

- 1.5. 添加下面这行以启用特权升级。务必使用两个空格缩进这一行（与上方的关键字对齐），以注明这属于第一个 play。

```
become: yes
```

- 1.6. 添加下面这行以定义 **tasks** 列表的开头。请使用两个空格缩进这一行（与上方的关键字对齐），以注明这属于第一个 play。

```
tasks:
```

- 2. 作为第一个 play 中的第一项任务，定义一个任务以确保 httpd 和 firewalld 软件包为最新版本。

务必使用四个空格缩进任务的第一行。在第一个 play 的 **tasks** 关键字下，添加以下几行。

```
- name: latest version of httpd and firewalld installed
  yum:
    name:
      - httpd
      - firewalld
    state: latest
```

第一行为任务提供一个描述性名称。第二行使用六个空格缩进，并调用 **yum** 模块。下一行使用八个空格缩进，是 **name** 关键字。它指定 **yum** 模块应确保哪些软件包为最新版本。**yum** 模块的 **name** 关键字（与任务的名称不同）可以取软件包列表为值，此列表在后续两行中缩进十个空格。列表后是缩进 8 个空格的 **state** 关键字，它指定 **yum** 模块应确保安装最新版本的软件包。

- 3. 添加一项任务到第一个 play 的列表中，以确保 **/var/www/html/index.html** 中含有正确的内容。

添加以下几行，以定义 **/var/www/html/index.html** 的内容。务必使用四个空格缩进第一行。

```
- name: test html page is installed
  copy:
    content: "Welcome to the example.com intranet!\n"
    dest: /var/www/html/index.html
```

第一个条目为任务提供一个描述性名称。第二个条目使用六个空格缩进，并调用 **copy** 模块。剩余的条目使用八个空格缩进，并传递必要的参数，以确保网页中含有正确的內容。

- 4. 在 play 中再定义两项任务，以确保 **firewalld** 服务正在运行并将在引导时启动，并且允许连接 **httpd** 服务。

- 4.1. 添加以下几行，以确保 **firewalld** 服务已经启用并在运行。务必使用四个空格缩进第一行。

```
- name: firewalld enabled and running
  service:
    name: firewalld
    enabled: true
    state: started
```

第一个条目为任务提供一个描述性名称。第二个条目使用八个空格缩进，并调用 **service** 模块。剩余的条目使用十个空格缩进，并传递必要的参数，以确保 firewalld 服务已经启用并已启动。

- 4.2. 添加以下几行，以确保 **firewalld** 允许通过远程系统连接 HTTP。务必使用四个空格缩进第一行。

```
- name: firewalld permits access to httpd service
  firewalld:
    service: http
    permanent: true
    state: enabled
    immediate: yes
```

第一个条目为任务提供一个描述性名称。第二个条目使用六个空格缩进，并调用 **firewalld** 模块。剩余的条目使用八个空格缩进，并传递必要的参数，以确保永久允许远程 HTTP 连接。

- 5. 添加最后一项任务到第一个 play 中，以确保 **httpd** 服务正在运行并将在引导时启动。

添加以下几行，以确保 **httpd** 服务已经启用并在运行。务必使用四个空格缩进第一行。

```
- name: httpd enabled and running
  service:
    name: httpd
    enabled: true
    state: started
```

第一个条目为任务提供一个描述性名称。第二个条目使用六个空格缩进，并调用 **service** 模块。剩余的条目使用八个空格缩进，并传递必要的参数，以确保 **httpd** 服务已经启用并在运行。

- 6. 在 **/home/student/playbook-multi/intranet.yml** 中，定义目标为 **localhost** 的第二个 play，它将测试 Intranet Web 服务器。它不需要特权升级。

6.1. 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 intranet web server
```

6.2. 添加下面这行，以注明该 play 适用于 **localhost** 受管主机。务必使用两个空格缩进该行，指明它包含在第二个 play 中。

```
hosts: localhost
```

6.3. 添加下面这行以禁用特权升级。务必使缩进与上方的 **hosts** 关键字对齐。

```
become: no
```

- 6.4. 将下面这行添加到 `/home/student/playbook-multi/intranet.yml` 文件，以定义 **tasks** 列表的开头。务必使用两个空格缩进该行，指明它包含在第二个 play 中。

