Task Error Handling in Ansible

By: Er. Vikas Nehra (M. Tech, B. Tech), Experience: 15 + Years

Session - 19 Agenda:

1. Task Error Handling in Ansible:
(a) Block & Rescue
(b) Ignore Errors
(c) Failed When

Task Error Handling in Ansible:

When Ansible receives a non-zero return code from a command or a failure from a module, by default it stops executing on that host and continues on other hosts. However, in some circumstances you may want different behavior. Sometimes a non-zero return code indicates success. Sometimes you want a failure on one host to stop execution on all hosts. Ansible provides tools and settings to handle these situations and help you get the behavior, output, and reporting you want.

There are many options available in Ansible to handle the task errors like:

(a) Block & Rescue
(b) Ignore Errors
(c) Failed When

Block & Rescue:

Blocks create logical groups of tasks.

Grouping tasks with blocks:

All tasks in a block inherit directives applied at the block level. Most of what you can apply to a single task (with the exception of loops) can be applied at the block level, so blocks make it much easier to set data or directives common to the tasks. The directive does not affect the block itself; it is only inherited by the tasks enclosed by a block. For example, a when statement is applied to the tasks within a block, not to the block itself.

Let's understand it with an example. Suppose we want to install httpd web server in all the nodes which are running on RHEL. We create an ansible playbook to install the httpd package and start the service. We have intentionally not used become keyword anywhere in this playbook.

$ vim blocks.yml

```
---
- name: Ansible blocks example
  hosts: all
  tasks:
    - name: Installing HTTPD latest package
      yum:
          name: httpd
          state: latest

    - name: Starting HTTPD service.
      service:
          name: httpd
          state: started
...
```

Let's execute this playbook and confirm that it will give us the error because these tasks require the admin or superuser privileges.

$ ansible-playbook blocks.yml

Task Error Handling in Ansible

By: Er. Vikas Nehra (M. Tech, B. Tech), Experience: 15 + Years

Now, let's allow the ansible user to use sudo access while running these tasks, for this when are going to mention become keyword against each task (not on the entire playbook) where sudo access is needed. There may be such n number of tasks and sometimes we don't want to use become option for all the tasks using become keyword globally for the playbook.

$ vim blocks.yml

```
---
- name: Ansible blocks example
  hosts: all
  tasks:
    - name: Installing HTTPD latest package
      yum:
         name: httpd
         state: latest
      become: yes

    - name: Starting HTTPD service.
      service:
         name: httpd
         state: started
      become: yes
...
```

Now, execute this playbook again or do the dry run to verify the working.

$ ansible-playbook blocks.yml -C

This time the playbook will execute on all the nodes and it will install the httpd server on the nodes where yum module is supported. If there are a large number of such tasks when the requirement is similar (like running it with sudo, same OS etc.), when can group such tasks using block option in the playbook.

$ vim blocks.yml

```
---
- name: Ansible blocks example
  hosts: all
  tasks:
     - block:
          - name: Installing HTTPD latest package
            yum:
               name: httpd
               state: latest
          - name: Starting HTTPD service.
            service:
              name: httpd
              state: started
       become: yes
       when: ansible_facts['distribution'] == 'RedHat'
...
```

Let's execute the playbook.

$ ansible-playbook blocks.yml

Task Error Handling in Ansible

By: Er. Vikas Nehra (M. Tech, B. Tech), Experience: 15 + Years

Handling errors with blocks:

You can control how Ansible responds to task errors using blocks with rescue and always sections. Rescue blocks specify tasks to run when an earlier task in a block fails. This approach is similar to exception handling in many programming languages. Ansible only runs rescue blocks after a task returns a 'failed' state. Bad task definitions and unreachable hosts will not trigger the rescue block.

Let's understand it with the help of an example. Suppose we need to configure EPEL repository in the RHEL 9 nodes only, first let's verify that there is no EPEL repository present in the node machines.

$ ansible all -m command -a 'ls /etc/yum.repos.d/epel.repo'
$ ansible all -m command -a 'dnf repolist all'

Let's create an ansible playbook for configuring the EPEL repo in RHEL9 based machines.

$ vim block_rescue.yml

```
---
- name: block & rescue example
  hosts: all
  become: true
  tasks:
    - block:
        - name: Installing the EPEL repository RPM
          command: sudo dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm -y
        - name: Verify the EPEL repository file.
          shell: ls /etc/yum.repos.d/epel.repo
      when: ansible_distribution_major_version == '9'
      rescue:
        - name: Copying EPEL repository file from the Ansible control node.
          copy:
            src: /tmp/epel.repo
            dest: /etc/yum.repos.d/epel.repo
...
```

Let's execute the playbook.

$ ansible-playbook block_rescue.yml

Verify the tasks using Ansible ah-hoc command.

$ ansible all -m command -a 'ls /etc/yum.repos.d/epel.repo'
$ ansible all -m command -a 'dnf repolist all'

Here you can see that the rescue task didn't execute because the regular tasks executed successfully. In case if any of the regular tasks get failed then only recuse task will execute.

Let's remove the EPEL repository from the managed nodes using Ansible Ad-hoc command.

$ ansible all -m command -a 'sudo dnf remove epel-release-9-5.el9.noarch -y'

Verify that there is no EPEL repository present now.

