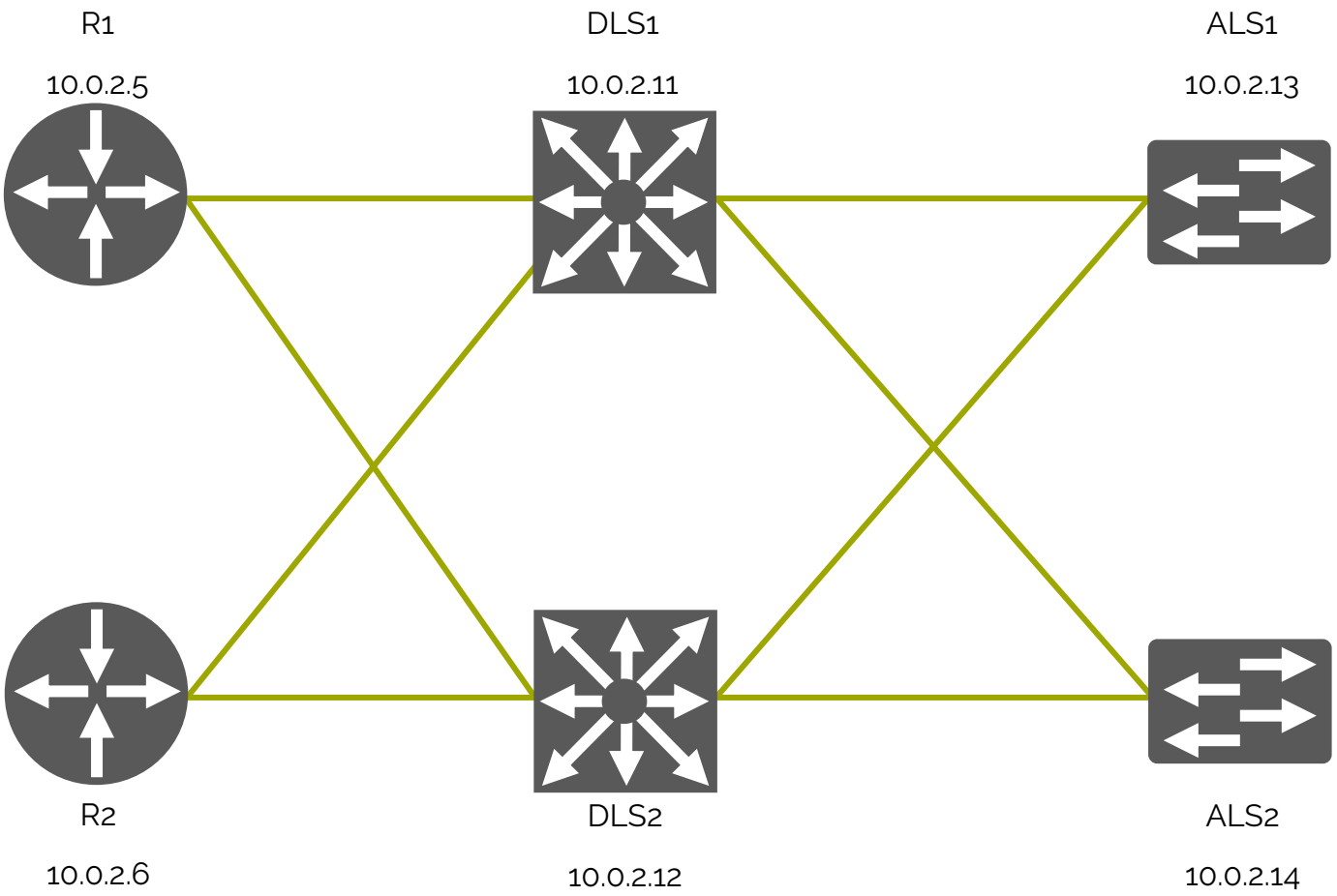

Image 4: Architecture example

The configuration files for the different devices can be downloaded by clicking on the devices above, or by downloading them as a compressed folder with this link.

10.2 Step 2 – Setting up your Linux and Ansible

Linux Host

Now that the Cisco devices are configured it's time to get the Linux Host in working order. For walkthrough of this walkthrough we will use Ubuntu 20.04 as it's the most commonly used Linux system - you can use any distro you want but for the sake of streamlining the walkthrough we will proceed with Ubuntu 20.04. In case you chose a different distro please refer to Chapter 2 for all the requirements and installation steps.

Once you get your Linux machine installed by going through all the steps, open up your CLI and type in the following commands.