```
tasks:
```

- 7. 在第二个 play 中添加一个任务，然后使用 `uri` 模块从 `http://servera.lab.example.com` 请求内容。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。

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 intranet web server
  uri:
    url: http://servera.lab.example.com
    return_content: yes
    status_code: 200
```

第一行为任务提供一个描述性名称。第二行使用六个空格缩进，并调用 `uri` 模块。剩余的行使用八个空格缩进，并传递必要的参数，以从控制节点向受管主机执行一个 Web 内容查询并验证收到的状态代码。`return_content` 关键字可确保将服务器的响应添加到任务结果中。

- 8. 验证最终的 `/home/student/playbook-multi/intranet.yml` playbook 是否反映下列结构化内容，然后保存并关闭文件。

```
---
- name: Enable intranet services
  hosts: servera.lab.example.com
  become: yes
  tasks:
    - name: latest version of httpd and firewalld installed
      yum:
        name:
          - httpd
          - firewalld
        state: latest

    - name: test html page is installed
      copy:
        content: "Welcome to the example.com intranet!\n"
        dest: /var/www/html/index.html

    - name: firewalld enabled and running
      service:
        name: firewalld
        enabled: true
        state: started

    - name: firewalld permits access to httpd service
      firewalld:
```

```

        service: http
        permanent: true
        state: enabled
        immediate: yes

        - name: httpd enabled and running
          service:
            name: httpd
            enabled: true
            state: started

        - name: Test intranet web server
          hosts: localhost
          become: no
          tasks:
            - name: connect to intranet web server
              uri:
                url: http://servera.lab.example.com
                return_content: yes
                status_code: 200

```

- 9. 运行 `ansible-playbook --syntax-check` 命令，验证 `/home/student/playbook-multi/intranet.yml` playbook 的语法。

```
[student@workstation playbook-multi]$ ansible-playbook --syntax-check intranet.yml
playbook: intranet.yml
```

- 10. 使用 `-v` 选项执行该 playbook，输出每个任务的详细结果。通读生成的输出，确保所有任务都已成功完成。验证对 `http://servera.lab.example.com` 的 HTTP GET 请求提供正确的内容。

```
[student@workstation playbook-multi]$ ansible-playbook -v intranet.yml
...output omitted...

PLAY [Enable intranet services] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [latest version of httpd and firewalld installed] ****
changed: [servera.lab.example.com] => {"changed": true, ...output omitted...

TASK [test html page is installed] ****
changed: [servera.lab.example.com] => {"changed": true, ...output omitted...

TASK [firewalld enabled and running] ****
ok: [servera.lab.example.com] => {"changed": false, ...output omitted...

TASK [firewalld permits http service] ****
changed: [servera.lab.example.com] => {"changed": true, ...output omitted...

TASK [httpd enabled and running] ****
```

```
changed: [servera.lab.example.com] => {"changed": true, ...output omitted...  
  
PLAY [Test intranet web server] *****  
  
TASK [Gathering Facts] *****  
ok: [localhost]  
  
TASK [connect to intranet web server] *****  
ok: [localhost] => {"accept_ranges": "bytes", "changed": false, "connection": "close", "content"❶: "Welcome to the example.com intranet!\n", "content_length": "37", "content_type": "text/html; charset=UTF-8", "cookies": {}, "cookies_string": "", "date": "...output omitted...", "etag": "\"25-5790ddbcc5a48\"", "last_modified": "...output omitted...", "msg": "OK (37 bytes)", "redirected": false, "server": "Apache/2.4.6 (Red Hat Enterprise Linux)", "status"❷: 200, "url": "http://servera.lab.example.com"}  
  
PLAY RECAP *****  
localhost : ok=2     changed=0      unreachable=0    failed=0  
servera.lab.example.com : ok=6     changed=4      unreachable=0    failed=0
```

- ❶ 服务器回复了所需的内容 **Welcome to the example.com intranet!\n**。
❷ 服务器回复了 HTTP 状态代码 **200**。

