Red Hat Enterprise Linux Automation with Ansible

Chapter 5. Deploying Files to Managed Hosts ☆

Abstract

Goal	Deploy, manage, and adjust files on hosts managed by Ansible.
Objectives	• Create, install, edit, and remove files on managed hosts, and manage permissions, ownership, SELinux context, and other characteristics of those files. • Deploy files to managed hosts that are customized by using Jinja2 templates.

Sections	• Modifying and Copying Files to Hosts (and Guided Exercise) • Deploying Custom Files with Jinja2 Templates (and Guided Exercise)
Lab	• Deploying Files to Managed Hosts

Modifying and Copying Files to Hosts

Objectives

After completing this section, you should be able to create, install, edit, and remove files on managed hosts, and manage permissions, ownership, SELinux context, and other characteristics of those files.

Describing Files Modules

Red Hat Ansible Automation Platform ships with a large collection of modules (the "module library") that are developed as part of the upstream Ansible project. To make it easier to organize, document, and manage them, they are organized into groups based on function in the documentation and when installed on a system.

The `Files` modules library includes modules allowing you to accomplish most tasks related to Linux file management, such as creating, copying, editing, and modifying permissions and other attributes of files. The following table provides a list of frequently used file management modules:

Table 5.1. Commonly Used Files Modules

Module name	Module description
`blockinfile`	Insert, update, or remove a block of multiline text surrounded by customizable marker lines.
`copy`	Copy a file from the local or remote machine to a location on a managed host. Similar to the `file` module, the `copy` module can also set file attributes, including SELinux context.
`fetch`	This module works like the `copy` module, but in reverse. This module is used for fetching files from remote machines to the control node and storing them in a file tree, organized by host name.
`file`	Set attributes such as permissions, ownership, SELinux contexts, and time stamps of regular files, symlinks, hard links, and directories. This module can also create or remove regular files, symlinks, hard links, and directories. A number of other file-related modules support the same options to set attributes as the `file` module, including the `copy` module.
`lineinfile`	Ensure that a particular line is in a file, or replace an existing line using a back-reference regular expression. This module is primarily useful when you want to change a single line in a file.
`stat`	Retrieve status information for a file, similar to the Linux `stat` command.
`synchronize`	A wrapper around the `rsync` command to make common tasks quick and easy. The `synchronize` module is not intended to provide access to the full power of the `rsync` command, but does make the most common invocations easier to implement. You may still need to call the `rsync` command directly via the `run command` module depending on your use case.

Automation Examples with Files Modules

Creating, copying, editing, and removing files on managed hosts are common tasks that you can implement using modules from the `Files` modules library. The following examples show ways that you can use these modules to automate common file management tasks.

Ensuring a File Exists on Managed Hosts

Use the `file` module to touch a file on managed hosts. This works like the `touch` command, creating an empty file if it does not exist, and updating its modification time if it does exist. In this example, in addition to touching the file, Ansible ensures that the owning user, group, and permissions of the file are set to specific values.

```
- name: Touch a file and set permissions
  file:
    path: /path/to/file
    owner: user1
    group: group1
    mode: 0640
    state: touch
```

Example outcome:

```
[user@host ~]$ ls -l file
-rw-r-----.  user1 group1 0 Nov 25 08:00 file
```

Modifying File Attributes

You can use the `file` module to ensure that a new or existing file has the correct permissions or SELinux type as well.

For example, the following file has retained the default SELinux context relative to a user's home directory, which is not the desired context.

```
[user@host ~]$ ls -Z samba_file
-rw-r--r--.  owner group unconfined_u:object_r:user_home_t:s0 samba_file
```

The following task ensures that the SELinux context type attribute of the `samba_file` file is the desired `samba_share_t` type. This behavior is similar to the Linux `chcon` command.

```
- name: SELinux type is set to samba_share_t
  file:
    path: /path/to/samba_file
    setype: samba_share_t
```

Example outcome:

```
[user@host ~]$ ls -Z samba_file
-rw-r--r--.  owner group unconfined_u:object_r:samba_share_t:s0 samba_file
```

> **NOTE**
>
> File attribute parameters are available in multiple file management modules. Run the `ansible-doc file` and `ansible-doc copy` commands for additional information.

Making SELinux File Context Changes Persistent

The `file` module acts like `chcon` when setting file contexts. Changes made with that module could be unexpectedly undone by running `restorecon`. After using `file` to set the context, you can use `sefcontext` from the collection of `System` modules to update the SELinux policy like `semanage fcontext`.

```
- name: SELinux type is persistently set to samba_share_t
  sefcontext:
    target: /path/to/samba_file
    setype: samba_share_t
    state: present
```

Example outcome:

```
[user@host ~]$ ls -Z samba_file
-rw-r--r--.  owner group unconfined_u:object_r:samba_share_t:s0 samba_file
```

> **IMPORTANT**
>
> The `sefcontext` module updates the default context for the target in the SELinux policy, but does not change the context on existing files.

