

9. 验证 Web 服务器现在是否响应 HTTPS 请求，使用自签名自定义证书加密连接。Web 服务器响应与字符串 **Configured for both HTTP and HTTPS** 匹配。

```
[student@workstation control-review]$ curl -k -vvv https://serverb.lab.example.com
* About to connect() to serverb.lab.example.com port 443 (#0)
*   Trying 172.25.250.11...
* Connected to serverb.lab.example.com (172.25.250.11) port 443 (#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
* skipping SSL peer certificate verification
* SSL connection using TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
* Server certificate:
...output omitted...
*   start date: Nov 13 15:52:18 2018 GMT
*   expire date: Aug 09 15:52:18 2021 GMT
*   common name: serverb.lab.example.com
...output omitted...
< Accept-Ranges: bytes
< Content-Length: 36
< Content-Type: text/html; charset=UTF-8
<
Configured for both HTTP and HTTPS.
* Connection #0 to host serverb.lab.example.com left intact
```

评估

在 workstation 上运行 **lab control-review grade** 命令，确认是否成功完成本练习。更正报告的所有错误并重新运行脚本，直到成功为止。

```
[student@workstation ~]$ lab control-review grade
```

完成

运行 **lab control-review finish** 命令，进行实验后清理。

```
[student@workstation ~]$ lab control-review finish
```

总结

在本章中，您学到了：

- 循环用于迭代一组值，例如一个简单的字符串列表或一个散列或字典列表。
- 条件用于仅在符合特定条件时执行任务或 play。
- 处理程序是特殊的任务，它们在收到其他任务通知时在 play 的末尾执行。
- 仅当任务报告其在受管主机上更改了某些内容时，才会通知处理程序。
- 对任务进行配置，从而通过忽略任务失败、在任务失败时强制调用处理程序来处理错误，即使任务成功时也将它标记为失败，或者覆盖导致任务被标记为已更改的行为。
- 块用于将任务分组为单元，并且根据块中的所有任务是否都已成功来执行其他任务。

章 6

在被管理节点上创建文件或目录

目标

在 Ansible 管理的主机上部署、管理和调整文件。

培训目标

- 在受管主机上创建、安装、编辑和删除文件，以及管理这些文件的权限、所有权、SELinux 上下文和其他特征。
- 使用 Jinja2 模板自定义文件，并将它们部署到受管主机。

章节

- 修改文件并将其复制到主机（及引导式练习）
- 使用 Jinja2 模板部署自定义文件（及引导式练习）

实验

- 将文件部署到受管主机

修改文件并将其复制到主机

培训目标

学完本节后，您应能够在受管主机上创建、安装、编辑和删除文件，以及管理这些文件的权限、所有权、SELinux 上下文和其他特征。

描述文件模块

红帽 Ansible 引擎附带了作为上游 Ansible 项目的一部分开发的一个大型模块集合（“模块库”）。为了便于整理、记录和管理这些模块，已根据文档中的功能以及在系统上安装的时间将它们分成多个组。

Files 模块库包含的模块允许您完成与 Linux 文件管理相关的大多数任务，如创建、复制、编辑和修改文件的权限和其他属性。下表提供了常用文件管理模块的列表：

常用文件模块

模块名称	模块说明
<code>blockinfile</code>	插入、更新或删除由可自定义标记线包围的多行文本块。
<code>copy</code>	将文件从本地或远程计算机复制到受管主机上的某个位置。类似于 <code>file</code> 模块， <code>copy</code> 模块还可以设置文件属性，包括 SELinux 上下文。
<code>fetch</code>	此模块的作用和 <code>copy</code> 模块类似，但以相反方式工作。此模块用于从远程计算机获取文件到控制节点，并将它们存储在按主机名组织的文件树中。
<code>file</code>	设置权限、所有权、SELinux 上下文以及常规文件、符号链接、硬链接和目录的时间戳等属性。此模块还可以创建或删除常规文件、符号链接、硬链接和目录。其他多个与文件相关的模块支持与 <code>file</code> 模块相同的属性设置选项，包括 <code>copy</code> 模块。
<code>lineinfile</code>	确保特定行位于某个文件中，或使用反向引用正则表达式来替换现有行。此模块主要在您想要更改文件中的某一行时使用。
<code>stat</code>	检索文件的状态信息，类似于 Linux <code>stat</code> 命令。
<code>synchronize</code>	围绕 <code>rsync</code> 命令的一个打包程序，可加快和简化常见任务。 <code>synchronize</code> 模块无法提供对 <code>rsync</code> 命令的完整功能的访问权限，但确实使最常见的调用更容易实施。根据您的用例，您可能仍需要通过 <code>run command</code> 模块直接调用 <code>rsync</code> 命令。

