Red Hat Enterprise Linux Automation with Ansible

FEEDBACK TRANSLATIONS CERTIFICATE OF ATTENDANCE

Guided Exercise: Handling Task Failure ☆

In this exercise, you will explore different ways to handle task failure in an Ansible Playbook.

Outcomes

You should be able to:

- Ignore failed commands during the execution of playbooks.
- Force execution of handlers.
- Override what constitutes a failure in tasks.
- Override the `changed` state for tasks.
- Implement `block`, `rescue`, and `always` in playbooks.

On `workstation`, run the lab start script to confirm the environment is ready for the lab to begin. This script creates the working directory, `/home/student/control-errors`.

```
[student@workstation ~]$ lab control-errors start
```

Procedure 4.3. Instructions

1. On `workstation.lab.example.com`, change to the `/home/student/control-errors` project directory.

   ```
   [student@workstation ~]$ cd ~/control-errors
   [student@workstation control-errors]$
   ```

2. The lab script created an Ansible configuration file as well as an inventory file that contains the server `servera.lab.example.com` in the `databases` group. Review the file before proceeding.

3. Create the `playbook.yml` playbook, which contains a play with two tasks. Write the first task with a deliberate error to cause failure.

   3.1. Open the playbook in a text editor. Define three variables: `web_package` with a value of `http`, `db_package` with a value of `mariadb-server`, and `db_service` with a value of `mariadb`. These variables will be used to install the required packages and start the server.

   The `http` value is an intentional error in the package name. The file should read as follows:

```
---
- name: Task Failure Exercise
  hosts: databases
  vars:
    web_package: http
    db_package: mariadb-server
    db_service: mariadb
```

3.2. Define two tasks that use the `yum` module and the two variables, `web_package` and `db_package`. The tasks will install the required packages. The tasks should read as follows:

```
  tasks:
    - name: Install {{ web_package }} package
      yum:
        name: "{{ web_package }}"
        state: present

    - name: Install {{ db_package }} package
      yum:
        name: "{{ db_package }}"
        state: present
```

4. Run the playbook and watch the output of the play.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ***************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Install http package] ****************************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failures": ["No package http available."], "msg": "Fa
iled to install some of the specified packages", "rc": 1, "results": []}

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=1    changed=0    unreachable=0    failed=1
```

The task failed because there is no existing package called `http`. Because the first task failed, the second task was not run.

5. Update the first task to ignore any errors by adding the `ignore_errors` keyword. The tasks should read as follows:

```
  tasks:
    - name: Install {{ web_package }} package
      yum:
        name: "{{ web_package }}"
        state: present
      ignore_errors: yes

    - name: Install {{ db_package }} package
      yum:
        name: "{{ db_package }}"
        state: present
```

6. Run the playbook again and watch the output of the play.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ***************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Install http package] ****************************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failures": ["No package http available."], "msg": "Fa
iled to install some of the specified packages", "rc": 1, "results": []}
...ignoring

TASK [Install mariadb-server package] ******************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=3    changed=1    unreachable=0    failed=0
```

Despite the fact that the first task failed, Ansible executed the second one.

7. In this step, you will set up a `block` keyword so you can experiment with how they work.

   7.1. Update the playbook by nesting the first task in a `block` clause. Remove the line that sets `ignore_errors: yes`. The block should read as follows:

   ```
   - name: Attempt to set up a webserver
     block:
       - name: Install {{ web_package }} package
         yum:
           name: "{{ web_package }}"
           state: present
   ```

   7.2. Nest the task that installs the mariadb-server package in a `rescue` clause. The task will execute if the task listed in the `block` clause fails. The block clause should read as follows:

   ```
     rescue:
       - name: Install {{ db_package }} package
         yum:
           name: "{{ db_package }}"
           state: present
   ```

   7.3. Finally, add an `always` clause to start the database server upon installation using the `service` module. The clause should read as follows:

   ```
     always:
       - name: Start {{ db_service }} service
         service:
           name: "{{ db_service }}"
           state: started
   ```

   7.4. The completed task section should read as follows:

```
tasks:
  - name: Attempt to set up a webserver
    block:
      - name: Install {{ web_package }} package
        yum:
          name: "{{ web_package }}"
          state: present
    rescue:
      - name: Install {{ db_package }} package
        yum:
          name: "{{ db_package }}"
          state: present
    always:
      - name: Start {{ db_service }} service
        service:
          name: "{{ db_service }}"
          state: started
```

8. Now run the playbook again and observe the output.

8.1. Run the playbook. The task in the block that makes sure `web_package` is installed fails, which causes the task in the `rescue` block to run. Then the task in the `always` block runs.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ***************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Install http package] ****************************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failures": ["No package http available."], "ms
g": "Failed to install some of the specified packages", "rc": 1, "results": []}

TASK [Install mariadb-server package] ******************************************
ok: [servera.lab.example.com]

TASK [Start mariadb service] ***************************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=3    changed=1    unreachable=0    failed=1
```

8.2. Edit the playbook, correcting the value of the `web_package` variable to read `httpd`. That will cause the task in the block to succeed the next time you run the playbook.

