Ansible Roles

Raúl Estrada

Octubre 2020

Roles

We have been working on few Ansible playbooks, and as you can imagine, there is a lot that can be abstracted from them into generic units of work. As of now, with our current knowledge of Ansible, the best thing we can do is use a naming convention for playbooks and files so that we don't mix them, but Ansible provides a better approach to this: roles.

Think of roles as common reusable capabilities as you do with modules in software: a highly cohesive set of playbooks, variables, and resources that work together for one purpose. For example, if we are managing `nginx` , it makes sense to have all the related resources in a single module (role, in this case) in order to improve the reusability as well as clarity of the code.

One option would be including playbooks using Ansible features. Although we did not talk about it, with Ansible, it is possible to include YAML files with tasks to create dependencies, as shown in the following snippets:

```
---
- include: play-include.yml
- hosts: all
 user: root
 tasks:
 - debug: msg="I am the main playbook"
 - include: tasks-include.yml
```

Let's explain what is going on. We can see two files included:

- The first include is what Ansible calls a play include. It is a fully functional playbook as is, which gets included in another playbook.
- The second include is what Ansible calls a task include. It only includes a list of tasks.

This can be explained easily by looking at the content of the two files. First, look at the content of `play-include.yml` :

Copy

```
---
- hosts: all
 user: root
 tasks:
 - debug: msg="I am a play include"
```

Second, look at the content of `tasks-include.yml` :

Copy

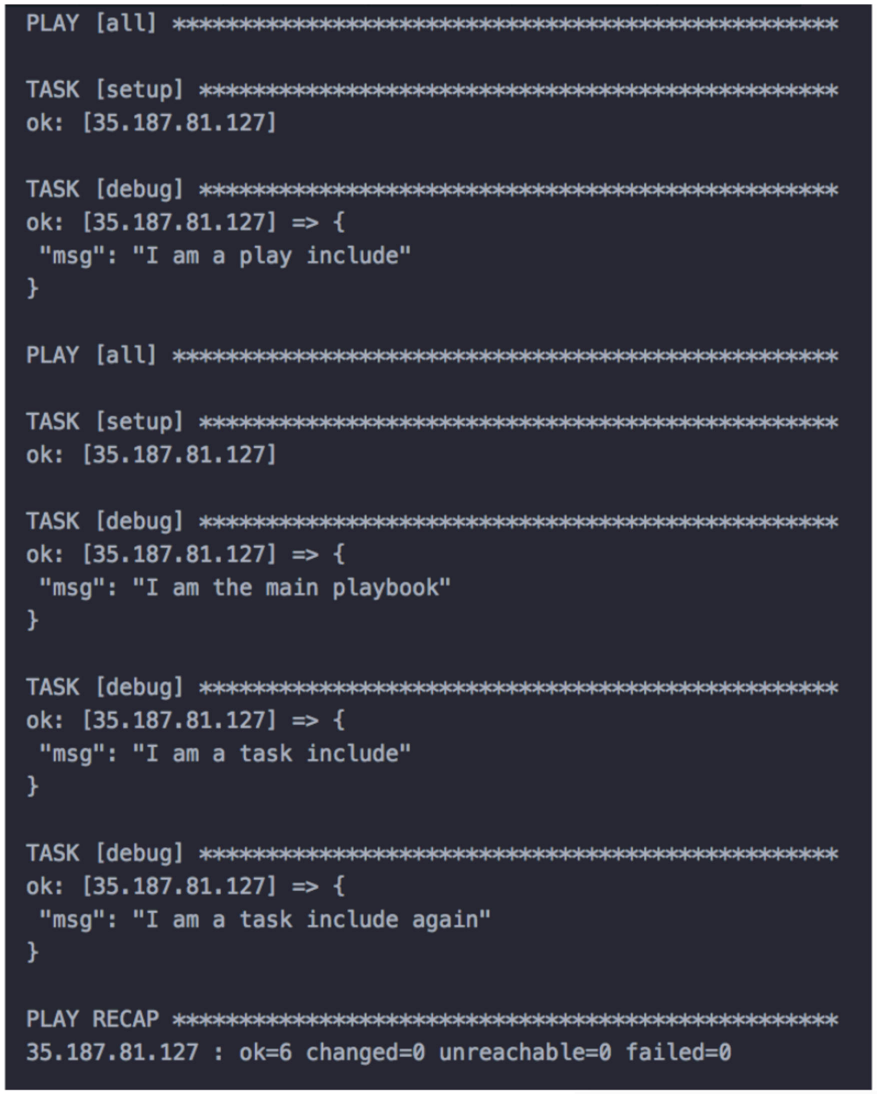
```
---
- debug: msg="I am a task include"
- debug: msg="I am a task include again"
```