FILES 模块的自动化示例

在受管主机上创建、复制、编辑和删除文件是您可以使用 **Files** 模块库中的模块实施的常见任务。以下示例显示了可以使用这些模块自动执行常见文件管理任务的方式。

确保受管主机上存在文件

使用 `file` 模块处理受管主机上的文件。其工作方式与 `touch` 命令类似，如果不存在则创建一个空文件，如果存在，则更新其修改时间。在本例中，除了处理文件之外，Ansible 还确保将文件的拥有用户、组和权限设置为特定值。

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

示例结果：

```
$ ls -l file
-rw-r----- user1 group1 0 Nov 25 08:00 file
```

修改文件属性

您可以使用 `file` 模块，确保新的或现有的文件具有正确的权限和 SELinux 类型。

例如，以下文件保留了相对于用户主目录的默认 SELinux 上下文，这不是所需的上下文。

```
$ ls -Z samba_file
-rw-r--r-- owner group unconfined_u:object_r:user_home_t:s0 samba_file
```

以下任务确保了 `samba_file` 文件的 SELinux 上下文类型属性是所需的 `samba_share_t` 类型。此行为与 Linux `chcon` 命令类似。

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

示例结果：

```
$ ls -Z samba_file
-rw-r--r-- owner group unconfined_u:object_r:samba_share_t:s0 samba_file
```



注意

文件属性参数在多个文件管理模块中可用。运行 `ansible-doc file` 和 `ansible-doc copy` 命令以获取其他信息。

使 SELinux 文件上下文更改具有持久性

设置文件上下文时，`file` 模块的行为与 `chcon` 类似。通过运行 `restorecon`，可能会意外地撤销使用该模块所做的更改。使用 `file` 设置上下文后，您可以使用 `System` 模块集合中的 `sefcontext` 来更新 SELinux 策略，如 `semanage fcontext`。

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

示例结果：

```
$ ls -Z samba_file
-rw-r--r--  owner group unconfined_u:object_r:samba_share_t:s0 samba_file
```



重要

`sefcontext` 模块更新 SELinux 策略中目标的默认上下文，但不更改现有文件的上下文。

在受管主机上复制和编辑文件

在此示例中，`copy` 模块用于将位于控制节点上的 Ansible 工作目录中的文件复制到选定的受管主机。

默认情况下，此模块假定设置了 `force: yes`。这会强制该模块覆盖远程文件（如果存在但包含与正在复制的文件不同的内容）。如果设置 `force: no`，则它仅会将该文件复制到受管主机（如果该文件尚不存在）。

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

要从受管主机检索文件，请使用 `fetch` 模块。这可用于在将参考系统分发给其他受管主机之前从参考系统中检索诸如 SSH 公钥之类的文件。

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

要确保现有文件中存在特定的单行文本，请使用 `lineinfile` 模块：

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

要将文本块添加到现有文件，请使用 `blockinfile` 模块：

```
- 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
```



注意

使用 `blockinfile` 模块时，注释块标记插入到块的开头和结尾，以确保幂等性。

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

您可以使用该模块的 `marker` 参数，帮助确保将正确的注释字符或文本用于相关文件。

从受管主机中删除文件

从受管主机中删除文件的基本示例是使用 `file` 模块和 `state: absent` 参数。`state` 参数对于许多模块是可选的。出于多个原因，您应始终明确您的意图，即您是需要 `state: present` 还是 `state: absent`。一些模块也支持其他选项。默认值可能会在某个时候发生变化，但也许最重要的是可以更轻松地根据您的任务了解系统应处于的状态。

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

检索受管主机上的文件状态

`stat` 模块检索文件的事实，类似于 Linux `stat` 命令。参数提供检索文件属性、确定文件校验和等功能。

`stat` 模块返回一个包含文件状态数据的值的散列字典，允许您使用单独的变量引用各条信息。

以下示例注册 `stat` 模块的结果，然后显示它检查的文件的 MD5 校验和。（也可使用更现代的 SHA256 算法；这里使用 MD5 以提高易读性。）

```
- 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 }}"
```

结果应类似于下文：

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

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

有关 `stat` 模块返回的值的信息由 `ansible-doc` 记录，或者您可以注册一个变量并显示其内容以查看可用内容：

```
- 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
```

同步控制节点和受管主机之间的文件

`synchronize` 模块是一个围绕 `rsync` 工具的打包程序，它简化了 playbook 中的常见文件管理任务。`rsync` 工具必须同时安装在本地和远程主机上。默认情况下，在使用 `synchronize` 模块时，“本地主机”是同步任务所源自的主机（通常是控制节点），而“目标主机”是 `synchronize` 连接到的主机。

以下示例将位于 Ansible 工作目录中的文件同步到受管主机：

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

有很多种方法可以使用 `synchronize` 模块及其许多参数，包括同步目录。运行 `ansible-doc synchronize` 命令查看其他参数和 playbook 示例。



