Notify & Handlers in Ansible

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

Session - 18 Agenda:

1. Notify & Handlers

NOTIFY & HANDLERS:

Let's understand this with the help of an example, suppose we have two tasks:

1. Change in the configuration file.

2. Restart the service.

The second task (restart the service) should be executed only when there is a change in the configuration file (first task). Here the second task is dependent on the first task. For such cases we use notifiers and handlers in Ansible. We don't prefer when statement for such tasks because it makes our playbook complex and lengthy. Similarly notify & handlers can be used for such more tasks like, updating the package and restarting the service, patching and rebooting the managed nodes etc.

Handlers: running operations on change

Sometimes you want a task to run only when a change is made on a machine. For example, you may want to restart a service if a task updates the configuration of that service, but not if the configuration is unchanged. Ansible uses handlers to address this use case. Handlers are tasks that only run when notified.

Notifying handlers:

Tasks can instruct one or more handlers to execute using the notify keyword. The notify keyword can be applied to a task and accepts a list of handler names that are notified on a task change. Alternately, a string containing a single handler name can be supplied as well.

Using Ansible Handlers:

As earlier mentioned, handlers are just like other tasks in a playbook, the difference being that they are triggered using the notify directive, and are only run when there is a change of state.

To get the most out of using handlers, here are some points to keep in mind.

a) A handler should have a globally unique name within the playbook.
b) If multiple handlers are defined with the same name, only the first one will be called. The remaining handlers will be ignored.
c) Handlers always run in the order in which they are defined in the handlers section, and not in the notify section.
d) If the state of a task remains unchanged, the handler will not run. As such, handlers help in achieving idempotency. Hence a handler task only runs if there is a change in state, else it doesn't.
e) To define a handler, the notify and handlers directives are used. The notify directive triggers the execution of the task(s) specified in the handlers section.

Let us now explore how to define and call Ansible handlers.

Notify & Handlers in Ansible

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

A single task and a handler:

Consider the following playbook that installs Apache and starts it on a target system.

```
$ vim singletask_handler.yml
---
- name: Install Apache on RHEL server
  hosts: node1
  become: true
  tasks:
    - name: Install the latest version of Apache
      dnf:
        name: httpd
        state: latest
      notify:
       - Start Apache

  handlers:
    - name: Start Apache
      service:
        name: httpd
        state: started
...
```

Let's execute the playbook now.

```
$ ansible-playbook singletask_handler.yml
```

We can verify the job by running the ansible ad-hoc command to check the httpd package.

```
$ ansible node1 -m command -a 'rpm -qi httpd'
```

The playbook consists of a regular task and a handler. The regular task installs the Apache HTTP server on the target system. Once installed the notify directive calls the handler task which starts the Apache service. During playbook runtime, the regular task is executed first followed by the handler. If the playbook is executed again, the handler task will not run for the simple reason that both the state and configuration of the Apache service remain unchanged.

Multiple tasks and handlers:

In most cases, you will be dealing with multiple tasks which might require multiple handlers. The following playbook contains two regular tasks and two handlers broken down as follows:

1. Regular tasks:

a) Installing the latest version of Apache

b) Configuring Apache

2. Handlers:

a) Starting Apache Service

b) Configuring the firewall to allow the incoming http traffic.

Let's create an ansible playbook for the same.

```
$ vim multitask_handler.yml
```

Notify & Handlers in Ansible

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

```
---
- name: Install Apache on RHEL server
  hosts: node1
  become: true
  tasks:
    - name: Install the latest version of Apache
      dnf:
        name: httpd
        state: latest
    - name: Configure Apache
      copy:
        content: "Nehra Classes Are Awesome."
        dest: /var/www/html/index.html
        owner: apache
        group: apache
        mode: 0644
      notify:
      - Configure Firewall
      - Start Apache
  handlers:
    - name: Start Apache
      service:
        name: httpd
        state: started
    - name: Configure Firewall
      firewalld:
        permanent: yes
        immediate: yes
        service: http
        state: enabled
...
```

Now check the playbook for errors.

$ ansible-playbook multitask_handler.yml --syntax-check

In case if firewalld module is not present in your ansible environment, you can install it manually from ansible galaxy by running the below command.

$ ansible-galaxy collection install ansible.posix

Now let's execute the playbook.