完成

在 workstation 上，运行 **lab playbook-multi finish** 命令来清理本练习中创建的资源。

```
[student@workstation ~]$ lab playbook-multi finish
```

本引导式练习到此结束。

► 开放研究实验

实施 PLAYBOOK

任务执行清单

在本实验中，您将使用 playbook 配置和执行对受管主机的管理任务。

成果

您应当能够通过构建并执行 playbook，在受管主机上安装、配置和验证 Web 服务与数据库服务的状态。

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 playbook-review start** 命令。此功能将确保受管主机 serverb.lab.example.com 可在网络上访问。它也将确保控制节点上安装了正确的 Ansible 配置文件和清单文件。

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

workstation 上已经为 Ansible 项目创建了工作目录 **/home/student/playbook-review**。该目录已填充了 **ansible.cfg** 配置文件和 **inventory** 文件。此清单文件中已定义了受管主机 **serverb.lab.example.com**。



注意

本实验中使用的 playbook 与您在本章上一引导式练习中编写的 playbook 非常相似。如果不从头开始创建本实验的 playbook，您可以将该练习中的 playbook 用作本实验的起点。

如果您这样做，请小心操作以将正确的主机设为目标并更改任务，从而与本练习的说明相符合。

1. 新建一个 playbook **/home/student/playbook-review/internet.yml**，再添加所需的条目，以开始名为 **Enable internet services** 的第一个 play，并指定它适用的受管主机 **serverb.lab.example.com**。添加一个条目来启用特权升级，再添加一个条目来启动任务列表。
2. 将必要的条目添加到 **/home/student/playbook-review/internet.yml** 文件，以定义一个任务来安装 firewalld、httpd、mariadb-server、php 和 php-mysqlnd 软件包的最新版本。
3. 将所需的条目添加到 **/home/student/playbook-review/internet.yml** 文件中，以定义防火墙配置任务。它们应确保 firewalld 服务处于 **enabled** 和 **running** 状态，并且允许访问 httpd 服务。
4. 添加必要的任务，以确保 httpd 和 mariadb 服务处于 **enabled** 和 **running** 状态。

5. 添加所需的条目，以定义用于生成测试用 Web 内容的最后一个任务。使用 `get_url` 模块将 `http://materials.example.com/labs/playbook-review/index.php` 复制到受管主机上的 `/var/www/html/`。
6. 在 `/home/student/playbook-review/internet.yml` 中，为要在控制节点上执行的任务定义另一个 play。此 play 将测试对 Web 服务器的访问，该服务器应在 `serverb` 受管主机上运行。此 play 不要求特权升级，并将在 `localhost` 受管服务器上运行。
7. 添加一个任务，以利用 `uri` 模块从控制节点测试 `serverb` 上运行的 Web 服务。检查返回状态代码 `200`。
8. 验证 `internet.yml` playbook 的语法。
9. 使用 `ansible-playbook` 命令运行 playbook。通读生成的输出，确保所有任务都已成功完成。

评估

从 `workstation` 计算机运行 `lab playbook-review grade` 命令，以评测您的工作。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 playbook-review grade
```

完成

在 `workstation` 上，运行 `lab playbook-review finish` 脚本来清理本实验中创建的资源。

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

本实验到此结束。

► 解决方案

实施 PLAYBOOK

任务执行清单

在本实验中，您将使用 playbook 配置和执行对受管主机的管理任务。

成果

您应当能够通过构建并执行 playbook，在受管主机上安装、配置和验证 Web 服务与数据库服务的状态。

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 playbook-review start** 命令。此功能将确保受管主机 serverb.lab.example.com 可在网络上访问。它也将确保控制节点上安装了正确的 Ansible 配置文件和清单文件。

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

workstation 上已经为 Ansible 项目创建了工作目录 **/home/student/playbook-review**。该目录已填充了 **ansible.cfg** 配置文件和 **inventory** 文件。此清单文件中已定义了受管主机 **serverb.lab.example.com**。



注意

本实验中使用的 playbook 与您在本章上一引导式练习中编写的 playbook 非常相似。如果不从头开始创建本实验的 playbook，您可以将该练习中的 playbook 用作本实验的起点。

如果您这样做，请小心操作以将正确的主机设为目标并更改任务，从而与本练习的说明相符合。

1. 新建一个 playbook **/home/student/playbook-review/internet.yml**，再添加所需的条目，以开始名为 **Enable internet services** 的第一个 play，并指定它适用的受管主机 **serverb.lab.example.com**。添加一个条目来启用特权升级，再添加一个条目来启动任务列表。
 - 1.1. 将下列条目添加到 **/home/student/playbook-review/internet.yml** 的开头，以开始 YAML 格式。


