Red Hat Enterprise Linux Automation with Ansible

➜ NEXT (/ROL/APP/COURSES/RH294-8.4/PAGES/CH09S02)

VIDEO CLASSROOM

Chapter 9. Automating Linux Administration Tasks

Abstract

Goal	Automate common Linux system administration tasks with Ansible.
Objectives	• Subscribe systems, configure software channels and repositories, enable module streams, and manage RPM packages on managed hosts. • Manage Linux users and groups, configure SSH, and modify Sudo configuration on managed hosts. • Manage service startup, schedule processes with at, cron, and systemd, reboot, and control the default boot target on managed hosts. • Partition storage devices, configure LVM, format partitions or logical volumes, mount file systems, and add swap files or spaces. • Configure network settings and name resolution on managed hosts, and collect network-related Ansible facts.
Sections	• Managing Software and Subscriptions (Guided Exercise) • Managing Users and Authentication (Guided Exercise) • Managing the Boot Process and Scheduled Processes (Guided Exercise) • Managing Storage (Guided Exercise) • Managing Network Configuration (Guided Exercise)
Lab	• Automating Linux Administration Tasks

Managing Software and Subscriptions

Objectives

After completing this section, you should be able to subscribe systems, configure software channels and repositories, enable module streams, and manage RPM packages on managed hosts.

Managing Packages with Ansible

The `yum` Ansible module uses the *Yum Package Manager* on the managed hosts to handle the package operations. The following example is a playbook that installs the httpd package on the `servera.lab.example.com` managed host.

```
---
- name: Install the required packages on the web server
  hosts: servera.lab.example.com
  tasks:
    - name: Install the httpd packages
      yum:
        name: httpd     (1)
        state: present  (2)
```

1. The `name` keyword gives the name of the package to install.
2. The `state` keyword indicates the expected state of the package on the managed host:

   `present`
   Ansible installs the package if it is not already there.

   `absent`
   Ansible removes the package if it is installed.

   `latest`
   Ansible updates the package if it is not already at the most recent available version. If the package is not installed, Ansible installs it.

The following table compares some usage of the `yum` Ansible module with the equivalent `yum` command.

Ansible task	Yum command

`- name: Install httpd`
`  yum:`
`  name: httpd`
`  state: present`
	`yum install httpd`

`- name: Install or update httpd`
`  yum:`
`  name: httpd`
`  state: latest`
	`yum update httpd` or `yum install httpd` if the package is not yet installed.

`- name: Update all packages`
`  yum:`
`  name: '*'`
`  state: latest`
	`yum update`

`- name: Remove httpd`
`  yum:`
`  name: httpd`
`  state: absent`
	`yum remove httpd`

`- name: Install Development Tools`
`  yum:`
`  name: '@Development Tools'` (1)
`  state: present`
 	
(1) With the `yum` Ansible module, you must prefix group names with `@`. Remember that you can retrieve the list of groups with the `yum group list` command.
	`yum group install "Development Tools"`

`- name: Remove Development Tools`
`  yum:`
`  name: '@Development Tools'`
`  state: absent`
	`yum group remove "Development Tools"`

`- name: Inst perl AppStream module`
`  yum:`
`  name: '@perl:5.26/minimal'` (1)
`  state: present`
 	
(1) To manage a Yum AppStream module, prefix its name with `@`. The syntax is the same as with the `yum` command. For example, you can omit the profile part to use the default profile: `@perl:5.26`. Remember that you can list the available Yum AppStream modules with the `yum module list` command.
	`yum module install perl:5.26/minimal`

Run the `ansible-doc yum` command for additional parameters and playbook examples.

Optimizing Multiple Package Installation

To operate on several packages, the `name` keyword accepts a list. The following example shows a playbook that installs three packages on `servera.lab.example.com`.

```
---
- name: Install the required packages on the web server
  hosts: servera.lab.example.com
  tasks:
    - name: Install the packages
      yum:
        name:
          - httpd
          - mod_ssl
          - httpd-tools
        state: present
```

With this syntax, Ansible installs the packages in a single Yum transaction. This is equivalent to running the `yum install httpd mod_ssl httpd-tools` command.

A commonly seen but less efficient and slower version of this task is to use a loop.

```
---
- name: Install the required packages on the web server
  hosts: servera.lab.example.com
  tasks:
    - name: Install the packages
      yum:
        name: "{{ item }}""
        state: present
      loop:
        - httpd
        - mod_ssl
        - httpd-tools
```

Avoid using this method as it requires the module to perform three individual transactions, one for each package.

Gathering Facts about Installed Packages

The `package_facts` Ansible module collects the installed package details on managed hosts. It sets the `ansible_facts.packages` variable with the package details.

