Managing Playbook

Parallelism

- Parallelism is the word used for Ansible defaults ability to interact with multiple hosts at the same time.
- The parallel processes spawned by Ansible are known as forks.
- The defaults number of forks in Ansible is 5.
- The defaults number of forks can be changed by editing the forks parameter in ansible.cfg
- The number of forks can be changed on a per command basis by using the -f flag.
- Ansible allows for rolling updates using the serial keywords.
- In theory, Ansible could simultaneously connect to all hosts in the play for each task. This works fine for small lists of hosts. But, if the play targets hundreds of hosts, this can put a heavy load on the control node.
- The maximum number of simultaneous connections that Ansible makes is controlled by the forks parameter in the Ansible configuration file. It is set to **5** by default, which can be verified using one of the following.
- *$ ansible-config view | grep fork*
- *$ ansible-config dump |grep -i forks*
- The default for forks is set to be very conservative. If your control node is managing Linux hosts, most tasks will run on the managed hosts and the control node has less load. In this case, you can usually set forks to a much higher value, possibly closer to 100, and see performance improvements.
- You can override the default setting for forks from the command line in the Ansible configuration file. Both the **ansible** and the **ansible-playbook** commands offer the **-f** or **--forks** options to specify the number of forks to use.

Managing play with forks

- Normally, when Ansible runs a play, it makes sure that all managed hosts have completed each task before starting the next task. After all managed hosts have completed all tasks, then any notified handlers are run.
- However, running all tasks on all hosts can lead to undesirable behaviour. For example, if a play updates a cluster of load balanced web servers, it might need to take each web server out of service while the update takes place. If

all the servers are updated in the same play, they could all be out of service at the same time.

- One way to avoid this problem is to use the serial keyword to run the hosts through the play in batches. Each batch of hosts will be run through the entire play before the next batch is started.
- Test the parallelism with below ad-hoc command and observe the output.
    - $ ansible active -m shell -a " sleep 5 ; echo ' hi ' "
    - $ ansible active -f 1 -m shell -a " sleep 5 ; echo ' hi ' "
- You can also change the value of forks from the ansible.cfg file.

```
---
  - name:
    hosts: active
    gather_facts: no
    serial: 2
    tasks:
    - debug:
        msg: " This is forks test"
```

Now change the forks value in play book and observe the output.

Import and Include Files

- There are two operations that Ansible can use to bring content into a playbook. You can *include* content, or you can *import* content.
- Include tasks is dynamic, if you have dynamic variable then use include tasks.
- Imports tasks is static, if you don't have dynamic variable then use imports tasks
- Imports statement are pre-process at the time playbook are parsed
- Include statement are processed as they encounter during the execution of the playbook.

Importing Playbook

- The import_playbook directive allows you to import external files containing lists of plays into a playbook. In other words, you can have a master playbook that imports one or more additional playbooks.

Managing Playbook

Importing Task Files

- You can statically import a task file into a play inside a playbook by using the import_tasks feature. When you import a task file, the tasks in that file are directly inserted when the playbook is parsed. The location of import_tasks in the playbook controls where the tasks are inserted and the order in which multiple imports are run.
- When using the import_tasks feature, conditional statements set on the import, such as **when**, are applied to each of the tasks that are imported.
- You cannot use loops with the import_tasks feature.
- If you use a variable to specify the name of the file to import, then you cannot use a host or group inventory variable.

Create multiple tasks and then all tasks will be imported in one playbook.

- **Tasks -1 Installing HTTPD**

```
---
- name: Install the httpd package
  yum:
    name: httpd
    state: latest
- name: Start the httpd service
  service:
    name: httpd
    enabled: true
    state: started
```

- **Tasks -2 configuring Firewall**

```
---
- name: Start the firewall
  service:
    name: firewalld
    enabled: true
    state: started
- name: Open the port for http
  firewalld:
    service: http
    immediate: true
    permanent: true
    state: enabled
```

Managing Playbook

- **Tasks -3 configuring Firewall**

```
---
- name: Create placeholder file
  copy:
    content: " Hello Welcome to import module.\n"
    dest: /var/www/html/index.html
```

- **Create a Playbook**

```
---
  - name: Importing Tasks from
    hosts: active
    gather_facts: no
    become: true
    tasks:
      - name:
        import_tasks: tasks/tasks1.yml
      - name:
        import_tasks: tasks/tasks2.yml
      - name:
        import_tasks: tasks/tasks3.yml
```

Including Task Files

- The include_tasks feature does not process content in the playbook until the play is running and that part of the play is reached. The order in which playbook content is processed impacts how they include tasks feature works.
- When using the include_tasks feature, conditional statements such as **when** set on the include determine whether or not the tasks are included in the play at all.
- If you run **ansible-playbook --list-tasks** to list the tasks in the playbook, then tasks in the included task files are not displayed. The tasks that include the task files are displayed. By comparison, the import_tasks feature would not list tasks that import task files, but instead would list the individual tasks from the imported task files.
- You cannot use **ansible-playbook --start-at-task** to start playbook execution from a task that is in an included task file.

- You cannot use a **notify** statement to trigger a handler name that is in an included task file. You can trigger a handler in the main playbook that includes an entire task file, in which case all tasks in the included file will run.

Use Cases for Task Files

- If new servers require complete configuration, then administrators could create various sets of tasks for creating users, installing packages, configuring services, configuring privileges, etc. Each of these sets of tasks could be managed through a separate self-contained task file.
- If servers are managed collectively by the developers, the system administrators, and the database administrators, then every organization can write its own task file which can then be reviewed and integrated by the system manager.

```
---
  - name: Instaling multiple packages with condition
    hosts: public
    gather_facts: true
    become: yes
    tasks:
      - name: Installing HTTP on RedHat
        yum:
          name: httpd
          state: present
        when: ansible_os_family == "RedHat"
      - name: Installing apache on Ubuntu
        apt:
          name: apache2
          state: present
        when: ansible_os_family == "Debian"
      - name: Starting http Service in RedHat
        service:
          name: httpd
          state: started
          enabled: true
        when: ansible_os_family == "RedHat"
      - name: starting apache service in Debian
        service:
          name: apache2
          state: started
          enabled: true
        when: ansible_os_family == "Debian"
```

Managing Playbook

- **Create a tasks with name: instaling_pkg_RedHat.yml**

```
---
- name: Installing HTTP on RedHat
  yum:
    name: httpd
    state: present
```

- **Create a tasks with name: instaling_pkg_Debian.yml**

```
---
- name: Installing apache on Ubuntu
  apt:
    name: apache2
    state: present
```

- **Create a tasks with name: starting_service_RedHat.yml**

```
---
- name: Starting http Service in RedHat
  service:
    name: httpd
    state: started
    enabled: true
```

- **Create a tasks with name: starting_service_Debian.yml**

```
--
- name: starting apache service in Debian
  service:
    name: apache2
    state: started
    enabled: true
```

- **Create a Playbook**

```
---
- name:
  hosts: public
  become: yes
  gather_facts: yes
  tasks:
  - include_tasks: instaling_pkg_{{ansible_os_family}}.yml
  - include_tasks: starting_service_{{ansible_os_family}}.yml
```