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

These commands will update your distribution, add the required repository for Ansible and install Ansible. After completing the installation steps we recommend to test out if Ansible is working by using the following command in your CLI:

```
ansible all -m ping -v
```

If everything went correctly you are done with the basic configuration and you're ready to proceed onto the Ansible configuration proceed to implementing Ansible for this specific use case. If you received an error try to read over the guide again and verify you've followed the steps correctly.

10.3 Step 3 – Creation of the Inventory file

Now that everything is set up and the connections have been tested it's time to aggregate the addresses into 1 inventory file for Ansible to use. To do this we need to open up the CLI on our Linux host and navigate to the /etc/ansible directory.

```
cd /etc/ansible
```

Once we're there we will add a file called "Hosts". This file will provide us with a working environment for Ansible.
The Hosts file will be the inventory file (as described in Chapter 2) containing the address information about the IOS devices. It will also contain the connection information and password for the IOS devices for Ansible to push/receive data from these devices.

```
Sudo nano Hosts
```

This will open up the Nano editor and enable us to type in all the devices which would be used later on with our playbooks. For now you can use the code provided below and once you're done we will explain the code line by line or you can revisit Chapter 3 for a detailed explanation on Inventory files.

```
[all:vars]
ansible_become=yes
ansible_become_method=enable
ansible_network_os=IOS
ansible_user=cisco
ansible_password=cisco
ansible_connection=network_cli

[Routers]
Router1 ansible_host=10.0.2.5
Router2 ansible_host=10.0.2.6

[L2Switches]
Switch1 ansible_host=10.0.2.13
Switch2 ansible_host=10.0.2.14

[L3Switches]
DLS1 ansible_host=10.0.2.11
DLS2 ansible_host=10.0.2.12
```

Table 1: explanation inventory file

Hosts file	Explanation
[all:vars]	Marking followed by all set up variables.
ansible_become=yes	Root access for Ansible.
ansible_become_method=enable	Enables the change of privilege modes.
ansible_network_os=IOS	Defines the OS of targets for Ansible.
ansible_user=cisco	Defines the user which Ansible will use to connect.
ansible_password=cisco	Defines the password which Ansible will use to connect.
ansible_connection=network_cli	Defines where Ansible will connect to.
[IOS]	Groups the devices under it to group called "IOS".
Router1 ansible_host=10.0.1.14	Sets up name and tells Ansible which IP the host has.

Finally we can test our groups by using the ping Ansible command again. Instead of defining the all group, the newly created Routers, L2Switches or L3Switches groups can be used to only ping the members of said group.

```
ansible Routers -m ping -v
ansible L2Switches -m ping -v
ansible L3Switches -m ping -v
```

10.4 Step 4 – Usage of playbooks and modules

Having achieved connectivity using Ansible, we will now discuss the modules and playbooks that we will use to amend the configuration as we discussed in the introduction. (For an explanation on modules please revisit chapter 6)

To streamline the commands that need to be used, we will use Ansible Galaxy. To find any relevant modules simply go to *galaxy.ansible.com* and search for Cisco. This returns multiple modules and collections provided by Cisco. We will use the collection cisco.IOS . To install this collection go to your Ansible host machine and, using the CLI, enter the following command:

```
ansible-galaxy collection install cisco.IOS
```

The addition of this collection will change the way we configured Cisco devices (as described in Chapter 9) to a more specialized way of doing instead of the broad commands that were available with IOS_config.

What we want to achieve with Ansible playbooks is idempotency. Idempotency means that whatever function you call it will always provide the same result without fault. To explain it with a simple mathematical example – think of multiplication by zero. It doesn't matter how often you will do it, the result will always remain the same zero.
With automation you don't want to configure every device separately, that wouldn't save you any time at all, but instead you want to be able to send a task to multiple machines at once, where the tasks are repeatable but the result will stay fixed. And that's exactly what we're trying to achieve with the new configurations for the network devices.

To recap what we needed to do from the introduction step. We need to implement a new *Banner* message, set up *ACLs*, configure *VLANs*, install and check the implementation of *SNMP*.

Starting off with the banner**.** The banner is something that's company-wide for the majority of devices. For a banner we can make a playbook that will use all the hosts from our inventory file. And because we installed the cisco.ios collection it might be a good idea to check the modules included, to see if there are any specific modules that will help us with the setup.

This can be done by revisiting Ansible galaxy website and following the links provided until the GitHub explanation of provided modules or by visiting https://github.com/ansible-collections/cisco.IOS/tree/main/docs .

