解决方案

管理变量和事实

任务执行清单

在本实验中，您将编写并运行使用变量、机密和事实的 Ansible Playbook。

成果

您应能够在 playbook 中定义变量和使用事实，以及使用加密文件中定义的变量。

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 data-review start`** 命令。该脚本将创建 **`/home/student/data-review`** 工作目录，并为它填充 Ansible 配置文件和主机清单。受管主机 `serverb.lab.example.com` 在此清单中定义为 `webserver` 主机组的成员。开发人员要求您编写一个 Ansible Playbook，以便在 `serverb.lab.example.com` 上自动设置 Web 服务器环境，该 playbook 使用基本身份验证控制用户对其网站的访问。

`files` 子目录包含：

- **`httpd.conf`** 配置文件，供 Apache Web 服务用于进行基本的身份验证
- **`.htaccess`** 文件，用于控制对 Web 服务器文档根目录的访问
- **`htpasswd`** 文件，包含被允许的用户的凭据

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

1. 在工作目录中，创建 **`playbook.yml`** playbook，并将 **`webserver`** 主机组添加为受管主机。定义以下 play 变量：

变量

变量	值
`firewall_pkg`	**`firewalld`**
`firewall_svc`	**`firewalld`**
`web_pkg`	**`httpd`**
`web_svc`	**`httpd`**
`ssl_pkg`	**`mod_ssl`**
`httpdconf_src`	**`files/httpd.conf`**
`httpdconf_dest`	**`/etc/httpd/conf/httpd.conf`**

海量视频题库 myitpub.com QQ:556462 www.52myit.com

变量	值
htaccess_src	**files/.htaccess**
secrets_dir	**/etc/httpd/secrets**
secrets_src	**files/htpasswd**
secrets_dest	**"{{ secrets_dir }}/htpasswd"**
web_root	**/var/www/html**

1.1. 更改到 **/home/student/data-review** 工作目录。

```
[student@workstation ~]$ cd ~/data-review
[student@workstation data-review]$
```

1.2. 创建 **playbook.yml** playbook 文件并在文本编辑器中编辑它。文件的开头应如下所示：

```
---
- name: install and configure webserver with basic auth
  hosts: webserver
  vars:
    firewall_pkg: firewalld
    firewall_svc: firewalld
    web_pkg: httpd
    web_svc: httpd
    ssl_pkg: mod_ssl
    httpdconf_src: files/httpd.conf
    httpdconf_dest: /etc/httpd/conf/httpd.conf
    htaccess_src: files/.htaccess
    secrets_dir: /etc/httpd/secrets
    secrets_src: files/htpasswd
    secrets_dest: "{{ secrets_dir }}/htpasswd"
    web_root: /var/www/html
```

2. 向 play 中添加一个 **tasks** 部分。编写一个任务，确保安装最新版本的必要软件包。这些软件包由 `firewall_pkg`、`web_pkg` 和 `ssl_pkg` 变量定义。

2.1. 通过向 playbook 中添加以下行，定义 **tasks** 部分的开头：

```
tasks:
```

2.2. 将以下行添加到 playbook 以定义使用 `yum` 模块安装所需软件包的任务。

```
  - name: latest version of necessary packages installed
    yum:
      name:
        - "{{ firewall_pkg }}"
        - "{{ web_pkg }}"
        - "{{ ssl_pkg }}"
      state: latest
```

海量视频题库 myitpub.com QQ:5565462
www.52myit.com

3. 向 playbook 添加第二个任务，确保由 `httpdconf_src` 变量指定的文件已被复制（使用 `copy` 模块）到受管主机上的 `httpdconf_dest` 变量指定的位置。该文件应归 `root` 用户和 `root` 组所有。同时将文件的权限设置为 0644。

   将以下行添加到 playbook 以定义一个任务，该任务使用 `copy` 模块将 `httpdconf_src` 变量定义的文件内容复制到 `httpdconf_dest` 变量指定的位置。

```
- name: configure web service
  copy:
    src: "{{ httpdconf_src }}"
    dest: "{{ httpdconf_dest }}"
    owner: root
    group: root
    mode: 0644
```

4. 添加第三个任务，该任务使用 `file` 模块创建受管主机上的 `secrets_dir` 变量指定的目录。此目录包含用于 Web 服务基本身份验证的密码文件。该文件应归 `apache` 用户和 `apache` 组所有。将 0500 设置为文件权限。

   将以下行添加到 playbook 以定义使用 `file` 模块创建 `secrets_dir` 变量定义的目录的任务。

```
- name: secrets directory exists
  file:
    path: "{{ secrets_dir }}"
    state: directory
    owner: apache
    group: apache
    mode: 0500
```

5. 添加第四个任务，该任务使用 `copy` 模块置入要用于 Web 用户基本身份验证的 htpasswd 文件。来源应由 `secrets_src` 变量定义。目标应由 `secrets_dest` 变量定义。该文件应归 `apache` 用户和组所有。Set **`0400`** as the file permissions.

```
- name: htpasswd file exists
  copy:
    src: "{{ secrets_src }}"
    dest: "{{ secrets_dest }}"
    owner: apache
    group: apache
    mode: 0400
```