参考文献

`ansible-doc(1)`、`chmod(1)`、`chown(1)`、`rsync(1)`、`stat(1)` 和 `touch(1)` 帮助手册

Files 模块

https://docs.ansible.com/ansible/latest/modules/list_of_files_modules.html

► 指导练习

修改文件并将其复制到主机

在本练习中，您将使用标准 Ansible 模块在受管主机上创建、安装、编辑和删除文件，以及管理这些文件的权限、所有权和 SELinux 上下文。

成果

您应能够：

- 按主机名从受管主机检索文件并在本地存储它们。
- 创建使用通用文件管理模块（例如 `copy`、`file`、`lineinfile` 和 `blockinfile`）的 playbook。

在你开始之前

以 `student` 用户身份并使用 `student` 作为密码登录 `workstation`。

在 `workstation` 上，运行 `lab file-manage start` 命令。该脚本将创建 `file-manage` 项目目录，并下载练习所需的 Ansible 配置文件和主机清单文件。

```
[student@workstation ~]$ lab file-manage start
```

- 1. 以 `student` 用户身份，在 `workstation` 上更改到 `/home/student/file-manage` 工作目录。在当前工作目录中，创建名为 `secure_log_backups.yml` 的 playbook。配置该 playbook 以使用 `fetch` 模块从每个受管主机检索 `/var/log/secure` 日志文件，并将它们存储在控制节点上。该 playbook 应创建 `secure-backups` 目录，并以每个受管主机的主机名命名子目录。将备份文件存储在各自的子目录中。

- 1.1. 前往 `/home/student/file-manage` 工作目录。

```
[student@workstation ~]$ cd ~/file-manage  
[student@workstation file-manage]$
```

- 1.2. 创建 `secure_log_backups.yml` playbook，使其包含以下初始内容：

```
---  
- name: Use the fetch module to retrieve secure log files  
  hosts: all  
  remote_user: root
```

- 1.3. 添加任务到 `secure_log_backups.yml` playbook，该任务从受管主机检索 `/var/log/secure` 日志文件并将其存储在 `~/file-manage/secure-backups` 目录中。如果目录 `~/file-manage/secure-backups` 不存在，`fetch` 模块就会创建该目录。使用 `flat: no` 参数，确保将主机名、路径和文件名附加到目标的默认行为：

```
tasks:  
  - name: Fetch the /var/log/secure log file from managed hosts  
    fetch:  
      src: /var/log/secure  
      dest: secure-backups  
      flat: no
```

- 1.4. 在运行 playbook 之前, 请运行 **ansible-playbook --syntax-check secure_log_backups.yml** 命令以验证其语法。更正所有错误, 再继续下一步。

```
[student@workstation file-manage]$ ansible-playbook --syntax-check \  
> secure_log_backups.yml
```

```
playbook: secure_log_backups.yml
```

- 1.5. 运行 **ansible-playbook secure_log_backups.yml** 以执行此 playbook:

```
[student@workstation file-manage]$ ansible-playbook secure_log_backups.yml  
PLAY [Use the fetch module to retrieve secure log files] *****  
  
TASK [Gathering Facts] *****  
ok: [servera.lab.example.com]  
ok: [serverb.lab.example.com]  
  
TASK [Fetch the /var/log/secure file from managed hosts] *****  
changed: [serverb.lab.example.com]  
changed: [servera.lab.example.com]  
  
PLAY RECAP *****  
servera.lab.example.com : ok=2     changed=1     unreachable=0     failed=0  
serverb.lab.example.com : ok=2     changed=1     unreachable=0     failed=0
```

- 1.6. 验证 playbook 结果:

```
[student@workstation file-manage]$ tree -F secure-backups  
secure-backups  
├── servera.lab.example.com/  
│   └── var/  
│       └── log/  
│           └── secure  
└── serverb.lab.example.com/  
    └── var/  
        └── log/  
            └── secure
```

- 2. 在当前工作目录中, 创建 playbook **copy_file.yml**。以 root 用户身份配置该 playbook, 将 **/home/student/file-manage/files/users.txt** 文件复制到所有受管主机。

- 2.1. 将以下初始内容添加到 **copy_file.yml** playbook:

```
---
- name: Using the copy module
hosts: all
remote_user: root
```

2.2. 添加一个任务，以使用 `copy` 模块将 `/home/student/file-manage/files/users.txt` 文件复制到所有受管主机。使用 `copy` 模块为 `users.txt` 文件设置以下参数：

参数	值
src	<code>files/users.txt</code>
dest	<code>/home/devops/users.txt</code>
owner	<code>devops</code>
group	<code>devops</code>
mode	<code>u+rw,g-wx,o-rwx</code>
setype	<code>samba_share_t</code>

```
tasks:
- name: Copy a file to managed hosts and set attributes
  copy:
    src: files/users.txt
    dest: /home/devops/users.txt
    owner: devops
    group: devops
    mode: u+rw,g-wx,o-rwx
    setype: samba_share_t
```

