RUNNING TASKS CONDITIONALLY

Ansible can use conditionals to execute tasks or plays when certain conditions are met. For example, a conditional can be used to determine available memory on a managed host before Ansible installs or configures a service.

Conditionals allow administrators to differentiate between managed hosts and assign them functional roles based on the conditions that they meet. Playbook variables, registered variables, and Ansible facts can all be tested with conditionals. Operators to compare strings, numeric data, and Boolean values are available.

The following scenarios illustrate the use of conditionals in Ansible:

- A hard limit can be defined in a variable (for example, min_memory) and compared against the available memory on a managed host.
- The output of a command can be captured and evaluated by Ansible to determine whether or not a task completed before taking further action. For example, if a program fails, then a batch is skipped.
- Use Ansible facts to determine the managed host network configuration and decide which template file to send (for example, network bonding or trunking).
- The number of CPUs can be evaluated to determine how to properly tune a web server.

#vim conditions.yml

```
---
 - name : Writing Playbook with conditions
   hosts : db
   tasks :
    - name : Installing Database
      yum :
        name : mariadb
        state : latest
```

when :

ansible_nodename == 'p1.example.com'

:wq!

HANDLING TASK FAILURE

Ansible evaluates the return code of each task to determine whether the task succeeded or failed. Normally, when a task fails Ansible immediately aborts the rest of the play on that host, skipping all subsequent tasks.

However, sometimes you might want to have play execution continue even if a task fails. For example, you might expect that a particular task could fail, and you might want to recover by running some other task conditionally. There are a number of Ansible features that can be used to manage task errors.

Ignoring Task Failure

By default, if a task fails, the play is aborted. However, this behavior can be overridden by ignoring failed tasks. You can use the ignore_errors keyword in a task to accomplish this.

The following snippet shows how to use ignore_errors in a task to continue playbook execution on the host even if the task fails. For example, if the notapkg package does not exist then the yum module fails, but having ignore_errors set to yes allows execution to continue.

```
- name: Latest version of notapkg is installed
  yum:
    name: notapkg
    state: latest
  ignore_errors: yes
```

Ansible Tags

If you have a large playbook, it becomes useful to be able to run only a specific part of it rather than running everything in the playbook. Ansible supports a tag attribute for this reason.

When you apply tags on things, then you can control whether they are executed by adding command-line options.

#vim tags.yml

```
---
 - name : Creating Users
   hosts : db
   tasks :
     - name : Creating user31 ( Task 1 )
       user :
         name : user31
         state : present
         shell : /bin/sh
         createhome : yes
         home : /home/user31

     - name : Creating user32 ( task 2 )
       user :
         name : user32
         state : present
         shell : /bin/sh
         createhome : yes
```

```
    home : /home/user32
- name : Creating user39 ( task 3 )
  user :
    name : user39
    state : present
    shell : /bin/sh
    createhome : yes
    home : /home/user33
  tags :
    - create

- name : Creating user34 ( Task 4  )
  user :
    name : user34
    state : present
    shell : /bin/sh
    createhome : yes
    home : /home/user34
  tags :
    - create
- name : Creating user35 ( Task 5 )
  user :
    name : user35
    state : present
```

shell : /bin/sh

createhome : yes

home : /home/user35

- name : Creating user36 (Task 6)

user :

name : user36

state : present

shell : /bin/sh

createhome : yes

home : /home/user36

- name : Creating user37 (Task 7)

user :

name : user37

state : present

shell : /bin/sh

createhome : yes

home : /home/user37

- name : Creating user38 (Task 8)

user :

name : user38

state : present

shell : /bin/sh

createhome : yes

home : /home/user38

tags :

- create

:wq!

#ansible-playbook tags.yml --list-tasks **(Lists all the tasks in Play book tags.yml)**

#ansible-playbook tags.yml --tags create --check **(verifying the tasks enabled with Tags)**

#ansible-playbook tags.yml --tags **create (Performing selected tasks enabled with tag : create)**

#ansible-playbook tags.yml --tags <tag1>,<tag2>,<tag3> **(Performing multiple selected tasks enabled with different Tags)**