Using the explanation provided in the GitHub docs and combining it with the examples provided from Cisco we can now easily make a playbook that will be idempotent and will satisfy our needs.

```
---
- name: Configure default info on all devices.
  hosts: all
  tasks:
    - name: Configure hostname and domain name.
      cisco.ios.ios_system:
        hostname: "{{ inventory_hostname }}"

    - name: Configure motd banner.
      cisco.ios.ios_banner:
        banner: motd
        text: |
          This device is for authorized personnel only.
          If you have not been provided with permission to
          access this device - disconnect at once.
        state: present

    - name: Configure login banner
      cisco.ios.ios_banner:
        banner: login
        text: |
          *** Ensure that you update the system configuration ***
          *** documentation after making system changes.      ***
        state: present
```

To reiterate what is happening in this playbook we will go through it line by line. Or you can revisit Chapter 4 and learn more about playbooks in detail.

Table 2: Explanation playbook banner

Configuration	Explanation
- name	The name to identify what our playbook does.
hosts: all	In this case all hosts contained in our inventory file.
- name	Name of a specific task (will be included in the output window).
cisco.ios.ios_banner	Name of the module used – when specified all its commands can be used below in the same task
banner: login	Specifies which banner should be configured on the remote device. In Ansible 2.4 and earlier only login and motd were supported.
text :	The banner text that should be present in the remote device running configuration. This argument accepts a multiline string, with no empty lines. Requires state=present.
state:	Specifies whether or not the configuration is present in the current devices active running configuration.

Running this playbook would configure the hostnames and domain name. These names are defined in the inventory file. The output of the playbook would look like the following in image 6.

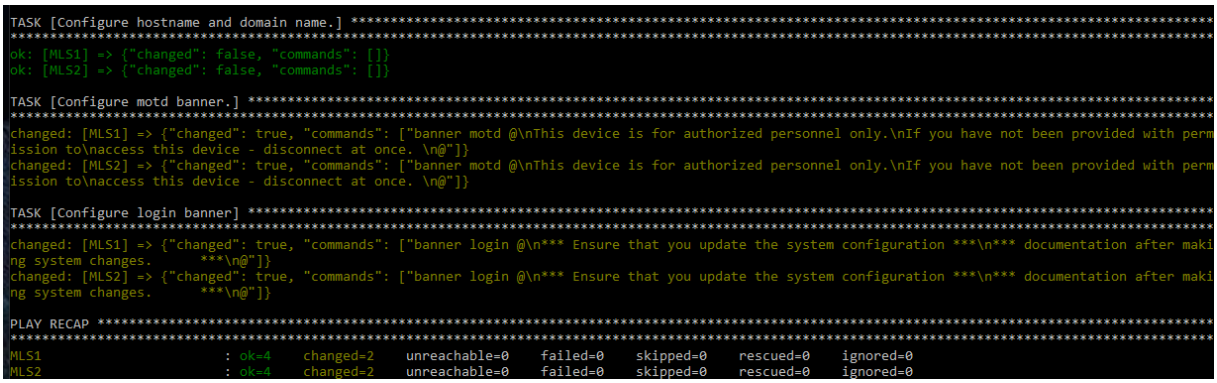

Image 6: Output playbook default information devices

Next on our list are the ACLs. Every network needs some form of security and in our case we will place the ACLs on the Distribution Layer Switches which are L3 switches. Unlike the previous playbook we will now specify just one group with the ACLs allowing for an SSH connection to or from the 10.1.1.0, 10.1.2.0 and 10.1.3.0 networks.
We will accomplish this using the cisco.ios.ios_acls module.

```
---
- name: Overwrite all ACL configuration.
  hosts: L3Switches
  tasks:
    - name: Merge provided configuration with device configuration
      cisco.ios.ios_acls:
        config:
        - afi: ipv4
          acls:
          - name: std_acl
            acl_type: standard
            aces:
            - grant: deny
              source:
                address: 192.168.1.200
            - grant: deny
              source:
                address: 192.168.2.0
                wildcard_bits: 0.0.0.255
        state: merged
```

Notable differences from the breakdown of our previous playbook are listed in the table 3 below.

Table 3: Explanation playbook ACL

Configuration	Explanation
hosts: L3Switches	Use all hosts from L3Switches group.
cisco.ios.ios_acls	A more general use module focused on line configuration.
- afi	The Address Family Indicator (AFI) for the Access Control Lists (ACL). This can be IPv4 or IPv6.
grant	This specifies the action. It can be permit or deny.

Going further down the list we arrive at VLANs**.** As our company uses two VLANs it is imperative that the switches know of them and are able to transfer the frames tagged with those VLAN-IDs.

