ANSIBLE

HANDLING TASKS

❖ HANDLING TASKS:

- In Ansible, handlers are typically used to start, reload, restart, and stop services.
- Sometimes you want a task to run only when a change is made on a machine.
  E.g.: you may want to restart a service if a task updates the configuration of that service, but not if the configuration is unchanged. Ansible uses handlers to address this use case.
- Handlers are tasks that only run **when notified.**
- By default, handlers are executed **last regardless** of their location in the playbook.

A SINGLE TASK AND A HANDLER:

```
- hosts: webservers
  become: true
  become_user: root
  tasks:
    - name: Install the latest version of Apache
      dnf:
        name: httpd
        state: latest
      notify:
       - Start Apache
  handlers:
     - name: Start Apache
       service:
         name: httpd
         state: started
```

MULTIPLE TASKS AND HANDLERS:

```
- hosts: webservers
  become: true
  become_user: root
  tasks:
    - name: Install the latest version of Apache
      yum:
        name: httpd
        state: latest
    - name: Configure Apache
      copy:
        src: /home/raju/index.html
        dest: /var/www/html
        owner: apache
        group: apache
        mode: 0644
      notify:
      - Configure Firewall
      - Start Apache
  handlers:
     - name: Start Apache
       service:
         name: httpd
         state: started
```

```
- name: Configure Firewall
  firewalld:
    permanent: yes
    immediate: yes
    service: http
    state: enabled
```

➢ HANDLING TASK FAILURE:

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

Ignoring Task Failure:

By default a task fails, the play is aborted. However, this behavior can be overridden by ignoring faild tasks.

You can use the ignore_error keyword in a task to accomplish this.

Example:

```
- hosts: server
  become: true
  become_user: root
  tasks:
    - name: Restart a service
      service:
        name: not a service
        state: restart
    - name: Copy a script
      copy:
        src: /tmp/script.sh
        dest: /opt
...


$ansible-playbook --syntax-check task1.yml
$ansible-playbook task1.yml -K
```

➢ IGNORE_ERRORS:

- Ansible console output becomes much harder to inspect because your it will contain lots of red (failed) task around, so scrolling to the right line would be much harder.
- It will trigger Ansible debugger if you configured ANSIBLE_STRATEGY=debug, even if you are likely not to want this making the use of debugger kinda useless if you have lots of such ignore_errors.

```
$vi task2.sh
- hosts: webservers
  become: true
  become_user: root
  tasks:
    - name: Restart a service
      service:
        name: not a service
        state: restart
      ignore_errors: yes
  - name: Copy a script
    copy:
      src: /tmp/script.sh
      dest: /opt
...
$ansible-playbook --syntax-check task2.yml
$ansible-playbook task1.yml -K
```

REGISTER:

- Ansible register is a way to capture the output from task execution and store it in a variable.

➢ FAILED_WHEN AND CHANGED_WHEN:

- we are going to see how to use conditional statements of Ansible such as when, changed_when, failed_when and where to use them appropriately and how it works. By these conditional modules, Ansible provides a way for us to define when should ansible run a certain task or consider the executed task as Success or failure.
- Long Story Short, these modules give us a way to make ansible do something when a certain condition is met or satisfied.
- The primary purpose of the failed_when and changed_when statements are to determine whether the task is actually successful or failure
- let us cover each conditional statements one by one with examples.

FAILD_WHEN:

- Use **failed_when** to make the playbook fail checking a condition.

```
$vi task3.yml
- hosts: webservers
  become: true
  become_user: root
  tasks:
    - name: Restart a service
      service:
        name: not a service
        state: restart
      ignore_errors: yes

   - name: Copy a script
      copy:
        src: /tmp/script.sh
        dest: /tmp

   - name: Run the script
      shell: sh /tmp/script.sh
      register: command_result

   - debug: msg="{{ command_result.stdout }}"
```

```
$vi task4.sh
- hosts: webservers
  become: true
  become_user: root
  tasks:
    - name: Restart a service
      service:
        name: not a service
        state: restart
      ignore_errors: yes
    - name: Copy a script
      copy:
        src: /tmp/script.sh
        dest: /tmp
    - name: Run the script
      shell: sh /tmp/script.sh
      register: command_result
      failed_when: "'raju' in command_result.stdout"
    - debug: msg="{{ command_result.stdout }}"
    - name: Restart the HTTPD
      service:
        name: httpd
        state: restarted
```

CHANGED_WHEN:

- The changed_when keyword can be used to control when a task reports that it has changed.

```
- hosts: webservers
  become: true
  tasks:
    - name: Restart a service
      service:
        name: not a service
        state: started
      ignore_errors: yes
    - name: Copy a script
      copy:
        src: /tmp/script.sh
        dest: /tmp
    - name: Run the script
      shell: sh /tmp/script.sh
      register: command_result
      changed_when: "'success' in command_result.stdout"
      notify:
       - restart_apache
  handlers:
     - name: restart_apache
       service:
         name: httpd
         state: restarted
```