Lookup plugins

- Enabling lookup plugins
- Using lookup plugins
- Forcing lookups to return lists: query and wantlist=True
- Plugin list

Lookup plugins are an Ansible-specific extension to the Jinja2 templating language. You can use lookup plugins to access data from outside sources (files, databases, key/value stores, APIs, and other services) within your playbooks. Like all `templating <playbooks_templating>`, lookups execute and are evaluated on the Ansible control machine. Ansible makes the data returned by a lookup plugin available using the standard templating system. You can use lookup plugins to load variables or templates with information from external sources. You can `create custom lookup plugins <developing_lookup_plugins>`.

- Lookups are executed with a working directory relative to the role or play, as opposed to local tasks, which are executed relative to the executed script. - Pass `wantlist=True` to lookups to use in Jinja2 template "for" loops. - By default, lookup return values are marked as unsafe for security reasons. If you trust the outside source for your lookup accesses, pass `allow_unsafe=True` to allow Jinja2 templates to evaluate lookup values.

- Some lookups pass arguments to a shell. When using variables from a remote/untrusted source, use the |quote filter to ensure safe usage.

Enabling lookup plugins

Ansible enables all lookup plugins it can find. You can activate a custom lookup by either dropping it into a `lookup_plugins` directory adjacent to your play, inside the `plugins/lookup/` directory of a collection you have installed, inside a standalone role, or in one of the lookup directory sources configured in `ansible.cfg <ansible_configuration_settings>`.

Using lookup plugins

You can use lookup plugins anywhere you can use templating in Ansible: in a play, in variables file, or a Jinja2 template for the `template <template_module>` module. For more information on using lookup plugins, see `playbooks_lookups`.

```
vars:
  file_contents: "{{ lookup('file', 'path/to/file.txt') }}"
```

Lookups are an integral part of loops. Wherever you see `with_`, the part after the underscore is the name of a lookup. For this reason, lookups are expected to output lists; for example, `with_items` uses the `items <items_lookup>` lookup:

```
tasks:
  - name: count to 3
    debug: msg={{ item }}
    with_items: [1, 2, 3]
```

You can combine lookups with `filters <playbooks_filters>`, `tests <playbooks_tests>` and even each other to do some complex data generation and manipulation. For example:

```
tasks:
  - name: Complicated chained lookups and filters
    debug: msg="find the answer here:\n{{ lookup('url', 'https://google.com/search/?q=' +
item|urlencode)|join(' ') }}"
    with_nested:
      - "{{ lookup('consul_kv', 'bcs/' + lookup('file', '/the/question') + ', host=localhost,
port=2000')|shuffle }}"
      - "{{ lookup('sequence', 'end=42 start=2 step=2')|map('log', 4)|list) }}"
      - ['a', 'c', 'd', 'c']
```

2.6

You can control how errors behave in all lookup plugins by setting `errors` to `ignore`, `warn`, or `strict`. The default setting is `strict`, which causes the task to fail if the lookup returns an error. For example:

To ignore lookup errors:

```
- name: if this file does not exist, I do not care .. file plugin itself warns anyway ...
  debug: msg="{{ lookup('file', '/nosuchfile', errors='ignore') }}"
```

```
[WARNING]: Unable to find '/nosuchfile' in expected paths (use -vvvvv to see paths)

ok: [localhost] => {
    "msg": ""
}
```

To get a warning instead of a failure:

```
- name: if this file does not exist, let me know, but continue
  debug: msg="{{ lookup('file', '/nosuchfile', errors='warn') }}"
```

```
[WARNING]: Unable to find '/nosuchfile' in expected paths (use -vvvvv to see paths)

[WARNING]: An unhandled exception occurred while running the lookup plugin 'file'. Error was a
<class 'ansible.errors.AnsibleError'>, original message: could not locate file in lookup:
/nosuchfile

ok: [localhost] => {
    "msg": ""
}
```

To get a fatal error (the default):

```
- name: if this file does not exist, FAIL (this is the default)
  debug: msg="{{ lookup('file', '/nosuchfile', errors='strict') }}"
```

```
[WARNING]: Unable to find '/nosuchfile' in expected paths (use -vvvvv to see paths)

fatal: [localhost]: FAILED! => {"msg": "An unhandled exception occurred while running the
lookup plugin 'file'. Error was a <class 'ansible.errors.AnsibleError'>, original message:
could not locate file in lookup: /nosuchfile"}
```

Forcing lookups to return lists: `query` and `wantlist=True`

2.5

In Ansible 2.5, a new Jinja2 function called `query` was added for invoking lookup plugins. The difference between `lookup` and `query` is largely that `query` will always return a list. The default behavior of `lookup` is to return a string of comma-separated values. `lookup` can be explicitly configured to return a list using `wantlist=True`.

This feature provides an easier and more consistent interface for interacting with the new `loop` keyword while maintaining backward compatibility with other uses of `lookup`.

The following examples are equivalent:

```
lookup('dict', dict_variable, wantlist=True)


query('dict', dict_variable)
```

As demonstrated above, the behavior of `wantlist=True` is implicit when using `query`.

Additionally, `q` was introduced as a short form of `query`:

```
q('dict', dict_variable)
```

Plugin list

You can use `ansible-doc -t lookup -l` to see the list of available plugins. Use `ansible-doc -t lookup <plugin name>` to see plugin-specific documentation and examples.

`about_playbooks`
An introduction to playbooks

`inventory_plugins`
Ansible inventory plugins

`callback_plugins`
Ansible callback plugins

`filter_plugins`
Jinja2 filter plugins

`test_plugins`
Jinja2 test plugins

`Communication<communication>`
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide

---

□□□ #2
□ □□□ □□□ 2026-10-31 14:41:23 UTC
□ □□□ □□□ 2026-10-31 14:41:24 UTC