2.3. 使用 `ansible-playbook --syntax-check copy_file.yml` 命令，验证 `copy_file.yml` playbook 的语法。

```
[student@workstation file-manage]$ ansible-playbook --syntax-check copy_file.yml
playbook: copy_file.yml
```

2.4. 运行该 playbook：

```
[student@workstation file-manage]$ ansible-playbook copy_file.yml
PLAY [Using the copy module] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [Copy a file to managed hosts and set attributes] ****
```

```
changed: [servera.lab.example.com]
changed: [serverb.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2      changed=1      unreachable=0      failed=0
serverb.lab.example.com : ok=2      changed=1      unreachable=0      failed=0
```

- 2.5. 以用户 **devops** 的身份使用临时命令执行 **ls -Z** 命令，验证受管主机上的 **users.txt** 文件的属性。

```
[student@workstation file-manage]$ ansible all -m command -a 'ls -Z' -u devops
servera.lab.example.com | CHANGED | rc=0 >>
unconfined_u:object_r:samba_share_t:s0 users.txt

serverb.lab.example.com | CHANGED | rc=0 >>
unconfined_u:object_r:samba_share_t:s0 users.txt
```

- 3. 在上一步中，为 **users.txt** 文件设置了 **samba_share_t** SELinux 类型字段。但是，现在确定应为 SELinux 文件上下文设置默认值。

在当前工作目录中，创建名为 **selinux_defaults.yml** 的 playbook。配置该 playbook 以使用 **file** 模块，确保 **user**、**role**、**type** 和 **level** 字段的默认 SELinux 上下文。



注意

在现实世界中，您也可以编辑 **copy_file.yml** 并删除 **setype** 关键词。

3.1. 创建 **selinux_defaults.yml** playbook:

```
---
- name: Using the file module to ensure SELinux file context
  hosts: all
  remote_user: root
  tasks:
    - name: SELinux file context is set to defaults
      file:
        path: /home/devops/users.txt
        seuser: _default
        serole: _default
        setype: _default
        selevel: _default
```

3.2. 使用 **ansible-playbook --syntax-check** **selinux_defaults.yml** 命令，验证 **selinux_defaults.yml** playbook 的语法。

```
[student@workstation file-manage]$ ansible-playbook --syntax-check
> selinux_defaults.yml

playbook: selinux_defaults.yml
```

3.3. 运行该 playbook:

```
[student@workstation file-manage]$ ansible-playbook selinux_defaults.yml
PLAY [Using the file module to ensure SELinux file context] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [SELinux file context is set to defaults] ****
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
serverb.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

- 3.4. 以用户 `devops` 的身份使用临时命令执行 `ls -Z` 命令，验证 `unconfined_u:object_r:user_home_t:s0` 的默认文件属性。

```
[student@workstation file-manage]$ ansible all -m command -a 'ls -Z' -u devops
servera.lab.example.com | CHANGED | rc=0 >>
unconfined_u:object_r:user_home_t:s0 users.txt

serverb.lab.example.com | CHANGED | rc=0 >>
unconfined_u:object_r:user_home_t:s0 users.txt
```

- 4. 在当前工作目录中，创建名为 `add_line.yml` 的 playbook。配置该 playbook 以使用 `lineinfile` 模块将 `This line was added by the lineinfile module.` 这一行附加到所有受管主机上的 `/home/devops/users.txt` 文件。

- 4.1. 创建 `add_line.yml` playbook:

```
---
- name: Add text to an existing file
  hosts: all
  remote_user: devops
  tasks:
    - name: Add a single line of text to a file
      lineinfile:
        path: /home/devops/users.txt
        line: This line was added by the lineinfile module.
        state: present
```

- 4.2. 使用 `ansible-playbook --syntax-check add_line.yml` 命令，验证 `add_line.yml` playbook 的语法。

```
[student@workstation file-manage]$ ansible-playbook --syntax-check add_line.yml
playbook: add_line.yml
```

- 4.3. 运行该 playbook:

```
[student@workstation file-manage]$ ansible-playbook add_line.yml
PLAY [Add text to an existing file] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [Add a single line of text to a file] ****
changed: [servera.lab.example.com]
changed: [serverb.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
serverb.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

4.4. 以 devops 用户身份使用 command 模块和 cat 选项，验证受管主机上的 **users.txt** 文件的内容。

```
[student@workstation file-manage]$ ansible all -m command \
> -a 'cat users.txt' -u devops
serverb.lab.example.com | CHANGED | rc=0 >>
This line was added by the lineinfile module.

servera.lab.example.com | CHANGED | rc=0 >>
This line was added by the lineinfile module.
```

