Ansible 2.0 Porting Guide

This section discusses the behavioral changes between Ansible 1.x and Ansible 2.0.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with Ansible Changelog for 2.0 to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at :ref`porting guides <porting_guides>`.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\porting_guides\[ansible-devel][docs][docsite][rst][porting_guides]porting_guide_2.0.rst`, **line 15**); *backlink*
>
> Unknown interpreted text role "ref".

Topics

Playbook

This section discusses any changes you may need to make to your playbooks.

```
# Syntax in 1.9.x
- debug:
    msg: "{{ 'test1_junk 1\\\\3' | regex_replace('(.*)_junk (.*)', '\\\\1 \\\\2') }}"
# Syntax in 2.0.x
- debug:
    msg: "{{ 'test1_junk 1\\3' | regex_replace('(.*)_junk (.*)', '\\1 \\2') }}"

# Output:
"msg": "test1 1\\3"
```

To make an escaped string that will work on all versions you have two options:

```
- debug: msg="{{ 'test1_junk 1\\3' | regex_replace('(.*)_junk (.*)', '\\1 \\2') }}"
```

uses key=value escaping which has not changed. The other option is to check for the ansible version:

```
"{{ (ansible_version|version_compare('2.0', 'ge'))|ternary( 'test1_junk 1\\3' | regex_replace('(.*)_junk (.*)', '\\1 \\2') , 'test1_junk
```

- trailing newline When a string with a trailing newline was specified in the playbook via yaml dict format, the trailing newline was stripped. When specified in key=value format, the trailing newlines were kept. In v2, both methods of specifying the string will keep the trailing newlines. If you relied on the trailing newline being stripped, you can change your playbook using the following as an example:

  ```
  # Syntax in 1.9.x
  vars:
    message: >
      Testing
      some things
  tasks:
  - debug:
      msg: "{{ message }}"

  # Syntax in 2.0.x
  vars:
    old_message: >
      Testing
      some things
    message: "{{ old_message[:-1] }}"
  - debug:
      msg: "{{ message }}"
  # Output
  "msg": "Testing some things"
  ```

- Behavior of templating DOS-type text files changes with Ansible v2.

  A bug in Ansible v1 causes DOS-type text files (using a carriage return and newline) to be templated to Unix-type text files (using only a newline). In Ansible v2 this long-standing bug was finally fixed and DOS-type text files are preserved correctly. This may be confusing when you expect your playbook to not show any differences when migrating to Ansible v2, while in fact you will see every DOS-type file being completely replaced (with what appears to be the exact same content).

- When specifying complex args as a variable, the variable must use the full jinja2 variable syntax (`` `{{var_name}}` ``) - bare variable names there are no longer accepted. In fact, even specifying args with variables has been deprecated, and will not be allowed in future versions:

  ```
  ---
  - hosts: localhost
    connection: local
    gather_facts: false
    vars:
      my_dirs:
        - { path: /tmp/3a, state: directory, mode: 0755 }
        - { path: /tmp/3b, state: directory, mode: 0700 }
    tasks:
      - file:
        args: "{{item}}" # <- args here uses the full variable syntax
        with_items: "{{my_dirs}}"
  ```

- porting task includes
- More dynamic. Corner-case formats that were not supposed to work now do not, as expected.
- variables defined in the yaml dict format https://github.com/ansible/ansible/issues/13324
- templating (variables in playbooks and template lookups) has improved with regard to keeping the original instead of turning everything into a string. If you need the old behavior, quote the value to pass it around as a string.
- Empty variables and variables set to null in yaml are no longer converted to empty strings. They will retain the value of *None*. You can override the *null_representation* setting to an empty string in your config file by setting the :envvar:`ANSIBLE_NULL_REPRESENTATION` environment variable.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\porting_guides\[ansible-devel][docs][docsite][rst][porting_guides]porting_guide_2.0.rst, line 98); *backlink*

Unknown interpreted text role "envvar".

- Extras callbacks must be enabled in ansible.cfg. Copying is no longer necessary but you must enable them in ansible.cfg.
- dnf module has been rewritten. Some minor changes in behavior may be observed.
- win_updates has been rewritten and works as expected now.
- from 2.0.1 onwards, the implicit setup task from gather_facts now correctly inherits everything from play, but this might cause issues for those setting *environment* at the play level and depending on *ansible_env* existing. Previously this was ignored but now might issue an 'Undefined' error.

Deprecated

While all items listed here will show a deprecation warning message, they still work as they did in 1.9.x. Please note that they will be removed in 2.2 (Ansible always waits two major releases to remove a deprecated feature).

