Ansible Modules

Ansible modules are discrete units of code which can be used from the command line or in a playbook task.

The modules also referred to as task plugins or library plugins in the Ansible.

Ansible ships with several modules that are called **module library**, which can be executed directly or remote hosts through the playbook.

Users can also write their modules. These modules can control like **services, system resources, files,** or **packages,** etc. and handle executing system commands.

Let's see how to execute three different modules from the command line.

1. ansible webservers -m service -a "name=httpd state=started"
2. ansible webservers -m ping
3. ansible webservers -m command -a "/sbin/reboot -t now"

Each module supports taking arguments. Mainly all modules take **key=value** arguments, space delimited.

Some module takes no arguments, and the shell/command modules take the string of the command which you want to execute.

From playbook, Ansible modules execute in a very similar way, such as:

1. - name: reboot the servers
2. command: /sbin/reboot -t now

Here is another way to pass arguments to a module that is using **YAML syntax**, and it is also called complex args.

```
1. - name: restart webserver
2.   service:
3.     name: httpd
4.     state: restarted
```

Technically, all modules return **JSON** format data, though command line or playbooks, you don't need to know much about that. If you're writing your module, it means you do not have to write modules in any particular language which you get to choose.

Modules should be idempotent and avoid making any changes if they detect that the current state matches the desired final state. When using Ansible playbooks, these modules can trigger "**change events**" in the form of notifying "**handlers**" to run additional tasks.

Documentation for each module can be accessed from the command line with the Ansible-doc tool:

Ansible Shell

Ansible shell module is designed to execute the shell commands against the target UNIX based hosts. Ansible can run except any high complexes commands with pipes, redirection. And you can also perform the shell scripts using the Ansible shell module.

The main advantage of the Ansible shell is except any high complexes commands with pipes and semicolons can be a disadvantage from the security perspective as a single mistake could cost a lot and break the system integrity.

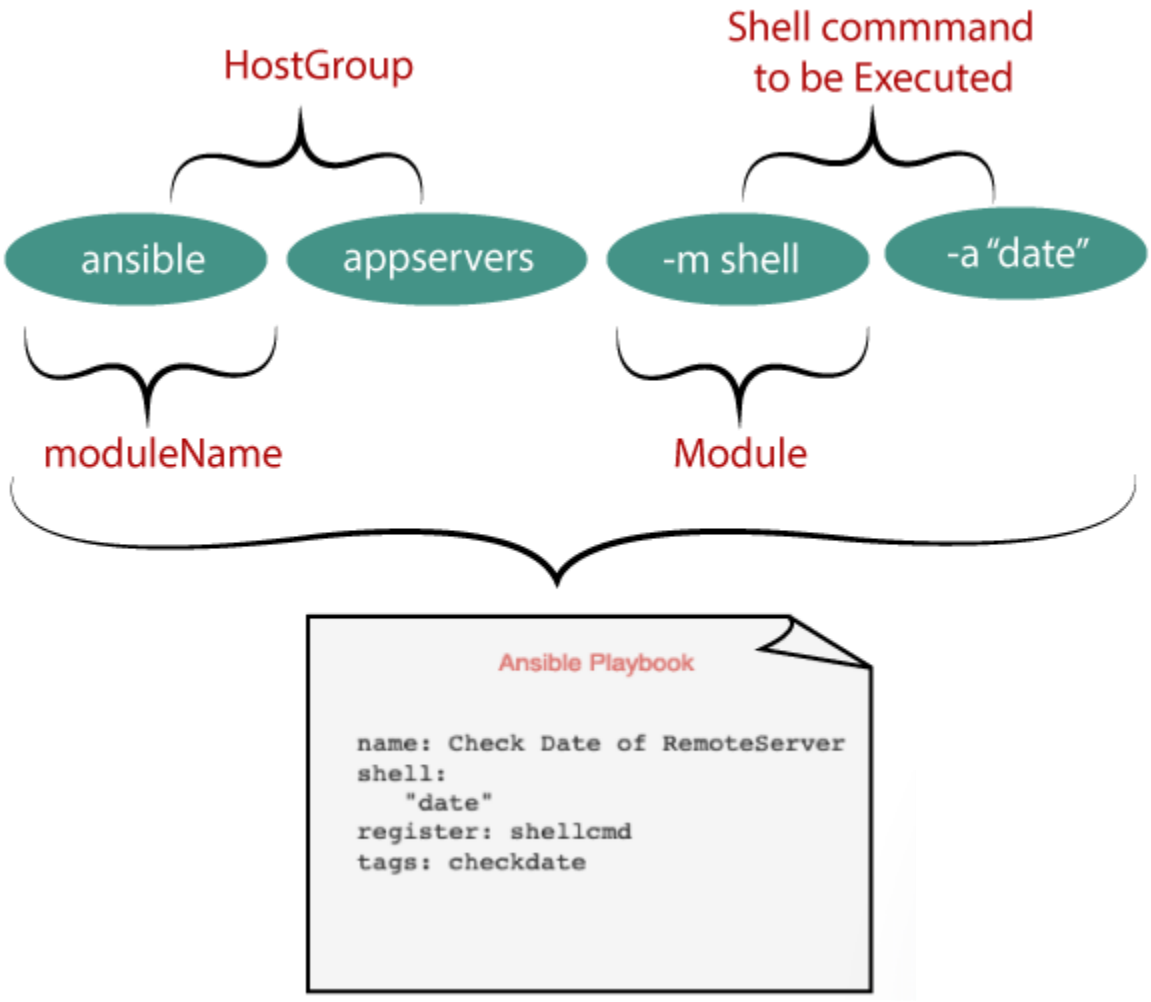
- The Ansible shell module is designed to work only with LINUX based machines and not for the windows. For windows, you should use the **win_shell**
- Ansible shell module can be used to execute shell scripts. Ansible has a dedicated module named script, which is used to copy the shell script from the control machine to the remote server.