```
---
```

 1.2. 添加下列条目，以使用名称 **Enable internet services** 注明 play 的开头。


```
- name: Enable internet services
```

1.3. 添加下列条目，注明该 play 适用于 **serverb** 受管主机。

```
hosts: serverb.lab.example.com
```

1.4. 添加下列条目以启用特权升级。

```
become: yes
```

1.5. 添加下列条目以定义 **tasks** 列表的开头。

```
tasks:
```

2. 将必要的条目添加到 **/home/student/playbook-review/internet.yml** 文件，以定义一个任务来安装 firewalld、httpd、mariadb-server、php 和 php-mysqlnd 软件包的最新版本。

```
- name: latest version of all required packages installed
  yum:
    name:
      - firewalld
      - httpd
      - mariadb-server
      - php
      - php-mysqlnd
    state: latest
```

3. 将所需的条目添加到 **/home/student/playbook-review/internet.yml** 文件中，以定义防火墙配置任务。它们应确保 **firewalld** 服务处于 **enabled** 和 **running** 状态，并且允许访问 **httpd** 服务。

```
- name: firewalld enabled and running
  service:
    name: firewalld
    enabled: true
    state: started

- name: firewalld permits http service
  firewalld:
    service: http
    permanent: true
    state: enabled
    immediate: yes
```

4. 添加必要的任务，以确保 **httpd** 和 **mariadb** 服务处于 **enabled** 和 **running** 状态。

```
- name: httpd enabled and running
  service:
    name: httpd
    enabled: true
    state: started

- name: mariadb enabled and running
```

```
service:  
  name: mariadb  
  enabled: true  
  state: started
```

5. 添加所需的条目，以定义用于生成测试用 Web 内容的最后一个任务。使用 `get_url` 模块将 `http://materials.example.com/labs/playbook-review/index.php` 复制到受管主机上的 `/var/www/html/`。

```
- name: test php page is installed  
  get_url:  
    url: "http://materials.example.com/labs/playbook-review/index.php"  
    dest: /var/www/html/index.php  
    mode: 0644
```

6. 在 `/home/student/playbook-review/internet.yml` 中，为要在控制节点上执行的任务定义另一个 play。此 play 将测试对 Web 服务器的访问，该服务器应在 `serverb` 受管主机上运行。此 play 不要求特权升级，并将在 `localhost` 受管服务器上运行。

- 6.1. 添加下列条目，使用名称 `Test internet web server` 注明第二个 play 的开头。

```
- name: Test internet web server
```

- 6.2. 添加下列条目，注明该 play 适用于 `localhost` 受管主机。

```
hosts: localhost
```

- 6.3. 将下面这行添加到 `hosts` 关键字的后面，为第二个 play 禁用特权升级。

```
become: no
```

- 6.4. 在 `/home/student/playbook-review/internet.yml` 文件中添加一个条目，以定义 `tasks` 列表的开头。

```
tasks:
```

7. 添加一个任务，以利用 `uri` 模块从控制节点测试 `serverb` 上运行的 Web 服务。检查返回状态代码 `200`。

```
- name: connect to internet web server  
  uri:  
    url: http://serverb.lab.example.com  
    status_code: 200
```

8. 验证 `internet.yml` playbook 的语法。

```
[student@workstation playbook-review]$ ansible-playbook --syntax-check \  
> internet.yml  
  
playbook: internet.yml
```

9. 使用 **ansible-playbook** 命令运行 playbook。通读生成的输出，确保所有任务都已成功完成。

