Advanced Syntax

The advanced YAML syntax examples on this page give you more control over the data placed in YAML files used by Ansible. You can find additional information about Python-specific YAML in the official PyYAML Documentation.

- Unsafe or raw strings
- YAML anchors and aliases: sharing variable values

Unsafe or raw strings

When handling values returned by lookup plugins, Ansible uses a data type called `unsafe` to block templating. Marking data as unsafe prevents malicious users from abusing Jinja2 templates to execute arbitrary code on target machines. The Ansible implementation ensures that unsafe values are never templated. It is more comprehensive than escaping Jinja2 with `{% raw %} ... {% endraw %}` tags.

You can use the same `unsafe` data type in variables you define, to prevent templating errors and information disclosure. You can mark values supplied by :ref:`vars_prompts<unsafe_prompts>` as unsafe. You can also use `unsafe` in playbooks. The most common use cases include passwords that allow special characters like `{` or `%`, and JSON arguments that look like templates but should not be templated. For example:

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

```
---
mypassword: !unsafe 234%234{435lkj{{lkjsdf
```

In a playbook:

```
---
hosts: all
vars:
    my_unsafe_variable: !unsafe 'unsafe % value'
tasks:
    ...
```

For complex variables such as hashes or arrays, use `!unsafe` on the individual elements:

```
---
my_unsafe_array:
    - !unsafe 'unsafe element'
    - 'safe element'

my_unsafe_hash:
    unsafe_key: !unsafe 'unsafe value'
```

YAML anchors and aliases: sharing variable values

YAML anchors and aliases help you define, maintain, and use shared variable values in a flexible way. You define an anchor with `&`, then refer to it using an alias, denoted with `*`. Here's an example that sets three values with an anchor, uses two of those values with an alias, and overrides the third value:

```
---
...
vars:
    app1:
        jvm: &jvm_opts
            opts: '-Xms1G -Xmx2G'
            port: 1000
            path: /usr/lib/app1
    app2:
        jvm:
            <<: *jvm_opts
            path: /usr/lib/app2
...
```

Here, `app1` and `app2` share the values for `opts` and `port` using the anchor `&jvm_opts` and the alias `*jvm_opts`. The value for `path` is merged by `<<` or merge operator.

Anchors and aliases also let you share complex sets of variable values, including nested variables. If you have one variable value that

includes another variable value, you can define them separately:

```
vars:
  webapp_version: 1.0
  webapp_custom_name: ToDo_App-1.0
```

This is inefficient and, at scale, means more maintenance. To incorporate the version value in the name, you can use an anchor in `app_version` and an alias in `custom_name`:

```
vars:
  webapp:
      version: &my_version 1.0
      custom_name:
          - "ToDo_App"
          - *my_version
```

Now, you can re-use the value of `app_version` within the value of `custom_name` and use the output in a template:

```
---
- name: Using values nested inside dictionary
  hosts: localhost
  vars:
    webapp:
        version: &my_version 1.0
        custom_name:
            - "ToDo_App"
            - *my_version
  tasks:
  - name: Using Anchor value
    ansible.builtin.debug:
        msg: My app is called "{{ webapp.custom_name | join('-') }}".
```

You've anchored the value of `version` with the `&my_version` anchor, and re-used it with the `*my_version` alias. Anchors and aliases let you access nested values inside dictionaries.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_advanced_syntax.rst, line 115)

Unknown directive type "seealso".

```
.. seealso::

   :ref:`playbooks_variables`
       All about variables
   :ref:`complex_data_manipulation`
       Doing complex data manipulation in Ansible
   `User Mailing List <https://groups.google.com/group/ansible-project>`_
       Have a question?  Stop by the google group!
   :ref:`communication_irc`
       How to join Ansible chat channels
```