Data manipulation

In many cases, you need to do some complex operation with your variables, while Ansible is not recommended as a data processing/manipulation tool, you can use the existing Jinja2 templating in conjunction with the many added Ansible filters, lookups and tests to do some very complex transformations.

Let's start with a quick definition of each type of plugin:

- lookups: Mainly used to query 'external data', in Ansible these were the primary part of loops using the `with_<lookup>` construct, but they can be used independently to return data for processing. They normally return a list due to their primary function in loops as mentioned previously. Used with the `lookup` or `query` Jinja2 operators.
- filters: used to change/transform data, used with the `|` Jinja2 operator.
- tests: used to validate data, used with the `is` Jinja2 operator.

Loops and list comprehensions

Most programming languages have loops (`for`, `while`, and so on) and list comprehensions to do transformations on lists including lists of objects. Jinja2 has a few filters that provide this functionality: `map`, `select`, `reject`, `selectattr`, `rejectattr`.

- map: this is a basic for loop that just allows you to change every item in a list, using the 'attribute' keyword you can do the transformation based on attributes of the list elements.
- select/reject: this is a for loop with a condition, that allows you to create a subset of a list that matches (or not) based on the result of the condition.
- selectattr/rejectattr: very similar to the above but it uses a specific attribute of the list elements for the conditional statement.

Use a loop to create exponential backoff for retries/until.

```
- name: retry ping 10 times with exponential backoff delay
  ping:
  retries: 10
  delay: '{{item|int}}'
  loop: '{{ range(1, 10)|map('pow', 2) }}'
```

Extract keys from a dictionary matching elements from a list

The Python equivalent code would be:

```
chains = [1, 2]
for chain in chains:
    for config in chains_config[chain]['configs']:
        print(config['type'])
```

There are several ways to do it in Ansible, this is just one 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)complex_data_manipulation.rst, line 57)

Error in "code-block" directive: unknown option: "emphasize-lines".

```
.. code-block:: YAML+Jinja
 :emphasize-lines: 4
 :caption: Way to extract matching keys from a list of dictionaries

  tasks:
    - name: Show extracted list of keys from a list of dictionaries
      ansible.builtin.debug:
        msg: "{{ chains | map('extract', chains_config) | map(attribute='configs') | flatten | map(attribute='type
      vars:
        chains: [1, 2]
        chains_config:
            1:
                foo: bar
                configs:
                    - type: routed
                      version: 0.1
                    - type: bridged
                      version: 0.2
            2:
                foo: baz
                configs:
                    - type: routed
                      version: 1.0
                    - type: bridged
                      version: 1.1
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)complex_data_manipulation.rst, line 84)

Error in "code-block" directive: unknown option: "caption".

```
.. code-block:: ansible-output
 :caption: Results of debug task, a list with the extracted keys

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)complex_data_manipulation.rst, line 97)

Error in "code-block" directive: unknown option: "caption".

```
.. code-block:: YAML+Jinja
 :caption: Get the unique list of values of a variable that vary per host

    vars:
        unique_value_list: "{{ groups['all'] | map ('extract', hostvars, 'varname') | list | unique}}"
```

Find mount point

In this case, we want to find the mount point for a given path across our machines, since we already collect mount facts, we can use the following:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)complex_data_manipulation.rst, line 111)

Error in "code-block" directive: unknown option: "caption".

```
.. code-block:: YAML+Jinja
 :caption: Use selectattr to filter mounts into list I can then sort and select the last from
 :emphasize-lines: 8

   - hosts: all
     gather_facts: True
     vars:
        path: /var/lib/cache
     tasks:
     - name: The mount point for {{path}}, found using the Ansible mount facts, [-1] is the same as the 'last' fil
       ansible.builtin.debug:
        msg: "{{(ansible_facts.mounts | selectattr('mount', 'in', path) | list | sort(attribute='mount'))[-1]['mou
```

Omit elements from a list

The special `omit` variable ONLY works with module options, but we can still use it in other ways as an identifier to tailor a list of elements:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)complex_data_manipulation.rst, line 132)

Error in "code-block" directive: unknown option: "caption".

```
.. code-block:: YAML+Jinja
 :caption: Inline list filtering when feeding a module option
 :emphasize-lines: 3, 6

    - name: Enable a list of Windows features, by name
      ansible.builtin.set_fact:
        win_feature_list: "{{ namestuff | reject('equalto', omit) | list }}"
      vars:
        namestuff:
          - "{{ (fs_installed_smb_v1 | default(False)) | ternary(omit, 'FS-SMB1') }}"
          - "foo"
          - "bar"
```

Another way is to avoid adding elements to the list in the first place, so you can just use it directly:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)complex_data_manipulation.rst, line 148)

Error in "code-block" directive: unknown option: "caption".

```
.. code-block:: YAML+Jinja
 :caption: Using set_fact in a loop to increment a list conditionally
 :emphasize-lines: 3, 4, 6

    - name: Build unique list with some items conditionally omitted
      ansible.builtin.set_fact:
         namestuff: ' {{ (namestuff | default([])) | union([item]) }}'
      when: item != omit
      loop:
          - "{{ (fs_installed_smb_v1 | default(False)) | ternary(omit, 'FS-SMB1') }}"
          - "foo"
          - "bar"
```

Combine values from same list of dicts

Combining positive and negative filters from examples above, you can get a 'value when it exists' and a 'fallback' when it doesn't.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)complex_data_manipulation.rst, line 169)

