Red Hat

Ansible Fundamentals

Running a One-Off Task with Ad Hoc Commands

Objective

This module explains how Ansible automation tasks can use ad hoc commands to execute a single Ansible task quickly.

Running Ad-Hoc Commands

Ad Hoc Commands

- Ad hoc commands are simple, one line operations that are run without writing a playbook.
- They are useful for quick tests and changes.
- For example, to start a service or ensure a line exists in a file.
- Ad hoc commands have limitations.

Ansible Modules

- Ansible provides *modules*, code that can be used to automate particular tasks

- Some uses of modules:
  - Ensure users exist with certain settings
  - Make sure the latest version of a software package is installed
  - Deploy a configuration file to a server
  - Enable a network service and make sure that it is running

- Most modules are *idempotent*, which means they only make changes if a change is needed. Idempotent modules can be run safely multiple times.

- An ad hoc command runs one module on the specified managed hosts.

Running Ad Hoc Commands

- The **`ansible`** command runs an ad hoc command
- Its **`host-pattern`** argument specifies the managed hosts to run on.
- Its **`-m`** option names the module that Ansible should run.
- Its **`-a`** option takes a list of all arguments required by the module.

```
ansible host-pattern -m module [-a 'module arguments'] [-i inventory]
```

Running Ad Hoc Commands

- One of the simplest ad hoc commands uses the **ping** module.
- It does not send an ICMP ping to the managed host.
- It checks to see if Ansible modules written in Python can be run on the managed host.

```
[user@controlnode ~]$ ansible all -m ping
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

Overriding Default Configuration Settings

- To override a default configuration setting there are several different options.
- These options override the configuration in the **ansible.cfg** configuration file.

    - **`-k`** or **`--ask-pass`** will prompt for the connection password.
    - **`-u`** ***`REMOTE_USER`*** overrides the **`remote_user`** setting in **ansible.cfg**.
    - **`-b`** option enables privilege escalation, running operations with **become: yes**.
    - **`-K`** or **`--ask-become-pass`** will prompt for the privilege escalation password.
    - **`--become-method`** will override the default privilege escalation method. The default is **sudo**. Find valid choices using **`ansible-doc -t become -l`**.

Ansible Modules

- Most modules take arguments to control them.
- Use the **`-a`** option to pass arguments.

- This example uses the **user** module to make sure the user *newbie* is present and has the UID number 4000.

```
[user@controlnode ~]$ ansible -m user -a 'name=newbie uid=4000 state=present' \
> servera.lab.example.com
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": true,
    "comment": "",
    "createhome": true,
    "group": 4000,
    "home": "/home/newbie",
    "name": "newbie",
    "shell": "/bin/bash",
    "state": "present",
    "system": false,
    "uid": 4000
}
```

Selected Ansible Command-line Options

- A number of command line directives can be used to override options from the Ansible configuration file:

Configuration File Directives	Command-line Option
inventory	`--inventory, --inventory-file, -i`
remote_user	`--user, -u`
become	`--become, -b`
become-method	`--become-method`
become_user	`--become-user`
become_ask_pass	`--ask-become-pass, -K`

Selecting Modules for Ad Hoc Commands

Objective

This module explains how Ansible ad hoc commands leverage Ansible modules to perform singular command line interactions with managed hosts.

Finding Information about Ansible Modules

- The **`ansible-doc -l`** command lists all modules installed on a system.
- The name and a description of the module are displayed.
- Thousands of modules are available: consider piping the output into **grep** to filter the result.

- The same information is available from the Ansible website: https://docs.ansible.com/ansible/latest/modules/modules_by_category.html

Selected Ansible Modules

There are thousands of modules to perform tasks. Some selected modules include:

- File Modules:
  - **`copy`**: Copy a local file to the manages host
  - **`file`**: Set permissions and other properties of files
  - **`lineinfile`**: Ensure a particular line is or is not in a file
  - **`synchronize`**: Synchronize content using rsync
- Software package modules:
  - **`yum`**: Manage packages using YUM
  - **`dnf`**: Manage packages using DNF
  - **`gem`**: Manage Ruby gems

Selected Ansible Modules

- System Modules:
    - **`firewalld`**: Manage arbitrary ports and services using firewalld
    - **`reboot`**: Reboot a machine
    - **`service`**: Manage services
    - **`user`**: Add, remove, and manage user accounts
- Net Tools modules:
    - **`get_url`**: Download files over HTTP, HTTPS, or FTP
    - **`nmcli`**: Manage networking
    - **`uri`**: Interact with web services and communicate with APIs

Getting Information about an Ansible Module

- **`ansible-doc`** can also provide information on how to use a module.
- For example:

```
[user@controlnode ~]$  ansible-doc ping
> PING     (/usr/lib/python3.7/site-packages/ansible/modules/system/ping.py)

        A trivial test module, this module always returns `pong' on successful
        contact. It does not make sense in playbooks, but it is useful from
        `/usr/bin/ansible' to verify the ability to login and that a usable
        Python is configured. This is NOT ICMP ping, this is just a trivial test
        module that requires Python on the remote-node. For Windows targets, use
        the [win_ping] module instead. For Network targets, use the [net_ping]
        module instead.