海量视频题库 myitpub.com QQ:556546 2
www.52myit.com

6. 添加第五个任务，该任务使用 copy 模块在 Web 服务器的文档根目录中创建 **.htaccess** 文件。将 htaccess_src 变量指定的文件复制到 **{{web_root}}/.htaccess**。该文件应归 apache 用户和 apache 组所有。将 0400 设置为文件权限。

   将以下行添加到 playbook 以定义一个任务，该任务使用 copy 模块以通过 htaccess_src 变量定义的文件创建 **.htaccess** 文件。

```
- name: .htaccess file installed in docroot
  copy:
    src: "{{ htaccess_src }}"
    dest: "{{ web_root }}/.htaccess"
    owner: apache
    group: apache
    mode: 0400
```

7. 添加第六个任务，该任务使用 copy 模块在 web_root 变量指定的目录中创建 Web 内容文件 **index.html**。该文件应包含消息 “HOSTNAME (IPADDRESS) has been customized by Ansible.”；其中，**HOSTNAME** 是受管主机的完全限定主机名，**IPADDRESS** 是其 IPv4 IP 地址。使用 copy 模块的 **content** 选项指定文件的内容，使用 Ansible 事实指定主机名和 IP 地址。

   将以下行添加到 playbook 以定义使用 copy 模块在 web_root 变量定义的目录中创建 **index.html** 文件的任务。使用从受管主机检索的 ansible_facts['fqdn'] 和 ansible_facts['default_ipv4']['address'] Ansible 事实，为此文件填充指定的内容。

```
- name: create index.html
  copy:
    content: "{{ ansible_facts['fqdn'] }} ({{ ansible_facts['default_ipv4']
['address'] }}) has been customized by Ansible.\n"
    dest: "{{ web_root }}/index.html"
```

8. 添加第七个任务，该任务使用 service 模块在受管主机上启用并启动防火墙服务。

   将以下行添加到 playbook 以定义使用 service 模块启用并启动防火墙服务的任务。

```
- name: firewall service enabled and started
  service:
    name: "{{ firewall_svc }}"
    state: started
    enabled: true
```

9. 添加第八个任务，该任务使用 firewalld 模块允许用户所需的 https 服务访问受管主机上的 Web 服务。此防火墙更改应该是持久的，且应立即进行。

   将以下行添加到 playbook 以定义使用 firewalld 模块打开 Web 服务的 HTTPS 端口的任务。

```
- name: open the port for the web server
  firewalld:
    service: https
    state: enabled
    immediate: true
    permanent: true
```

海量视频题库 myitpub.com QQ:556462
www.52myit.com

10. 添加最后一个任务，该任务使用 `service` 模块在受管主机上启用并启动 Web 服务，以使所有配置更改生效。Web 服务的名称由 `web_svc` 变量定义。

```
- name: web service enabled and started
  service:
    name: "{{ web_svc }}"
    state: started
    enabled: true
```

11. 定义目标为 `localhost` 的第二个 play，它将测试 Web 服务器身份验证。它不需要特权升级。定义名为 `web_user` 的变量，值为 **`guest`**。

11.1.添加下面这行，以定义第二个 play 的开头。注意这里没有缩进。

```
- name: test web server with basic auth
```

11.2.添加下面这行，以注明该 play 适用于 `localhost` 受管主机。

```
hosts: localhost
```

11.3.添加下面这行以禁用特权升级。

```
become: no
```

11.4.添加以下行，以定义变量列表和 `web_user` 变量。

```
vars:
  web_user: guest
```

12. 向 play 添加一个指令，该指令从名为 **`vars/secret.yml`** 的变量文件添加其他变量。此文件包含一个指定 Web 用户密码的变量。您将在稍后的实验中创建此文件。

定义任务列表的开头。

12.1.使用 `vars_files` 关键字，将以下行添加到 playbook 中，以指示 Ansible 使用在 **`vars/secret.yml`** 变量文件中找到的变量。

```
vars_files:
  - vars/secret.yml
```

12.2.添加下面这行以定义 **`tasks`** 列表的开头。

```
tasks:
```

13. 在第二个 play 中添加两个任务。

第一个任务使用 `uri` 模块并利用基本身份验证从 `https://serverb.lab.example.com` 请求内容。请注意，`serverb` 出示的证书将不受信任，因此您需要避免证书验证。该任务应验证返回 HTTP 状态代码是否为 **`200`**。配置任务以将返回的内容放入任务结果变量中。将任务结果注册到变量中。

第二个任务使用 `debug` 模块来打印从 Web 服务器返回的内容。

海量视频题库 mp.youb.com QQ:5565462
www.52mvit.com

13.1.添加以下几行，以创建从控制节点验证 Web 服务的任务。务必使用四个空格缩进第一行。

```
  - name: connect to web server with basic auth
    uri:
      url: https://serverb.lab.example.com
      validate_certs: no
      force_basic_auth: yes
      user: "{{ web_user }}"
      password: "{{ web_pass }}"
      return_content: yes
      status_code: 200
    register: auth_test
```