Let see the syntax of how to use the Ansible shell module in the playbook and Adhoc:

Syntax of Ansible shell module in a playbook

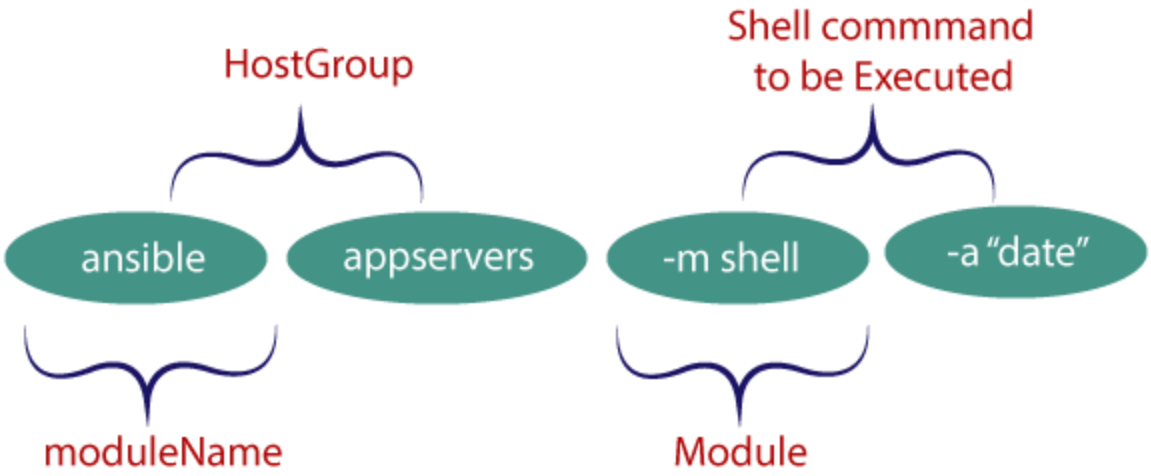
The beauty of the playbook is the way it looks and written. A playbook is written in YAML so it can be easily understood.