   * This module is maintained by The Ansible Core Team
OPTIONS (= is mandatory):

- data
        Data to return for the `ping' return value.
        If this parameter is set to `crash', the module will cause an exception.
        [Default: pong]
        type: str

[...output omitted…]
```

Ad Hoc Command Example: User Creation

- You can use the **ansible-doc -l** command to discover the **user** module

```
user                                                          Manage user accounts
```

- Then run **ansible-doc user** to find out how the module works:
  - **name** (or **user**) is a mandatory argument, takes the user name of the account.
  - **uid** specifies the UID number that the account must have.
  - **state** specifies whether the user must be present or must be absent from the managed host. This can be used to remove user accounts.
  - There are many other options to adjust the user's account settings and password.
- To make sure user newbie with UID 4000 exists on all managed hosts, you can run:

```
ansible all -m user -a 'name=newbie uid=4000 state=present'
```

Ad Hoc Command Example: Group Creation

- You can use the **ansible-doc -l** command to discover the **group** module

```
group                                                           Add or remove groups
```

- Then run **ansible-doc group** to find out how the module works:
  - **name** is a mandatory argument, takes the group name.
  - **gid** specifies the GID number that the group must have.
  - **state** specifies whether the group must be present or must be absent from the managed host. This can be used to remove group accounts.
  - There are many other options to adjust the user's account settings and password.
- To make sure group developers with UID 2000 exists on all managed hosts, you can run:

```
ansible all -m group -a 'name=developers gid=2000 state=present'
```

Ad Hoc Command Example: Group Management

- The **user** module can be used to adjust group membership.
- Run **ansible-doc user** again to find out how the module works:
    - **group** sets the user's primary group
    - **groups** is a list of other groups to assign the user
    - **append** controls whether to add the groups to the user's current list (if any) or replace the list

- To add user *newbie* to group *developers* and group *wheel*, without changing the user's primary group or removing *newbie* from other groups:

```
ansible all -m user -a 'name=newbie groups=developers,wheel append=yes state=present'
```

Ad Hoc Command Example: Software Package Installation

- You can use the **ansible-doc -l** command to discover the **package** module

```
package                                                                  Generic OS package manager
```

- Then run **ansible-doc package** to find out how the module works:
  - **name** is the name of a package or a list of packages to manage
  - **state** specifies whether the package must be present, absent, or the latest version. This can be used to remove or update packages.
- To make sure the httpd package is installed on all hosts, you can run:

```
ansible all -m package -a 'name=httpd state=present'
```

- Other modules are also available, including **yum**, **dnf**, and **apt**, that work in a similar way but which might support more sophisticated options specific to those managers.

Command Modules

- There are a handful of modules that run commands directly on the managed host
- You can use these if no other module is available to do what you need
- They are **not idempotent**: you must make sure that they are safe to run twice when using them

- **command** runs a single command on the remote system
- **shell** runs a command on the remote system's shell (redirection and other features work)
- **raw** simply runs a command with no processing (can be dangerous)

- In general, you should use regular modules if you can before resorting to these

Running Arbitrary Commands on Managed Hosts

- The **command** module allows administrators to run arbitrary commands.
- It cannot access shell environment variables or perform shell operations such as redirection and piping.

```
[user@controlnode ~]$ ansible mymanagedhosts -m command -a /usr/bin/hostname
host1.lab.example.com | CHANGED | rc=0 >>
host1.lab.example.com
host2.lab.example.com | CHANGED | rc=0 >>
host2.lab.example.com
```

Red Hat

Running Arbitrary Commands on Managed Hosts

- The **shell** module is used when commands require shell processing.

```
[user@controlnode ~]$ ansible localhost -m command -a set
localhost | FAILED | rc=2 >>
[Errno 2] No such file or directory
[user@controlnode ~]$ ansible localhost -m shell -a set
localhost | CHANGED | rc=0 >>
BASH=/bin/sh
```

- Both the **command** and **shell** modules require a working Python installation on the managed host.
- The **raw** module can run commands directly using the remote shell, bypassing the module subsystem.
- This is useful when managing systems that cannot have Python installed (for example a network router).

When to Use Ad Hoc Commands

- Ad hoc commands are useful when you need to make one quick change to a large number of systems
- This can be very powerful when you need to make a simple change quickly

- However, they have a number of disadvantages:
  - You can only call one module, which limits them to simple changes
  - You have to type the same command again to rerun it, options can get complex
  - They are still semi-manual in nature

- In many cases, a better approach will be to use Ansible Playbooks
  - Playbooks can run multiple modules with conditionals and other processing
  - Playbooks are text files that can be tracked in version control systems
  - Playbooks can be easily rerun with a simple command

Red Hat