Conditionals

Topics

Often the result of a play may depend on the value of a variable, fact (something learned about the remote system), or previous task result. In some cases, the values of variables may depend on other variables. Further, additional groups can

be created to manage hosts based on whether the hosts match other criteria. There are many options to control execution flow in Ansible.

Let's dig into what they are.

The When Statement

Sometimes you will 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 Jinja2 expression (see *Variables*). It's actually pretty simple:

```
tasks:
  - name: "shutdown Debian flavored systems"
    command: /sbin/shutdown -t now
    when: ansible_os_family == "Debian"
```

You can also use parentheses to group conditions:

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

A number of Jinja2 "filters" can also be used in when statements, some of which are unique and provided by Ansible. Suppose we want to ignore the error of one statement and then decide to do something conditionally based on success or

failure:

```
tasks:
  - command: /bin/false
    register: result
    ignore_errors: True
  - command: /bin/something
    when: result|failed
  - command: /bin/something_else
    when: result|success
  - command: /bin/still/something_else
    when: result|skipped
```

Note that was a little bit of foreshadowing on the ‘register’ statement. We’ll get to it a bit later in this chapter.

As a reminder, to see what facts are available on a particular system, you can do:

```
ansible hostname.example.com -m setup
```

Tip: Sometimes you’ll get back a variable that’s a string and you’ll want to do a math operation comparison on it. You can do this like so:

```
tasks:
  - shell: echo "only on Red Hat 6, derivatives, and later"
    when: ansible_os_family == "RedHat" and ansible_lsb.major_release|int >= 6
```

Note
the above example requires the lsb_release package on the target host in order to return the ansible_lsb.major_release fact.

Variables defined in the playbooks or inventory can also be used. An example may be the execution of a task based on a variable's boolean value:

```
vars:
  epic: true
```

Then a conditional execution might look like:

```
tasks:
    - shell: echo "This certainly is epic!"
      when: epic
```

or:

```
tasks:
    - shell: echo "This certainly isn't epic!"
      when: not epic
```

If a required variable has not been set, you can skip or fail using Jinja2's *defined* test. For example:

```
tasks:
    - shell: echo "I've got '{{ foo }}' and am not afraid to use it!"
      when: foo is defined

    - fail: msg="Bailing out. this play requires 'bar'"
      when: bar is not defined
```

This is especially useful in combination with the conditional import of vars files (see below).

Note that when combining *when* with *with_items* (see *Loops*), 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, which is covered in *Developing Modules*. 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 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. Note this does not work with playbook includes, just task includes. All the tasks get evaluated, but the conditional is applied to each and every task:

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

Or with a role:

```
- hosts: webservers
  roles:
     - { role: debian_stock_config, when: ansible_os_family == 'Debian' }
```

You will note a lot of 'skipped' output by default in Ansible when using this approach on systems that don't match the criteria. Read up on the 'group_by' module in the *About Modules* docs for a more streamlined way to accomplish the same thing.

Conditional Imports

> **Note**
> This is an advanced topic that is infrequently used. You can probably skip this section.

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:

```
---
- hosts: all
  remote_user: root
  vars_files:
    - "vars/common.yml"
    - [ "vars/{{ ansible_os_family }}.yml", "vars/os_defaults.yml" ]
  tasks:
  - name: make sure apache is running
    service: name={{ apache }} state=running
```

Note
The variable 'ansible_os_family' is being interpolated into the list of filenames being defined for vars_files.

As a reminder, the various YAML files contain just keys and values:

```
---
# for vars/CentOS.yml
apache: httpd
somethingelse: 42
```

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'. Pretty simple.

To use this conditional import feature, you'll need facter or ohai installed prior to

running the playbook, but you can of course push this out with Ansible if you like:

```
# 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"
```

Ansible's approach to configuration – separating variables from tasks, keeps your playbooks from turning into arbitrary code with ugly nested ifs, conditionals, and so on - and results in more streamlined & auditable configuration rules – especially because there are a minimum of decision points to track.

Selecting Files And Templates Based On Variables

> **Note**
> This is an advanced topic that is infrequently used. You can probably skip this section.

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.

The following example shows how to template out a configuration file that was very different between, say, CentOS and Debian:

```
- 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. It looks like this (in an obviously trivial example):

```
- name: test play
  hosts: all

  tasks:

      - shell: cat /etc/motd
        register: motd_contents

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

As shown previously, the registered variable’s string contents are accessible with

the ‘stdout’ value. The registered result can be used in the “with_items” of a task if it is converted into a list (or already is a list) as shown below. “stdout_lines” is already available on the object as well though you could also call “home_dirs.stdout.split()” if you wanted, and could split by other fields:

```
- name: registered variable usage as a with_items list
  hosts: all

  tasks:

      - name: retrieve the list of home directories
        command: ls /home
        register: home_dirs

      - name: add home dirs to the backup spooler
        file: path=/mnt/bkspool/{{ item }} src=/home/{{ item }} state=link
        with_items: home_dirs.stdout_lines
        # same as with_items: home_dirs.stdout.split()
```

See also

Playbooks
An introduction to playbooks

Playbook Roles and Include Statements
Playbook organization by roles

Best Practices
Best practices in playbooks

Conditionals
Conditional statements in playbooks

Variables
All about variables

User Mailing List
Have a question? Stop by the google group!

irc.freenode.net
#ansible IRC chat channel

Previous Next