13.2.使用 debug 模块创建第二个任务。从 Web 服务器返回的内容将以 **content** 键的形式添加到注册的变量中。

```
  - debug:
      var: auth_test.content
```

13.3.完成的 playbook 应当如下所示：

```
---
- name: install and configure webserver with basic auth
  hosts: webserver
  vars:
    firewall_pkg: firewalld
    firewall_svc: firewalld
    web_pkg: httpd
    web_svc: httpd
    ssl_pkg: mod_ssl
    httpdconf_src: files/httpd.conf
    httpdconf_dest: /etc/httpd/conf/httpd.conf
    htaccess_src: files/.htaccess
    secrets_dir: /etc/httpd/secrets
    secrets_src: files/htpasswd
    secrets_dest: "{{ secrets_dir }}/htpasswd"
    web_root: /var/www/html
  tasks:
    - name: latest version of necessary packages installed
      yum:
        name:
          - "{{ firewall_pkg }}"
          - "{{ web_pkg }}"
          - "{{ ssl_pkg }}"
        state: latest

    - name: configure web service
      copy:
        src: "{{ httpdconf_src }}"
        dest: "{{ httpdconf_dest }}"
        owner: root
        group: root
```

海量视频题库 myitpub.com QQ:5565462
www.52myit.com

```
        mode: 0644

    - name: secrets directory exists
      file:
        path: "{{ secrets_dir }}"
        state: directory
        owner: apache
        group: apache
        mode: 0500

    - name: htpasswd file exists
      copy:
        src: "{{ secrets_src }}"
        dest: "{{ secrets_dest }}"
        owner: apache
        group: apache
        mode: 0400

    - name: .htaccess file installed in docroot
      copy:
        src: "{{ htaccess_src }}"
        dest: "{{ web_root }}/.htaccess"
        owner: apache
        group: apache
        mode: 0400

    - name: create index.html
      copy:
        content: "{{ ansible_facts['fqdn'] }} ({{ ansible_facts['default_ipv4']
['address'] }}) has been customized by Ansible.\n"
        dest: "{{ web_root }}/index.html"

    - name: firewall service enable and started
      service:
        name: "{{ firewall_svc }}"
        state: started
        enabled: true

    - name: open the port for the web server
      firewalld:
        service: https
        state: enabled
        immediate: true
        permanent: true

    - name: web service enabled and started
      service:
        name: "{{ web_svc }}"
        state: started
        enabled: true

- name: test web server with basic auth
  hosts: localhost
  become: no
  vars:
```

海量视频题库 myitpub.com QQ:5565462
www.52myit.com

```
    - web_user: guest
  vars_files:
    - vars/secret.yml
  tasks:
    - name: connect to web server with basic auth
      uri:
        url: https://serverb.lab.example.com
        validate_certs: no
        force_basic_auth: yes
        user: "{{ web_user }}"
        password: "{{ web_pass }}"
        return_content: yes
        status_code: 200
      register: auth_test

    - debug:
        var: auth_test.content
```

13.4.保存并关闭该 **playbook.yml** 文件。

14. 创建一个使用 Ansible Vault 加密、名为 **vars/secret.yml** 的文件。它应该将 web_pass 变量设置为 **redhat**，这将是 Web 用户的密码。

14.1.在工作目录中创建一个名为 **vars** 的子目录。

```
[student@workstation data-review]$ mkdir vars
```

14.2.使用 Ansible Vault 创建加密变量文件 **vars/secret.yml**。将加密文件的密码设置为 **redhat**。

```
[student@workstation data-review]$ ansible-vault create vars/secret.yml
New Vault password: redhat
Confirm New Vault password: redhat
```

14.3.将下列变量定义添加到该文件。

```
web_pass: redhat
```

14.4.保存并关闭该文件。

15. 运行 **playbook.yml** playbook。验证是否从 Web 服务器成功返回了内容，该内容是否与先前任务中配置的内容相匹配。

15.1.运行该 playbook 前，通过运行 **ansible-playbook --syntax-check** 来验证其语法是否正确。使用 **--ask-vault-pass** 提示输入 vault 密码。系统提示输入密码时，请输入 **redhat**。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 data-review]$ ansible-playbook --syntax-check \
> --ask-vault-pass playbook.yml
Vault password: redhat

playbook: playbook.yml
```

海量视频题库 myitpub.com QQ:3565462
www.52myit.com

15.2.使用 **ansible-playbook** 命令，通过 **--ask-vault-pass** 选项运行该 playbook。系统提示输入密码时，请输入 **redhat**。

```
[student@workstation data-review]$ ansible-playbook playbook.yml --ask-vault-pass
Vault password: redhat
PLAY [Install and configure webserver with basic auth] ********************

...output omitted...