- 5. 在当前工作目录中，创建名为 **add_block.yml** 的 playbook。配置该 playbook 以使用 **blockinfile** 模块将以下文本块附加到所有受管主机上的 **/home/devops/users.txt** 文件。

```
This block of text consists of two lines.
They have been added by the blockinfile module.
```

5.1. 创建 **add_block.yml** playbook:

```
---
- name: Add block of text to a file
  hosts: all
  remote_user: devops
  tasks:
    - name: Add a block of text to an existing file
      blockinfile:
        path: /home/devops/users.txt
        block: |
          This block of text consists of two lines.
          They have been added by the blockinfile module.
        state: present
```

5.2. 使用 **ansible-playbook --syntax-check add_block.yml** 命令，验证 **add_block.yml** playbook 的语法。

```
[student@workstation file-manage]$ ansible-playbook --syntax-check add_block.yml  
playbook: add_block.yml
```

5.3. 运行该 playbook:

```
[student@workstation file-manage]$ ansible-playbook add_block.yml  
PLAY [Add block of text to a file] *****  
TASK [Gathering Facts] *****  
ok: [serverb.lab.example.com]  
ok: [servera.lab.example.com]  
  
TASK [Add a block of text to an existing file] *****  
changed: [servera.lab.example.com]  
changed: [serverb.lab.example.com]  
  
PLAY RECAP *****  
servera.lab.example.com : ok=2     changed=1      unreachable=0    failed=0  
serverb.lab.example.com : ok=2     changed=1      unreachable=0    failed=0
```

5.4. 使用 command 模块和 cat 命令，以验证受管主机上的 /home/devops/users.txt 文件的正确内容。

```
[student@workstation file-manage]$ ansible all -m command \  
> -a 'cat users.txt' -u devops  
serverb.lab.example.com | CHANGED | rc=0 >>  
This line was added by the lineinfile module.  
# BEGIN ANSIBLE MANAGED BLOCK  
This block of text consists of two lines.  
They have been added by the blockinfile module.  
# END ANSIBLE MANAGED BLOCK  
  
servera.lab.example.com | CHANGED | rc=0 >>  
This line was added by the lineinfile module.  
# BEGIN ANSIBLE MANAGED BLOCK  
This block of text consists of two lines.  
They have been added by the blockinfile module.  
# END ANSIBLE MANAGED BLOCK
```

- 6. 在当前工作目录中，创建名为 remove_file.yml 的 playbook。配置该 playbook 以使用 file 模块删除所有受管主机上的 /home/devops/users.txt 文件。

6.1. 创建 remove_file.yml playbook:

```
---  
- name: Use the file module to remove a file  
  hosts: all  
  remote_user: devops  
  tasks:
```

```
- name: Remove a file from managed hosts
  file:
    path: /home/devops/users.txt
    state: absent
```

6.2. 使用 **ansible-playbook --syntax-check remove_file.yml** 命令，验证 **remove_file.yml** playbook 的语法。

```
[student@workstation file-manage]$ ansible-playbook --syntax-check remove_file.yml
playbook: remove_file.yml
```

6.3. 运行该 playbook:

```
[student@workstation file-manage]$ ansible-playbook remove_file.yml
PLAY [Use the file module to remove a file] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [Remove a file from managed hosts] ****
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com      : ok=2      changed=1      unreachable=0      failed=0
serverb.lab.example.com      : ok=2      changed=1      unreachable=0      failed=0
```

6.4. 使用一个临时命令来执行 **ls -l** 命令，确认受管主机上不再存在 **users.txt** 文件。

```
[student@workstation file-manage]$ ansible all -m command -a 'ls -l'
serverb.lab.example.com | CHANGED | rc=0 >>
total 0

servera.lab.example.com | CHANGED | rc=0 >>
total 0
```

完成

在 workstation 上，运行 **lab file-manage finish** 脚本来清理本练习。

```
[student@workstation ~]$ lab file-manage finish
```

本引导式练习到此结束。

使用 JINJA2 模板部署自定义文件

培训目标

学完本节后，您应能够使用 Jinja2 模板将自定义文件部署到受管主机。

构建文件模板

红帽 Ansible 引擎有许多模块可用于修改现有文件。其中包括 `lineinfile` 和 `blockinfile` 等等。但是，它们并不总是易于有效和正确地使用。

管理文件的一种更强大的方法是为其构建模板。通过此方法，您可以使用 Ansible 变量和事实编写一个模板配置文件，在部署该文件时自动为受管主机自定义此模板配置文件。这样更容易控制，也更不易出错。

JINJA2 简介

Ansible 将 Jinja2 模板系统用于模板文件。Ansible 还使用 Jinja2 语法来引用 playbook 中的变量，因此您已经对如何使用它有了一些了解。

