Chapter : 05

IMPLEMENTING TASK CONTROL

LOOPS

Loops saves administrators from the need to write multiple tasks that use the same module.

Keyword = loop
value = item

```
- service:
    name: "{{ item }}"
    state: started
  loop:
    - firewalld
    - chronyd
```

```
- user:
    name: "{{ item.name }}"
    state: present
    groups: "{{ item.groups }}"
  loop:
    - name: user1
      groups: group1
    - name: user2
      groups: group2
```

Previously instead of loop 'with_*' were used.

```
vars:
  data:
    - user1
    - user2
tasks:
  debug: | msg: "{{item }}" | with_items: "{{data}}"
```

```
task:
  - shell: "my name is {{item}}"
    loop:
      - one
      - two
    register: echo_results
  - debug:
  - debug:
      msg: " {{ item.stdout }} "
    loop: "{{echo_results.results }}"
```

When

The when statement is used to run a task conditionally.

```
- hosts: all
  vars:
    my_service: httpd
  tasks:
    yum:
      name: " {{ my_service }}"
    when: my_service is defined.
```

Example of conditions:

equals to string ' == " " '
equals to numeric ==
less than ' < '
greater than ' > '
less than or equal to ' <= '
greater than or equal to ' >= '
not equal to ' != '

variable exist	min_memory 'is defined'
variable does not exist	" " 'is not defined'
true, 1, True, yes	'memory_available'
false, 0, False, no	'not memory_available'
1st var value is present as value in 2nd var	var_1 in var_2

example of last case:

```
  :
  vars:
     supported_software:
           - ~~soft1 os~~ - redhat
           - ~~soft2 os~~ - ubuntu
  :
  when: ansible_distribution in supported_software.
```

~~→~~ when condition can be combined using 'and' or 'or' keyword. & grouped with parenthesis.

either condition → 'or' keyword, / One cond. has to be true

```
when: ansible_distribution == "RedHat" or ansible_distribution
                                     == "Fedora"
```

And condition → 'and' keyword / both condition has to be true

```
~~when: ansible_distribution == "RedHat" and ansible_distribution == "Fedora"~~
when: ansible_distribution == "Redhat" and ansible_distribution_version
                                              == "7.5"
```

Another way to use 'and' condition is

```
when:
    - variable = output result
    - variable = output result
```

Combination of 'and' & 'or' with () parenthesis

```
when: >                                   ( > = split for multiple lines. )
    ( ansible_distribution == "RedHat" and
      ansible_distribution_major_version == "7" )
  or
    ( —— " —— == "Fedora" and
      —— " ——        "28" )
```

Combination of loop & conditional tasks

```
- name: install mariadb-server if enough space on root.
  yum:
     name: mariadb-server
     state: latest
  loop: "{{ ansible_mount }}"
  when: item.mount == "/" and item.size_available > 300000000
```

Implementing Handlers

Ansible modules are idempotent. But in some cases when changes are done in config files, there is requirement to reload the service; in such cases handlers play the role.

Handlers are tasks that respond to a notification triggered by other tasks.

If one or more tasks notify the handler, the handler will run exactly once after all other tasks in the play have compleated.

Handlers ⇒ inactive tasks ⇒ triggered by "notify" statement

```
tasks:
  - name: Copying template.
   template:
        src: /var/lib/template/file.1
        dest: ~~/file~~ /etc/httpd/conf.d/file 2
    notify:
       - restart apache.

handlers:
  - name: restart apache.
    service:
        name: httpd.
        state: restarted.
```

→ for two notify: example:

```
 tasks:
         :
     notify:
        - restart apache
        - restart mysql.
  handlers:
         :
    - name: restart apache
         :
    - name: restart mysql.
        name: mariadb
        State: ~~resta~~ restarted.
```

Note:

1) Handlers always runs in the order specified under handler; Not in the order specified in notify.

2) Handlers name must be unique (if two names are there) if common/same name is given by mistake, then only one handler will run.

3) Even if more than on task notifies same handler name then also handler only run once.

4) Handler only ~~-cha~~ runs; whenever 'notify' is 'changed' in running playbook.

Ø - Handling Task Failures.

When a task fails Ansible immediately aborts the rest of the play on that host, skipping all subsequent tasks. ⇒ This setting is by default.

This can be overridden by using 'ignore_errors' keyword in a task.

```
- name : package installation
  yum:
      name : mariabb      (as mariabb does not exist)
      State : present
  ignore_errors: yes
```

(This does not abort play & move forward to execute further play & playbook tasks)

Handler & Task failure

if a task fails; play is aborted; Thus the handler is not run.; Default setting
To override this; "force_handlers: yes" is used.

```
- hosts: all
  ignore
  force_handlers: yes.
  tasks:
    - yum:
        name: mariadb
        state: present
      notify: reboot
  handlers:
    - name: reboot
      reboot
```

Runs the handler though the task is failed and play might get abort.

failed_when

failed_when keyword on the task specifies which conditions indicate that the task has failed.

```
tasks:
  - name: run a script
    shell: /usr/local/bin/create.sh
    register: command_result
    failed_when: "Password Missing" in command_result.stdout
```

fail

```
tasks:
  - name : Run a script
    shell :  /usr/local/bin/user_creation.sh
    register: comand_result
    ignore_errors : yes
  - name : Running a fail task
    fail :
       msg: "The Password is missing in the output"
    when: "Password Missing" in command_result.stdout
```

change_when

shell, command modules always show changed = True ; To supress this "changed_when: false" is set so that it will only reports "ok" or "failed"

```
- name: get kerberos credentials as "admin"
  shell: echo "{{ krb_admin_pass }}" | kinit -f admin
  changed_when: false
```

Block rescue & always

```
- name: block example
  hosts: all
  tasks :
    - name: installing package
      block:
          - yum:
              :
      when: ansible_distribution == "RedHat"
```

Block: Defines main task to run.

Rescue: Defines task to run, if the tasks defined in the block clause fail.

Always: Defines the tasks that will always run independently of the success or failure of tasks defined in the block and rescue clauses.

"The when condition on a block clause also applies to its rescue and always clauses if present"

```
tasks:
  - name: Upgrade DB
    block:
      - name: upgrade the database
        shell:
          cmd: /usr/local/lib/upgrade-database.
    rescue:
      - name: revert the database upgrade
        shell:
          cmd: /usr/local/lib/revert-database
    always:
      - name: always restart the database
        service:
          name: mariadb
          state: restarted.
```