$ ansible all -m command -a 'ls /etc/yum.repos.d/epel.repo'
$ ansible all -m command -a 'dnf repolist all'

Task Error Handling in Ansible

By: Er. Vikas Nehra (M. Tech, B. Tech), Experience: 15 + Years

Let's create some errors in the first task, so that EPEL package installation will get failed.

$ vim block_rescue.yml

```
---
- name: block & rescue example
  hosts: all
  become: true
  tasks:
    - block:
        - name: Installing the EPEL repository RPM
          command: sudo dnf install https://dl.fredoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm -y
        - name: Verify the EPEL repository file.
          shell: ls /etc/yum.repos.d/epel.repo
      when: ansible_distribution_major_version == '9'
      rescue:
        - name: Copying EPEL repository file from the Ansible control node.
          copy:
            src: /tmp/epel.repo
            dest: /etc/yum.repos.d/epel.repo
...
```

Let's execute the playbook.

$ ansible-playbook block_rescue.yml

Verify the tasks using Ansible ah-hoc command.

$ ansible all -m command -a 'ls /etc/yum.repos.d/epel.repo'

$ ansible all -m command -a 'dnf repolist all'

Here you can see that rescue task executed successfully.

Task Error Handling in Ansible

By: Er. Vikas Nehra (M. Tech, B. Tech), Experience: 15 + Years

Ignore Errors:

By default, Ansible stops executing tasks on a host when a task fails on that host. You can use ignore_errors to continue on in spite of the failure. The ignore_errors directive only works when the task is able to run and returns a value of 'failed'.

Let's check the status of the cockpit service in the managed nodes.

$ ansible all -m command -a 'systemctl is-active cockpit.socket'

Let's create an ansible playbook to check and start the cockpit service.

$ vim ignore_errors.yml

```
---
- name: ignore errors example
  hosts: RHEL
  become: true
  tasks:
      - name: Checking the cockpit service.
        command: systemctl is-active cockpit.socket
        register: abc
      - debug:
          msg: "{{ abc }}"
...
```

Let's execute the playbook to check whether the cockpit service is running or not in the managed node.

$ ansible-playbook ignore_errors.yml

Here you can see that the task is not executing because ansible thinks that the command is returning an error, which is actually not true. Sometime we get outputs like inactive, warning, failed etc. on the Linux CLI while executing the commands (these are the actual outputs which we get after executing the commands, these are not the errors). Ansible considers such outputs as errors and therefore don't push the task in such cases. In order to overcome this problem, we tell ansible to ignore such errors by using the keyword ignore_errors in the task.

$ vim ignore_errors.yml

```
---
- name: ignore errors example
  hosts: RHEL
  become: true
  tasks:
      - name: Checking the cockpit service.
        command: systemctl is-active cockpit.socket
        register: abc
        ignore_errors: yes
      - debug:
          msg: "{{ abc }}"
...
```

Let's execute the playbook to check whether the cockpit service is running or not in the managed node.

$ ansible-playbook ignore_errors.yml

Task Error Handling in Ansible

By: Er. Vikas Nehra (M. Tech, B. Tech), Experience: 15 + Years

This time you can see that the playbook is running and it is showing us that output of the command as inactive.

Let's start the service is not running and restart it if it was running already.

$ vim ignore_errors.yml

```
---
- name: ignore errors example
  hosts: node1
  become: true
  tasks:
      - name: Checking the cockpit service.
        command: systemctl is-active cockpit.socket
        register: abc
        ignore_errors: yes
      - debug:
          msg: "{{ abc }}"
      - name: Starting the cockpit service if not running.
        service:
            name: cockpit.socket
            state: started
        when: abc.rc !=0
      - name: Restarting the cockpit service if already running.
        service:
            name: cockpit.socket
            state: restarted
        when: abc.rc ==0

...
```

Now, execute the playbook.

$ ansible-playbook ignore_errors.yml

This time when have started/restarted the cockpit service successfully using ignore_errors option in the playbook. Make sure you only use this keyword if you want to use ignore_errors to continue on in spite of the failure.

Task Error Handling in Ansible

By: Er. Vikas Nehra (M. Tech, B. Tech), Experience: 15 + Years

Failed When:

Ansible lets you define what “failure” means in each task using the failed_when conditional. As with all conditionals in Ansible, lists of multiple failed_when conditions are joined with an implicit and meaning the task only fails when all conditions are met.

Let’s suppose if we want to install & configure apache web server on all the nodes except on machines which are running on RHEL (means we don’t want to install apache web server in RHEL based machines) in our environment. We can easily do it using failed_when conditional in the ansible playbook.

$ vim failed_when.yml

```
---
- name: failed when example
  hosts: all
  become: true
  tasks:
     - debug:
           msg: "Incompatible Operating System."
       failed_when: ansible_distribution == 'RedHat'

     - name: Installing Apache Web Server
       apt:
          name: apache2
          state: latest
...
```

Let’s execute the playbook.

$ ansible-playbook failed_when.yml

Verify the same using ansible ad-hoc command.

$ ansible RHEL -m command -a 'rpm -qi httpd'

$ ansible node2 -m command -a 'apt list installed apache2'

Thanks