```
[student@workstation playbook-review]$ ansible-playbook internet.yml
PLAY [Enable internet services] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]

TASK [latest version of all required packages installed] ****
changed: [serverb.lab.example.com]

TASK [firewalld enabled and running] ****
ok: [serverb.lab.example.com]

TASK [firewalld permits http service] ****
changed: [serverb.lab.example.com]

TASK [httpd enabled and running] ****
changed: [serverb.lab.example.com]

TASK [mariadb enabled and running] ****
changed: [serverb.lab.example.com]

TASK [test php page installed] ****
changed: [serverb.lab.example.com]

PLAY [Test internet web server] ****
TASK [Gathering Facts] ****
ok: [localhost]

TASK [connect to internet web server] ****
ok: [localhost]

PLAY RECAP ****
localhost : ok=2    changed=0    unreachable=0    failed=0
serverb.lab.example.com : ok=7    changed=5    unreachable=0    failed=0
```

评估

从 workstation 计算机运行 **lab playbook-review grade** 命令，以评测您的工作。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 playbook-review grade
```

完成

在 workstation 上，运行 **lab playbook-review finish** 脚本来清理本实验中创建的资源。

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

总结

在本章中，您学到了：

- play 是一个有序的任务列表，这些任务针对清单中选定的主机运行。
- playbook 是一个文本文件，其中包含由一个或多个按顺序运行的 play 组成的列表。
- Ansible Playbook 以 YAML 格式编写。
- YAML 文件的结构中使用空格缩进来代表数据层次结构。
- 任务的实施使用标准化代码，这些代码打包为 Ansible 模块。
- **ansible-doc** 命令可以列出已安装的模块，同时提供关于如何在 playbook 中使用模块的文档和示例代码片段。
- **ansible-playbook** 命令用于验证 playbook 语法并运行 playbook。

章 4

管理变量和事实

目标

编写使用变量的 playbook 来简化 playbook 的管理，以引用有关受管主机的信息。

培训目标

- 创建和引用影响特定主机或主机组、play 或全局环境的变量，并描述变量优先级的工作方式。
- 使用 Ansible Vault 加密敏感变量，并运行引用 Vault 加密变量文件的 playbook。
- 使用 Ansible 事实引用有关受管主机的数据，并在受管主机上配置自定义事实。

章节

- 管理变量（及引导式练习）
- 管理机密（及引导式练习）
- 管理事实（及引导式练习）

实验

- 管理变量和事实

管理变量

培训目标

学完本节后，您应能够在 playbook 中创建和引用影响特定主机或主机组、play 或全局环境的变量，并描述变量优先级的工作方式。

ANSIBLE 变量简介

Ansible 支持利用变量来存储值，并在 Ansible 项目的所有文件中重复使用这些值。这可以简化项目的创建和维护，并减少错误的数量。

通过变量，您可以轻松地在 Ansible 项目中管理给定环境的动态值。例如，变量可能包含下面这些值：

- 要创建的用户
- 要安装的软件包
- 要重新启动的服务
- 要删除的文件
- 要从互联网检索的存档

命名变量

变量的名称必须以字母开头，并且只能含有字母、数字和下划线。

下表说明了无效和有效变量名称之间的差别。

无效和有效的 Ansible 变量名称的示例

| 无效的变量名称 | 有效的变量名称 |
|-----------------|-----------------------------------|
| web server | web_server |
| remote.file | remote_file |
| 1st file | file_1 file1 |
| remoteserver\$1 | remote_server_1 remote_server1 |

定义变量

可在 Ansible 项目中的多个位置定义变量。不过，这可简化为三个基本的范围级别：

- 全局范围：从命令行或 Ansible 配置设置的变量

- Play 范围：在 play 和相关结构中设置的变量
- 主机范围：由清单、事实收集或注册的任务，在主机组和个别主机上设置的变量

如果在多个级别上定义了相同名称的变量，则采用优先级别最高的变量。窄范围优先于更广泛的范围：由清单定义的变量将被 playbook 定义的变量覆盖，后者将被命令行中定义的变量覆盖。