The below image demonstrates how an Adhoc command would be transformed as a play of an Ansible playbook.



Syntax of Ansible shell module in Adhoc

The below image shows a quick syntax of the Ansible shell module in Adhoc manner.



Example

To execute a single command in a single task using a Shell or command module. Suppose you want to get the date of the remote server. And the remote server is under the hostgroup which name is testservers.

Step 1: Login to the Ansible server.

Step 2: Below is an example that executes a single command using the Shell module in a remote host.

```
1. ---
2. -name: Shell command example
3. Hosts: testservers
4. tasks:
5. -name: check date with the shell command
6. shell:
7. "date"
8. register: datecmd
9. tags: datecmd
10. -debug: msg= "{{datecmd.stdout}}"
```

In the above example, we are running our playbook against a hostgroup named **testservers** and executing a simple date command and saving the output of that command into a **Register** variable named **datecmd**.

At the last line, we retrieve the registered variable and printing only the date command output stored in the **stdout** property of **datecmd**.

Example 2: Execute multiple commands in a single shell:

The Shell can accept various commands together in a single shell play. Also, you can write your shell script with the Ansible shell module.

In the below example, we grouped some shell commands to execute a controlled and clean tomcat restart.

The playbook is designed to execute the following steps in order, such as:

- Stop the tomcatServer
- Clear the cache
- Truncate the log file
- Start the instance

```
---
 - name: Shell Examples
   hosts: testservers
   tasks:
   - name: Clear Cache and Restart tomcat
     become: yes
     delay: 10
     async: 10
     poll: 50
     shell: |
       echo -e "\n Change directory to the Tomcat"
       cd tomcat8/
       echo -e "\n Present working directory is" `pwd`

       echo -e "\n Stopping the tomcat instance"
```

```
16.       bin/shutdown.sh
17.       echo -e "\n Clearning the tmp and work directory of tomcat"
18.       rm -rfv tmp/*
19.       rm -rfv work/*
20.       echo -e "\nTruncate the log file"
21.       > logs/catalina.out
22.       echo -e "\nDirectory listing"
23.       ls -lrtd logs/catalina.out
24.       echo -e "\nStarting the instance"
25.       bin/startup.sh
26.     args:
27.       chdir: "/apps/tomcat/"
28.     register: fileout
29.     tags: fileout
30.   - debug: msg="{{ fileout.stdout_lines }}"
```

Ansible Copy

Ansible provides the functionality of copying the files and directories with the help of copy and fetch modules. The copy module is versatile.

The copy module is used to copy files and folders from the local machine to the remote servers. And the fetch module to copy data from the remote machine to the local machine.

If you want to copy files after substituting with variables, such as config files with IP changes, or you can use the template module also. You can perform a lot of complicated tasks with this module.

Copying Files from Local to Remote

The copy module is used to check the file set in the **src** parameter, on the local machine. And then, it will copy the data to the remote machine path specified in the **destpath**.

In the below example, we will copy the **sample.txt** file in the home directory of the local machine, to the destination is the **/tmp** directory on the remote server. as long as we are not specifying any permission for the file, the default permission for the remote file is set as -**rw-rw-r-(0664)**.

```
- hosts: blocks
  tasks:
  - name: Ansible copy file to a remote server
    copy:
      src: ~/sample.txt
      dest: /tmp
```

Case 1: If the file is already present on the remote server, but the source file's content is different, then the destination file will be modified. You can control this by setting the force parameter. The default is set to yes. So it modifies the file by default.

If you don't want the file to be modified, if the source file is different, then you can set it No. The following task will copy the file if the file does not exist on the remote server.

```
- hosts: blocks
  tasks:
  - name: Ansible copy file force
    copy:
      src: ~/sample.txt
      dest: /tmp
      force: no
```

Case 2: If the file did not found on the local machine, the Ansible throw an error.