TASK [connect to web server with basic auth] ***********************************
ok: [localhost]

TASK [debug] *******************************************************************
ok: [localhost] => {
    "auth_test.content": "serverb.lab.example.com (172.25.250.11) has been
 customized by Ansible.\n"
}

PLAY RECAP *********************************************************************
localhost                  : ok=3    changed=0    unreachable=0    failed=0
serverb.lab.example.com    : ok=10   changed=8    unreachable=0    failed=0
```

评估

在 workstation 上运行 **lab data-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 data-review grade
```

完成

在 workstation 上，运行 **lab data-review finish** 命令来清理本练习。

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

海量视频题库 myitpub.com QQ:5565462
www.52myit.com

总结

在本章中，您学到了：

- Ansible 变量让管理员能够在整个 Ansible 项目中跨文件重复利用值。
- 可以为清单文件中的主机和主机组定义变量。
- 可以使用事实和外部文件为 playbook 定义变量。也可以在命令行中定义这些变量。
- **`register`** 关键字可用于在变量中捕获命令的输出。
- Ansible Vault 是保护使用 Ansible playbook 进行部署时所用的密码散列和私钥等敏感数据的一种方式。
- Ansible 事实是 Ansible 从受管主机自动检测到的变量。

海量视频题库 myitpub.com QQ:556546 2
www.52myit.com

章 5

实施任务控制

目标

在 Ansible Playbook 中管理任务控制、处理程序和任务错误。

培训目标

- 实施循环来编写高效的任务，从而控制运行任务的时间。
- 实施仅在另一个任务更改托管主机时运行的任务。
- 控制任务失败时发生的情况，以及导致任务失败的条件。

章节

- 编写循环和条件任务（及引导式练习）
- 实施处理程序（及引导式练习）
- 处理任务失败（练习手册）

实验

- 实施任务控制

海量视频题库 myitpub.com QQ:5565462
www.52myit.com

编写循环和条件任务

培训目标

学完完本节后，您应能够使用循环来编写高效的任务，并使用条件来控制运行任务的时间。

利用循环迭代任务

通过利用循环，管理员无需编写多个使用同一模块的任务。例如，他们不必编写五个任务来确保存在五个用户，而是只需编写一个任务来对含有五个用户的列表迭代，从而确保它们都存在。

Ansible 支持使用 **loop** 关键字对一组项目迭代任务。您可以配置循环以利用列表中的各个项目、列表中各个文件的内容、生成的数字序列或更为复杂的结构来重复任务。本节介绍迭代项目列表的简单循环。有关更高级的循环方案，请参阅文档。

简单循环

简单循环对一组项目迭代任务。**loop** 关键字添加到任务中，将应对其迭代任务的项目列表取为值。循环变量 **item** 保存每个迭代过程中使用的值。

请思考以下代码片段，它使用两次 `service` 模块来确保两个网络服务处于运行状态：

```
- name: Postfix is running
  service:
    name: postfix
    state: started

- name: Dovecot is running
  service:
    name: dovecot
    state: started
```

这两个任务可以重新编写为使用一个简单循环，从而只需一个任务来确保两个服务都在运行：

```
- name: Postfix and Dovecot are running
  service:
    name: "{{ item }}"
    state: started
  loop:
    - postfix
    - dovecot
```

可以通过一个变量提供 **loop** 所使用的列表。在以下示例中，变量 `mail_services` 含有需要处于运行状态的服务的列表。

```
vars:
  mail_services:
    - postfix
    - dovecot
```

海量视频题库 myitpub.com QQ:5565462
www.52myit.com

```
tasks:
  - name: Postfix and Dovecot are running
    service:
      name: "{{ item }}"
      state: started
    loop: "{{ mail_services }}"
```

循环散列或字典列表

loop 列表不需要是简单值列表。在以下示例中，列表中的每个项实际上是散列或字典。示例中的每个散列或字典具有两个键，即 **name** 和 **groups**，当前 **item** 循环变量中每个键的值可以分别通过 item.name 和 item.groups变量来检索。

```
- name: Users exist and are in the correct groups
  user:
    name: "{{ item.name }}"
    state: present
    groups: "{{ item.groups }}"
  loop:
    - name: jane
      groups: wheel
    - name: joe
      groups: root
```

上一任务的结果是用户jane存在且为组wheel的成员，并且用户joe存在且为组 root的成员。

较早样式的循环关键字

在 Ansible 2.5 之前，大多数 playbook 使用不同的循环语法。提供了多个循环关键字，前缀为 **with_** ，后跟 Ansible 查找插件（一项高级功能，本课程中未加以详细介绍）的名称。这种循环语法在现有 playbook 中很常见，但在将来的某个时候可能会被弃用。

下表列出了几个示例：

较早样式的 Ansible 循环

循环关键字	描述
with_items	行为与简单列表的 **loop** 关键字相同，例如字符串列表或散列/字典列表。但与 **loop** 不同的是，如果为 **with_items** 提供了列表的列表，它们将被扁平化为单级列表。循环变量 **item** 保存每次迭代过程中使用的列表项。
with_file	此关键字需要控制节点文件名列表。循环变量 **item** 在每次迭代过程中保存文件列表中相应文件的内容。
with_sequence	此关键字不需要列表，而是需要参数来根据数字序列生成值列表。循环变量 **item** 在每次迭代过程中保存生成的序列中的一个生成项的值。

playbook 中的 **with_items** 的示例如下所示：

海量视频题库 myitpub.com QQ:556462
www.52myit.com

```
  vars:
    data:
      - user0
      - user1
      - user2
  tasks:
    - name: "with_items"
      debug:
        msg: "{{ item }}"
      with_items: "{{ data }}"
```