Ansible 文档中详细探讨了变量优先级，本节末尾的“参考”部分中提供了其链接。

PLAYBOOK 中的变量

变量在 Ansible Playbook 中发挥着重要作用，因为它们可以简化 playbook 中变量数据的管理。

在 Playbook 中定义变量

编写 playbook 时，您可以定义自己的变量，然后在任务中调用这些值。例如，名为 `web_package` 的变量可以使用值 `httpd` 来定义。然后，任务可以使用 `yum` 模块调用该变量来安装 httpd 软件包。

Playbook 变量可以通过多种方式定义。一种常见方式是将变量放在 playbook 开头的 `vars` 块中：

```
- hosts: all
  vars:
    user: joe
    home: /home/joe
```

也可以在外部文件中定义 playbook 变量。此时不使用 playbook 中的 `vars` 块，可以改为使用 `vars_files` 指令，后面跟上相对于 playbook 位置的外部变量文件名称列表：

```
- hosts: all
  vars_files:
    - vars/users.yml
```

而后，可以使用 YAML 格式在这一/这些文件中定义 playbook 变量：

```
user: joe
home: /home/joe
```

在 Playbook 中使用变量

声明了变量后，管理员可以在任务中使用这些变量。若要引用变量，可将变量名称放在双花括号 (`{}{}`) 内。在任务执行时，Ansible 会将变量替换为其值。

```
vars:
  user: joe

tasks:
  # This line will read: Creates the user joe
  - name: Creates the user {{ user }}
    user:
      # This line will create the user named Joe
      name: "{{ user }}"
```

**重要**

当变量用作开始一个值的第一元素时，必须使用引号。这可以防止 Ansible 将变量引用视为 YAML 字典的开头。如果缺少引号，将显示下列消息：

```
yum:
```

```
  name: {{ service }}
```

```
  ^ here
```

We could be wrong, but this one looks like it might be an issue with missing quotes. Always quote template expression brackets when they start a value. For instance:

```
with_items:
```

```
- {{ foo }}
```

Should be written as:

```
with_items:
```

```
- "{{ foo }}"
```

主机变量和组变量

直接应用于主机的清单变量分为两大类：主机变量，应用于特定主机；以及组变量，应用于一个主机组或一组主机组中的所有主机。主机变量优先于组变量，但 playbook 中定义的变量的优先级比这两者更高。

若要定义主机变量和组变量，一种方法是直接在清单文件中定义。这是较旧的做法，不建议采用，但您仍可能会遇到。

- 定义 `demo.example.com` 的 `ansible_user` 主机变量：

```
[servers]
demo.example.com  ansible_user=joe
```

- 定义 `servers` 主机组的 `user` 组变量。

```
[servers]
demo1.example.com
demo2.example.com
```

```
[servers:vars]
user=joe
```

- 定义 `servers` 组的 `user` 组变量，该组由两个主机组组成，每个主机组有两个服务器。

```
[servers1]
demo1.example.com
demo2.example.com
```

```
[servers2]
demo3.example.com
demo4.example.com
```

```
[servers:children]
servers1
servers2

[servers:vars]
user=joe
```

此做法存在的一些缺点，它使得清单文件更难以处理，在同一文件中混合提供了主机和变量信息，而且采用的也是过时的语法。

使用目录填充主机和组变量

定义主机和机组的变量的首选做法是在与清单文件或目录相同的工作目录中，创建 **group_vars** 和 **host_vars** 两个目录。这两个目录分别包含用于定义组变量和主机变量的文件。



重要

建议的做法是使用 **host_vars** 和 **group_vars** 目录定义清单变量，而不直接在清单文件中定义它们。

为了定义用于 **servers** 组的组变量，需要创建名为 **group_vars/servers** 的 YAML 文件，然后该文件的内容将使用与 playbook 相同的语法将变量设置为值：

```
user: joe
```