The following playbook calls the `package_facts` module, the `debug` module to display the content of the `ansible_facts.packages` variable, and the `debug` module again to view the version of the installed NetworkManager package.

```
---
- name: Display installed packages
  hosts: servera.lab.example.com
  tasks:
    - name: Gather info on installed packages
      package_facts:
        manager: auto

    - name: List installed packages
      debug:
        var: ansible_facts.packages

    - name: Display NetworkManager version
      debug:
        msg: "Version {{ansible_facts.packages['NetworkManager'][0].version}}"
      when: "'NetworkManager' in ansible_facts.packages"
```

When run, the playbook displays the package list and the version of the NetworkManager package:

```
[user@controlnode ~]$ ansible-playbook lspackages.yml

PLAY [Display installed packages] ***************************************

TASK [Gathering Facts] **************************************************
ok: [servera.lab.example.com]

TASK [Gather info on installed packages] ********************************
ok: [servera.lab.example.com]

TASK [List installed packages] ******************************************
ok: [servera.lab.example.com] => {
    "ansible_facts.packages": {
        "NetworkManager": [
            {
                "arch": "x86_64",
                "epoch": 1,
                "name": "NetworkManager",
                "release": "14.el8",
                "source": "rpm",
                "version": "1.14.0"
            }
        ],
...output omitted...
        "zlib": [
            {
                "arch": "x86_64",
                "epoch": null,
                "name": "zlib",
                "release": "10.el8",
                "source": "rpm",
                "version": "1.2.11"
            }
        ]
    }
}

TASK [Display NetworkManager version] ***********************************
ok: [servera.lab.example.com] => {
    "msg": "Version 1.14.0"
}

PLAY RECAP **************************************************************
servera.lab.example.com    : ok=4    changed=0    unreachable=0    failed=0
```

Reviewing Alternative Modules to Manage Packages

The `yum` Ansible module works on managed hosts that are using the Yum Package Manager. For other package managers, Ansible usually provides a dedicated module. For example, the `dnf` module manages packages on operating systems such as Fedora using the *DNF package manager*. The `apt` module uses the *APT package tool* available on Debian or Ubuntu. The `win_package` module can install software on Microsoft Windows systems.

The following playbook uses conditionals to select the appropriate module in an environment composed of Red Hat Enterprise Linux and Fedora systems.

```
---
- name: Install the required packages on the web servers
  hosts: webservers
  tasks:
    - name: Install httpd on RHEL
      yum:
        name: httpd
        state: present
      when: "ansible_distribution == 'RedHat'"

    - name: Install httpd on Fedora
      dnf:
        name: httpd
        state: present
      when: "ansible_distribution == 'Fedora'"
```

As an alternative, the generic `package` module automatically detects and uses the package manager available on the managed hosts. With the `package` module, you can rewrite the previous playbook as follows.

```
---
- name: Install the required packages on the web servers
  hosts: webservers
  tasks:
    - name: Install httpd
      package:
        name: httpd
        state: present
```

However, notice that the `package` module does not support all the features that the more specialized modules provide. Also, operating systems often have different names for the packages they provide. For example, the package that installs the Apache HTTP Server is httpd on Red Hat Enterprise Linux and apache2 on Ubuntu. In that situation, you still need a conditional for selecting the correct package name depending on the operating system of the managed host.

Registering and Managing Systems with Red Hat Subscription Management

To entitle your new Red Hat Enterprise Linux systems to product subscriptions, Ansible provides the `redhat_subscription` and `rhsm_repository` modules. These modules interface with the *Red Hat Subscription Management* tool on the managed hosts.

Registering and Subscribing New systems

The first two tasks you usually perform with the Red Hat Subscription Management tool is to register the new system and attach an available subscription.

Without Ansible, you perform these tasks with the `subscription-manager` command:

```
[user@host ~]$ subscription-manager register --username=yourusername \
> --password=yourpassword
[user@host ~]$ subscription-manager attach --pool=poolID
```

Remember that you list the available pools in your account with the `subscription-manager list --available` command.

The `redhat_subscription` Ansible module performs the registration and the subscription in one task.

```
- name: Register and subscribe the system
  redhat_subscription:
    username: yourusername
    password: yourpassword
    pool_ids: poolID
    state: present
```

A `state` keyword set to `present` indicates to register and to subscribe the system. When it is set to `absent`, the module unregisters the system.

Enabling Red Hat Software Repositories

The next task after the subscription is to enable Red Hat software repositories on the new system.

Without Ansible, you usually execute the `subscription-manager` command for that purpose:

```
[user@host ~]$ subscription-manager repos \
> --enable "rhel-8-for-x86_64-baseos-rpms" \
> --enable "rhel-8-for-x86_64-baseos-debug-rpms"
```

Remember that you can list the available repositories with the `subscription-manager repos --list` command.

With Ansible, use the `rhsm_repository` module:

```
- name: Enable Red Hat repositories
  rhsm_repository:
    name:
      - rhel-8-for-x86_64-baseos-rpms
      - rhel-8-for-x86_64-baseos-debug-rpms
    state: present
```

Configuring a Yum Repository

To enable support for a third-party repository on a managed host, Ansible provides the `yum_repository` module.