- Bare variables in `with_` loops should instead use the `"{{ var }}"` syntax, which helps eliminate ambiguity.
- The ansible-galaxy text format requirements file. Users should use the YAML format for requirements instead.
- Undefined variables within a `with_` loop's list currently do not interrupt the loop, but they do issue a warning; in the future, they will issue an error.
- Using dictionary variables to set all task parameters is unsafe and will be removed in a future version. For example:

```
- hosts: localhost
  gather_facts: no
  vars:
    debug_params:
      msg: "hello there"
  tasks:
    # These are both deprecated:
    - debug: "{{debug_params}}"
    - debug:
      args: "{{debug_params}}"

    # Use this instead:
    - debug:
        msg: "{{debug_params['msg']}}"
```

- Host patterns should use a comma (,) or colon (:) instead of a semicolon (;) to separate hosts/groups in the pattern.
- Ranges specified in host patterns should use the [x:y] syntax, instead of [x-y].
- Playbooks using privilege escalation should always use "become*" options rather than the old su*/sudo* options.
- The "short form" for vars_prompt is no longer supported. For example:

```
vars_prompt:
    variable_name: "Prompt string"
```

- Specifying variables at the top level of a task include statement is no longer supported. For example:

```
- include_tasks: foo.yml
    a: 1
```

Should now be:

```
- include_tasks: foo.yml
  vars:
    a: 1
```

- Setting any_errors_fatal on a task is no longer supported. This should be set at the play level only.
- Bare variables in the *environment* dictionary (for plays/tasks/etc.) are no longer supported. Variables specified there should use the full variable syntax: '{{foo}}'.
- Tags (or any directive) should no longer be specified with other parameters in a task include. Instead, they should be specified as an option on the task. For example:

```
- include_tasks: foo.yml tags=a,b,c
```

  Should be:

```
- include_tasks: foo.yml
  tags: [a, b, c]
```

- The first_available_file option on tasks has been deprecated. Users should use the with_first_found option or lookup ('first_found', â€¦) plugin.

Other caveats

Here are some corner cases encountered when updating. These are mostly caused by the more stringent parser validation and the capture of errors that were previously ignored.

- Bad variable composition:

```
with_items: myvar_{{rest_of_name}}
```

  This worked 'by accident' as the errors were retemplated and ended up resolving the variable, it was never intended as valid syntax and now properly returns an error, use the following instead.:

```
hostvars[inventory_hostname]['myvar_' + rest_of_name]
```

- Misspelled directives:

```
- task: dostuf
  becom: yes
```

  The task always ran without using privilege escalation (for that you need *become*) but was also silently ignored so the play 'ran' even though it should not, now this is a parsing error.

- Duplicate directives:

```
- task: dostuf
  when: True
  when: False
```

  The first *when* was ignored and only the 2nd one was used as the play ran w/o warning it was ignoring one of the directives, now this produces a parsing error.

- Conflating variables and directives:

```
- role: {name=rosy, port=435 }

# in tasks/main.yml
- wait_for: port={{port}}
```

  The *port* variable is reserved as a play/task directive for overriding the connection port, in previous versions this got conflated with a variable named *port* and was usable later in the play, this created issues if a host tried to reconnect or was using a non caching connection. Now it will be correctly identified as a directive and the *port* variable will appear as undefined, this now forces the use of non conflicting names and removes ambiguity when adding settings and variables to a role invocation.

- Bare operations on *with_*:

```
with_items: var1 + var2
```

An issue with the 'bare variable' features, which was supposed only template a single variable without the need of braces ({{)}}, would in some versions of Ansible template full expressions. Now you need to use proper templating and braces for all expressions everywhere except conditionals (*when*):

```
with_items: "{{var1 + var2}}"
```

The bare feature itself is deprecated as an undefined variable is indistinguishable from a string which makes it difficult to display a proper error.

Porting plugins

In ansible-1.9.x, you would generally copy an existing plugin to create a new one. Simply implementing the methods and attributes that the caller of the plugin expected made it a plugin of that type. In ansible-2.0, most plugins are implemented by subclassing a base class for each plugin type. This way the custom plugin does not need to contain methods which are not customized.

Lookup plugins

- lookup plugins ; import version

Connection plugins

- connection plugins

Action plugins

- action plugins

Callback plugins

Although Ansible 2.0 provides a new callback API the old one continues to work for most callback plugins. However, if your callback plugin makes use of :attr:`self.playbook`, :attr:`self.play`, or :attr:`self.task` then you will have to store the values for these yourself as ansible no longer automatically populates the callback with them. Here's a short snippet that shows you how:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\porting_guides\[ansible-devel][docs][docsite][rst][porting_guides]porting_guide_2.0.rst`, line 243); ***backlink***

Unknown interpreted text role "attr".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\porting_guides\[ansible-devel][docs][docsite][rst][porting_guides]porting_guide_2.0.rst`, line 243); ***backlink***