类似地，为了定义用于特定主机的主机变量，需要在 **host_vars** 目录中创建名称与主机匹配的文件来存放主机变量。

下面的示例更加详细地说明了这一做法。例如在一个场景中，需要管理两个数据中心，并在 **~/project/inventory** 清单文件中定义数据中心主机：

```
[admin@station project]$ cat ~/project/inventory
[datacenter1]
demo1.example.com
demo2.example.com

[datacenter2]
demo3.example.com
demo4.example.com

[datacenters:children]
datacenter1
datacenter2
```

- 如果需要为两个数据中心的所有服务器定义一个通用值，可以为 **datacenters** 主机组设置一个组变量：

```
[admin@station project]$ cat ~/project/group_vars/datacenters
package: httpd
```

- 如果要为每个数据中心定义不同的值，可以为每个数据中心机组设置组变量。

```
[admin@station project]$ cat ~/project/group_vars/datacenter1
package: httpd
[admin@station project]$ cat ~/project/group_vars/datacenter2
package: apache
```

- 如果要为每一数据中心的各个主机定义不同的值，则在单独的主机变量文件中定义变量：

```
[admin@station project]$ cat ~/project/host_vars/demo1.example.com
package: httpd
[admin@station project]$ cat ~/project/host_vars/demo2.example.com
package: apache
[admin@station project]$ cat ~/project/host_vars/demo3.example.com
package: mariadb-server
[admin@station project]$ cat ~/project/host_vars/demo4.example.com
package: mysql-server
```

示例项目 **project** 的目录结构如果包含上面的所有示例文件，将如下所示：

```
project
├── ansible.cfg
├── group_vars
│   ├── datacenters
│   │   ├── datacenters1
│   │   └── datacenters2
├── host_vars
│   ├── demo1.example.com
│   ├── demo2.example.com
│   ├── demo3.example.com
│   └── demo4.example.com
└── inventory
└── playbook.yml
```

从命令行覆盖变量

清单变量可被 playbook 中设置的变量覆盖，这两种变量又可通过在命令行中传递参数到 **ansible** 或 **ansible-playbook** 命令来覆盖。在命令行上设置的变量称为额外变量。

当您需要覆盖一次性运行的 playbook 的变量的已定义值时，额外变量非常有用。例如：

```
[user@demo ~]$ ansible-playbook main.yml -e "package=apache"
```

使用数组作为变量

除了将与同一元素相关的配置数据（软件包列表、服务列表和用户列表等）分配到多个变量外，管理员也可以使用数组。这种做法的一个好处在于，数组是可以浏览的。

例如，假设下列代码片段：

```
user1_first_name: Bob
user1_last_name: Jones
user1_home_dir: /users/bjones
user2_first_name: Anne
user2_last_name: Cook
user2_home_dir: /users/acook
```

这可以改写成名为 **users** 的数组：

```
users:
  bjones:
    first_name: Bob
    last_name: Jones
    home_dir: /users/bjones
  acook:
    first_name: Anne
    last_name: Cook
    home_dir: /users/acook
```

然后，您可以使用以下变量来访问用户数据：

```
# Returns 'Bob'
users.bjones.first_name

# Returns '/users/acook'
users.acook.home_dir
```

由于变量被定义为 Python 字典，因此可以使用替代语法。

```
# Returns 'Bob'
users['bjones']['first_name']

# Returns '/users/acook'
users['acook']['home_dir']
```



重要

如果键名称与 Python 方法或属性的名称（如 **discard**、**copy** 和 **add** 等）相同，点表示法可能会造成问题。使用方括号表示法有助于避免冲突和错误。

两种语法都有效，但为了方便故障排除，红帽建议在任何给定 Ansible 项目的所有文件中一致地采用一种语法。

使用已注册变量捕获命令输出

管理员可以使用 **register** 语句捕获命令输出。输出保存在一个临时变量中，稍后在 playbook 中可用于调试用途或者达成其他目的，例如基于命令输出的特定配置。

以下 playbook 演示了如何为调试用途捕获命令输出：

```

---
- name: Installs a package and prints the result
  hosts: all
  tasks:
    - name: Install the package
      yum:
        name: httpd
        state: installed
        register: install_result

    - debug: var=install_result

