ANSIBLE

Google™ Custom Search

# The Ansible Configuration File

**Topics**

- command_warnings
- connection_plugins
- deprecation_warnings
- display_skipped_hosts
- error_on_undefined_vars
- executable
- filter_plugins
- force_color
- force_handlers
- forks
- gathering
- hash_behaviour
- hostfile
- host_key_checking
- inventory
- jinja2_extensions
- library
- log_path
- lookup_plugins
- module_lang
- module_name
- nocolor
- nocows
- pattern
- poll_interval
- private_key_file
- remote_port
- remote_tmp
- remote_user
- roles_path
- sudo_exe

Certain settings in Ansible are adjustable via a configuration file. The stock configuration should be sufficient for most users, but there may be reasons you would want to change them.

Changes can be made and used in a configuration file which will be processed in the following order:

```
* ANSIBLE_CONFIG (an environment variable)
* ansible.cfg (in the current directory)
* .ansible.cfg (in the home directory)
* /etc/ansible/ansible.cfg
```

Prior to 1.5 the order was:

```
* ansible.cfg (in the current directory)
* ANSIBLE_CONFIG (an environment variable)
* .ansible.cfg (in the home directory)
* /etc/ansible/ansible.cfg
```

Ansible will process the above list and use the first file found. Settings in files are not merged.

## Getting the latest configuration

If installing ansible from a package manager, the latest ansible.cfg should be present in /etc/ansible, possibly as a ".rpmnew" file (or other) as appropriate in the case of updates.

If you have installed from pip or from source, however, you may want to create this file in order to override default settings in Ansible.

You may wish to consult the ansible.cfg in source control for all of the possible latest values.

## Environmental configuration

Ansible also allows configuration of settings via environment variables. If these environment variables are set, they will override any setting loaded from the configuration file. These variables are for brevity not defined here, but look in ‘constants.py’ in the source tree if you want to use these. They are mostly considered to be a legacy system as compared to the config file, but are equally valid.

## Explanation of values by section

The configuration file is broken up into sections. Most options are in the “general” section but some sections of the file are specific to certain connection types.

### General defaults

In the [defaults] section of ansible.cfg, the following settings are tunable:

#### action_plugins

Actions are pieces of code in ansible that enable things like module execution, templating, and so forth.

This is a developer-centric feature that allows low-level extensions around Ansible to be loaded from different locations:

```
action_plugins = ~/.ansible/plugins/action_plugins/:/usr/share/ansible_plugins/action_plugins
```

Most users will not need to use this feature. See *Developing Plugins* for more details.

#### ansible_managed

Ansible-managed is a string that can be inserted into files written by Ansible’s config templating system, if you use a string like:

```
{{ ansible_managed }}
```

The default configuration shows who modified a file and when:

```
ansible_managed = Ansible managed: {file} modified on %Y-%m-%d %H:%M:%S by {uid} on {host}
```

This is useful to tell users that a file has been placed by Ansible and manual changes are likely to be overwritten.

Note that if using this feature, and there is a date in the string, the template will be reported changed each time as the date is updated.

### ask_pass

This controls whether an Ansible playbook should prompt for a password by default. The default behavior is no:

```
ask_pass=True
```

If using SSH keys for authentication, it’s probably not needed to change this setting.

### ask_sudo_pass

Similar to ask_pass, this controls whether an Ansible playbook should prompt for a sudo password by default when sudoing. The default behavior is also no:

```
ask_sudo_pass=True
```

Users on platforms where sudo passwords are enabled should consider changing this setting.

### bin_ansible_callbacks

New in version 1.8.

Controls whether callback plugins are loaded when running /usr/bin/ansible. This may be used to log activity from the command line, send notifications, and so on. Callback plugins are always loaded for /usr/bin/ansible-playbook if present and cannot be disabled:

```
bin_ansible_callbacks=False
```

Prior to 1.8, callbacks were never loaded for /usr/bin/ansible.

