

Error Handling

Ansible Advanced

Agenda

- Error Handling
- Using blocks
- Handlers and Failure
- Execution strategy

Error Handling In Practice

Error Handling

How Ansible behaves with failure

- If you get an error on a task when running on a host, Ansible's default behavior is to continue executing the task on the other hosts
- The task will not break for everyone just because it failed on one host
- However, the playbook execution for the failed host will be halted at the point of the error, and Ansible will not execute subsequent tasks on that specific host
- The other hosts will continue to execute the remaining tasks in the playbook.

Error Handling: ignore_errors

- This directive allows a task to fail without stopping the entire playbook
- It's useful when you expect a task might fail and it's okay if it does.

```
- name: Try to install a package that might not exist
  apt:
    name: some_unknown_package
    state: present
  ignore_errors: yes
```

Error Handling: failed_when

- Allows you to define custom conditions for what constitutes a failure
- It can be used to override the default failure conditions of a module

```
- name: Check free disk space
  ansible.builtin.stat:
    path: /opt/data
  register: result
  failed_when: result.stat.size > 5000000
```

Error Handling: failed_when

- Another example of using **failed_when** to generate an error based on task normal output

```
- name: Execute a command that might fail
  command: "some-command"
  register: command_result
  failed_when: "'ERROR' in command_result.stderr"
```

Error Handling: any_errors_fatal

- If set to **yes**, any error will be considered fatal and will stop all hosts' execution
- Useful when you want to ensure that if any host fails, no further tasks should be executed on any host
- Can be used on tasks or playbook level

```
- hosts: web_servers
  any_errors_fatal: true
  tasks:
    - name: Ensure configuration file is present
      ansible.builtin.copy:
        src: /src/config.conf
        dest: /dest/config.conf
```

Error Handling: max_fail_percentage

- If you're running a task on multiple hosts, you can use this directive to continue execution even if a certain percentage of hosts fail
- Can be set on task or playbook level

```
- hosts: web_servers
  max_fail_percentage: 30
  tasks:
    - name: Update application
      ansible.builtin.git:
        repo: https://github.com/example/app.git
        dest: /opt/app
```

Unreachable hosts

- Unreachable hosts have a different behavior from failed tasks
- If Ansible cannot connect to a host, it marks that host as 'UNREACHABLE' and removes it from the list of active hosts for the run
- Using **ignore_errors** don't have any effect on unreachable hosts

Use ignore_unreachable

- When you set **ignore_unreachable: yes** for a task or playbook, Ansible will not fail the playbook for that specific task/block if a host becomes unreachable.
- Instead, it will ignore the unreachable host and continue executing the remaining tasks for that host, as well as continue executing tasks for other hosts.
- This can be useful in scenarios where temporary network disruptions are expected, and you don't want the entire playbook run to be affected.

```
- name: Sample Playbook with ignore_unreachable
hosts: all
tasks:
  - name: This task will ignore unreachable hosts
    command: echo "Running a command"
    ignore_unreachable: yes

  - name: This task will fail if a host is unreachable
    command: echo "Another command"
```

Demo: Error Handling

Ansible Advanced

Blocks

Error Handling

What are Ansible blocks

- Blocks in Ansible provide a way to group related tasks together
- You may apply particular task attributes, such as looping, conditional execution, and error handling, to the entire group
- They can significantly simplify playbook logic and make error handling more streamlined

Error handling with blocks

- One of the most common uses of blocks is for error handling
- Blocks can be combined with **rescue** and **always** sections to define tasks that should be run if there's an error in the block or tasks that should always run regardless of success or failure

Basic Error Handling

```
- hosts: localhost
  tasks:
    - block:
        - name: This will fail
          command: /nonexistent/command

    rescue:
        - name: This will run if there's a failure in the block
          debug:
            msg: "An error occurred!"
```

Using always for Cleanup Tasks

```
- hosts: localhost
  tasks:
    - block:
        - name: Create a temporary file
          tempfile:
            state: file
            register: temp_file

        - name: This will fail
          command: /nonexistent/command

    rescue:
      - name: Handle the error
        debug:
          msg: "An error occurred!"

    always:
      - name: Cleanup temporary file
        file:
          path: "{{ temp_file.path }}"
          state: absent
```

Demo: Blocks

Ansible Advanced

Handlers and Failure

Error Handling

What are handlers?

