How Network Automation is Different

Network automation uses the basic Ansible concepts, but there are important differences in how the network modules work. This introduction prepares you to understand the exercises in this guide.

Execution on the control node

Unlike most Ansible modules, network modules do not run on the managed nodes. From a user's point of view, network modules work like any other modules. They work with ad hoc commands, playbooks, and roles. Behind the scenes, however, network modules use a different methodology than the other (Linux/Unix and Windows) modules use. Ansible is written and executed in Python. Because the majority of network devices can not run Python, the Ansible network modules are executed on the Ansible control node, where `ansible` or `ansible-playbook` runs.

Network modules also use the control node as a destination for backup files, for those modules that offer a `backup` option. With Linux/Unix modules, where a configuration file already exists on the managed node(s), the backup file gets written by default in the same directory as the new, changed file. Network modules do not update configuration files on the managed nodes, because network configuration is not written in files. Network modules write backup files on the control node, usually in the *backup* directory under the playbook root directory.

Multiple communication protocols

Because network modules execute on the control node instead of on the managed nodes, they can support multiple communication protocols. The communication protocol (XML over SSH, CLI over SSH, API over HTTPS) selected for each network module depends on the platform and the purpose of the module. Some network modules support only one protocol; some offer a choice. The most common protocol is CLI over SSH. You set the communication protocol with the `ansible_connection` variable:

Value of ansible_connection	Protocol	Requires	Persistent?
ansible.netcommon.network_cli	CLI over SSH	network_os setting	yes
ansible.netcommon.netconf	XML over SSH	network_os setting	yes
ansible.netcommon.httpapi	API over HTTP/HTTPS	network_os setting	yes
local	depends on provider	provider setting	no

Note

`ansible.netcommon.httpapi` deprecates `eos_eapi` and `nxos_nxapi`. See :ref:`httpapi_plugins` for details and an example.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\getting_started\[ansible-devel][docs][docsite][rst][network][getting_started]network_differences.rst`, line 32); *backlink*

Unknown interpreted text role "ref".

The `ansible_connection: local` has been deprecated. Please use one of the persistent connection types listed above instead. With persistent connections, you can define the hosts and credentials only once, rather than in every task. You also need to set the `network_os` variable for the specific network platform you are communicating with. For more details on using each connection type on various platforms, see the :ref:`platform-specific <platform_options>` pages.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\getting_started\[ansible-devel][docs][docsite][rst][network][getting_started]network_differences.rst`, line 34); *backlink*

Unknown interpreted text role "ref".

Collections organized by network platform

A network platform is a set of network devices with a common operating system that can be managed by an Ansible collection, for example:

- Arista: arista.eos
- Cisco: cisco.ios, cisco.iosxr, cisco.nxos
- Juniper: junipernetworks.junos
- VyOS vyos.vyos

All modules within a network platform share certain requirements. Some network platforms have specific differences - see the :ref:`platform-specific <platform_options>` documentation for details.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\getting_started\[ansible-devel][docs][docsite][rst][network][getting_started]network_differences.rst, line 47); *backlink***
>
> Unknown interpreted text role "ref".

Privilege Escalation: `enable` mode, `become`, and `authorize`

Several network platforms support privilege escalation, where certain tasks must be done by a privileged user. On network devices this is called the `enable` mode (the equivalent of `sudo` in *nix administration). Ansible network modules offer privilege escalation for those network devices that support it. For details of which platforms support `enable` mode, with examples of how to use it, see the :ref:`platform-specific <platform_options>` documentation.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\getting_started\[ansible-devel][docs][docsite][rst][network][getting_started]network_differences.rst, line 54); *backlink***
>
> Unknown interpreted text role "ref".

Using `become` for privilege escalation

Use the top-level Ansible parameter `become: yes` with `become_method: enable` to run a task, play, or playbook with escalated privileges on any network platform that supports privilege escalation. You must use either `connection: network_cli` or `connection: httpapi` with `become: yes` with `become_method: enable`. If you are using `network_cli` to connect Ansible to your network devices, a `group_vars` file would look like:

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: cisco.ios.ios
ansible_become: yes
ansible_become_method: enable
```

For more information, see :ref:`Become and Networks<become_network>`

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\getting_started\[ansible-devel][docs][docsite][rst][network][getting_started]network_differences.rst, line 68); *backlink***
>
> Unknown interpreted text role "ref".