This time we only need to make these changes on the Layer 2 switches. So our goal is to create two new VLANs on each of the layer 2 switches and set the interfaces to access mode for the correct interfaces. As previously mentioned during the banner setup it's always a good idea to check whether or not your collection has modules specialized for your tasks which helps with idempotency and most of the time makes it an easier and clearer to read playbook.
The *cisco.ios* collection also provides us with an *ios_vlans* and *ios_l3_interfaces* module. These modules simplify the actions we need, and are easy to use owing to the well documented module information on the GitHub page of the collection with examples.

Having a clear goal and the help of the documentation we can now write our playbook containing 3 tasks. Creation of the first VLAN, second VLAN and lastly setting up the access mode on the interfaces for our VLANs.

```
- name: Set vlans
  hosts: L3Switches
  tasks:
    - name: Configure Vlans
      cisco.ios.ios_vlans:
        config:
        - name: Users
          vlan_id: 10
          state: active
        - name: Servers
          vlan_id: 20
          state: active
        state: overridden

    - name: Replace L3 configuration
      cisco.ios.ios_l3_interfaces:
        config:
        - name: vlan10
          ipv4:
          - address: 10.0.3.1/24
        - name: vlan20
          ipv4:
          - address: 10.0.4.1/24
        state: replaced
```

```
    - name: Configure interfaces
      cisco.ios.ios_l2_interfaces:
        config:
        - name: Ethernet2/1
          mode: trunk
        - name: Ethernet2/2
          mode: trunk
        - name: Ethernet2/3
          mode: trunk
        state: replaced
```

With this we should be able to differentiate between each line of the playbook noticing the group change, different modules used their breakdown as it looks almost identical as to what you would normally write on your switch but slightly more organized.

The output of the VLAN playbook should give you the following output as in image 7.

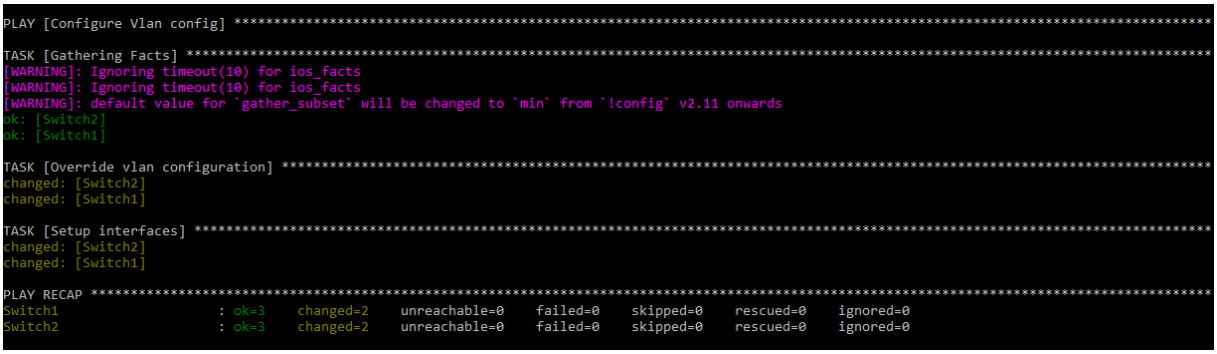

Image 7: Output playbook VLAN

Lastly we will create a playbook that checks for SNMP and collects the data, removes it and installs the newest version.
Seeing as Ansible works with a push model and each of our goals needs different results we will have to split each of our planned actions into different playbooks.

“Company A” as many other companies values data and wants to be up to date on the state of its devices to diagnose and improve over time. The usual configuration that they have runs on SNMP version 3. It is important to first check for the current diagnostics data and then upgrade the SNMP version that is set up on the new devices as they might have an older version configured.

While there is an SNMP specific module in the cisco.ios collection, the same idea as with ACLs applies. The cisco.ios.ios_config module provides an easy way to visualize and manage SNMP, as opposed to SNMP in the SNMP specific module.

So to check whether or not the devices are configured with SNMP we will run a show run command that grabs the lines which include “snmp-server” if they exist at all and show them directly in the output screen of the CLI on the Ansible host. To do this we should write the following playbook:

```
---
- name: Get SNMP data from all devices.
  hosts: all
  tasks:

    - name: Get SNMP data from running config.
      IOS_command:
        commands:
          - show running-config | include snmp-server
      register: if_data

    - name: Show SNMP data from running config.
      debug:
        var: if_data['stdout_lines'][0]
```

Now while this playbook looks quite similar to the ones we’ve used before there are a few small additions which are explained below.

Table 4: Explanation playbook SNMP

Configuration	Explanation
Register:if_data	Register saves data output as a variable.
Debug:	Prints statements during execution.
Var: `if_data['stdout_lines'][0]`	The Variable “if_data” procured from register are put on screen thanks to the stdout_lines.