Declaring a Yum Repository

When run, the following playbook declares a new repository on `servera.lab.example.com`.

```
---
- name: Configure the company Yum repositories
  hosts: servera.lab.example.com
  tasks:
    - name: Ensure Example Repo exists
      yum_repository:
        file: example    (1)
        name: example-internal
        description: Example Inc. Internal YUM repo
        baseurl: http://materials.example.com/yum/repository/
        enabled: yes
        gpgcheck: yes    (2)
        state: present   (3)
```

(1) The `file` keyword gives the name of the file to create under the `/etc/yum.repos.d/` directory. The module automatically adds the `.repo` extension to that name.

2. Typically, software providers digitally sign RPM packages using GPG keys. By setting the `gpgcheck` keyword to `yes`, the RPM system verifies package integrity by confirming that the package was signed by the appropriate GPG key. It refuses to install a package if the GPG signature does not match. Use the `rpm_key` Ansible module, described later on, to install the required GPG public key.

3. When you set the `state` keyword to `present`, Ansible creates or updates the `.repo` file. When `state` is set to `absent`, Ansible deletes the file.

The resulting `/etc/yum.repos.d/example.repo` file on `servera.lab.example.com` is as follows.

```
[example-internal]
baseurl = http://materials.example.com/yum/repository/
enabled = 1
gpgcheck = 1
name = Example Inc. Internal YUM repo
```

The `yum_repository` module exposes most of the Yum repository configuration parameters as keywords. Run the `ansible-doc yum_repository` command for additional parameters and playbook examples.

> **NOTE**
>
> Some third-party repositories provide the configuration file and the GPG public key as part of an RPM package that can be downloaded and installed using the `yum install` command. For example, the *Extra Packages for Enterprise Linux (EPEL)* project provides the https://dl.fedoraproject.org/pub/epel/epel-release-latest-*VER*.noarch.rpm package that deploys the `/etc/yum.repos.d/epel.repo` configuration file. For this repository, use the `yum` Ansible module to install the EPEL package instead of the `yum_repository` module.

Importing an RPM GPG key

When the `gpgcheck` keyword is set to `yes` in the `yum_repository` module, you also need to install the GPG key on the managed host. The `rpm_key` module in the following example deploys on `servera.lab.example.com` the GPG public key hosted on a remote web server.

```
---
- name: Configure the company Yum repositories
  hosts: servera.lab.example.com
  tasks:
    - name: Deploy the GPG public key
      rpm_key:
        key: http://materials.example.com/yum/repository/RPM-GPG-KEY-example
        state: present

    - name: Ensure Example Repo exists
      yum_repository:
        file: example
        name: example-internal
        description: Example Inc. Internal YUM repo
        baseurl: http://materials.example.com/yum/repository/
        enabled: yes
        gpgcheck: yes
        state: present
```

> **REFERENCES**
>
> `yum`(8), `yum.conf`(5), and `subscription-manager`(8) man pages
>
> yum — Manages packages with the yum package manager — Ansible Documentation (https://docs.ansible.com/ansible/2.9/modules/yum_module.html)
>
> package_facts — package information as facts — Ansible Documentation (https://docs.ansible.com/ansible/2.9/modules/package_facts_module.html)
>
> redhat_subscription — Manage registration and subscriptions to RHSM using the subscription-manager command — Ansible Documentation (https://docs.ansible.com/ansible/2.9/modules/redhat_subscription_module.html)
>
> rhsm_repository — Manage RHSM repositories using the subscription-manager command — Ansible Documentation (https://docs.ansible.com/ansible/2.9/modules/rhsm_repository_module.html)
>
> yum_repository — Add or remove YUM repositories — Ansible Documentation (https://docs.ansible.com/ansible/2.9/modules/yum_repository_module.html)
>
> rpm_key — Adds or removes a gpg key from the rpm db — Ansible Documentation (https://docs.ansible.com/ansible/2.9/modules/rpm_key_module.html)

➜ NEXT (/ROL/APP/COURSES/RH294-8.4/PAGES/CH09S02)

Privacy Policy (http://s.bl-1.com/h/cZrgWbQn?url=https://www.redhat.com/en/about/privacy-policy?extIdCarryOver=true&sc_cid=701f2000001D8QoAAK)

Terms of Use (https://www.redhat.com/en/about/terms-use)

Release Notes (https://learn.redhat.com/t5/Red-Hat-Learning-Subscription/Red-Hat-Learning-Subscription-Release-Notes/ba-p/22952)

Red Hat Training Policies (http://s.bl-1.com/h/cZrb2DXG?url=https://www.redhat.com/en/about/red-hat-training-policies)

All policies and guidelines (https://www.redhat.com/en/about/all-policies-guidelines)

Cookie Preferences