How to Limit Execution by Tasks

This is a very important execution strategy where one needs to execute only one execution and not the entire playbook. **For example**, suppose you only want to stop a server (in case a production issue comes) and then post applying a patch you would like to only start the server.

Here in original playbook stop and start were a part of different roles in the same playbook but this can be handled with the usage of tags. We can provide different tags to different roles (which in turn will have tasks) and hence based on the tags provided by the executor only that specified role/task gets executed. So for the above example provided, we can add tags like the following −

```
- {role: start-tomcat, tags: ['install']}}
```

The following command helps in using tags −

```
ansible-playbook -i hosts <your yaml> --tags "install" -vvv
```

With the above command, only the start-tomcat role will be called. The tag provided is case-sensitive. Ensure exact match is being passed to the command.

How to Limit Execution by Hosts

There are two ways to achieve the execution of specific steps on specific hosts. For a specific role, one defines the hosts - as to which specific hosts that specific role should be run.

Example

```
- hosts: <A>
   environment: "{{your env}}"
   pre_tasks:
      - debug: msg = "Started deployment.
      Current time is {{ansible_date_time.date}}
{{ansible_date_time.time}} "

   roles:
      - {role: <your role>, tags: ['<respective tag>']}
   post_tasks:
      - debug: msg = "Completed deployment.
      Current time is {{ansible_date_time.date}}
{{ansible_date_time.time}}"

- hosts: <B>
   pre_tasks:
```

```
      - debug: msg = "started....
      Current time is {{ansible_date_time.date}}
{{ansible_date_time.time}} "

   roles:
      - {role: <your role>, tags: ['<respective tag>']}
   post_tasks:
      - debug: msg = "Completed the task..
      Current time is {{ansible_date_time.date}}
{{ansible_date_time.time}}"
```

As per the above example, depending on the hosts provided, the respective roles will only be called. Now my hosts A and B are defined in the hosts (inventory file).

Alternate Solution

A different solution might be defining the playbook's hosts using a variable, then passing in a specific host address via **--extra-vars** −

```
# file: user.yml   (playbook)
---
- hosts: '{{ target }}'
   user: ...
playbook contd….
```

Running the Playbook

```
ansible-playbook user.yml --extra-vars "target = "<your host
variable>"
```

If {{ target }} isn't defined, the playbook does nothing. A group from the hosts file can also be passed through if need be. This does not harm if the extra vars is not provided.

Playbook targeting a single host

```
$ ansible-playbook user.yml --extra-vars "target = <your hosts
variable>" --listhosts
```

The most common strategies for debugging Ansible playbooks are using the modules given below −

Debug and Register

These two are the modules available in Ansible. For debugging purpose, we need to use the two modules judiciously. Examples are demonstrated below.

Use Verbosity

With the Ansible command, one can provide the verbosity level. You can run the commands with verbosity level one (-v) or two (-vv).

Important Points

In this section, we will go through a few examples to understand a few concepts.

If you are not quoting an argument that starts with a variable. For example,

```
vars:
   age_path: {{vivek.name}}/demo/

{{vivek.name}}
```

This will throw an error.

Solution

```
vars:
   age_path: "{{vivek.name}}/demo/" - marked in yellow is the fix.

How to use register -> Copy this code into a yml file say test.yml
and run it
---
#Tsting
- hosts: tomcat-node
   tasks:

   - shell: /usr/bin/uptime
      register: myvar
      - name: Just debugging usage
         debug: var = myvar
```

When I run this code via the command Ansible-playbook -i hosts test.yml, I get the output as shown below.

If you see the yaml , we have registered the output of a command into a variable – **myvar** and just printed the output.

The text marked yellow, tells us about property of the variable –myvar that can be used for further flow control. This way we can find out about the properties that are exposed of a particular variable. The following debug command helps in this.

```
$ ansible-playbook -i hosts test.yml

PLAY [tomcat-node]
*****************************************************************
***************
*****************************************************************
************** ******************************

TASK [Gathering Facts]
*****************************************************************
*************
*****************************************************************
```

```
************** *************************
Monday 05 February 2018  17:33:14 +0530 (0:00:00.051) 0:00:00.051
*******
ok: [server1]

TASK [command]
*****************************************************************
*************
*****************************************************************
************* *********************************
Monday 05 February 2018  17:33:16 +0530 (0:00:01.697) 0:00:01.748
*******
changed: [server1]

TASK [Just debugging usage]
*****************************************************************
*************
*****************************************************************
************* ********************
Monday 05 February 2018  17:33:16 +0530 (0:00:00.226) 0:00:01.974
*******
ok: [server1] => {
   "myvar": {
      "changed": true,
      "cmd": "/usr/bin/uptime",
      "delta": "0:00:00.011306",
      "end": "2018-02-05 17:33:16.424647",
      "rc": 0,
      "start": "2018-02-05 17:33:16.413341",
      "stderr": "",
      "stderr_lines": [],
      "stdout": " 17:33:16 up 7 days, 35 min,  1 user,  load
average: 0.18, 0.15, 0.14",
      "stdout_lines": [
         " 17:33:16 up 7 days, 35 min,  1 user,  load average:
0.18, 0.15, 0.14"
      ]
   }
}

PLAY RECAP
*****************************************************************
*********
*****************************************************************
***************
 ************************************
server1 : ok = 3    changed = 1    unreachable = 0    failed = 0
```

Common Playbook Issues

In this section, we will learn about the a few common playbook issues. The issues are −

- Quoting
- Indentation

Playbook is written in yaml format and the above two are the most common issues in yaml/playbook.

Yaml does not support tab based indentation and supports space based indentation, so one needs to be careful about the same.

Note − once you are done with writing the yaml , open this site(https://editor.swagger.io/) and copy paste your yaml on the left hand side to ensure that the yaml compiles properly. This is just a tip.

Swagger qualifies errors in warning as well as error.