**Handle exceptions in Ansible Playbooks with block and rescue**

Have you ever gotten to the end of your Ansible Playbook execution and found you needed to:

1. Rescue from errors or partial execution of your tasks
2. Capture a summary of the results per host for further revision

What is a block?

A block is a logical grouping of tasks within a playbook that can be executed as a single unit. This makes it easy to manage complex playbooks by breaking them down into smaller, more manageable parts.

You can use blocks to apply options to a group of tasks and avoid repeating code, like in this example from the documentation.

tasks:

- name: Install, configure, and start Apache

block:

- name: Install httpd and memcached

ansible.builtin.yum:

name:

- httpd

- memcached

state: present

- name: Apply the foo config template

ansible.builtin.template:

src: templates/src.j2

dest: /etc/foo.conf

- name: Start service bar and enable it

ansible.builtin.service:

name: bar

state: started

enabled: True

when: ansible\_facts['distribution'] == 'CentOS'

become: true

become\_user: root

ignore\_errors: true

Notice that the keywords when, become, become\_user, and ignore\_errors are all applied to the block.

How to use blocks and rescue in Ansible

Blocks and rescue work together to provide error-handling capabilities in Ansible. Use the rescue keyword in association with a block to define a set of tasks that will be executed if an error occurs in the block. You can use the rescue tasks to handle errors, log messages, or take other actions to recover from the error.

Here is a high-level example:

**---**

- hosts: <hosts>

tasks:

- block:

- <task1>

- <task2>

- <task3>

rescue:

- <rescue\_task1>

- <rescue\_task2>

- <rescue\_task3>

always:

- <always\_task>

You define tasks under the **block** keyword, which could be as simple as invoking the ansible.builtin.ping module, or you could have a combination of multiple tasks and including/importing roles.

The associated **rescue** keyword is where the playbook execution will be sent, *for each host*, if anything fails along the block.

Finally, the **always** section executes for *all nodes*, no matter if they succeed or fail.

Some key ideas from this structure:

1. **rescue** and **always** are optional features, which I will use for the specific purpose of demonstrating this "recover and summary" logic.
2. When your playbook runs against a considerable number of hosts, handling the individual results becomes harder to track. This is how the ideas discussed in this article can help.

For the following example, the inventory file contains:

**[nodes]**

node1

node2

node3

Here is the playbook:

**---**

- name: Test block/rescue

hosts: nodes

gather\_facts: false

tasks:

- name: Main block

block:

- name: Role 1

ansible.builtin.include\_role:

name: role1

- name: Role 2

ansible.builtin.include\_role:

name: role2

- name: Accumulate success

ansible.builtin.set\_fact:

\_result:

host: "{{ inventory\_hostname }}"

status: "OK"

interfaces: "{{ ansible\_facts['interfaces'] }}"

rescue:

- name: Accumulate failure

ansible.builtin.set\_fact:

\_result:

host: "{{ inventory\_hostname }}"

status: "FAIL"

always:

- name: Tasks that will always run after the main block

block:

- name: Collect results

ansible.builtin.set\_fact:

\_global\_result: "{{ (\_global\_result | default([])) + [hostvars[item]['\_result']] }}"

loop: "{{ ansible\_play\_hosts }}"

- name: Classify results

ansible.builtin.set\_fact:

\_result\_ok: "{{ \_global\_result | selectattr('status', 'equalto', 'OK') | list }}"

\_result\_fail: "{{ \_global\_result | selectattr('status', 'equalto', 'FAIL') | list }}"

- name: Display results OK

ansible.builtin.debug:

msg: "{{ \_result\_ok }}"

when: (\_result\_ok | length ) > 0

- name: Display results FAIL

ansible.builtin.debug:

msg: "{{ \_result\_fail }}"

when: (\_result\_fail | length ) > 0

delegate\_to: localhost

run\_once: true

...

Think about this playbook as an illustration of some logic that could be applied to a complex automation in the real world. Yes, you could run simpler actions to recover or issue a notification about the failure, but you want a summary of all results. Then you can use this summary in the **always** section to automate sending a notification by email or writing the individual results into a database.

Automation advice

Also, the variables starting with \_ are my personal naming convention preference... there's no special meaning in Ansible for that.