Now we are going to execute the playbooks from earlier and see what the output is. Save the content of the first playbook on a file called `tasks.yml` and use the same inventory as on all the examples from earlier. Now run the following command:

Copy

```
ansible-playbook -i inventory tasks.yml
```

Once the execution has finished, let's examine the output, which should be very similar to the following one:



```
PLAY [all] ***************************************************

TASK [setup] *************************************************
ok: [35.187.81.127]

TASK [debug] *************************************************
ok: [35.187.81.127] => {
 "msg": "I am a play include"
}

PLAY [all] ***************************************************

TASK [setup] *************************************************
ok: [35.187.81.127]

TASK [debug] *************************************************
ok: [35.187.81.127] => {
 "msg": "I am the main playbook"
}

TASK [debug] *************************************************
ok: [35.187.81.127] => {
 "msg": "I am a task include"
}

TASK [debug] *************************************************
ok: [35.187.81.127] => {
 "msg": "I am a task include again"
}

PLAY RECAP ***************************************************
35.187.81.127 : ok=6 changed=0 unreachable=0 failed=0
```

Let's explain this:

1. The play include (`play-include.yml`) gets executed by outputting the debug message in there.
2. The debug task in the main playbook gets executed.
3. The task includes (`tasks-include.yml`) gets executed by executing the two debug messages included there.

It is not very complicated, but it gets easier if you play around a bit with the playbooks.

Although the preceding example can lead to a very clean and reusable set of files, there is a much better way of doing this: using roles. Roles are isolated sets of functionalities that allow an easy maintenance cycle like any other software component.

Following the preceding example, we can rewrite it using three roles:

- The play include (`play-include.yml`)
- The main tasks (`tasks.yml`)
- The tasks include (`tasks-include.yml`)

In order to start creating roles, first, create a new folder called `ansible-roles` and a folder called `roles` inside the same one. One thing that was not mentioned earlier is the fact that it is a good practice to create a set of folders to hold Ansible resources: tasks folders to hold the tasks, files folder to store all the files that need to be transferred to the remote hosts, and so on. In general, I agree with this setup, but for the examples, we just simplified it in order to make everything easier. For roles, this setup is mandatory. We need to create the folders as appropriated. In this case, as we are only going to use tasks to demonstrate how roles work; we will create the folder tasks inside of every role because otherwise, we won't execute the tasks from the role.

Inside the **roles** folder, we are going to create another folder called `play-include` , which is going to be the equivalent to `play-include.yml` from the preceding example but in the form of a role.

Now it is time to create our first role playbook: create a file called `main.yml` and place it inside the `play-include/tasks/` folder. This is the content of the `main.yml file` :

```
---
- debug: msg="I am a play include"
```

Now it is time to add a second role called `main-tasks` by creating a folder in **roles** and adding a file called `main.yml` inside of `roles/main-tasks/tasks` :

Copy

```
---
- debug: msg="I am the main playbook"
```

And our third and last role is called `tasks-include`. Just create the folder as earlier (inside the roles folder) and add a file called `main.yml` to it inside of the tasks folder:

Copy

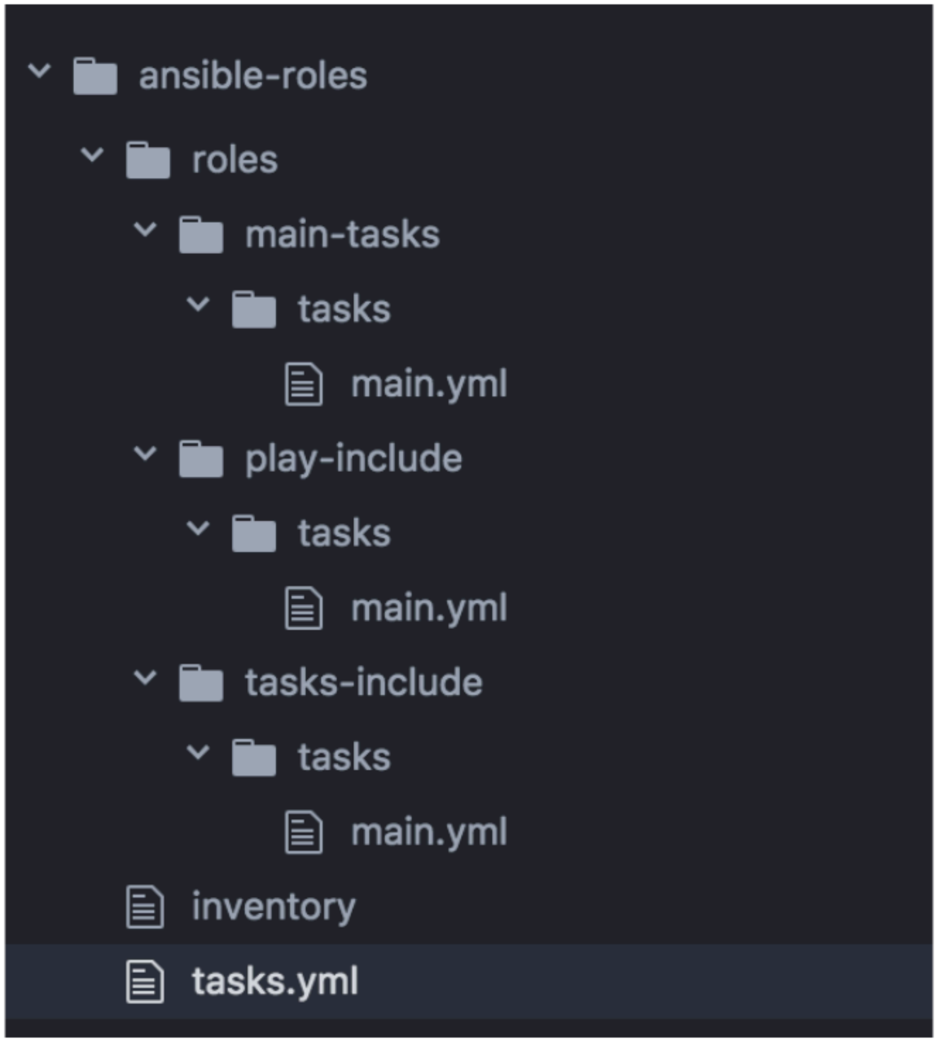
```
---
- debug: msg="I am a task include"
- debug: msg="I am a task include again"
```

And that's it. You have created three roles that can be reused across different Ansible projects. Now it is time to use them. Create a file called `tasks.yml` in the root folder of your project (in my case, `ansible-roles`) and add the following content:

Copy

```
---
- hosts: all
 user: root
 roles:
 - main-tasks
 - play-include
 - tasks-include
```

This is how your project should look after adding all the files from earlier:



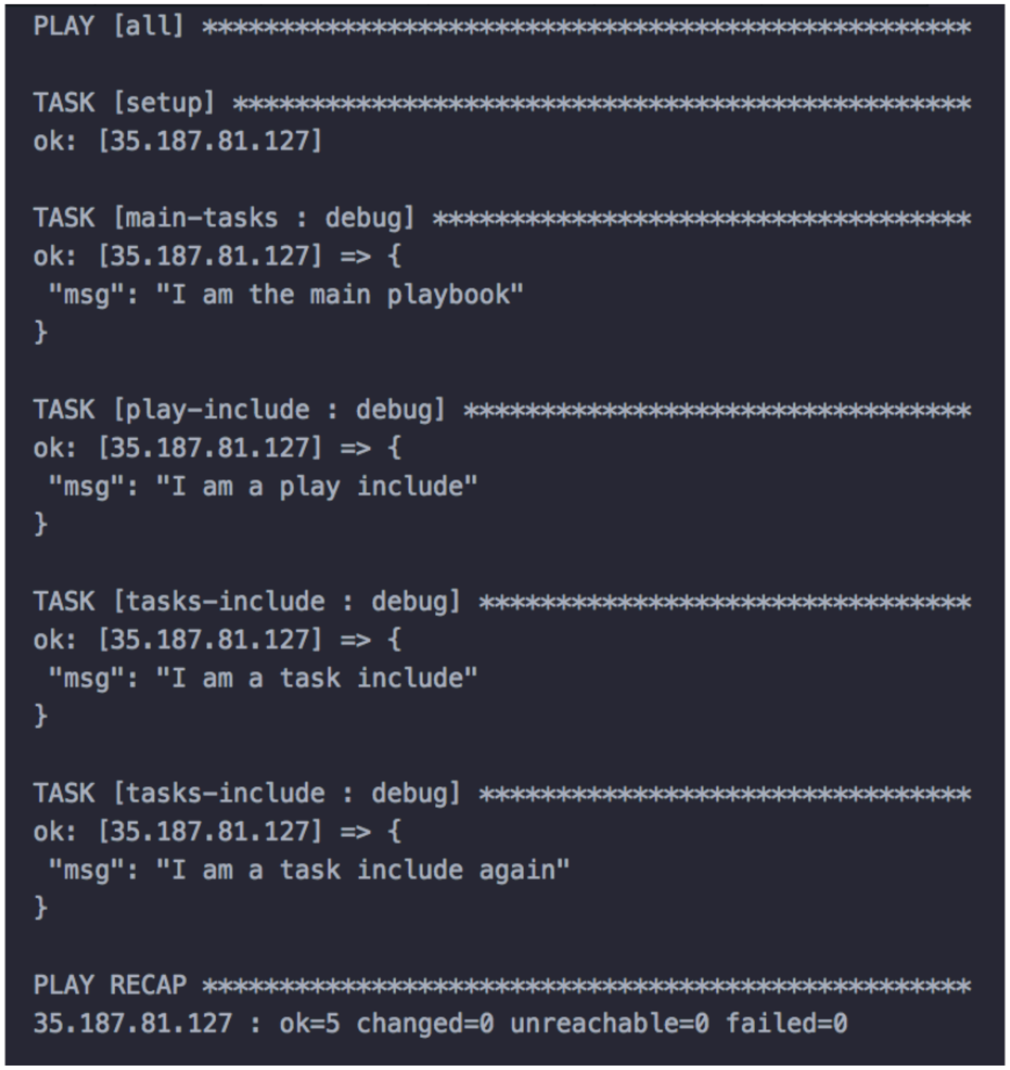
```
ansible-roles
  roles
    main-tasks
      tasks
        main.yml
    play-include
      tasks
        main.yml
    tasks-include
      tasks
        main.yml
  inventory
  tasks.yml
```

The inventory is the same one as the previous examples (remember, the recommendation was to reuse the same VM). Now it is time to run our playbook:

Copy

```
ansible-playbook -i inventory tasks.yml
```

This will produce output similar to the following one:



```
PLAY [all] **************************************************

TASK [setup] ************************************************
ok: [35.187.81.127]

TASK [main-tasks : debug] ***********************************
ok: [35.187.81.127] => {
 "msg": "I am the main playbook"
}

TASK [play-include : debug] *********************************
ok: [35.187.81.127] => {
 "msg": "I am a play include"
}

TASK [tasks-include : debug] ********************************
ok: [35.187.81.127] => {
 "msg": "I am a task include"
}

TASK [tasks-include : debug] ********************************
ok: [35.187.81.127] => {
 "msg": "I am a task include again"
}

PLAY RECAP **************************************************
35.187.81.127 : ok=5 changed=0 unreachable=0 failed=0
```

If we compare the output from the previous example, we can see that it is virtually the same except for the legend of the task, which indicates the role that the task is coming from.

In roles, we can also define variables and access to the variables defined in the global scope as well as many other features. As stated earlier, Ansible is big enough to write an entire book just on it, so we are scratching the surface of the important parts (under my criteria). As usual, the documentation in Ansible is pretty good, and if you want to learn more about roles, the information can be found at https://docs.ansible.com/ansible-container/roles/index.html.

If I can give you some advice regarding Ansible, it would be that you should always try to use roles. It doesn't matter how big or simple your project is; you will find out very soon that the isolation and reusability that roles provide at pretty much no cost are quite beneficial.