> **重要**
> 从 Ansible 2.5 开始，建议使用 **loop** 关键字编写循环。
>
> 但是，您仍然应该了解旧语法，尤其是 **with_items**，因为它广泛用于现有 playbook。您可能会遇到继续使用 **with_*** 关键字进行循环的 playbook 和角色。

使用旧语法的任何任务都可以转换为结合使用 **loop** 和 Ansible 过滤器。您不需要知道如何使用 Ansible 过滤器来执行此操作。有关如何将旧循环转换为新语法，以及如何循环不是简单列表的项的示例，Ansible 用户指南第 从 with_X 迁移至循环 [https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#migrating-from-with-x-to-loop] 节中的 Ansible 文档提供了良好的参考。

您可能会遇到包含 **with_*** 关键字的旧 playbook 中的任务。

高级循环技术不在本课程的讨论范围内。本课程中的所有迭代任务都可以使用 **with_items** 或 **loop** 关键字实施。

将 Register 变量与 Loop 一起使用

register 关键字也可以捕获循环任务的输出。以下代码片段显示了循环任务中 register 变量的结构：

```
[student@workstation loopdemo]$ cat loop_register.yml
---
- name: Loop Register Test
  gather_facts: no
  hosts: localhost
  tasks:
    - name: Looping Echo Task
      shell: "echo This is my item: {{ item }}"
      loop:
        - one
        - two
      register: echo_results ❶

    - name: Show echo_results variable
      debug:
        var: echo_results ❷
```

❶ **echo_results** 变量已注册。

海量视频题库 myitpub.com QQ:556546 2
www.52myit.com

❷ **echo_results** 变量的内容显示在屏幕上。

运行上面的 playbook 会产生以下输出：

```
[student@workstation loopdemo]$ ansible-playbook loop_register.yml
PLAY [Loop Register Test] ******************************************************

TASK [Looping Echo Task] *******************************************************
...output omitted...
TASK [Show echo_results variable] ******************************************
ok: [localhost] => {
    "echo_results": {❶
        "changed": true,
        "msg": "All items completed",
        "results": [❷
            {❸
                "_ansible_ignore_errors": null,
                ...output omitted...
                "changed": true,
                "cmd": "echo This is my item: one",
                "delta": "0:00:00.011865",
                "end": "2018-11-01 16:32:56.080433",
                "failed": false,
                ...output omitted...
                "item": "one",
                "rc": 0,
                "start": "2018-11-01 16:32:56.068568",
                "stderr": "",
                "stderr_lines": [],
                "stdout": "This is my item: one",
                "stdout_lines": [
                    "This is my item: one"
                ]
            },
            {❹
                "_ansible_ignore_errors": null,
                ...output omitted...
                "changed": true,
                "cmd": "echo This is my item: two",
                "delta": "0:00:00.011142",
                "end": "2018-11-01 16:32:56.828196",
                "failed": false,
                ...output omitted...
                "item": "two",
                "rc": 0,
                "start": "2018-11-01 16:32:56.817054",
                "stderr": "",
                "stderr_lines": [],
                "stdout": "This is my item: two",
                "stdout_lines": [
                    "This is my item: two"
                ]
            }
```

海量视频题库 myitpub.com QQ:5565462
www.52myit.com

```
        ]❺
    }
}
...output omitted...
```

❶ **{** 字符表示 echo_results 变量的开头由键值对组成。
❷ **results** 键包含上一个任务的结果。**[** 字符表示列表的开头。
❸ 第一项的任务元数据的开头（由 **item** 键表示）。**echo** 命令的输出可在 **stdout** 键中找到。
❹ 第二项的任务结果元数据的开头。
❺ **]** 字符表示 **results** 列表的结尾。

在上面，**results** 键包含一个列表。在下面，修改了 playbook，使第二个任务迭代此列表：

```
[student@workstation loopdemo]$ cat new_loop_register.yml
---
- name: Loop Register Test
  gather_facts: no
  hosts: localhost
  tasks:
    - name: Looping Echo Task
      shell: "echo This is my item: {{ item }}"
      loop:
        - one
        - two
      register: echo_results

    - name: Show stdout from the previous task.
      debug:
        msg: "STDOUT from previous task: {{ item.stdout }}"
      loop: "{{ echo_results['results'] }}"
```

执行上述 playbook 后，输出为：

```
PLAY [Loop Register Test] ******************************************************

TASK [Looping Echo Task] *******************************************************
...output omitted...

TASK [Show stdout from the previous task.] *************************************
ok: [localhost] => (item={...output omitted...}) => {
    "msg": "STDOUT from previous task: This is my item: one"
}
ok: [localhost] => (item={...output omitted...}) => {
    "msg": "STDOUT from previous task: This is my item: two"
}
...output omitted...
```