- Handlers are tasks that only run when notified by another task
- They are triggered by a "**notify**" directive in a task
- Just like regular tasks in Ansible, handlers are idempotent
- Handlers run after all tasks are completed in a particular play, not immediately after the task that notified them
- If multiple tasks notify the same handler, the handler will run only once, after all tasks are completed
- The handler only runs if the task that notifies the handler returns a **CHANGED** result

Common use cases

- Restarting a service after its configuration file has been modified
- Reloading a service after a new package has been installed
- Any scenario where a specific action should only be taken in response to a change
- Can be used for error handling

Force handlers on error

- If a task notifies a handler but another task fails later in the play, by default the handler does not run on that host, which may leave the host in an unexpected state
- For example, a task could update a configuration file and notify a handler to restart some service. If a task later in the same play fails, the configuration file might be changed but the service will not be restarted.

Force handlers on error

- You can change this behavior with:
- **--force-handlers** command-line option
- Including **force_handlers: True** in a play
- Adding **force_handlers = True** to **ansible.cfg**
- When handlers are forced, Ansible will run all notified handlers on all hosts, even hosts with failed tasks

Handlers on Error Handling

- Use handlers to perform specific actions in response to errors or failures during playbook execution
- By using blocks, you can group tasks and define error handling for the entire block
- If any task within the block fails, the tasks in the rescue section will execute
- You can notify a handler from within the rescue section to take specific actions in response to errors.

Handlers on Error Handling

```
- hosts: localhost
  tasks:
    - block:
        - name: Update service configuration
          copy:
            src: new_config.conf
            dest: /etc/service/config.conf
          notify: Restart service

    rescue:
        - name: Restore backup configuration
          copy:
            src: backup_config.conf
            dest: /etc/service/config.conf
          notify: Restart service

  handlers:
    - name: Restart service
      service:
        name: myservice
        state: restarted
```

Demo: Handlers

Ansible Advanced

Execution Strategy

Error Handling

Linear Execution

- The linear strategy, which is the default, ensures that each task is executed on all hosts before moving on to the next task.
- If you have a list of tasks and a group of hosts, Ansible will take the first task and execute it on all hosts, one by one (or in parallel batches if you've set a forks value greater than 1).
- Only after the first task is completed on all hosts will Ansible move on to the second task.
- Use Cases
 - When you need to ensure that a particular state is set across all hosts before proceeding.
 - When tasks have dependencies that need to be met on all hosts before moving on.
- Example
 - If you have tasks A, B, and C and hosts 1, 2, and 3, the order of execution will be: A1, A2, A3, B1, B2, B3, C1, C2, C3.

Free Execution

- The free strategy allows each host to process its list of tasks as quickly as possible without waiting for other hosts
- Hosts will execute their list of tasks independently and may finish at different times
- One host might be on task 1 while another host could be on task 3, depending on factors like network speed, host performance, etc.
- Use Cases:
 - When you want to complete the playbook run as quickly as possible and don't have inter-host task dependencies.
 - When working with a large number of hosts where waiting for each task to complete on every host would be inefficient.
- Example
 - If you have tasks A, B, and C and hosts 1, 2, and 3, the order of execution could be: A1, A2, A3, B1, C1, B2, B3, C2, C3. Host 1 moved through all its tasks before Host 2 completed its second task.

Comparison

Order

- **linear**: Ensures a strict order of task execution across all hosts.
- **free**: Allows hosts to process tasks as quickly as they can, without a strict order.

Efficiency

- **linear**: Can be slower, especially with a large number of hosts, as each task must complete on all hosts before moving to the next.
- **free**: Can be faster as hosts work through tasks independently.

Predictability

- **linear**: More predictable as tasks are executed in a specific order across hosts.
- **free**: Less predictable as hosts may be on different tasks at any given time.

Use Case

- **linear**: Best for playbooks where task order and consistency across hosts are crucial.
- **free**: Best for playbooks where speed is more important than strict task order.

How to change execution strategy

- Playbook Level

- You can set the strategy for a specific playbook by adding the **strategy** directive at the beginning of your playbook

```
- hosts: all
  strategy: free
  tasks:
```

- Ansible Configuration

- If you want to set the free strategy as the default for all playbooks, you can do so in the Ansible configuration file (ansible.cfg)

```
[defaults]
strategy = free
```

