Brandon Williams

Ansible variable precedence

January 7, 2018
Tags: ansible docker

I have been using Ansible professionally now for 15 months and let me first say that I really enjoy using Ansible over the other automation tools I have used in the past. However, with all tools, it is not without it's quirks, particularly, Ansible variables. I would like expand on this and give a concrete example of how variable precedence has caused my current team some pain. All examples below can be found in this repo.

We will start with where variables can be set. When we set a variable via the command line for the run of the playbook, the variable is availble in the playbook. This is demonstrated below and code availble via example 1.

```
ansible-playbook example-01.yml -i hosts -e 'variable_one=set-from-commandline'
```

```
# example-01.yml
---

- hosts: localhost
  connection: local
  tasks:
    - debug:
        var: variable_one
```

In example 2, we will still set the variable via the command line but also set via the playbook. What we learn here is that the command line declaration of the variable wins.

```
ansible-playbook example-02.yml -i hosts -e 'variable_two=set-from-commandline'
```

```
- hosts: localhost
  connection: local
  vars:
    variable_two: 'set-via-vars-in-playbook'
  tasks:
    - debug:
        var: variable_two
```

In example 3, we can see that vars section will be the first (and only) declaration of the variable.

```
ansible-playbook example-03.yml -i hosts
```

```
# example-03.yml
---

- hosts: localhost
  connection: local
  vars:
    variable_three: 'set-via-vars-in-playbook'
  tasks:
    - debug:
        var: variable_three
```

In this fourth example, we will set the variable via the `vars` block and via the `set_fact` module. We will notice that the first encounter of the variable is set to the `vars` block value and when we do the `set_fact` we can reset the value to what we would like. So from this example, we will see that we can set a variable to a new value during a playbook execution if needed. The code for example 4.

```
ansible-playbook example-04.yml -i hosts
```

```
# example-04.yml
---

- hosts: localhost
```

```
    variable_four: 'set-via-vars-in-playbook'
  tasks:
    - debug:
        var: variable_four
    - set_fact:
        variable_four: 'set-via-set-fact'
    - debug:
        var: variable_four
```

Now we will take a look at how and when variables are available from an `include_vars` file. In this example, I show that the variable is not available until after we have processed the `include_vars` command in the playbook. The code for example 5.

```
ansible-playbook example-05.yml -i hosts
```

```
# example-05.yml
---

- hosts: localhost
  connection: local
  tasks:
    - debug:
        var: variable_include_vars_five
    - include_vars: 'variable_include_vars_five.yml'
    - debug:
        var: variable_include_vars_five
```

```
# variable_include_vars_five.yml
---

variable_include_vars_five: 'set-via-include-vars-file'
```

Remember when we encounter the first declaration of a variable in a playbook, it will be set to that until we do a `set_fact` or change it via an `include_vars`. If we revist the `vars` block (or set via the command line)

include_vars. The code for example 6.

```
ansible-playbook example-06.yml -i hosts
```

```
# example-06.yml
---

- hosts: localhost
  connection: local
  vars:
    variable_include_vars_six: 'set-via-vars-in-playbook'
  tasks:
    - debug:
        var: variable_include_vars_six
    - include_vars: 'variable_include_vars_six.yml'
    - debug:
        var: variable_include_vars_six
```

```
# variable_include_vars_six.yml
---

variable_include_vars_six: 'set-via-include-vars-file'
```

If we do multiple `include_vars` and change the value in each, the variable will change after each file is included. The code for example 7.

```
ansible-playbook example-07.yml -i hosts
```

```
# example-07.yml
---

- hosts: localhost
  connection: local
  tasks:
    - debug:
        var: variable_include_vars_seven
    - include_vars: 'first_variable_include_vars_seven.yml'
    - debug:
        var: variable_include_vars_seven
    - include_vars: 'second_variable_include_vars_seven.yml'
```

```
# first_variable_include_vars_seven.yml
---

variable_include_vars_seven: 'set-via-first-include-vars-file'
```

```
# second_variable_include_vars_seven.yml
---

variable_include_vars_seven: 'set-via-second-include-vars-file'
```

In example 8, we will return to setting variables from the command line. In this example, we will set the variable from the command line and from an `include_vars` file. We will notice something interesting in that the variable is always set to the command line value, even after the `include_vars`.

```
ansible-playbook example-08.yml -i hosts -e "variable_eight=set-via-commandline"
```

```
# example-08.yml
---

- hosts: localhost
  connection: local
  tasks:
    - debug:
        var: variable_eight
    - include_vars: 'variable_include_vars_eight.yml'
    - debug:
        var: variable_eight
```

```
# variable_include_vars_eight.yml
---

variable_eight: 'set-via-include-vars-file'
```

In example 9, we will again return to setting variables from the command line. In this example, we will set the variable from the command line and

the variable is always set to the command line value, even after the `set_fact`. Now looking at the Ansible documentation for variable precedence, we will notice (if you scroll down to the 2.x section) that `extra vars` always win precedence (where `extra vars` are command line variables).

```
ansible-playbook example-09.yml -i hosts -e "variable_nine=set-via-commandline"
```

```
# example-09.yml
---

- hosts: localhost
  connection: local
  tasks:
    - debug:
        var: variable_nine
    - set_fact:
        variable_nine: 'set-via-set-fact'
    - debug:
        var: variable_nine
```

In example 10, I would like to show the example that caused us a lot of trouble. If you read the section the documentation, you will notice that the `role defaults` takes the highest precedence, right behind `extra vars`, in both 1.x and 2.x versions of Ansible. Our team had not seen this section of the documentation and had been expecting that the first declaration of a variable was the value of that variable. So, we were very surprised when some of our playbooks started failing for what seemed like odd reasons. You will see that even before the role has been applied, the variables for that role are now in the global scope and available to any `pre_tasks`. We were doing a check to see if a variable had been defined, and since the role defaults was defining the variable it was always true, we would do some certain processing. This behavior differs from what we see above with the `include_vars` where the variable is not available until

behavior for the roles.

```
ansible-playbook example-10.yml -i hosts
```

```
# example-10.yml
---

- hosts: localhost
  connection: local
  pre_tasks:
    - debug:
        msg: 'pre-tasks'
    - debug:
        var: variable_ten
  roles:
    - role: variable_role
  tasks:
    - debug:
        msg: 'tasks'
    - debug:
        var: variable_ten
```

```
# roles/variable_role/defaults/main.yml
---

variable_ten: 'set-via-role-defaults'
```

```
# roles/variable_role/tasks/main.yml
---

- debug:
    msg: 'role'
- debug:
    var: variable_ten
```

In conclusion, I highly recommend reading the documentation for the tools that you use. While the Ansible documentation does warn us of the behavior we encounter. I hope that these examples provide living documentation. This was not an exhaustive list, but I wanted to highlight

Brandon Williams

we had originally anticipated.

See my github profile.

Follow me on Twitter.

Brandon Williams