有条件地运行任务

Ansible 可使用 conditionals 在符合特定条件时执行任务或 play。例如，可以利用一个条件在 Ansible 安装或配置服务前确定受管主机上的可用内存。

海量视频题库 myitpub.com QQ:5565462 www.52myit.com

管理员可利用条件来区分不同的受管主机，并根据它们所符合的条件来分配功能角色。Playbook 变量、注册的变量和 Ansible 事实都可通过条件来进行测试。可以使用比较字符串、数字数据和布尔值的运算符。

以下场景说明了在 Ansible 中使用条件的情况：

- 可以在变量中定义硬限制（如 `min_memory`）并将它与受管主机上的可用内存进行比较。
- Ansible 可以捕获并评估命令的输出，以确定某一任务在执行进一步操作前是否已经完成。例如，如果某一程序失败，则将跳过批处理。
- 可以利用 Ansible 事实来确定受管主机网络配置，并且决定要发送的模板文件（如，网络绑定或中继）。
- 可以评估 CPU 的数量，来确定如何正确调节某一 Web 服务器。
- 将注册的变量与预定义的变量进行比较，以确定服务是否已更改。例如，测试服务配置文件的 MD5 校验和以查看服务是否已更改。

条件任务语法

when 语句用于有条件地运行任务。它取要测试的条件作为值。如果条件满足，则运行任务。如何条件不满足，则跳过任务。

可以测试的一个最简单条件是某一布尔变量是 true 还是 false。以下示例中的 **when** 语句导致任务仅在 **run_my_task** 为 true 时运行：

```
---
- name: Simple Boolean Task Demo
  hosts: all
  vars:
    run_my_task: true

  tasks:
    - name: httpd package is installed
      yum:
        name: httpd
      when: run_my_task
```

以下示例稍微复杂一些，它测试 `my_service` 变量是否具有值。若有值，则将 `my_service` 的值用作要安装的软件包的名称。如果未定义 `my_service` 变量，则跳过任务且不显示错误。

```
---
- name: Test Variable is Defined Demo
  hosts: all
  vars:
    my_service: httpd

  tasks:
    - name: "{{ my_service }} package is installed"
      yum:
        name: "{{ my_service }}"
      when: my_service is defined
```

下表显示了管理员在处理条件时可使用的一些运算：

海量视频题库 myitpub.com QQ:5565462
www.52myit.com

示例条件

操作	示例
等于（值为字符串）	`ansible_machine == "x86_64"`
等于（值为数字）	`max_memory == 512`
小于	`min_memory < 128`
大于	`min_memory > 256`
小于等于	`min_memory <= 256`
大于等于	`min_memory >= 512`
不等于	`min_memory != 512`
变量存在	`min_memory is defined`
变量不存在	`min_memory is not defined`
布尔变量是 **true**。**1**、**True** 或 **yes** 的求值为 **true**。	`memory_available`
布尔变量是 **false**。**0**、**False** 或 **no** 的求值为 **false**。	`not memory_available`
第一个变量的值存在，作为第二个变量的列表中的值	`ansible_distribution in supported_distros`

上表中的最后一个条目初看起来有些令人困惑。下例演示了它的工作原理。

在示例中，`ansible_distribution` 变量是在 **Gathering Facts** 任务期间确定的事实，用于标识托管主机的操作系统分发。变量 `supported_distros` 由 playbook 作者创建，包含该 playbook 支持的操作系统分发列表。如果 `ansible_distribution` 的值在 `supported_distros` 列表中，则条件通过且任务运行。

```
---
- name: Demonstrate the "in" keyword
  hosts: all
  gather_facts: yes
  vars:
    supported_distros:
      - RedHat
      - Fedora
  tasks:
    - name: Install httpd using yum, where supported
      yum:
        name: http
        state: present
      when: ansible_distribution in supported_distros
```

海量视频题库 myitpub.com QQ:556462
www.52myit.com

重要

注意 **when** 语句的缩进。由于 **when** 语句不是模块变量，它必须通过缩进到任务的最高级别，放置在模块的外面。

任务是 YAML 散列/字典，**when** 语句只是任务中的又一个键，就如任务的名称以及它所使用的模块一样。通常的惯例是将可能存在的任何 **when** 关键字放在任务名称和模块（及模块参数）的后面。

测试多个条件

一个 **when** 语句可用于评估多个条件。为此，可以使用 **and** 或 **or** 关键字组合条件，并使用括号分组条件。

下列代码片段显示了如何表达多个条件的一些示例。

- 如果任一条件为真时满足条件语句，则应当使用 **or** 语句。例如，如果计算机上运行的是红帽企业 Linux 或 Fedora，则下述条件得到满足：

```
when: ansible_distribution == "RedHat" or ansible_distribution == "Fedora"
```