使用分隔符

变量和逻辑表达式置于标记或分隔符之间。例如，Jinja2 模板将 `{% EXPR %}` 用于表达式或逻辑（如循环），而 `{{ EXPR }}` 则用于向最终用户输出表达式或变量的结果。后一标记在呈现时将被替换为一个或多个值，对最终用户可见。使用 `{# COMMENT #}` 语法括起不应出现在最终文件中的注释。

在下例中，第一行中含有不会包含于最终文件中的注释。第二行中引用的变量被替换为所引用的事实的值。

```
{# /etc/hosts line #}
{{ ansible_facts['default_ipv4']['address'] }}    {{ ansible_facts['hostname'] }}
```

构建 JINJA2 模板

Jinja2 模板由多个元素组成：数据、变量和表达式。在呈现 Jinja2 模板时，这些变量和表达式被替换为对应的值。模板中使用的变量可以在 playbook 的 `vars` 部分中指定。可以将受管主机的事实用作模板中的变量。



注意

请记住，可以使用 `ansible system_hostname -i inventory_file -m setup` 命令来获取与受管主机相关的事实。

下例演示了如何使用变量及 Ansible 从受管主机检索的事实创建 `/etc/ssh/sshd_config` 的模板。当执行相关的 playbook 时，任何事实都将被替换为所配置的受管主机中对应的值。

**注意**

包含 Jinja2 模板的文件不需要有任何特定的文件扩展名（如 **.j2**）。但是，提供此类文件扩展名可能会使您更容易记住它是模板文件。

```

# {{ ansible_managed }}
# DO NOT MAKE LOCAL MODIFICATIONS TO THIS FILE AS THEY WILL BE LOST

Port {{ ssh_port }}
ListenAddress {{ ansible_facts['default_ipv4']['address'] }}

HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
HostKey /etc/ssh/ssh_host_ed25519_key

SyslogFacility AUTHPRIV

PermitRootLogin {{ root_allowed }}
AllowGroups {{ groups_allowed }}

AuthorizedKeysFile /etc/.rht_authorized_keys .ssh/authorized_keys

PasswordAuthentication {{ passwords_allowed }}

ChallengeResponseAuthentication no

GSSAPIAuthentication yes
GSSAPICleanupCredentials no

UsePAM yes

X11Forwarding yes
UsePrivilegeSeparation sandbox

AcceptEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY LC_MESSAGES
AcceptEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
AcceptEnv LC_IDENTIFICATION LC_ALL LANGUAGE
AcceptEnv XMODIFIERS

Subsystem sftp /usr/libexec.openssh/sftp-server

```

部署 JINJA2 模板

Jinja2 模板是功能强大的工具，可用于自定义要在受管主机上部署的配置文件。创建了适用于配置文件的 Jinja2 模板后，它可以通过 **template** 模块部署到受管主机上，该模块支持将控制节点中的本地文件转移到受管主机。

若要使用 **template** 模块，请使用下列语法。与 **src** 键关联的值指定来源 Jinja2 模板，而与 **dest** 键关联的值指定要在目标主机上创建的文件。

```
tasks:
  - name: template render
    template:
      src: /tmp/j2-template.j2
      dest: /tmp/dest-config-file.txt
```



注意

`template` 模块还允许您指定已部署文件的所有者（拥有该文件的用户）、组、权限和 SELinux 上下文，就像 `file` 模块一样。它也可以取用 `validate` 选项运行任意命令（如 `visudo -c`），在将文件复制到位之前检查该文件的语法是否正确。

有关更多详细信息，请参阅 [ansible-doc template](#)。

管理模板文件

为避免系统管理员修改 Ansible 部署的文件，最好在模板顶部包含注释，以指示不应手动编辑该文件。

可使用 `ansible_managed` 指令中设置的“Ansible managed”字符串来执行此操作。这不是正常变量，但可以在模板中用作一个变量。`ansible_managed` 指令在 `ansible.cfg` 文件中设置：

```
ansible_managed = Ansible managed
```

要将 `ansible_managed` 字符串包含在 Jinja2 模板内，请使用下列语法：

```
{{ ansible_managed }}
```

控制结构

您可以在模板文件中使用 Jinja2 控制结构，以减少重复键入，为 play 中的每个主机动态输入条目，或者有条件地将文本插入到文件中。

使用循环

Jinja2 使用 `for` 语句来提供循环功能。在下例中，`user` 变量替换为 `users` 变量中包含的所有值，一行一个值。

```
{% for user in users %}
  {{ user }}
{% endfor %}
```

以下示例模板使用 `for` 语句逐一运行 `users` 变量中的所有值，将 `myuser` 替换为各个值，但值为 `root` 时除外。

```
{# for statement #}
{% for myuser in users if not myuser == "root" %}
User number {{ loop.index }} - {{ myuser }}
{% endfor %}
```