### callback_plugins

Callbacks are pieces of code in ansible that get called on specific events, permitting to trigger notifications.

This is a developer-centric feature that allows low-level extensions around Ansible to be loaded from different locations:

```
callback_plugins = ~/.ansible/plugins/callback_plugins/:/usr/share/ansible_plugins/callback_plugins
```

Most users will not need to use this feature. See *Developing Plugins* for more details

### command_warnings

New in version 1.8.

By default since Ansible 1.8, Ansible will warn when usage of the shell and command module appear to be simplified by using a default Ansible module instead. This can include reminders to use the ‘git’ module instead of shell commands to execute ‘git’. Using modules when possible over arbitrary shell commands can lead to more reliable and consistent playbook runs, and also easier to maintain playbooks:

```
command_warnings = False
```

These warnings can be silenced by adjusting the following setting or adding warn=yes or warn=no to the end of the command line parameter string, like so:

```
- name: usage of git that could be replaced with the git module
  shell: git update foo warn=yes
```

### connection_plugins

Connections plugin permit to extend the channel used by ansible to transport commands and files.

This is a developer-centric feature that allows low-level extensions around Ansible to be loaded from different locations:

```
connection_plugins = ~/.ansible/plugins/connection_plugins/:/usr/share/ansible_plugins/connection_plugins
```

Most users will not need to use this feature. See *Developing Plugins* for more details

## deprecation_warnings

New in version 1.3.

Allows disabling of deprecating warnings in ansible-playbook output:

```
deprecation_warnings = True
```

Deprecation warnings indicate usage of legacy features that are slated for removal in a future release of Ansible.

## display_skipped_hosts

If set to *False*, ansible will not display any status for a task that is skipped. The default behavior is to display skipped tasks:

```
display_skipped_hosts=True
```

Note that Ansible will always show the task header for any task, regardless of whether or not the task is skipped.

## error_on_undefined_vars

On by default since Ansible 1.3, this causes ansible to fail steps that reference variable names that are likely typoed:

```
error_on_undefined_vars=True
```

If set to False, any '{{ template_expression }}' that contains undefined variables will be rendered in a template or ansible action line exactly as written.

### executable

This indicates the command to use to spawn a shell under a sudo environment. Users may need to change this in rare instances to /bin/bash in rare instances when sudo is constrained, but in most cases it may be left as is:

```
executable = /bin/bash
```

### filter_plugins

Filters are specific functions that can be used to extend the template system.

This is a developer-centric feature that allows low-level extensions around Ansible to be loaded from different locations:

```
filter_plugins = ~/.ansible/plugins/filter_plugins/:/usr/share/ansible_plugins/filter_plugins
```

Most users will not need to use this feature. See *Developing Plugins* for more details

### force_color

This options forces color mode even when running without a TTY:

```
force_color = 1
```

### force_handlers

New in version 1.9.1.

This option causes notified handlers to run on a host even if a failure occurs on that host:

```
force_handlers = True
```

The default is False, meaning that handlers will not run if a failure has occurred on a host. This can also be set per play or on the command line. See `_handlers_and_failure` for more details.

### forks

This is the default number of parallel processes to spawn when communicating with remote hosts. Since Ansible 1.3, the fork number is automatically limited to the number of possible hosts, so this is really a limit of how much network and CPU load you think you can handle. Many users may set this to 50, some set it to 500 or more. If you have a large number of hosts, higher values will make actions across all of those hosts complete faster. The default is very very conservative:

```
forks=5
```

## gathering

New in 1.6, the ‘gathering’ setting controls the default policy of facts gathering (variables discovered about remote systems).

The value ‘implicit’ is the default, meaning facts will be gathered per play unless ‘gather_facts: False’ is set in the play. The value ‘explicit’ is the inverse, facts will not be gathered unless directly requested in the play.

The value ‘smart’ means each new host that has no facts discovered will be scanned, but if the same host is addressed in multiple plays it will not be contacted again in the playbook run. This option can be useful for those wishing to save fact gathering time.