- 使用 **and** 运算时，两个条件都必须为真，才能满足整个条件语句。例如，如果远程主机是红帽企业 Linux 7.5 主机，并且安装的内核是指定版本，则将满足以下条件：

```
when: ansible_distribution_version == "7.5" and ansible_kernel ==
 "3.10.0-327.el7.x86_64"
```

  when 关键字还支持使用列表来描述条件列表。向 **when** 关键字提供列表时，将使用 **and** 运算组合所有条件。下面的示例演示了使用 **and** 运算符组合多个条件语句的另一方式：

```
when:
  - ansible_distribution_version == "7.5"
  - ansible_kernel == "3.10.0-327.el7.x86_64"
```

  这种格式提高了可读性，而可读性是良好编写的 Ansible Playbook 的关键目标。

- 通过使用括号分组条件，可以表达更复杂的条件语句。这可确保正确解释它们。

  例如，如果计算机上运行的是红帽企业 Linux 7 或 Fedora 28，则下述条件语句得到满足。此示例使用大于字符 (>)，这样长条件就可以在 playbook 中分成多行，以便于阅读。

```
when: >
    ( ansible_distribution == "RedHat" and
      ansible_distribution_major_version == "7" )
    or
    ( ansible_distribution == "Fedora" and
    ansible_distribution_major_version == "28" )
```

组合循环和有条件任务

循环和条件可以组合使用。

海量视频题库 myitpub.com QQ:5565462
www.52myit.com

在下例中，**yum** 模块将安装 mariadb-server 软件包，只要 / 上挂载的文件系统具有超过 300 MB 的可用空间。**ansible_mounts** 事实是一组字典，各自代表一个已挂载文件系统的相关事实。循环迭代列表中的每一字典，只有找到了代表两个条件都为真的已挂载文件系统的字典时，条件语句才得到满足。

```
- name: install mariadb-server if enough space on root
  yum:
    name: mariadb-server
    state: latest
  loop: "{{ ansible_mounts }}"
  when: item.mount == "/" and item.size_available > 300000000
```

重要

对某个任务结合使用 **when** 和 **loop** 时，将对每个项检查 **when** 语句。

下面是组合使用条件和注册变量的另一个示例。下方标注的 playbook 只有在 postfix 服务处于运行状态时才会重新启动 httpd 服务。

```
---
- name: Restart HTTPD if Postfix is Running
  hosts: all
  tasks:
    - name: Get Postfix server status
      command: /usr/bin/systemctl is-active postfix ❶
      ignore_errors: yes❷
      register: result❸

    - name: Restart Apache HTTPD based on Postfix status
      service:
        name: httpd
        state: restarted
      when: result.rc == 0❹
```

❶ Postfix 是否在运行?

❷ 如果它不在运行并且命令失败，则不停止处理。

❸ 将模块的结果信息保存在名为 result 的变量中。

❹ 评估 Postfix 任务的输出。如果 **systemctl** 命令的退出代码为 0，则 Postfix 激活并且此任务重新启动 httpd 服务。

海量视频题库 myitpub.com QQ:5565462
www.52myit.com

参考文献

循环 — Ansible 文档
https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html

测试 — Ansible 文档
https://docs.ansible.com/ansible/latest/user_guide/playbooks_tests.html

条件 — Ansible 文档
https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html

什么是有效的变量名 — 变量 — Ansible 文档
https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#what-makes-a-valid-variable-name

海量视频题库 myitpub.com QQ:556546 2
www.52myit.com

指导练习

编写循环和条件任务

在本练习中，您将编写一个 playbook，其中包含具有条件和循环的任务。

成果

您应能够：

- 使用 **when** 关键字实施 Ansible 条件。
- 结合使用 **loop** 关键字和条件实施任务迭代。

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 control-flow start** 命令。该脚本将创建工作目录 **/home/student/control-flow**。

```
[student@workstation ~]$ lab control-flow start
```

1. 在 workstation.lab.example.com 上，更改到 **/home/student/control-flow** 项目目录。

```
[student@workstation ~]$ cd ~/control-flow
[student@workstation control-flow]$
```

2. 实验脚本将创建 Ansible 配置文件和清单文件。此库存文件包含 **database_dev** 主机组中的服务器 servera.lab.example.com 和 **database_prod** 主机组中的服务器 serverb.lab.example.com。检查文件后再继续。

```
[student@workstation control-flow]$ cat inventory
[database_dev]
servera.lab.example.com

[database_prod]
serverb.lab.example.com
```

3. 创建 **playbook.yml** playbook，它将含有一个包含两个任务的 play。使用 database_dev 主机组。第一个任务安装 MariaDB 必需的软件包，第二个任务确保 MariaDB 服务正在运行。

   3.1. 在文本编辑器中打开 playbook。为 mariadb_packages 变量定义两个值：**mariadb-server** 和 **python3-PyMySQL**。该 playbook 使用此变量来安装必要的软件包。该文件应当如下所示：

海量视频题库 mypub.com QQ:556462
www.52mvpt.com

```
---
- name: MariaDB server is running
  hosts: database_dev
  vars:
    mariadb_packages:
      - mariadb-server
      - python3-PyMySQL
```

