Red Hat Enterprise Linux Automation with Ansible

PREVIOUS (/ROL/APP/COURSES/RH294-8.4/PAGES/CH09S04) NEXT (/ROL/APP/COURSES/RH294-8.4/PAGES/CH09S06)

VIDEO CLASSROOM

Managing the Boot Process and Scheduled Processes

Objectives

After completing this section, you should be able to manage service startup, schedule processes with at, cron, and systemd, reboot, and control the default boot target on managed hosts.

Scheduling with the at Module

Quick one-time scheduling is done with the `at` module. You create the job for a future time to run and it is held until that time comes to execute. There are six parameters that come with this module. They are: command, count, script_file, state, unique, and units.

The at Module Example:

```
- name: remove tempuser.
  at:
    command: userdel -r tempuser
    count: 20
    units: minutes
    unique: yes
```

Table 9.3. Parameters

Parameter	Options	Comments
command	Null	A command that is scheduled to run.
count	Null	The count of units. (Must run with units)
script_file	Null	An existing script file to be executed in the future.
state	absent, present	The state adds or removes a command or script.
unique	yes, no	If a job is already running, it will not be executed again.
units	minutes/hours/days/weeks	The time denominations.

Appending Commands with the cron Module

When setting a jobs scheduled task the `cron` module is used. The cron module will append commands directly into the crontab of the user you designate.

The cron module example:

```
- cron:
    name: "Flush Bolt"
    user: "root"
    minute: 45
    hour: 11
    job: "php ./app/nut cache:clear"
```

This play uses a company's cache:clear command immediately flushes Bolt cache, removing cached files and directories.flushes cache of the CMS server every morning at 11:45.

Ansible will write the play to the crontab using the correct syntax as the user stated.

Checking the crontab will verify that it has been appended to.

Some commonly used parameters for the cron module are:

Table 9.4. Parameters

Parameter	Options	Comments
special_time	reboot, yearly, annually, monthly, weekly, daily, hourly	A set of reoccurring times.
state	absent, present	If set to present, it will create the command. Absent will remove it.

Parameter	Options	Comments
cron_file	Null	If you have large banks of servers to maintain then sometimes it is better to have a pre-written crontab file.
backup	yes, no	Backs up the crontab file prior to being edited.

Managing Services with the systemd and service Modules

For managing services or reloading daemons, Ansible has the `systemd` and the service modules. Service offers a basic set of options start, stop, restart, enable. The systemd module offers more configuration options. Systemd will allow you to do a daemon-reload where the service module will not.

The service Module Example:

```
- name: start nginx
  service:
    name: nginx
    state: started"
```

NOTE

The init daemon is being replaced by systemd. So in a lot of cases systemd will be the better option.

The systemd Module Example:

```
- name: reload web server
  systemd:
    name: apache2
    state: reload
    daemon-reload: yes
```

The Reboot Module

Another well used Ansible Systems Module is `reboot`. Considered safer than using the shell module to initiate shutdown. While running a play the reboot module will shut down the managed host, then wait until it is back up again prior to carrying on with the play.

The reboot module Example:

```
- name: "Reboot after patching"
  reboot:
    reboot_timeout: 180

- name: force a quick reboot
  reboot:
```

The Shell and Command Module

Like the service and the systemd modules, the shell and the command can interchange some tasks. The command module is considered more secure but some environment variables are not available. Also, stream operators will not work. If you need to stream your commands then shell module will do.

The shell module example:

```
- name: Run a templated variable (always use quote filter to avoid injection)
    shell: cat {{ myfile|quote }} (1)
```

(1) To sanitize any variables, It is suggested that you use `{{ var | quote }}` instead of just `{{ var }}`

The command module example:

```
- name: This command only
  command: /usr/bin/scrape_logs.py arg1 arg2
  args: (1)
    chdir: scripts/
    creates: /path/to/script
```

(1) You can pass arguments into the form to provide the options.

NOTE

The command module is considered more secure because it is not affected by the users environment.

Gathering facts on the managed host will allow you to access the environment variables. There is a sublist called ansible_env which has all the environment variables inside it.

```
---
- name:
  hosts: webservers
  vars:
    local_shell:  "{{ ansible_env }}"(1)
  tasks:
    - name: Printing all the environment variables in Ansible
      debug:
        msg: "{{ local_shell }}"
```

(1) You can isolate the variable you want to return by using the lookup plugin. `msg: "{{ lookup('env','USER','HOME','SHELL') }}"`

REFERENCES

at - Schedule the execution of a command or script file via the at command — Ansible Documentation (https://docs.ansible.com/ansible/2.9/modules/at_module.html)

cron - Manage cron.d and crontab entries — Ansible Documentation (https://docs.ansible.com/ansible/2.9/modules/cron_module.html)

reboot - Reboot a machine — Ansible Documentation (https://docs.ansible.com/ansible/2.9/modules/reboot_module.html)

service - Run services on a machine — Ansible Documentation (https://docs.ansible.com/ansible/2.9/modules/service_module.html)

PREVIOUS (/ROL/APP/COURSES/RH294-8.4/PAGES/CH09S04)   NEXT (/ROL/APP/COURSES/RH294-8.4/PAGES/CH09S06)