## hash_behaviour

Ansible by default will override variables in specific precedence orders, as described in *Variables*. When a variable of higher precedence wins, it will replace the other value.

Some users prefer that variables that are hashes (aka ‘dictionaries’ in Python terms) are merged. This setting is called ‘merge’. This is not the default behavior and it does not affect variables whose values are scalars (integers, strings) or arrays. We generally recommend not using this setting unless you think you have an absolute need for it, and playbooks in the official examples repos do not use this setting:

```
hash_behaviour=replace
```

The valid values are either ‘replace’ (the default) or ‘merge’.

### hostfile

This is a deprecated setting since 1.9, please look at Inventory for the new setting.

### host_key_checking

As described in *Getting Started*, host key checking is on by default in Ansible 1.3 and later. If you understand the implications and wish to disable it, you may do so here by setting the value to False:

```
host_key_checking=True
```

### inventory

This is the default location of the inventory file, script, or directory that Ansible will use to determine what hosts it has available to talk to:

```
inventory = /etc/ansible/hosts
```

It used to be called hostfile in Ansible before 1.9

### jinja2_extensions

This is a developer-specific feature that allows enabling additional Jinja2 extensions:

```
jinja2_extensions = jinja2.ext.do,jinja2.ext.i18n
```

If you do not know what these do, you probably don’t need to change this setting :)

## library

This is the default location Ansible looks to find modules:

```
library = /usr/share/ansible
```

Ansible knows how to look in multiple locations if you feed it a colon separated path, and it also will look for modules in the ”./library” directory alongside a playbook.

## log_path

If present and configured in ansible.cfg, Ansible will log information about executions at the designated location. Be sure the user running Ansible has permissions on the logfile:

```
log_path=/var/log/ansible.log
```

This behavior is not on by default. Note that ansible will, without this setting, record module arguments called to the syslog of managed machines. Password arguments are excluded.

For Enterprise users seeking more detailed logging history, you may be interested in *Ansible Tower*.

### lookup_plugins

This is a developer-centric feature that allows low-level extensions around Ansible to be loaded from different locations:

```
lookup_plugins = ~/.ansible/plugins/lookup_plugins/:/usr/share/ansible_plugins/lookup_plugins
```

Most users will not need to use this feature. See *Developing Plugins* for more details

### module_lang

This is to set the default language to communicate between the module and the system. By default, the value is ‘C’.

### module_name

This is the default module name (-m) value for /usr/bin/ansible. The default is the ‘command’ module. Remember the command module doesn’t support shell variables, pipes, or quotes, so you might wish to change it to ‘shell’:

```
module_name = command
```

### nocolor

By default ansible will try to colorize output to give a better indication of failure and status information. If you dislike this behavior you can turn it off by setting ‘nocolor’ to 1:

```
nocolor=0
```

### nocows

By default ansible will take advantage of cowsay if installed to make /usr/bin/ansible-playbook runs more exciting. Why? We believe systems management should be a happy experience. If you do not like the cows, you can disable them by setting ‘nocows’ to 1:

```
nocows=0
```

### pattern

This is the default group of hosts to talk to in a playbook if no “hosts:” stanza is supplied. The default is to talk to all hosts. You may wish to change this to protect yourself from surprises:

```
hosts=*
```

Note that /usr/bin/ansible always requires a host pattern and does not use this setting, only /usr/bin/ansible-playbook.

### poll_interval

For asynchronous tasks in Ansible (covered in *Asynchronous Actions and Polling*), this is how often to check back on the status of those tasks when an explicit poll interval is not supplied. The default is a reasonably moderate 15 seconds which is a tradeoff between checking in frequently and providing a quick turnaround when something may have completed:

```
poll_interval=15
```

## private_key_file

If you are using a pem file to authenticate with machines rather than SSH agent or passwords, you can set the default value here to avoid re-specifying `--private-key` with every invocation:

```
private_key_file=/path/to/file.pem
```

## remote_port

This sets the default SSH port on all of your systems, for systems that didn’t specify an alternative value in inventory. The default is the standard 22:

```
remote_port = 22
```

## remote_tmp

Ansible works by transferring modules to your remote machines, running them, and then cleaning up after itself. In some cases, you may not wish to use the default location and would like to change the path. You can do so by altering this setting:

```
remote_tmp = $HOME/.ansible/tmp
```

The default is to use a subdirectory of the user’s home directory. Ansible will then choose a random directory name inside this location.