`loop.index` 变量扩展至循环当前所处的索引号。它在循环第一次执行时值为 1，每一次迭代递增 1。

再如，此模板也使用了 `for` 语句，并且假定使用的清单文件中已定义了 `myhosts` 变量。此变量将包含要管理的主机的列表。使用下列 `for` 语句时，文件中将列出清单中 `myhosts` 组内的所有主机。

```
{% for myhost in groups['myhosts'] %}  
{{ myhost }}  
{% endfor %}
```

举一个更实际的例子，您可以使用该模板从主机事实动态生成 `/etc/hosts` 文件。假设您有以下 playbook：

```
- name: /etc/hosts is up to date  
hosts: all  
gather_facts: yes  
tasks:  
  - name: Deploy /etc/hosts  
    template:  
      src: templates/hosts.j2  
      dest: /etc/hosts
```

下述三行 `templates/hosts.j2` 模板从 `all` 组中的所有主机构造文件。（由于变量名称的长度，模板的中间行非常长。）它迭代组中的每个主机以获得 `/etc/hosts` 文件的三个事实。

```
{% for host in groups['all'] %}  
{{ hostvars['host']['ansible_facts']['default_ipv4']['address'] }}  
{{ hostvars['host']['ansible_facts']['fqdn'] }} {{ hostvars['host'] }  
['ansible_facts']['hostname'] }  
{% endfor %}
```

使用条件句

Jinja2 使用 `if` 语句来提供条件控制。如果满足某些条件，这允许您在已部署的文件中放置一行。

在以下示例中，仅当 `finished` 变量的值为 `True` 时，才可将 `result` 变量的值放入已部署的文件。

```
{% if finished %}  
{{ result }}  
{% endif %}
```



重要

您可以在 Ansible 模板中使用 Jinja2 循环和条件，但不能在 Ansible Playbook 中使用。

变量过滤器

Jinja2 提供了过滤器，更改模板表达式的输出格式（例如，输出到 JSON）。有适用于 YAML 和 JSON 等语言的过滤器。**to_json** 过滤器使用 JSON 格式化表达式输出，**to_yaml** 过滤器则使用 YAML 格式化表达式输出。

```
 {{ output | to_json }}  
 {{ output | to_yaml }}
```

也有其他过滤器，如 **to_nice_json** 和 **to_nice_yaml** 过滤器，它们将表达式输出格式化为 JSON 或 YAML 等人类可读格式。

```
 {{ output | to_nice_json }}  
 {{ output | to_nice_yaml }}
```

from_json 和 **from_yaml** 过滤器相应要求 JSON 或 YAML 格式的字符串，并对它们进行解析。

```
 {{ output | from_json }}  
 {{ output | from_yaml }}
```

变量测试

在 Ansible Playbook 中与 **when** 子句一同使用的表达式是 Jinja2 表达式。用于测试返回值的内置 Ansible 测试包括 **failed**、**changed**、**succeeded** 和 **skipped**。以下任务演示了如何在条件表达式内使用测试。

```
 tasks:  
 ...output omitted...  
 - debug: msg="the execution was aborted"  
   when: returnvalue is failed
```



参考文献

模板 - 将文件以模板形式分发到远程服务器 — Ansible 文档

https://docs.ansible.com/ansible/latest/modules/template_module.html

变量 — Ansible 文档

https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html

过滤器 — Ansible 文档

https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html

► 指导练习

使用 JINJA2 模板部署自定义文件

在本练习中，您将创建一个简单的模板文件，供您的 playbook 用来在每个受管主机上安装自定义的当日消息文件。

成果

您应能够：

- 构建模板文件。
- 在 playbook 中使用模板文件。

在你开始之前

以 student 用户身份并使用 student 作为密码登录 workstation。

在 workstation 上，运行 **lab file-template start** 命令。此脚本可确保在 workstation 上安装 Ansible，创建 /home/student/file-template 目录，并将 **ansible.cfg** 文件下载到该目录。

```
[student@workstation ~]$ lab file-template start
```



注意

本练习中使用的所有文件都位于 workstation 上的 /home/student/file-template/files 目录中供参考。

- 1. 在 workstation 上，导航到 /home/student/file-template 工作目录。
在当前工作目录中创建 **inventory** 文件。此文件配置两个组：**webservers** 和 **workstations**。将 servera.lab.example.com 系统包含在 **webservers** 组中，将 workstation.lab.example.com 系统包含在 **workstations** 组中。
- ```
[webservers]
servera.lab.example.com

[workstations]
workstation.lab.example.com
```

- 2. 为当日消息创建一个模板，并将其包括在当前工作目录中的 **motd.j2** 文件中。在模板中包含下列变量和事实：
- **ansible\_facts['fqdn']**，用于插入受管主机的 FQDN。
  - **ansible\_facts['distribution']** 和 **ansible\_facts['distribution\_version']**，用于提供分发信息。

