Ansible Conditionals

Eyes On Cloud

What are Conditionals?

- Conditionals are an important part of Ansible playbooks.
- Conditionals support decision-making and provide a way to take specific actions based on the outcomes of commands or tasks.
- In some cases, the values of variables may depend on other variables.
- There are many options to control execution flow in Ansible.
- These options are known as conditionals.

Types of Conditions

- When Statement
- Loops and Conditionals
- Loading in Custom Facts
- Applying ‘when’ to roles, imports, and includes
- Conditional Imports
- Selecting Files And Templates Based On Variables
- Register Variables

When Statement

- Sometimes you want to skip a particular step on a particular host.
- This could be something as simple as not installing a certain package if the operating system is a particular version, or it could be something like performing some cleanup steps if a filesystem is getting full.
- This is easy to do in Ansible with the when clause, which contains a raw Jinja2 expression without double curly braces.

Example

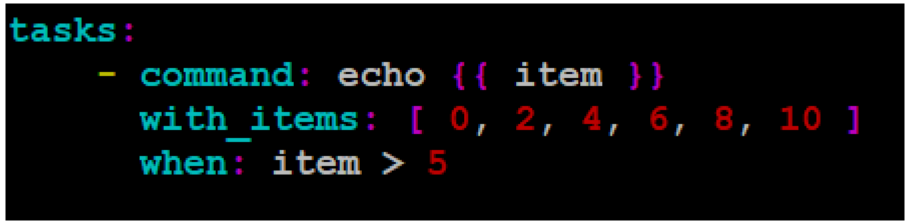
```
tasks:
  - name: "shut down Debain flavored systems"
    command: /sbin/shutdown -t now
    when: ansible_os_family == "Debian"
    # note that Ansible facts and vars like ansible_os_family can be used
    # directly in conditionals without double curly braces
```

- You can also use parentheses to group conditions:

```
tasks:
  - name: "shut down CentOS 6 and Debian 7 systems"
    command: /sbin/shutdown -t now
    when: (ansible_distribution == "CentOS" and ansible_distribution_major_version == "6" )or
          (ansible_distribution == "Debian" and ansible_distribution_major_version == "7")
```

Loops and Conditionals

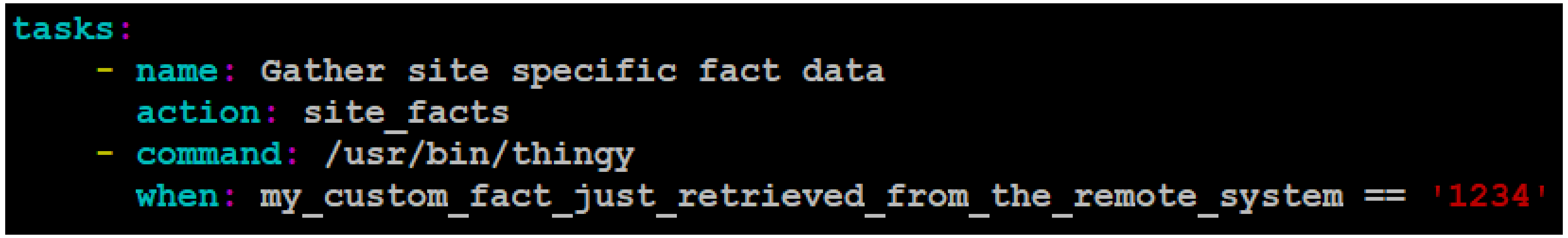
- Combining when with **with_items**, be aware that the when statement is processed separately for each item. This is by design:



```
tasks:
    - command: echo {{ item }}
      with_items: [ 0, 2, 4, 6, 8, 10 ]
      when: item > 5
```

Loading in Custom Facts

- It's also easy to provide your own facts if you want to run them, just make a call to your own custom fact gathering module at the top of your list of tasks, and variables returned there will be accessible to future tasks:



```
tasks:
    - name: Gather site specific fact data
      action: site_facts
    - command: /usr/bin/thingy
      when: my_custom_fact_just_retrieved_from_the_remote_system == '1234'
```

Applying ‘when’ to roles, imports, and includes

- Note that if you have several tasks that all share the same conditional statement, you can affix the conditional to a task include statement as below.
- All the tasks get evaluated, but the conditional is applied to each and every task:

```
- import_tasks: tasks/sometasks.yml
  when: "'reticulating splines' in output"
```