## remote_user

This is the default username ansible will connect as for /usr/bin/ansible-playbook. Note that /usr/bin/ansible will always default to the current user if this is not defined:

```
remote_user = root
```

## roles_path

The roles path indicate additional directories beyond the ‘roles/’ subdirectory of a playbook project to search to find Ansible roles. For instance, if there was a source control repository of common roles and a different repository of playbooks, you might choose to establish a convention to checkout roles in /opt/mysite/roles like so:

```
roles_path = /opt/mysite/roles
```

Additional paths can be provided separated by colon characters, in the same way as other pathstrings:

```
roles_path = /opt/mysite/roles:/opt/othersite/roles
```

Roles will be first searched for in the playbook directory. Should a role not be found, it will indicate all the possible paths that were searched.

### sudo_exe

If using an alternative sudo implementation on remote machines, the path to sudo can be replaced here provided the sudo implementation is matching CLI flags with the standard sudo:

```
sudo_exe=sudo
```

### sudo_flags

Additional flags to pass to sudo when engaging sudo support. The default is ‘-H’ which preserves the environment of the original user. In some situations you may wish to add or remove flags, but in general most users will not need to change this setting:

```
sudo_flags=-H
```

### sudo_user

This is the default user to sudo to if `--sudo-user` is not specified or ‘sudo_user’ is not specified in an Ansible playbook. The default is the most logical: ‘root’:

```
sudo_user=root
```

### system_warnings

New in version 1.6.

Allows disabling of warnings related to potential issues on the system running ansible itself (not on the managed hosts):

```
system_warnings = True
```

These may include warnings about 3rd party packages or other conditions that should be resolved if possible.

### timeout

This is the default SSH timeout to use on connection attempts:

```
timeout = 10
```

### transport

This is the default transport to use if “-c <transport_name>” is not specified to /usr/bin/ansible or /usr/bin/ansible-playbook. The default is ‘smart’, which will use ‘ssh’ (OpenSSH based) if the local operating system is new enough to support ControlPersist technology, and then will otherwise use ‘paramiko’. Other transport options include ‘local’, ‘chroot’, ‘jail’, and so on.

Users should usually leave this setting as ‘smart’ and let their playbooks choose an alternate setting when needed with the ‘connection:’ play parameter.

### vars_plugins

This is a developer-centric feature that allows low-level extensions around Ansible to be loaded from different locations:

```
vars_plugins = ~/.ansible/plugins/vars_plugins/:/usr/share/ansible_plugins/vars_plugins
```

Most users will not need to use this feature. See *Developing Plugins* for more details

### vault_password_file

New in version 1.7.

Configures the path to the Vault password file as an alternative to specifying `--vault-password-file` on the command line:

```
vault_password_file = /path/to/vault_password_file
```

As of 1.7 this file can also be a script. If you are using a script instead of a flat file, ensure that it is marked as executable, and that the password is printed to standard output. If your script needs to prompt for data, prompts can be sent to standard error.

## Paramiko Specific Settings

Paramiko is the default SSH connection implementation on Enterprise Linux 6 or earlier, and is not used by default on other platforms. Settings live under the [paramiko] header.

### record_host_keys

The default setting of yes will record newly discovered and approved (if host key checking is enabled) hosts in the user's hostfile. This setting may be inefficient for large numbers of hosts, and in those situations, using the ssh transport is definitely recommended instead. Setting it to False will improve performance and is recommended when host key checking is disabled:

```
record_host_keys=True
```

## OpenSSH Specific Settings

Under the [ssh_connection] header, the following settings are tunable for SSH connections. OpenSSH is the default connection type for Ansible on OSes that are new enough to support ControlPersist. (This means basically all operating systems except Enterprise Linux 6 or earlier).