$ ansible-playbook multitask_handler.yml

Verify the tasks:

$ curl node1
$ ansible node1 -m command -a 'rpm -qi httpd'
$ ansible node1 -m command -a 'cat /var/www/html/index.html'
$ ansible node1 -m command -a 'sudo firewall-cmd --list-all'

Notify & Handlers in Ansible

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

The first task installs Apache while the second one configures Apache by making changes to the default index.html file and applying the required group, user ownership, and file permissions.

During playbook runtime, the 'Start Apache' handler is executed first followed by the 'Configure Firewall' handler despite the latter appearing first in the notify section. As mentioned earlier, the order of execution is determined by the handlers section and not the notify section.

Grouping Handlers Using the 'listen' Directive:
Alternatively, you can group handlers using the listen keyword and call them using a single notify statement. In the following playbook, the notify directive is set to restart services which triggers all the handler tasks bearing the listen directive.

$ vim grouping_handlers.yml

```
---
- name: Grouping the handlers.
  hosts: node1
  become: true
  tasks:
    - name: Restart services on remote target
      command: echo "This task restarts the services."
      notify: "restart services"
  handlers:
    - name: Restart Apache
      service:
        name: httpd
        state: restarted
      listen: "restart services"
    - name: Restart Rsyslog
      service:
        name: rsyslog
        state: restarted
      listen: "restart services"
...
```

Let's execute the playbook.
$ ansible-playbook grouping_handlers.yml

Determine When Handlers Run:
By default, handlers run at the end of the play once all the regular tasks have been executed. If you want handlers to run before the end of the play or between tasks, add a meta task to flush them using the meta module. The meta task is defined as follows.

```
- name: Flush handlers
  meta: flush_handlers
```

The meta: flush_handlers task calls any handlers that have been notified at that point in the play.

Notify & Handlers in Ansible

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

The following playbook performs the following tasks in order:
1. Installation of EPEL package (Extra Package for Enterprise Linux)
2. Installation of Nginx web server
3. Installation of Neofetch

Let's create an ansible playbook to control when handlers run using meta directive.

$ vim handlers_meta.yml

```
---
- name: Control when handlers run using meta directive
  hosts: node3
  become: true
  tasks:
    - name: Adding the EPEL repository
      command: sudo dnf install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
    - name: Installing the Nginx web server
      dnf:
        name: nginx
        state: latest
      notify:
      - Start Nginx
    - name: Flush handlers
      meta: flush_handlers
    - name: Install Neofetch
      dnf:
        name: neofetch
        state: latest
  handlers:
    - name: Start Nginx
      service:
        name: nginx
        state: started
...
```

Let's execute the playbook.

$ ansible-playbook handlers_meta.yml

The meta: flush_handlers meta task triggers the Start Nginx handler to be executed at that point in the play, the remaining task follows suit. From the output of the playbook runtime, you can see that the handler runs just before the last task.

Verify the tasks using ansible ad-hoc commands.

$ ansible node3 -m command -a "dnf repolist all"
$ ansible node3 -m command -a "rpm -qi nginx"
$ ansible node3 -m command -a "neofetch"

Handling task errors:

By default, when a task fails in a particular host, subsequent tasks are not executed on that host and the playbook exits with an error. Consequently, the handler tasks are also ignored

and do not get executed.

Let's take a simple playbook example to illustrate this. The following playbook contains two tasks. The first task defined by the shell module is set to fail using the /bin/false argument. The second task, however, is configured to run successfully using the /bin/false argument.

$ vim fail_task.yml

```
---
- name: Simulate a task to fail on a host.
  hosts: node1
  tasks:
    - name: Set a task to fail
      shell: /bin/false
    - name: Set a task to run successfully
      shell: /bin/true
      notify: success
  handlers:
    - name: success
      debug:
        msg: "This task has been executed successfully."
...
```

Let's execute the playbook.

$ ansible-playbook fail_task.yml

As expected, the playbook executes and fails upon encountering an error in the first task. To ignore failed tasks, use the ignore_errors: yes property in the playbook as follows.

$ vim fail_task.yml

```
---
- name: Simulate a task to fail on a host.
  hosts: node1
  ignore_errors: yes
  tasks:
    - name: Set a task to fail
      shell: /bin/false
    - name: Set a task to run successfully
      shell: /bin/true
      notify: success
  handlers:
    - name: success
      debug:
        msg: "This task has been executed successfully."
...
```

Let's execute the playbook again.

$ ansible-playbook fail_task.yml

This time around, the failed task is ignored and the playbook proceeds to run the remaining tasks. The ignored flag section at the very bottom of the output displays the number of tasks that have been skipped due to failures or errors.