1. For this example, the roles in the main block don't do anything special. They represent the actions that you would put in the main block, which could fail at different points. In this simplified example, if a node succeeds, there will be a list of interfaces in the \_result variable. Otherwise, the status will be set to FAIL.
2. For each host the playbook is running on:
   1. If the actions proceed without errors, the task **Accumulate success** will execute.
   2. If the action fails in any of the roles, the flow goes to the rescue block *for each host*.
3. The **always** section collects the results saved in the variable \_result. Here is a little breakdown of the logic:
   1. Up to this point, each host has a variable in its **hostvars** structure, either with a success or failed status information.
   2. In the **Collect results** task, which runs once and is delegated to localhost, it captures the individual results and adds them to the list \_global\_result.
   3. The loop is done using the Ansible magic variable ansible\_play\_hosts\_all, which is a list of all hosts targeted by this playbook.
   4. **Classify results** does some filtering to create a list of all OK and failed results. You can use these in notifications, reports, or to send to a database (this example just displays them).

If you run this playbook and no node fails, there is no need for rescue, and the display should show that results are OK in all nodes:

PLAY [Test block/rescue] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Role 1] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [role1 : Execution of role 1] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [node1] => {

"changed": false,

"msg": "All assertions passed"

}

ok: [node2] => {

"changed": false,

"msg": "All assertions passed"

}

ok: [node3] => {

"changed": false,

"msg": "All assertions passed"

}

TASK [Role 2] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [role2 : Execution of role 2] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [node1]

ok: [node2]

ok: [node3]

TASK [role2 : Show network information] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

skipping: [node1]

skipping: [node2]

skipping: [node3]

TASK [Accumulate success] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [node1]

ok: [node2]

ok: [node3]

TASK [Collect results] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [node1 -> localhost] => (item=node1)

ok: [node1 -> localhost] => (item=node2)

ok: [node1 -> localhost] => (item=node3)

TASK [Classify results] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [node1 -> localhost]

TASK [Display results OK] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [node1 -> localhost] => {

"msg": [

{

"host": "node1",

"interfaces": [

"lo",

"enp7s0",

"enp1s0"

],

"status": "OK"

},

{

"host": "node2",

"interfaces": [

"lo",

"enp7s0",

"enp1s0"

],

"status": "OK"

},

{

"host": "node3",

"interfaces": [

"enp7s0",

"lo",

"enp1s0"

],

"status": "OK"

}

]

}

TASK [Display results FAIL] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

skipping: [node1]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

node1 : ok=6 changed=0 unreachable=0 failed=0 skipped=2 rescued=0 ignored=0

node2 : ok=3 changed=0 unreachable=0 failed=0 skipped=1 rescued=0 ignored=0

node3 : ok=3 changed=0 unreachable=0 failed=0 skipped=1 rescued=0 ignored=0

If you force a failure in some nodes, they will invoke the rescue section, and the summary will show the ones that succeeded and those that failed:

PLAY [Test block/rescue] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Role 1] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [role1 : Execution of role 1] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [node1] => {

"changed": false,

"msg": "All assertions passed"

}

fatal: [node2]: FAILED! => {

"assertion": "inventory\_hostname in nodes\_ok",

"changed": false,

"evaluated\_to": false,

"msg": "Assertion failed"

}

fatal: [node3]: FAILED! => {

"assertion": "inventory\_hostname in nodes\_ok",

"changed": false,

"evaluated\_to": false,

"msg": "Assertion failed"

}

TASK [Role 2] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [role2 : Execution of role 2] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [node1]

TASK [role2 : Show network information] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

skipping: [node1]

TASK [Accumulate success] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [node1]

TASK [Accumulate failure] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [node2]

ok: [node3]

TASK [Collect results] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [node1 -> localhost] => (item=node1)

ok: [node1 -> localhost] => (item=node2)

ok: [node1 -> localhost] => (item=node3)

TASK [Classify results] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [node1 -> localhost]

TASK [Display results OK] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [node1 -> localhost] => {

"msg": [

{

"host": "node1",

"interfaces": [

"enp7s0",

"enp1s0",

"lo"

],

"status": "OK"

}

]

}

TASK [Display results FAIL] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [node1 -> localhost] => {

"msg": [

{

"host": "node2",

"status": "FAIL"

},

{

"host": "node3",

"status": "FAIL"

}

]

}

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

node1 : ok=7 changed=0 unreachable=0 failed=0 skipped=1 rescued=0 ignored=0

node2 : ok=1 changed=0 unreachable=0 failed=0 skipped=0 rescued=1 ignored=0

node3 : ok=1 changed=0 unreachable=0 failed=0 skipped=0 rescued=1 ignored=0

Notice that even though there were failures, at the end, the Ansible output counts them as **rescued**.