Error in "code-block" directive: unknown option: "caption".

```
.. code-block:: YAML+Jinja
 :caption: Use selectattr and rejectattr to get the ansible_host or inventory_hostname as needed

    - hosts: localhost
      tasks:
        - name: Check hosts in inventory that respond to ssh port
          wait_for:
            host: "{{ item }}"
            port: 22
          loop: '{{ has_ah + no_ah }}'
          vars:
            has_ah: '{{ hostvars|dictsort|selectattr("1.ansible_host", "defined")|map(attribute="1.ansible_host")|
            no_ah: '{{ hostvars|dictsort|rejectattr("1.ansible_host", "defined")|map(attribute="0")|list }}'
```

Custom Fileglob Based on a Variable

This example uses Python argument list unpacking to create a custom list of fileglobs based on a variable.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)complex_data_manipulation.rst, line 191)

Error in "code-block" directive: unknown option: "caption".

```
.. code-block:: YAML+Jinja
  :caption: Using fileglob with a list based on a variable.

    - hosts: all
      vars:
        mygroups
          - prod
          - web
      tasks:
        - name: Copy a glob of files based on a list of groups
          copy:
            src: "{{ item }}"
            dest: "/tmp/{{ item }}"
          loop: '{{ q("fileglob", *globlist) }}'
          vars:
            globlist: '{{ mygroups | map("regex_replace", "^(.*)$", "files/\1/*.conf") | list }}'
```

Complex Type transformations

Jinja provides filters for simple data type transformations (`int`, `bool`, and so on), but when you want to transform data structures things are not as easy. You can use loops and list comprehensions as shown above to help, also other filters and lookups can be chained and used to achieve more complex transformations.

Create dictionary from list

In most languages it is easy to create a dictionary (a.k.a. map/associative array/hash and so on) from a list of pairs, in Ansible there are a couple of ways to do it and the best one for you might depend on the source of your data.

These example produces `{"a": "b", "c": "d"}`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)complex_data_manipulation.rst, line 228)

Error in "code-block" directive: unknown option: "caption".

```
.. code-block:: YAML+Jinja
 :caption: Simple list to dict by assuming the list is [key, value , key, value, ...]

  vars:
      single_list: [ 'a', 'b', 'c', 'd' ]
      mydict: "{{ dict(single_list | slice(2)) }}"
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)complex_data_manipulation.rst, line 236)

Error in "code-block" directive: unknown option: "caption".

```
.. code-block:: YAML+Jinja
 :caption: It is simpler when we have a list of pairs:

  vars:
      list_of_pairs: [ ['a', 'b'], ['c', 'd'] ]
      mydict: "{{ dict(list_of_pairs) }}"
```

Both end up being the same thing, with `slice(2)` transforming `single_list` to a `list_of_pairs` generator.

A bit more complex, using `set_fact` and a `loop` to create/update a dictionary with key value pairs from 2 lists:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)complex_data_manipulation.rst, line 249)

Error in "code-block" directive: unknown option: "caption".

```
.. code-block:: YAML+Jinja
 :caption: Using set_fact to create a dictionary from a set of lists
```

```
:emphasize-lines: 3, 4

    - name: Uses 'combine' to update the dictionary and 'zip' to make pairs of both lists
      ansible.builtin.set_fact:
        mydict: "{{ mydict | default({}) | combine({item[0]: item[1]}) }}"
      loop: "{{ (keys | zip(values)) | list }}"
      vars:
        keys:
          - foo
          - var
          - bar
        values:
          - a
          - b
          - c
```

This results in `{"foo": "a", "var": "b", "bar": "c"}`.

You can even combine these simple examples with other filters and lookups to create a dictionary dynamically by matching patterns to variable names:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)complex_data_manipulation.rst, line 272)

Error in "code-block" directive: unknown option: "caption".

```
.. code-block:: YAML+Jinja
 :caption: Using 'vars' to define dictionary from a set of lists without needing a task

    vars:
        xyz_stuff: 1234
        xyz_morestuff: 567
        myvarnames: "{{ q('varnames', '^xyz_') }}"
        mydict: "{{ dict(myvarnames|map('regex_replace', '^xyz_', '')|list | zip(q('vars', *myvarnames))) }}"
```

A quick explanation, since there is a lot to unpack from these two lines:

- The `varnames` lookup returns a list of variables that match "begin with `xyz_`".
- Then feeding the list from the previous step into the `vars` lookup to get the list of values. The `*` is used to 'dereference the list' (a pythonism that works in Jinja), otherwise it would take the list as a single argument.
- Both lists get passed to the `zip` filter to pair them off into a unified list (key, value, key2, value2, ...).
- The dict function then takes this 'list of pairs' to create the dictionary.

An example on how to use facts to find a host's data that meets condition X:

```
vars:
  uptime_of_host_most_recently_rebooted: "{{ansible_play_hosts_all | map('extract', hostvars, 'ansible_uptime_seconds')
```

An example to show a host uptime in days/hours/minutes/seconds (assumes facts where gathered).

```
- name: Show the uptime in days/hours/minutes/seconds
  ansible.builtin.debug:
    msg: Uptime {{ now().replace(microsecond=0) - now().fromtimestamp(now(fmt='%s') | int - ansible_uptime_seconds) }}
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)complex_data_manipulation.rst, line 307)

Unknown directive type "seealso".

```
.. seealso::

   :ref:`playbooks_filters`
       Jinja2 filters included with Ansible
   :ref:`playbooks_tests`
       Jinja2 tests included with Ansible
   `Jinja2 Docs <https://jinja.palletsprojects.com/>`_
      Jinja2 documentation, includes lists for core filters and tests
```