For example: fatal: [remote-machine-1]: FAILED!=> {"changed": false, "failed": true, "msg": "unable to find '~/sample.txt' in expected paths."}

Copying Directories from Local to Remote

You can also copy folders or directories using the Ansible copy module. If the '**src**' path is a directory, then it will be copied recursively. Or the entire directory will be copied.

There are two different variations for this task. Depending on whether you have the '/' character at the endpoint of the 'src' path or not.

The first method will create a directory on the remote server, with the name set in the src parameter. Then, it will copy and paste the content of the source folder into that directory.

If you want this behavior, then it doesn't give the '/' after the path in the src parameter.

in the below example, it will first create a directory named **copy_dir_ex** in the **/tmp** of the remote server.

```
1. - hosts: blocks
2.   tasks:
3.   - name: Ansible copy the directory to the remote server
4.     copy:
5.       src:/Users/mdtutorials2/Documents/Ansible/copy_dir_ex
6.       dest:/Users/mdtutorials2/Documents/Ansible/tmp
```

Copying Files between Directories on Remote Machine

Ansible copy allows you to copy the files from one directory to another on the same remote machine. But this is only for files, not for the directories. You can use the **remote_src parameter** to let Ansible know your intentions.

The below code will copy **/tmp/test.txt** to the home directory of the user **(/home/[username]/)**.

1. ---
2. -hosts: webservers
3. tasks:
4. -name: copy the file between directories on a remote server
5. copy:
6. src: /tmp/test.txt
7. dest: ~/test.txt
8. remote_src: yes

Ansible Command

Ansible command module is used to run any commands or run any scripts in the remote target machine. Or used to execute commands on a remote node.

The command module is used to run simple Linux commands on a remote node or server, which is a part of the host group or standalone server mentioned in the host group.

Ansible Command Module and Shell Module

The shell module is used when we need to execute a command in remote servers, in the shell of your choice. By default, the commands are run on the **/bin/sh** shell. You can make use of the various operations such as '|', '<', '>' etc. and environmental variables such as **$HOME**.

The command module does not process the commands through a shell. So it does not support the above operations.

You give the command you want to execute the same way you provide it on a UNIX shell, command name followed by the arguments.

1. - name: Executing a command using the shell module

```
2.  shell: ls -lrt > temp.txt
```

The first command lists all the files in the current folders and writes that to the file, temp.txt.

```
1.  - name: Executing a command using the command module
2.   command: hello.txt
```

The above example displays the content of the hello.txt file.

Changing the Default Directory

The command will always execute in the default directory. You can change and specify the directory path where you want to run the command using the **chdir** parameter. This parameter is available for both command and shell module.

You can also change the default shell by specifying the absolute path of the require shell in the executable parameter.

```
1.  - - hosts: loc
2.    tasks:
3.    - name: ansible command with chdir and executable parameters
4.      command: ls -lrt
5.      args:
6.        chdir: /home/Ansible/command_chdir_example
7.        executable: /bin/bash
```

In the above example, using the "Bourne Again Shell" by giving the obsolete the path **/bin/bash**. And changed the directory to /home/Ansible/command_chdir_example.

Executing Multiple Commands

If you need to run multiple commands, then you can give them to both shell and command modules using the "**with_items**".

Example 1:

```
1. - hosts: loc
2.   tasks:
3.   - name: Ansible command module multiple commands
4.     command: "touch {{ item }}"
5.     with_items:
6.       - hello.txt
7.       - hello1.txt
8.       - hello2.txt
9.     args: chdir: /root/ansible
```

Example 2:

```
1. - hosts: loc
2.   tasks:
3.   - name: Ansible shell module multiple commands
4.     shell: "cat {{ item }} | grep ansible"
5.     with_items:
6.       - hello.txt
7.       - hello1.txt
8.       - hello2.txt
9.     args:
10.      chdir: /root/ansible
```

In the above examples, we want to execute three files; hello.txt, hello1.txt, and hello2.txt. Since I give the {{item}} keyword in the command, it will be replaced with an element of the list in each iteration. Ensure that the level of indentation of "with_item" is on the same level as the module name.

,