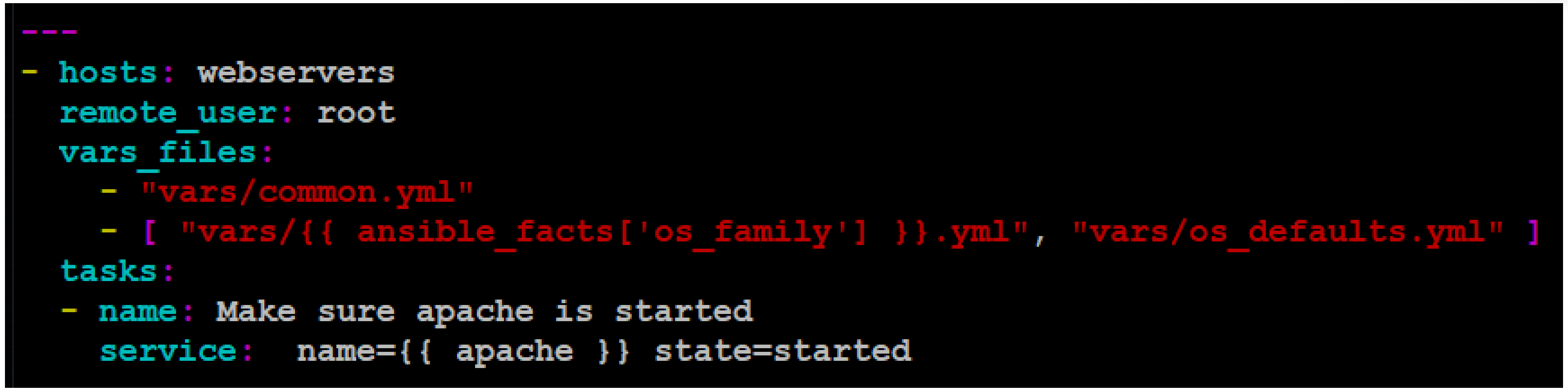
Conditional Imports

- Sometimes you will want to do certain things differently in a playbook based on certain criteria.
- Having one playbook that works on multiple platforms and OS versions is a good example.
- As an example, the name of the Apache package may be different between CentOS and Debian, but it is easily handled with a minimum of syntax in an Ansible Playbook:

Conditional Imports

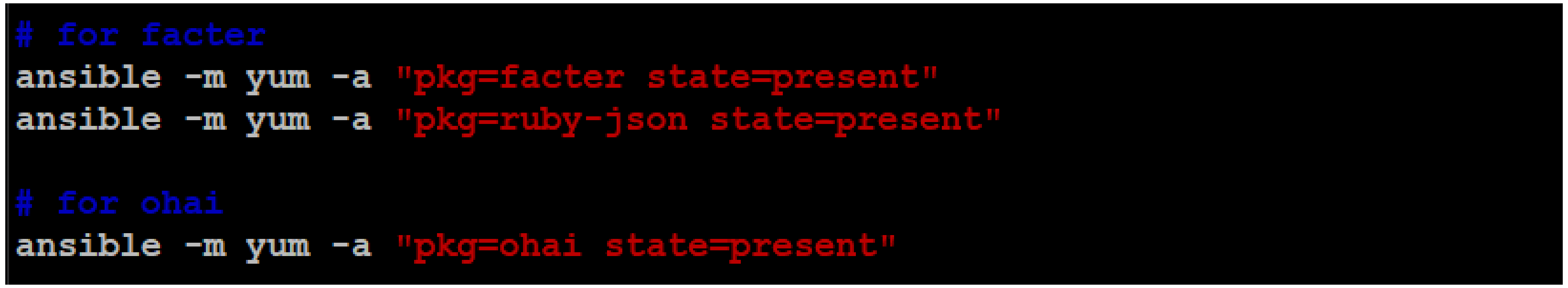
- How does this work? If the operating system was ‘CentOS’, the first file Ansible would try to import would be ‘vars/CentOS.yml’, followed by ‘/vars/os_defaults.yml’ if that file did not exist.
- If no files in the list were found, an error would be raised. On Debian, it would instead first look towards ‘vars/Debian.yml’ instead of ‘vars/CentOS.yml’, before falling back on ‘vars/os_defaults.yml’.

Example



```
---
- hosts: webservers
  remote_user: root
  vars_files:
    - "vars/common.yml"
    - [ "vars/{{ ansible_facts['os_family'] }}.yml", "vars/os_defaults.yml" ]
  tasks:
  - name: Make sure apache is started
    service:  name={{ apache }} state=started
```

- File:



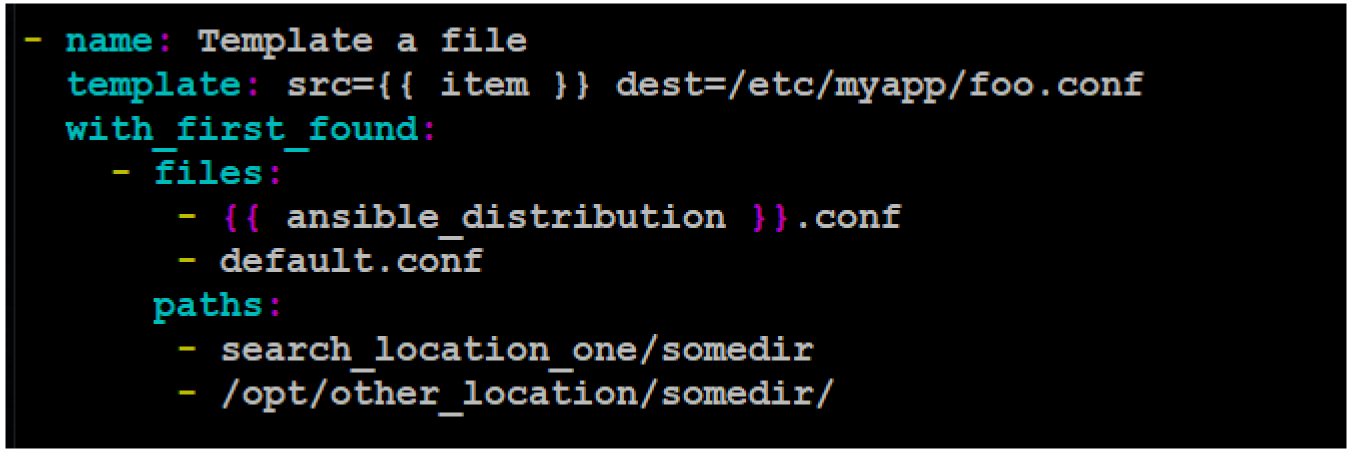
```
# for facter
ansible -m yum -a "pkg=facter state=present"
ansible -m yum -a "pkg=ruby-json state=present"

# for ohai
ansible -m yum -a "pkg=ohai state=present"
```

Selecting Files And Templates Based On Variables

- Sometimes a configuration file you want to copy, or a template you will use may depend on a variable.
- The following construct selects the first available file appropriate for the variables of a given host, which is often much cleaner than putting a lot of if conditionals in a template.

Example

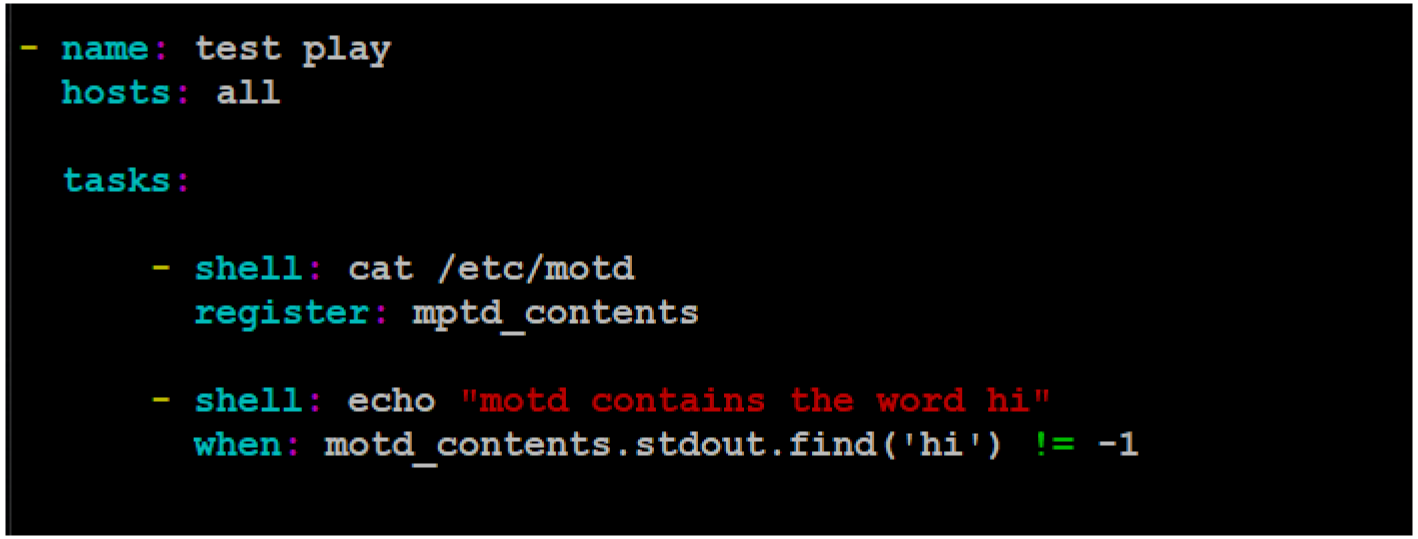


```
- name: Template a file
  template: src={{ item }} dest=/etc/myapp/foo.conf
  with_first_found:
    - files:
       - {{ ansible_distribution }}.conf
       - default.conf
      paths:
       - search_location_one/somedir
       - /opt/other_location/somedir/
```

Register Variables

- Often in a playbook it may be useful to store the result of a given command in a variable and access it later.
- Use of the command module in this way can in many ways eliminate the need to write site specific facts, for instance, you could test for the existence of a particular program.
- The 'register' keyword decides what variable to save a result in. The resulting variables can be used in templates, action lines, or when statements.

Example



```
- name: test play
  hosts: all

  tasks:

      - shell: cat /etc/motd
        register: mptd_contents

      - shell: echo "motd contains the word hi"
        when: motd_contents.stdout.find('hi') != -1
```