```
vars:
  web_package: httpd
  db_package: mariadb-server
  db_service: mariadb
```

8.3. Run the playbook again. This time, the task in the block does not fail. This causes the task in the `rescue` section to be ignored. The task in the `always` will still run.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ***************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Install httpd package] ***************************************************
changed: [servera.lab.example.com]

TASK [Start mariadb service] ***************************************************
ok: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=3    changed=1    unreachable=0    failed=0
```

9. This step explores how to control the condition that causes a task to be reported as "changed" to the managed host.

   9.1. Edit the playbook to add two tasks to the start of the play, preceding the `block`. The first task uses the `command` module to run the `date` command and register the result in the `command_result` variable. The second task uses the `debug` module to print the standard output of the first task's command.

```
  tasks:
    - name: Check local time
      command: date
      register: command_result

    - name: Print local time
      debug:
        var: command_result.stdout
```

   9.2. Run the playbook. You should see that the first task, which runs the `command` module, reports `changed`, even though it did not change the remote system; it only collected information about the time. That is because the `command` module cannot tell the difference between a command that collects data and a command that changes state.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ***************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Check local time] ********************************************************
changed: [servera.lab.example.com]

TASK [Print local time] ********************************************************
ok: [servera.lab.example.com] => {
    "command_result.stdout": "mié mar 27 08:07:08 EDT 2019"
}

TASK [Install httpd package] ***************************************************
ok: [servera.lab.example.com]

TASK [Start mariadb service] ***************************************************
ok: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=5    changed=1    unreachable=0    failed=0
```

   If you run the playbook again, the `Check local time` task returns `changed` again.

   9.3. That `command` task should not report `changed` every time it runs because it is not changing the managed host. Because you know that the task will never change a managed host, add the line `changed_when: false` to the task to suppress the change.

```
  tasks:
    - name: Check local time
      command: date
      register: command_result
      changed_when: false

    - name: Print local time
      debug:
        var: command_result.stdout
```

9.4. Run the playbook again and notice that the task now reports `ok`, but the task is still being run and is still saving the time in the variable.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ***************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Check local time] ********************************************************
ok: [servera.lab.example.com]

TASK [Print local time] ********************************************************
ok: [servera.lab.example.com] => {
    "command_result.stdout": "mié mar 27 08:08:36 EDT 2019"
}

TASK [Install httpd package] ***************************************************
ok: [servera.lab.example.com]

TASK [Start mariadb service] ***************************************************
ok: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=5    changed=0    unreachable=0    failed=0
```

0. As a final exercise, edit the playbook to explore how the `failed_when` keyword interacts with tasks.

10.1. Edit the `Install {{ web_package }} package` task so that it reports as having failed when `web_package` has the value `httpd`. Because this is the case, the task will report failure when you run the play.

Be careful with your indentation to make sure the keyword is correctly set on the task.

```
    - block:
        - name: Install {{ web_package }} package
          yum:
            name: "{{ web_package }}"
            state: present
          failed_when: web_package == "httpd"
```

10.2. Run the playbook.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ***************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Check local time] ********************************************************
ok: [servera.lab.example.com]

TASK [Print local time] ********************************************************
ok: [servera.lab.example.com] => {
    "command_result.stdout": "mié mar 27 08:09:35 EDT 2019"
}

TASK [Install httpd package] ***************************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failed_when_result": true, "msg": "Nothing to d
o", "rc": 0, "results": ["Installed: httpd"]}

TASK [Install mariadb-server package] ******************************************
ok: [servera.lab.example.com]

TASK [Start mariadb service] ***************************************************
ok: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=5    changed=0    unreachable=0    failed=1
```

Look carefully at the output. The `Install httpd package` task *reports* that it failed, but it actually ran and made sure the package is installed first. The `failed_when` keyword changes the status the task reports *after* the task runs; it does not change the behavior of the task itself.

However, the reported failure might change the behavior of the rest of the play. Because that task was in a block and reported that it failed, the `Install mariadb-server package` task in the block's `rescue` section was run.

Finish

On `workstation`, run the `lab control-errors finish` script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab control-errors finish
```

This concludes the guided exercise.

← PREVIOUS (/ROL/APP/COURSES/RH294-8.4/PAGES/CH04S05) → NEXT (/ROL/APP/COURSES/RH294-8.4/PAGES/CH04S07)