Having this done and received data, we now know that there is an SNMP server configured on one/several of our devices which means that we can proceed to the removal. In case we haven’t received any data output we can instead proceed to the installation of SNMPv3.

The removal of SNMP is quite simple and possible the easiest playbook yet in our walkthrough. As usual you want to open your text editor in the CLI of your Ansible host and type / copy the following in:

```
---
- name: Remove any SNMP configuration.
  hosts: all
  tasks:
    - name: Clear SNMP configuration.
      IOS_config:
        lines:
          - no snmp-server
```

As we can see there was nothing new in this playbook as far as complexity goes which is good as the task itself was simple and shouldn't be made complicated. Now it's time to save it and run again by typing the following command:

```
Ansible-playbook play_name.yml
```

In case data was received the first time we ran our Check SNMP playbook, we can run it again to double check if there is still data showing up which shouldn't be there in this case.

And last but not least we have to configure SNMPv3**.** Now that we know for sure that none of our devices are configured with SNMP we can finally set it up to our needs. We will configure it to our needs with password authentication, version 3 and traps centred around OSPF and VLANs. We will also add a new feature to our Cisco playbook where the running config will be saved to the start-up configuration when modifications occur.

```
---
- name: Configure all devices with SNMPv3.
  hosts: all
  tasks:
    - name: Configure SNMP V3.
      IOS_config:
        lines:
          - snmp-server engineid remote 10.0.2.1 446172742E506F776
          - snmp-
server user ADMIN ADMINGROUP v3 auth md5 AUTHPASS priv aes 128 PRIVPASS
          - snmp-server group ADMINGROUP v3 priv
          - snmp-server host 10.0.2.1 traps version 3 priv ADMIN
          - snmp-server enable traps vlancreate
          - snmp-server enable traps vlandelete
          - snmp-server enable traps ospf
          - snmp-server enable traps ospf errors
    - name: save running to start up when modified
      cisco.IOS.IOS_config:
          save_when: modified
```

As previously explained we use IOS_config for simplicity and a neat code. The two noticeable changes this time come in the name of modules and the new task which helps with the longevity of your changes by saving your configuration in start-up config therefore not being reset by a reboot as would be the case with regular running config.

You may have noticed that we used the IOS_config module initially, followed by the cisco.IOS.IOS_config module. These are essentially the same if a newer version of Ansible is used, however if an older version of ansible is used it might be impossible to run only IOS_config, as abbreviations were not fully supported yet.

The second change is the second task which saves the running configuration to start-up configuration when there were any modifications. A simple and small task but definitely really important when you want your changes to be more than just temporary.

10.5 Step 5 – Ansible in retrospective

First of all – we would like to congratulate and thank you for following our Manual and guide so far.

Looking back at what we've achieved we can conclude a few things about automation and Ansible. Just in this chapter alone we learned the value of modules, idempotency and Ansible itself.

Starting with modules we found out how we can search for pre-made functionalities with galaxy and discovered how to find more information/explanation provided by the creators of these modules. This allowed us to expand on the base functionality that Ansible gives us for automation. Discovering this ability expands upon simple automation tasks and adds almost any automation task we can imagine. One thing of note however is that not all modules are made evenly as we discussed in step 4. In certain cases, specialized modules may not be optimal as these modules are often more convoluted in use. In these cases the ios_config module is recommended as this module exposes the standard IOS interface commands for use in ansible.

Next up we have Idempotency – as we learned before it is imperative for automation to be able to provide the one config on all devices which would provide the same results on all of the targets. If we ignore idempotency and proceed to write our automation tasks not caring for uniformity and deploying each task separately, we achieve nothing. Whilst nothing may be worded a bit too strongly, doing that and typing out your configs directly on the device has little to no difference.

And arriving at this point we have to talk about Ansible and the benefits of automation. A recurring theme in this manual is uniformity as well as the ease of code. Having used Ansible from the beginning of this walkthrough together it's safe to say that we haven't encountered any type of code that didn't come out as extremely difficult or convoluted. This is because Ansible tries to make it a selling point to be as easy to use as possible. Having a low barrier of entry helps while making both simple and complex automation tasks which can be differentiated with the use of modules. In case the pre-made modules would not be enough Ansible gives us the possibility of writing our own and or integrating coding languages like python with it, expanding on the functionalities. Now if we didn't use Ansible to automate our tasks we would be most likely forced to install an agent on our target devices making the setup step longer and more prone to mistakes or in other instances forced to re-copy our code for each device separately not adding to the process of automation but instead slightly cutting off the time you'd use while configurating your devices normally.