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.

Handler example

This playbook, `verify-apache.yml`, contains a single play with a handler.

```
---
- name: Verify apache installation
  hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
    - name: Ensure apache is at the latest version
      ansible.builtin.yum:
        name: httpd
        state: latest

    - name: Write the apache config file
      ansible.builtin.template:
        src: /srv/httpd.j2
        dest: /etc/httpd.conf
      notify:
      - Restart apache

    - name: Ensure apache is running
      ansible.builtin.service:
        name: httpd
        state: started

  handlers:
    - name: Restart apache
      ansible.builtin.service:
        name: httpd
        state: restarted
```

In this example playbook, the Apache server is restarted by the handler after all tasks complete in the play.

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. The following example demonstrates how multiple handlers can be notified by a single task:

```
tasks:
- name: Template configuration file
  ansible.builtin.template:
    src: template.j2
    dest: /etc/foo.conf
  notify:
    - Restart apache
    - Restart memcached

handlers:
  - name: Restart memcached
    ansible.builtin.service:
      name: memcached
      state: restarted

  - name: Restart apache
    ansible.builtin.service:
      name: apache
      state: restarted
```

In the above example the handlers are executed on task change in the following order: `Restart memcached`, `Restart apache`. Handlers are executed in the order they are defined in the `handlers` section, not in the order listed in the `notify` statement. Notifying the same handler multiple times will result in executing the handler only once regardless of how many tasks notify it. For example, if multiple tasks update a configuration file and notify a handler to restart Apache, Ansible only bounces Apache once to avoid unnecessary restarts.

Naming handlers

Handlers must be named in order for tasks to be able to notify them using the `notify` keyword.

Alternately, handlers can utilize the `listen` keyword. Using this handler keyword, handlers can listen on topics that can group multiple handlers as follows:

```
tasks:
  - name: Restart everything
    command: echo "this task will restart the web services"
    notify: "restart web services"

handlers:
  - name: Restart memcached
    service:
      name: memcached
      state: restarted
    listen: "restart web services"

  - name: Restart apache
    service:
      name: apache
      state: restarted
    listen: "restart web services"
```

Notifying the `restart web services` topic results in executing all handlers listening to that topic regardless of how those handlers are named.

This use makes it much easier to trigger multiple handlers. It also decouples handlers from their names, making it easier to share handlers among playbooks and roles (especially when using third-party roles from a shared source such as Ansible Galaxy).

Each handler should have a globally unique name. If multiple handlers are defined with the same name, only the last one defined is notified with `notify`, effectively shadowing all of the previous handlers with the same name. Alternately handlers sharing the same name can all be notified and executed if they listen on the same topic by notifying that topic.

There is only one global scope for handlers (handler names and listen topics) regardless of where the handlers are defined. This also includes handlers defined in roles.

Controlling when handlers run

By default, handlers run after all the tasks in a particular play have been completed. Notified handlers are executed automatically after each of the following sections, in the following order: `pre_tasks`, `roles` / `tasks` and `post_tasks`. This approach is efficient, because the handler only runs once, regardless of how many tasks notify it. For example, if multiple tasks update a configuration file and notify a handler to restart Apache, Ansible only bounces Apache once to avoid unnecessary restarts.

If you need handlers to run before the end of the play, add a task to flush them using the meta module (../collections/ansible/builtin/meta_module.html#meta-module), which executes Ansible actions:

```
tasks:
  - name: Some tasks go here
    ansible.builtin.shell: ...

  - name: Flush handlers
    meta: flush_handlers

  - name: Some other tasks
    ansible.builtin.shell: ...
```

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

Once handlers are executed, either automatically after each mentioned section or manually by the `flush_handlers` meta task, they can be notified and run again in later sections of the play.

Using variables with handlers

You may want your Ansible handlers to use variables. For example, if the name of a service varies slightly by distribution, you want your output to show the exact name of the restarted service for each target machine. Avoid placing variables in the name of the handler. Since handler names are templated early on, Ansible may not have a value available for a handler name like this:

```
handlers:
# This handler name may cause your play to fail!
- name: Restart "{{ web_service_name }}"
```

If the variable used in the handler name is not available, the entire play fails. Changing that variable mid-play **will not** result in newly created handler.

Instead, place variables in the task parameters of your handler. You can load the values using `include_vars` like this:

```
tasks:
  - name: Set host variables based on distribution
    include_vars: "{{ ansible_facts.distribution }}.yml"

handlers:
  - name: Restart web service
    ansible.builtin.service:
      name: "{{ web_service_name | default('httpd') }}"
      state: restarted
```

While handler names can contain a template, `listen` topics cannot.

Handlers in roles

Handlers from roles are not just contained in their roles but rather inserted into global scope with all other handlers from a play. As such they can be used outside of the role they are defined in. It also means that their name can conflict with handlers from outside the role. To ensure that a handler from a role is notified as opposed to one from outside the role with the same name, notify the handler by using its name in the following form: `role_name : handler_name`.

Handlers notified within the `roles` section are automatically flushed at the end of the `tasks` section, but before any `tasks` handlers.

Includes and imports in handlers

Notifying a dynamic include such as `include_task` as a handler results in executing all tasks from within the include. It is not possible to notify a handler defined inside a dynamic include.

Having a static include such as `import_task` as a handler results in that handler being effectively rewritten by handlers from within that import before the play execution. A static include itself cannot be notified; the tasks from within that include, on the other hand, can be notified individually.

Meta tasks as handlers

Since Ansible 2.14 meta tasks (../collections/ansible/builtin/meta_module.html#ansible-collections-ansible-builtin-meta-module) are allowed to be used and notified as handlers. Note that however `flush_handlers` cannot be used as a handler to prevent unexpected behavior.

Limitations

A handler cannot run `import_role` or `include_role`.