### ssh_args

If set, this will pass a specific set of options to Ansible rather than Ansible's usual defaults:

```
ssh_args = -o ControlMaster=auto -o ControlPersist=60s
```

In particular, users may wish to raise the ControlPersist time to encourage performance. A value of 30 minutes may be appropriate.

### control_path

This is the location to save ControlPath sockets. This defaults to:

```
control_path=%(directory)s/ansible-ssh-%%h-%%p-%%r
```

On some systems with very long hostnames or very long path names (caused by long user names or deeply nested home directories) this can exceed the character limit on file socket names (108 characters for most platforms). In that case, you may wish to shorten the string to something like the below:

```
control_path = %(directory)s/%%h-%%r
```

Ansible 1.4 and later will instruct users to run with “-vvvv” in situations where it hits this problem and if so it is easy to tell there is too long of a Control Path filename. This may be frequently encountered on EC2.

## scp_if_ssh

Occasionally users may be managing a remote system that doesn’t have SFTP enabled. If set to True, we can cause scp to be used to transfer remote files instead:

```
scp_if_ssh=False
```

There’s really no reason to change this unless problems are encountered, and then there’s also no real drawback to managing the switch. Most environments support SFTP by default and this doesn’t usually need to be changed.

## pipelining

Enabling pipelining reduces the number of SSH operations required to execute a module on the remote server, by executing many ansible modules without actual file transfer. This can result in a very significant performance improvement when enabled, however when using “sudo:” operations you must first disable ‘requiretty’ in /etc/sudoers on all managed hosts.

By default, this option is disabled to preserve compatibility with sudoers configurations that have requiretty (the default on many distros), but is highly recommended if you can enable it, eliminating the need for *Accelerated Mode*:

```
pipelining=False
```

## Accelerated Mode Settings

Under the [accelerate] header, the following settings are tunable for *Accelerated Mode*. Acceleration is a useful performance feature to use if you cannot enable pipelining in your environment, but is probably not needed if you can.

### accelerate_port

New in version 1.3.

This is the port to use for accelerated mode:

```
accelerate_port = 5099
```

### accelerate_timeout

New in version 1.4.

This setting controls the timeout for receiving data from a client. If no data is received during this time, the socket connection will be closed. A keepalive packet is sent back to the controller every 15 seconds, so this timeout should not be set lower than 15 (by default, the timeout is 30 seconds):

```
accelerate_timeout = 30
```

## accelerate_connect_timeout

New in version 1.4.

This setting controls the timeout for the socket connect call, and should be kept relatively low. The connection to the *accelerate_port* will be attempted 3 times before Ansible will fall back to ssh or paramiko (depending on your default connection setting) to try and start the accelerate daemon remotely. The default setting is 1.0 seconds:

```
accelerate_connect_timeout = 1.0
```

Note, this value can be set to less than one second, however it is probably not a good idea to do so unless you’re on a very fast and reliable LAN. If you’re connecting to systems over the internet, it may be necessary to increase this timeout.

## accelerate_daemon_timeout

New in version 1.6.

This setting controls the timeout for the accelerated daemon, as measured in minutes. The default daemon timeout is 30 minutes:

```
accelerate_daemon_timeout = 30
```

Note, prior to 1.6, the timeout was hard-coded from the time of the daemon’s launch. For version 1.6+, the timeout is now based on the last activity to the daemon and is configurable via this option.

### accelerate_multi_key

New in version 1.6.

If enabled, this setting allows multiple private keys to be uploaded to the daemon. Any clients connecting to the daemon must also enable this option:

```
accelerate_multi_key = yes
```

New clients first connect to the target node over SSH to upload the key, which is done via a local socket file, so they must have the same access as the user that launched the daemon originally.

Previous Next