🧪 Lab 12 — Error Handling in Ansible (`ignore_errors`, `failed_when`, `changed_when`)

Author: *Sandeep Kumar Sharma*

🕑 Learning Objectives

In this lab, you will learn: - How Ansible handles errors during execution - How to ignore task failures safely - How to define custom failure conditions - How to define custom change conditions - How to write more reliable and predictable automation

---

👙 Learning Outcomes

After completing this lab, you will: - Control when tasks should fail or continue - Prevent playbooks from stopping unnecessarily - Build intelligent and fault-tolerant automation - Understand how to override default Ansible behavior

---

🧰 Why Is Error Handling Important?

Not all errors should stop the automation.
Some failures are acceptable, some must be handled, and some must trigger custom conditions.

Error-handling gives you control over: - When a task should ignore failure - When a task should be marked as failed - When a task should be marked as changed or unchanged

---

🪙 SECTION A — Using `ignore_errors`

This allows tasks to continue even if they fail.

Create file:

```
nano ignore-errors.yml
```

Add:

```
---
- name: Demo ignore_errors
  hosts: dev
  become: yes

  tasks:
    - name: Try installing a package that doesn't exist
      package:
        name: no-such-package
        state: present
      ignore_errors: yes

    - name: This task still runs
      debug:
        msg: "Playbook continues even after failure"
```

Run:

```
ansible-playbook ignore-errors.yml
```

🏛️SECTION B — Using `failed_when` (Custom Failure Conditions)

This decides when a task should be marked as failed.

```
nano failed-when.yml
```

Add:

```
---
- name: Demo failed_when
  hosts: dev
  become: yes

  tasks:
    - name: Run a command
      shell: echo "hello"
      register: cmd_out

    - name: Fail if output contains the word 'hello'
```

```
    debug:
      msg: "Output: {{ cmd_out.stdout }}"
    failed_when: "'hello' in cmd_out.stdout"
```

Run:

```
ansible-playbook failed-when.yml
```

🏛️ SECTION C — Using `changed_when` (Custom Change Detection)

This controls when a task should or should not report "changed".

Create file:

```
nano changed-when.yml
```

Add:

```
---
- name: Demo changed_when
  hosts: dev
  become: yes

  tasks:
    - name: Check file
      stat:
        path: /etc/passwd
      register: file_out

    - name: Print output but mark unchanged
      debug:
        msg: "File exists"
      changed_when: false
```

Run:

```
ansible-playbook changed-when.yml
```

🏛️SECTION D — Real-World Example: Custom Error Handling for Services

Create file:

```
nano service-error-demo.yml
```

Add:

```
---
- name: Service check example
  hosts: dev
  become: yes

  tasks:
    - name: Try checking a service that may not exist
      shell: systemctl status customservice
      register: svc
      ignore_errors: yes

    - name: Fail only if service output contains 'failed'
      debug:
        msg: "Service Check: {{ svc.stdout }}"
      failed_when: "'failed' in svc.stdout"
```

Run:

```
ansible-playbook service-error-demo.yml
```

🧪Hands-On Checklist

- [ ] Use `ignore_errors` to continue execution
- [ ] Use `failed_when` to define custom error rules
- [ ] Use `changed_when` to control change reporting
- [ ] Combine all three for real-world scenarios

🩱 Lab Summary

In this lab, you learned: - How to manage errors using `ignore_errors` - How to create custom failure conditions with `failed_when` - How to control change reporting using `changed_when` - How to build smart and predictable automation

Next Lab: 👉**Lab 13 — Ansible Facts (Advanced Usage & Filters)**