9. Task automation

9. Task automation

In chapter 8 we started with the configuration of Cisco IOS devices with their hostnames and loopback interfaces. But Ansible has more modules for IOS, whether they are for interfaces or just to set the banner for the login, combining these modules could create powerful automated configuration of Cisco IOS devices. In this chapter we will go through some configuration examples and what exactly happens.

9.1 Environment

In chapter eight we described what the entries in the hosts file are, the entries in Ansible.cfg are just to make Ansible skip the SSH key checking. Thus, when adding a switch to test it will not require me to trust the key. I recommend to only keep the password in plain text for a development environment.

9.1.1 Environment used

The environment used has been configured with the following files in the /etc/ansible directory.

Hosts:

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

[IOS]
Router1 ansible_host=10.0.1.14
Router2 ansible_host=10.0.1.15

[Switches]
Switch1 ansible_host=10.0.1.254
Switch2 ansible_host=10.0.1.16
```

Ansible.cfg:

```
[defaults]
host_key_checking = no
[paramiko_connection]
host_key_checking = no
```

9.1.2 OSPF example

Taking OSPF as example to configure on a Cisco IOS device. For example, the following code can be used to configure OSPF on IOS devices.

```
- name: Set OSPF.
  ios_config:
    parents: router ospf 1
    lines:
      - network 10.2.0.0 0.0.0.255 area 0
```

Taking a closer look to the code we see that the module “ios_config” is used to configure OSPF, this module is used to configure Cisco IOS devices in configuration mode. This configuration can range from configuring the banner of a login to setting up SNMP.
As the name parents state, it will run the line(s) of code after getting into the router ospf 1 configuration mode. Thus, any lines that are added to “lines:” will be run in it. This could also be useful when setting up loopback interfaces or configuring physical interfaces.

9.2 Using host variables

In chapter 3 we talked about the host_vars and where they could be located, in this chapter we are working with the following directory structure:

```
Ansible
|host_vars
| | Switch1.yml
| |Switch2.yml
|Example.yml
```

Ansible will automatically look up a host file, whether it is in the working directory or in /etc/ansible/…, based on the names/IP addresses specified in the host file in/etc/ansible/host. With these variables we could easily create an ansible playbook that could serve as template as any data in it will be retrieved from the host vars file.

9.2.1 Configuration contents

Let's look to the following files and their configuration.

> **Note**
> Switch1.yml and Switch2.yml are almost the same config, except a different IP address.

Example.yml:

```
---
- name: Create loopback
  hosts: Switches
  tasks:
    - name: Set Loopbacks.
      with_items: "{{ local_loopback }}"
      ios_config:
        lines:
          - description {{ item.desc }}
          - ip address {{ item.ip_address }}
        parents: interface {{ item.name }}

    - name: save running to startup when modified
      ios_config:
        save_when: modified
```

Switch1.yml:

```
---
local_loopback:
  - name: Loopback1
    desc: 'Sample config'
    ip_address: 192.168.2.3 255.255.255.0
  - name: Loopback2
    desc: 'Sample config'
    ip_address: 192.168.3.3 255.255.255.0
```

The name "Switches" is specified, in the host file of Ansible I specified the group "Switches" contains "Switch1" and "Switch2", this will result that the playbook will be run on every device added to the group "Switches". The default behaviour for Ansible is to look for <ip_address>.yaml in any of the known hosts_vars locations. This could be either the current work directory or /etc/ansible/... . But in our case where the hosts are given a hostname in addition to their IP address Ansible will look for Switch1.yml and Switch2.yml. From here on we will call "Example.yml" the playbook, as it is an Ansible playbook.

In Switch1.yml I specified a variable local_loopback with two entries, named Loopback1 and Loopback2 both with their own IP address and description. Keep in mind that; name, description and ip_address are nothing more than names for variables.

In the playbook we want to configure an interface based on details specified in the host_var file “Switch1.yml” and “Switch2.yml”. Taking a closer look again we see that the playbook does not contain any predefined details anymore, as told before this playbook will be used as template to configure both switches.

As explained before, Ansible will automatically look for any host file with the same name as the items in the “hosts” file, thus there is no need to specify any location or file that needs to be used. To make things easier I will only mention Switch1 from now on, but keep in mind that with the specified group for the hosts in the playbook, it will run on both Switch1 and Switch2.

9.2.2 With items

We also added the line: “with_items:”, what this does is, ansible will look for an entry in the host_vars file of Switch1 with the name local_loopback resulting in Ansible knowing and able to provide data to where we need it later on in the task of configuring the loopback interfaces. “local_loopback” is also here nothing more than a variable name for data in it.

With the line “with_items” now defined we are able to get data from the file through calling {{ item.<variable> }}, in the line “parents” we want to get the name of the loopback addresses thus we need to call the variable {{ item.name }}, this will result in Ansible setting “interface loopback1” as parent for the configuration that will follow. As explained above, the variables in “lines” will result in: 'Sample config' and 192.168.2.3 255.255.255.0.

Remember that we talked before about how Ansible will run this playbook on both switches? The same logic could be applied to what happens next, in “Switch1.yml” we defined two names. This will result in ansible running the configuration for the parents: “interface Loopback1” and “interface Loopback2” with each their defined description and IP address.

> **Note**
> You may have noticed that the line “parents” is now underneath the line of “lines”, Ansible will read the task first and run the code in the order defined in the “ios_module” thus looking at “parents” first before running the code in “lines”.

> **Note**
> The quotes around the variables are only needed when the line the variable call is in is not a string, thus “with_items” will require quotes as Ansible needs to look for a variable and not a directory.

9.3 Cisco IOS configuration

This module may be one of the most important modules in Ansible for Cisco IOS, as you are able to change anything with it from a banner to configuring a VPN. But there are also a few more configuration options in the module to help you configure a Cisco IOS device, and I will provide some information of some of them.

9.3.1 Before & after

Either command could be used in any task to run a command, as the name implies, before or after a task. For example, we want to set some access list rules but we want to remove the old configuration first. Thus:

```
- name: load new acl into device
  ios_config:
    lines:
      - 10 permit ip host 192.0.2.1 any log
    parents: ip access-list extended test
    before: no ip access-list extended test
    match: exact
```

This will result in the list test being deleted before the list will be created again and configured. The command after will run after the task has configured the things specified.

9.3.2 Replace

The replace command could be quite useful when you want to be sure you want to replace just one line or a whole block. For example, we want to make sure an access list is exact the same as provided thus adding the line “replace” to the task.

```
- name: load new acl into device
  ios_config:
    lines:
      - 10 permit ip host 192.0.2.1 any log
    parents: ip access-list extended test
    replace: block
```

9.3.3 Diff against

Another useful module is diff_against, this module allows you to compare the intended config to the running config. Keep in mind when running this module in a playbook the argument --diff need to be passed with the command to run the playbook to show the results.

```
- name: check the running-config against master config
  ios_config:
    diff_against: intended
    intended_config: "{{ lookup('file', 'master.cfg') }}"
```

With the configuration as is, Ansible will look for a file named “master.cfg” and will compare the running config against the master config, and with the --diff argument passed with the command it will return the results of what is different between the two files.

9.3.4 Save

Another feature I would like to mention is the save function of “ios_config”. With this feature we could tell the device to save its configuration based on what happened, whether we want it to happen always, never, modified or when changed. The first two explain themselves, when defined in a playbook it will always or never save the configuration. But the other two may be a bit confusing.

When the flag is set to modified it will only save the running config when it has changed since the last save, while with the flag set to changed it will save when the playbook made a change to the configuration.