3.2. 定义一个使用 **yum** 模块和变量 `mariadb_packages` 的任务。该任务将安装所需的软件包。该任务应当如下所示：

```
  tasks:
    - name: MariaDB packages are installed
      yum:
        name: "{{ item }}"
        state: present
      loop: "{{ mariadb_packages }}"
```

3.3. 定义第二个任务来启动 `MariaDB` 服务。该任务应当如下所示：

```
    - name: Start MariaDB service
      service:
        name: mariadb
        state: started
        enabled: true
```

▶ **4.** 运行 playbook，并观察 play 的输出。

```
[student@workstation control-flow]$ ansible-playbook playbook.yml

PLAY [MariaDB server is running] ***********************************************

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

TASK [MariaDB packages are installed] ******************************************
changed: [servera.lab.example.com] => (item=mariadb-server)
changed: [servera.lab.example.com] => (item=python3-PyMySQL)

TASK [Start MariaDB service] ***************************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com     : ok=3    changed=2    unreachable=0    failed=0
```

▶ **5.** 更新第一个任务，使其仅在受管主机将红帽企业 Linux 用作其操作系统时才执行。更新 play 以使用 `database_prod` 主机组。该任务应当如下所示：

海量视频题库 myitpub.com QQ:5565462
www.52myit.com

```
- name: MariaDB server is running
  hosts: database_prod
  vars:
...output omitted...
  tasks:
    - name: MariaDB packages are installed
      yum:
        name: "{{ item }}"
        state: present
      loop: "{{ mariadb_packages }}"
      when: ansible_distribution == "RedHat"
```

6. 验证 database_prod 主机组中的受管主机是否将红帽企业 Linux 用作其操作系统。

```
[student@workstation control-flow]$ ansible database_prod -m command \
> -a 'cat /etc/redhat-release' -u devops --become
serverb.lab.example.com | CHANGED | rc=0 >>
Red Hat Enterprise Linux release 8.0 (Ootpa)
```

7. 再次运行 playbook，并观察 play 的输出。

```
[student@workstation control-flow]$ ansible-playbook playbook.yml

PLAY [MariaDB server is running] ***********************************************

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

TASK [MariaDB packages are installed] ******************************************
changed: [serverb.lab.example.com] => (item=mariadb-server)
changed: [serverb.lab.example.com] => (item=python3-PyMySQL)

TASK [Start MariaDB service] ***************************************************
changed: [serverb.lab.example.com]

PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=3    changed=2    unreachable=0    failed=0
```

Ansible 会执行该任务，因为 serverb.lab.example.com 使用了红帽企业 Linux。

完成

在 workstation 上，运行 **lab control-flow finish** 脚本来清理本练习中创建的资源。

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

本引导式练习到此结束。

海量视频题库 myitpub.com QQ:556462
www.52myit.com

实施处理程序

培训目标

学完本节后，您应能够实施仅在另一个任务更改托管主机时运行的任务。

ANSIBLE 处理程序

Ansible 模块设计为具有幂等性。这表示，在正确编写的 playbook 中，playbook 及其任务可以运行多次而不会改变受管主机，除非需要进行更改使受管主机进入所需的状态。

但有时候，在任务确实更改系统时，可能需要运行进一步的任务。例如，更改服务配置文件时可能要求重新加载该服务以便使更改的配置生效。

处理程序是响应由其他任务触发的通知的任务。仅当任务在受管主机上更改了某些内容时，任务才通知其处理程序。每个处理程序具有全局唯一的名称，在 playbook 中任务块的末尾触发。如果没有任务通过名称通知处理程序，处理程序就不会运行。如果一个或多个任务通知处理程序，处理程序就会在 play 中的所有其他任务完成后运行一次。因为处理程序就是任务，所以管理员可以在处理程序中使用他们将用于任何其他任务的模块。通常而言，处理程序被用于重新引导主机和重新启动服务。

处理程序可视为非活动任务，只有在使用 **notify** 语句显式调用时才会被触发。在下列代码片段中，只有配置文件更新并且通知了该任务时，**restart apache** 处理程序才会重新启动 Apache 服务器：

```
tasks:
  - name: copy demo.example.conf configuration template ❶
    template:
      src: /var/lib/templates/demo.example.conf.template
      dest: /etc/httpd/conf.d/demo.example.conf
    notify: ❷
      - restart apache ❸

handlers: ❹
  - name: restart apache ❺
    service: ❻
      name: httpd
      state: restarted
```

❶ 通知处理程序的任务。
❷ **notify** 语句指出该任务需要触发一个处理程序。
❸ 要运行的处理程序的名称。
❹ **handlers** 关键字表示处理程序任务列表的开头。
❺ 被任务调用的处理程序的名称。
❻ 用于该处理程序的模块。

在上例中，**restart apache** 处理程序只有在 **template** 任务通知已发生更改时才会触发。一个任务可以在其 **notify** 部分中调用多个处理程序。Ansible 将 **notify** 语句视为数组，并且迭代处理程序名称：

海量视频题库 mvitpub.com QQ:556462
www.52mvit.com