Copying and Editing Files on Managed Hosts

In this example, the `copy` module is used to copy a file located in the Ansible working directory on the control node to selected managed hosts.

By default this module assumes that `force: yes` is set. That forces the module to overwrite the remote file if it exists but contains different contents from the file being copied. If `force: no` is set, then it only copies the file to the managed host if it does not already exist.

```
- name: Copy a file to managed hosts
  copy:
    src: file
    dest: /path/to/file
```

To retrieve files from managed hosts use the `fetch` module. This could be used to retrieve a file such as an SSH public key from a reference system before distributing it to other managed hosts.

```
- name: Retrieve SSH key from reference host
  fetch:
    src: "/home/{{ user }}/.ssh/id_rsa.pub
    dest: "files/keys/{{ user }}.pub"
```

To ensure a specific single line of text exists in an existing file, use the `lineinfile` module:

```
- name: Add a line of text to a file
  lineinfile:
    path: /path/to/file
    line: 'Add this line to the file'
    state: present
```

To add a block of text to an existing file, use the `blockinfile` module:

```
- name: Add additional lines to a file
  blockinfile:
    path: /path/to/file
    block: |
      First line in the additional block of text
      Second line in the additional block of text
    state: present
```

NOTE

When using the `blockinfile` module, commented block markers are inserted at the beginning and end of the block to ensure idempotency.

```
# BEGIN ANSIBLE MANAGED BLOCK
First line in the additional block of text
Second line in the additional block of text
# END ANSIBLE MANAGED BLOCK
```

You can use the `marker` parameter to the module to help ensure that the right comment character or text is being used for the file in question.

Removing a File from Managed Hosts

A basic example to remove a file from managed hosts is to use the `file` module with the `state: absent` parameter. The `state` parameter is optional to many modules. You should always make your intentions clear whether you want `state: present` or `state: absent` for several reasons. Some modules support other options as well. It is possible that the default could change at some point, but perhaps most importantly, it makes it easier to understand the state the system should be in based on your task.

```
- name: Make sure a file does not exist on managed hosts
  file:
    dest: /path/to/file
    state: absent
```

Retrieving the Status of a File on Managed Hosts

The `stat` module retrieves facts for a file, similar to the Linux `stat` command. Parameters provide the functionality to retrieve file attributes, determine the checksum of a file, and more.

The `stat` module returns a hash dictionary of values containing the file status data, which allows you to refer to individual pieces of information using separate variables.

The following example registers the results of a `stat` module and then prints the MD5 checksum of the file that it checked. (The more modern SHA256 algorithm is also available; MD5 is being used here for easier legibility.)

```
- name: Verify the checksum of a file
  stat:
    path: /path/to/file
    checksum_algorithm: md5
  register: result

- debug
    msg: "The checksum of the file is {{ result.stat.checksum }}"
```

The outcome should be similar to the following:

```
TASK [Get md5 checksum of a file] ***************************************
ok: [hostname]

TASK [debug] ************************************************************
ok: [hostname] => {
    "msg": "The checksum of the file is 5f76590425303022e933c43a7f2092a3"
}
```

Information about the values returned by the `stat` module are documented by `ansible-doc`, or you can register a variable and display its contents to see what is available:

```
- name: Examine all stat output of /etc/passwd
  hosts: localhost

  tasks:
    - name: stat /etc/passwd
      stat:
        path: /etc/passwd
      register: results

    - name: Display stat results
      debug:
        var: results
```

Synchronizing Files Between the Control Node and Managed Hosts

The `synchronize` module is a wrapper around the `rsync` tool, which simplifies common file management tasks in your playbooks. The `rsync` tool must be installed on both the local and remote host. By default, when using the `synchronize` module, the "local host" is the host that the synchronize task originates on (usually the control node), and the "destination host" is the host that `synchronize` connects to.

The following example synchronizes a file located in the Ansible working directory to the managed hosts:

```
- name: synchronize local file to remote files
  synchronize:
    src: file
    dest: /path/to/file
```

There are many ways to use the `synchronize` module and its many parameters, including synchronizing directories. Run the `ansible-doc synchronize` command for additional parameters and playbook examples.

REFERENCES

`ansible-doc`(1), `chmod`(1), `chown`(1), `rsync`(1), `stat`(1) and `touch`(1) man pages

Files modules (https://docs.ansible.com/ansible/2.9/modules/list_of_files_modules.html)

➜ NEXT (/ROL/APP/COURSES/RH294-8.4/PAGES/CH05S02)