Ansible Network FAQ

Topics

How can I improve performance for network playbooks?

Consider `strategy: free` if you are running on multiple hosts

The `strategy` plugin tells Ansible how to order multiple tasks on multiple hosts. :ref:`Strategy<strategy_plugins>` is set at the playbook level.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\user_guide\(ansible-devel)(docs)(docsite)(rst)(network)(user_guide)faq.rst, line 19); *backlink***
>
> Unknown interpreted text role "ref".

The default strategy is `linear`. With strategy set to `linear`, Ansible waits until the current task has run on all hosts before starting the next task on any host. Ansible may have forks free, but will not use them until all hosts have completed the current task. If each task in your playbook must succeed on all hosts before you run the next task, use the `linear` strategy.

Using the `free` strategy, Ansible uses available forks to execute tasks on each host as quickly as possible. Even if an earlier task is still running on one host, Ansible executes later tasks on other hosts. The `free` strategy uses available forks more efficiently. If your playbook stalls on each task, waiting for one slow host, consider using `strategy: free` to boost overall performance.

Execute `show running` only if you absolutely must

The `show running` command is the most resource-intensive command to execute on a network device, because of the way queries are handled by the network OS. Using the command in your Ansible playbook will slow performance significantly, especially on large devices; repeating it will multiply the performance hit. If you have a playbook that checks the running config, then executes changes, then checks the running config again, you should expect that playbook to be very slow.

Use `ProxyCommand` only if you absolutely must

Network modules support the use of a :ref:`proxy or jump host<network_delegate_to_vs_ProxyCommand>` with the `ProxyCommand` parameter. However, when you use a jump host, Ansible must open a new SSH connection for every task, even if you are using a persistent connection type (`network_cli` or `netconf`). To maximize the performance benefits of the persistent connection types introduced in version 2.5, avoid using jump hosts whenever possible.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\user_guide\(ansible-devel)(docs)(docsite)(rst)(network)(user_guide)faq.rst, line 37); *backlink***
>
> Unknown interpreted text role "ref".

Set `--forks` to match your needs

Every time Ansible runs a task, it forks its own process. The `--forks` parameter defines the number of concurrent tasks - if you retain the default setting, which is `--forks=5`, and you are running a playbook on 10 hosts, five of those hosts will have to wait until a fork is available. Of course, the more forks you allow, the more memory and processing power Ansible will use. Since most network tasks are run on the control host, this means your laptop can quickly become cpu- or memory-bound.

Why is my output sometimes replaced with ********?

Ansible replaces any string marked `no_log`, including passwords, with `********` in Ansible output. This is done by design, to protect your sensitive data. Most users are happy to have their passwords redacted. However, Ansible replaces every string that matches your password with `********`. If you use a common word for your password, this can be a problem. For example, if you

choose `Admin` as your password, Ansible will replace every instance of the word `Admin` with `********` in your output. This may make your output harder to read. To avoid this problem, select a secure password that will not occur elsewhere in your Ansible output.

Why do the `*_config` modules always return `changed=true` with abbreviated commands?

When you issue commands directly on a network device, you can use abbreviated commands. For example, `int g1/0/11` and `interface GigabitEthernet1/0/11` do the same thing; `shut` and `shutdown` do the same thing. Ansible Network `*_command` modules work with abbreviations, because they run commands through the network OS.

When committing configuration, however, the network OS converts abbreviations into long-form commands. Whether you use `shut` or `shutdown` on `GigabitEthernet1/0/11`, the result in the configuration is the same: `shutdown`.

Ansible Network `*_config` modules compare the text of the commands you specify in `lines` to the text in the configuration. If you use `shut` in the `lines` section of your task, and the configuration reads `shutdown`, the module returns `changed=true` even though the configuration is already correct. Your task will update the configuration every time it runs.

To avoid this problem, use long-form commands with the `*_config` modules:

```
---
- hosts: all
  gather_facts: no
  tasks:
    - cisco.ios.ios_config:
        lines:
          - shutdown
        parents: interface GigabitEthernet1/0/11
```