- **system\_owner**, 用于系统所有者的电子邮件。此变量需要在 playbook 模板的 **vars** 部分中定义适当的值。

```
This is the system {{ ansible_facts['fqdn'] }}.
This is a {{ ansible_facts['distribution'] }} version
{{ ansible_facts['distribution_version'] }} system.
Only use this system with permission.
You can request access from {{ system_owner }}.
```

- 3. 在当前工作目录中，创建名为 **motd.yml** 的 playbook 文件。在 **vars** 部分中定义 **system\_owner** 变量，为 **template** 模块包含一项任务，它将 **motd.j2** Jinja2 模板映射到受管主机上的远程文件 **/etc/motd**。将所有者和组设置为 **root**，并将模式设置为 0644。

```

- name: configure SOE
 hosts: all
 remote_user: devops
 become: true
 vars:
 - system_owner: clyde@example.com
 tasks:
 - name: configure /etc/motd
 template:
 src: motd.j2
 dest: /etc/motd
 owner: root
 group: root
 mode: 0644
```

- 4. 在运行 playbook 之前，请使用 **ansible-playbook --syntax-check** 命令验证语法。如果报告任何错误，请更正后再继续下一步。您应看到类似于下文的输出：

```
[student@workstation file-template]$ ansible-playbook --syntax-check motd.yml
playbook: motd.yml
```

- 5. 运行 **motd.yml** playbook。

```
[student@workstation file-template]$ ansible-playbook motd.yml
PLAY [all] ****

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

TASK [template] ****
changed: [servera.lab.example.com]
changed: [workstation.lab.example.com]
```

```
PLAY RECAP ****
servera.lab.example.com : ok=2 changed=1 unreachable=0 failed=0
workstation.lab.example.com : ok=2 changed=1 unreachable=0 failed=0
```

- ▶ 6. 以 devops 用户身份登录 servera.lab.example.com，验证登录时是否正确显示该 MOTD。完成时注销。

```
[student@workstation file-template]$ ssh devops@servera.lab.example.com
This is the system servera.lab.example.com.
This is a RedHat version 8.0 system.
Only use this system with permission.
You can request access from clyde@example.com.
...output omitted...
[devops@servera ~]# exit
Connection to servera.lab.example.com closed.
```

## 完成

运行 **lab file-template finish** 命令，进行练习后清理。

```
[student@workstation ~]$ lab file-template finish
```

本引导式练习到此结束。

## ▶ 开放研究实验

# 将文件部署到受管主机

### 任务执行清单

在本实验中，您将运行一个 playbook，该 playbook 使用 Jinja2 模板在受管主机上创建自定义文件。

### 成果

您应能够：

- 构建模板文件。
- 在 playbook 中使用模板文件。

### 在你开始之前

以 student 用户身份并使用 student 作为密码登录 workstation。

在 workstation 上，运行 `lab file-review start` 命令。这可确保在 workstation 上安装 Ansible，创建 `/home/student/file-review` 目录，并将 `ansible.cfg` 文件下载到该目录。它也会将 `motd.yml`、`motd.j2`、`issue` 和 `inventory` 文件下载到 `/home/student/file-review/files` 目录。

```
[student@workstation ~]$ lab file-review start
```



#### 注意

本练习中使用的所有文件都位于 workstation 上的 `/home/student/file-review/files` 目录中。

1. 在 `/home/student/file-review` 目录中创建一个名为 `inventory` 的清单文件。此清单文件定义 `servers` 组，该组关联有 `serverb.lab.example.com` 受管主机。
2. 识别 `serverb.lab.example.com` 上的事实，其显示系统内存总量以及处理器数目。
3. 在当前目录中为当日消息创建一个名为 `motd.j2` 的模板。当 `devops` 用户登录 `serverb.lab.example.com` 时，应显示一条消息，其中显示系统的总内存和处理器数。使用 `ansible_facts['memtotal_mb']` 和 `ansible_facts['processor_count']` 事实为该消息提供内存信息。
4. 在当前目录中创建一个名为 `motd.yml` 的新 playbook 文件。使用 `template` 模块，配置之前创建的 `motd.j2` Jinja2 模板文件，以映射到受管主机上的文件 `/etc/motd`。此文件的所有者和组为 `root` 用户，其权限为 0644。使用 `stat` 和 `debug` 模块，创建相应的任务来验证受管主机上是否存在 `/etc/motd` 并且显示 `/etc/motd` 的文件信息。使用 `copy` 模块，将 `files/issue` 放入受管主机上的 `/etc/` 目录，并且使用与 `/etc/motd` 相同的所有权和权限。使用 `file` 模块确保 `/etc/issue.net` 是受管主机上 `/etc/issue` 的符号链接。配置该 playbook 使其使用 `devops` 用户，并将 `become` 参数设置为 `true`。