Use Ansible network roles

Roles are sets of Ansible defaults, files, tasks, templates, variables, and other Ansible components that work together. As you saw on :ref:`first_network_playbook`, moving from a command to a playbook makes it easy to run multiple tasks and repeat the same tasks in the same order. Moving from a playbook to a role makes it even easier to reuse and share your ordered tasks. You can look at :ref:`Ansible Galaxy <ansible_galaxy>`, which lets you share your roles and use others' roles, either directly or as inspiration.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\getting_started\(ansible-devel)(docs)(docsite)(rst)(network)(getting_started)network_roles.rst, line 8); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\getting_started\(ansible-devel)(docs)(docsite)(rst)(network)(getting_started)network_roles.rst, line 8); *backlink*

Unknown interpreted text role "ref".

Understanding roles

So what exactly is a role, and why should you care? Ansible roles are basically playbooks broken up into a known file structure. Moving to roles from a playbook makes sharing, reading, and updating your Ansible workflow easier. Users can write their own roles. So for example, you don't have to write your own DNS playbook. Instead, you specify a DNS server and a role to configure it for you.

To simplify your workflow even further, the Ansible Network team has written a series of roles for common network use cases. Using these roles means you don't have to reinvent the wheel. Instead of writing and maintaining your own `create_vlan` playbooks or roles, you can concentrate on designing, codifying and maintaining the parser templates that describe your network topologies and inventory, and let Ansible's network roles do the work. See the network-related roles on Ansible Galaxy.

A sample DNS playbook

To demonstrate the concept of what a role is, the example `playbook.yml` below is a single YAML file containing a two-task playbook. This Ansible Playbook configures the hostname on a Cisco IOS XE device, then it configures the DNS (domain name system) servers.

```
---
- name: configure cisco routers
  hosts: routers
  connection: ansible.netcommon.network_cli
  gather_facts: no
  vars:
    dns: "8.8.8.8 8.8.4.4"

  tasks:
   - name: configure hostname
     cisco.ios.ios_config:
       lines: hostname {{ inventory_hostname }}

   - name: configure DNS
     cisco.ios.ios_config:
       lines: ip name-server {{dns}}
```

If you run this playbook using the `ansible-playbook` command, you'll see the output below. This example used `-l` option to limit the playbook to only executing on the **rtr1** node.

```
[user@ansible ~]$ ansible-playbook playbook.yml -l rtr1

PLAY [configure cisco routers] *************************************************
```

```
TASK [configure hostname] ***********************************************
changed: [rtr1]

TASK [configure DNS] ****************************************************
changed: [rtr1]

PLAY RECAP **************************************************************
rtr1                       : ok=2    changed=2    unreachable=0    failed=0
```

This playbook configured the hostname and DNS servers. You can verify that configuration on the Cisco IOS XE **rtr1** router:

```
rtr1#sh run | i name
hostname rtr1
ip name-server 8.8.8.8 8.8.4.4
```

Convert the playbook into a role

The next step is to convert this playbook into a reusable role. You can create the directory structure manually, or you can use `ansible-galaxy init` to create the standard framework for a role.

```
[user@ansible ~]$ ansible-galaxy init system-demo
[user@ansible ~]$ cd system-demo/
[user@ansible system-demo]$ tree
.
â”œâ”€â”€ defaults
â”‚   â”œâ”€â”€ main.yml
â”œâ”€â”€ files
â”œâ”€â”€ handlers
â”‚   â”œâ”€â”€ main.yml
â”œâ”€â”€ meta
â”‚   â”œâ”€â”€ main.yml
â”œâ”€â”€ README.md
â”œâ”€â”€ tasks
â”‚   â”œâ”€â”€ main.yml
â”œâ”€â”€ templates
â”œâ”€â”€ tests
â”‚   â”œâ”€â”€ inventory
â”‚   â”œâ”€â”€ test.yml
â”œâ”€â”€ vars
  â”œâ”€â”€ main.yml
```

This first demonstration uses only the **tasks** and **vars** directories. The directory structure would look as follows:

```
[user@ansible system-demo]$ tree
.
â”œâ”€â”€ tasks
â”‚   â”œâ”€â”€ main.yml
â”œâ”€â”€ vars
    â”œâ”€â”€ main.yml
```

Next, move the content of the `vars` and `tasks` sections from the original Ansible Playbook into the role. First, move the two tasks into the `tasks/main.yml` file:

```
[user@ansible system-demo]$ cat tasks/main.yml
---
- name: configure hostname
  cisco.ios.ios_config:
    lines: hostname {{ inventory_hostname }}

- name: configure DNS
  cisco.ios.ios_config:
    lines: ip name-server {{dns}}
```

Next, move the variables into the `vars/main.yml` file:

```
[user@ansible system-demo]$ cat vars/main.yml
---
dns: "8.8.8.8 8.8.4.4"
```

Finally, modify the original Ansible Playbook to remove the `tasks` and `vars` sections and add the keyword `roles` with the name of the role, in this case `system-demo`. You'll have this playbook:

```
---
- name: configure cisco routers
  hosts: routers
  connection: ansible.netcommon.network_cli
  gather_facts: no

  roles:
```

```
    - system-demo
```