Unknown interpreted text role "attr".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\porting_guides\[ansible-devel][docs][docsite][rst][porting_guides]porting_guide_2.0.rst`, line 243); ***backlink***

Unknown interpreted text role "attr".

```
import os
from ansible.plugins.callback import CallbackBase

class CallbackModule(CallbackBase):
    def __init__(self):
        self.playbook = None
        self.playbook_name = None
        self.play = None
        self.task = None

    def v2_playbook_on_start(self, playbook):
        self.playbook = playbook
        self.playbook_name = os.path.basename(self.playbook._file_name)

    def v2_playbook_on_play_start(self, play):
        self.play = play

    def v2_playbook_on_task_start(self, task, is_conditional):
        self.task = task

    def v2_on_any(self, *args, **kwargs):
        self._display.display('%s: %s: %s' % (self.playbook_name,
        self.play.name, self.task))
```

Connection plugins

- connection plugins

Hybrid plugins

In specific cases you may want a plugin that supports both ansible-1.9.x *and* ansible-2.0. Much like porting plugins from v1 to v2, you need to understand how plugins work in each version and support both requirements.

Since the ansible-2.0 plugin system is more advanced, it is easier to adapt your plugin to provide similar pieces (subclasses, methods) for ansible-1.9.x as ansible-2.0 expects. This way your code will look a lot cleaner.

You may find the following tips useful:

- Check whether the ansible-2.0 class(es) are available and if they are missing (ansible-1.9.x) mimic them with the needed methods (for example, `__init__`)
- When ansible-2.0 python modules are imported, and they fail (ansible-1.9.x), catch the `ImportError` exception and perform the equivalent imports for ansible-1.9.x. With possible translations (for example, importing specific methods).
- Use the existence of these methods as a qualifier to what version of Ansible you are running. So rather than using version checks, you can do capability checks instead. (See examples below)
- Document for each if-then-else case for which specific version each block is needed. This will help others to understand how they have to adapt their plugins, but it will also help you to remove the older ansible-1.9.x support when it is deprecated.
- When doing plugin development, it is very useful to have the `warning()` method during development, but it is also important to emit warnings for deadends (cases that you expect should never be triggered) or corner cases (for example, cases where you expect misconfigurations).
- It helps to look at other plugins in ansible-1.9.x and ansible-2.0 to understand how the API works and what modules, classes and methods are available.

Lookup plugins

As a simple example we are going to make a hybrid `fileglob` lookup plugin.

```
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

import os
import glob

try:
```

```
    # ansible-2.0
    from ansible.plugins.lookup import LookupBase
except ImportError:
    # ansible-1.9.x

    class LookupBase(object):
        def __init__(self, basedir=None, runner=None, **kwargs):
            self.runner = runner
            self.basedir = self.runner.basedir

        def get_basedir(self, variables):
            return self.basedir

try:
    # ansible-1.9.x
    from ansible.utils import (listify_lookup_plugin_terms, path_dwim, warning)
except ImportError:
    # ansible-2.0
    from ansible.utils.display import Display
    warning = Display().warning

class LookupModule(LookupBase):

    # For ansible-1.9.x, we added inject=None as valid argument
    def run(self, terms, inject=None, variables=None, **kwargs):

        # ansible-2.0, but we made this work for ansible-1.9.x too !
        basedir = self.get_basedir(variables)

        # ansible-1.9.x
        if 'listify_lookup_plugin_terms' in globals():
            terms = listify_lookup_plugin_terms(terms, basedir, inject)

        ret = []
        for term in terms:
            term_file = os.path.basename(term)

            # For ansible-1.9.x, we imported path_dwim() from ansible.utils
            if 'path_dwim' in globals():
                # ansible-1.9.x
                dwimmed_path = path_dwim(basedir, os.path.dirname(term))
            else:
                # ansible-2.0
                dwimmed_path = self._loader.path_dwim_relative(basedir, 'files', os.path.dirname(term))

            globbed = glob.glob(os.path.join(dwimmed_path, term_file))
            ret.extend(g for g in globbed if os.path.isfile(g))

        return ret
```

Note

In the above example we did not use the `warning()` method as we had no direct use for it in the final version. However we left this code in so people can use this part during development/porting/use.

Connection plugins

- connection plugins

Action plugins

- action plugins

Callback plugins

- callback plugins

Connection plugins

- connection plugins

Porting custom scripts

Custom scripts that used the `ansible.runner.Runner` API in 1.x have to be ported in 2.x. Please refer to: :ref:`developing_api`

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\porting_guides\[ansible-devel][docs][docsite][rst][porting_guides]porting_guide_2.0.rst`, line 396); *backlink*

Unknown interpreted text role "ref".