To summarize, this demonstration now has a total of three directories and three YAML files. There is the `system-demo` folder, which represents the role. This `system-demo` contains two folders, `tasks` and `vars`. There is a `main.yml` is each respective folder. The `vars/main.yml` contains the variables from `playbook.yml`. The `tasks/main.yml` contains the tasks from `playbook.yml`. The `playbook.yml` file has been modified to call the role rather than specifying vars and tasks directly. Here is a tree of the current working directory:

```
[user@ansible ~]$ tree
.
â”œâ”€â”€ playbook.yml
â””â”€â”€ system-demo
    â”œâ”€â”€ tasks
    â”‚   â””â”€â”€ main.yml
    â””â”€â”€ vars
        â””â”€â”€ main.yml
```

Running the playbook results in identical behavior with slightly different output:

```
[user@ansible ~]$ ansible-playbook playbook.yml -l rtr1

PLAY [configure cisco routers] *************************************************

TASK [system-demo : configure hostname] ****************************************
ok: [rtr1]

TASK [system-demo : configure DNS] *********************************************
ok: [rtr1]

PLAY RECAP *********************************************************************
rtr1             : ok=2    changed=0    unreachable=0    failed=0
```

As seen above each task is now prepended with the role name, in this case `system-demo`. When running a playbook that contains several roles, this will help pinpoint where a task is being called from. This playbook returned `ok` instead of `changed` because it has identical behavior for the single file playbook we started from.

As before, the playbook will generate the following configuration on a Cisco IOS-XE router:

```
rtr1#sh run | i name
hostname rtr1
ip name-server 8.8.8.8 8.8.4.4
```

This is why Ansible roles can be simply thought of as deconstructed playbooks. They are simple, effective and reusable. Now another user can simply include the `system-demo` role instead of having to create a custom "hard coded" playbook.

Variable precedence

What if you want to change the DNS servers? You aren't expected to change the `vars/main.yml` within the role structure. Ansible has many places where you can specify variables for a given play. See :ref:`playbooks_variables` for details on variables and precedence. There are actually 21 places to put variables. While this list can seem overwhelming at first glance, the vast majority of use cases only involve knowing the spot for variables of least precedence and how to pass variables with most precedence. See :ref:`ansible_variable_precedence` for more guidance on where you should put variables.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\getting_started\(ansible-devel)(docs)(docsite)(rst)(network)(getting_started)network_roles.rst, line 190); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\getting_started\(ansible-devel)(docs)(docsite)(rst)(network)(getting_started)network_roles.rst, line 190); *backlink*

Unknown interpreted text role "ref".

Lowest precedence

The lowest precedence is the `defaults` directory within a role. This means all the other 20 locations you could potentially specify the variable will all take higher precedence than `defaults`, no matter what. To immediately give the vars from the `system-demo` role the least precedence, rename the `vars` directory to `defaults`.

```
[user@ansible system-demo]$ mv vars defaults
[user@ansible system-demo]$ tree
.
```

```
â”œâ”€â”€ defaults
â”‚   â””â”€â”€ main.yml
â”œâ”€â”€ tasks
â”‚   â””â”€â”€ main.yml
```

Add a new `vars` section to the playbook to override the default behavior (where the variable `dns` is set to 8.8.8.8 and 8.8.4.4). For this demonstration, set `dns` to 1.1.1.1, so `playbook.yml` becomes:

```
---
- name: configure cisco routers
  hosts: routers
  connection: ansible.netcommon.network_cli
  gather_facts: no
  vars:
    dns: 1.1.1.1
  roles:
    - system-demo
```

Run this updated playbook on **rtr2**:

```
[user@ansible ~]$ ansible-playbook playbook.yml -l rtr2
```

The configuration on the **rtr2** Cisco router will look as follows:

```
rtr2#sh run | i name-server
ip name-server 1.1.1.1
```

The variable configured in the playbook now has precedence over the `defaults` directory. In fact, any other spot you configure variables would win over the values in the `defaults` directory.

Highest precedence

Specifying variables in the `defaults` directory within a role will always take the lowest precedence, while specifying `vars` as extra vars with the `-e` or `--extra-vars=` will always take the highest precedence, no matter what. Re-running the playbook with the `-e` option overrides both the `defaults` directory (8.8.4.4 and 8.8.8.8) as well as the newly created `vars` within the playbook that contains the 1.1.1.1 dns server.

```
[user@ansible ~]$ ansible-playbook playbook.yml -e "dns=192.168.1.1" -l rtr3
```

The result on the Cisco IOS XE router will only contain the highest precedence setting of 192.168.1.1:

```
rtr3#sh run | i name-server
ip name-server 192.168.1.1
```

How is this useful? Why should you care? Extra vars are commonly used by network operators to override defaults. A powerful example of this is with the Job Template Survey feature on AWX or the :ref:`ansible_platform`. It is possible through the web UI to prompt a network operator to fill out parameters with a Web form. This can be really simple for non-technical playbook writers to execute a playbook using their Web browser.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\getting_started\(ansible-devel)(docs)(docsite)(rst)(network)(getting_started)network_roles.rst, line 252); *backlink***
>
> Unknown interpreted text role "ref".

Update an installed role

The Ansible Galaxy page for a role lists all available versions. To update a locally installed role to a new or different version, use the `ansible-galaxy install` command with the version and `--force` option. You may also need to manually update any dependent roles to support this version. See the role **Read Me** tab in Galaxy for dependent role minimum version requirements.

```
[user@ansible]$ ansible-galaxy install mynamespace.my_role,v2.7.1 --force
```

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\getting_started\(ansible-devel)(docs)(docsite)(rst)(network)(getting_started)network_roles.rst, line 264)**
>
> Unknown directive type "seealso".
>
> ```
> .. seealso::
>
>        `Ansible Galaxy documentation <https://galaxy.ansible.com/docs/>`_
>            Ansible Galaxy user guide
> ```