```

运行该 playbook 时，debug 模块用于将 `install_result` 注册变量的值转储到终端。

```

[user@demo ~]$ ansible-playbook playbook.yml
PLAY [Installs a package and prints the result] ****
TASK [setup] ****
ok: [demo.example.com]

TASK [Install the package] ****
ok: [demo.example.com]

TASK [debug] ****
ok: [demo.example.com] => {
    "install_result": {
        "changed": false,
        "msg": "",
        "rc": 0,
        "results": [
            "httpd-2.4.6-40.el7.x86_64 providing httpd is already installed"
        ]
    }
}

PLAY RECAP ****
demo.example.com : ok=3     changed=0     unreachable=0    failed=0

```



参考文献

清单 — Ansible 文档

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

变量 — Ansible 文档

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

变量优先级：我该将变量放到何处？

https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable

YAML 语法 — Ansible 文档

https://docs.ansible.com/ansible/latest/reference_appendices/YAMLTntax.html

► 指导练习

管理变量

在本练习中，您将在 playbook 中定义和使用变量。

成果

您应能够：

- 在 playbook 中定义变量。
- 创建使用定义的变量的任务。

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-variables start** 命令。此功能会创建 **data-variables** 工作目录，并为它填充 Ansible 配置文件和主机清单。

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

- 1. 以 student 用户身份，在 workstation 上更改到 **/home/student/data-variables** 目录。

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

- 2. 在接下来的几步中，您要创建一个 playbook，它将安装 Apache Web 服务器并开启相关的端口以使其服务可被访问。Playbook 查询 Web 服务器以确认其已设好并在运行。

创建 **playbook.yml** playbook 并在 **vars** 部分中定义以下变量：

变量

| 变量 | 描述 |
|------------------|------------------|
| web_pkg | 要安装的 Web 服务器软件包。 |
| firewall_pkg | 要安装的防火墙软件包。 |
| web_service | 要管理的 Web 服务。 |
| firewall_service | 要管理的防火墙服务。 |
| python_pkg | uri 模块所需的软件包。 |
| rule | 要打开的服务。 |

```
---  
- name: Deploy and start Apache HTTPD service  
  hosts: webserver  
  vars:  
    web_pkg: httpd  
    firewall_pkg: firewalld  
    web_service: httpd  
    firewall_service: firewalld  
    python_pkg: python3-PyMySQL  
    rule: http
```

- 3. 创建 **tasks** 块，再创建第一项任务，该任务应使用 **yum** 模块确保安装了必要软件包的最新版本。

```
tasks:  
- name: Required packages are installed and up to date  
  yum:  
    name:  
      - "{{ web_pkg }}"  
      - "{{ firewall_pkg }}"  
      - "{{ python_pkg }}"  
    state: latest
```



注意

您可以使用 **ansible-doc yum** 检查 **yum** 模块的语法。该语法显示其 **name** 指令将取该模块应处理的软件包列表，因此您不需要通过单独的任务来确保各个软件包为最新版本。

- 4. 创建两项任务，确保 **httpd** 和 **firewalld** 服务已经启动并已启用。

```
- name: The {{ firewall_service }} service is started and enabled  
  service:  
    name: "{{ firewall_service }}"  
    enabled: true  
    state: started  
  
- name: The {{ web_service }} service is started and enabled  
  service:  
    name: "{{ web_service }}"  
    enabled: true  
    state: started
```



注意

service 模块的工作方式与 **yum** 模块不同，如 **ansible-doc service** 中所述。其 **name** 指令取要处理的确切服务的名称。

您可以使用本课程中稍后介绍的 **loop** 关键字编写一个任务，确保启动并启用这两个服务。

- 5. 添加确保 `/var/www/html/index.html` 文件中存在特定内容的任务。

```
- name: Web content is in place
copy:
  content: "Example web content"
  dest: /var/www/html/index.html
```

- 6. 添加一项任务，该任务将使用 `firewalld` 模块确保为 `rule` 变量中指定的 `firewalld` 服务打开防火墙端口。

```
- name: The firewall port for {{ rule }} is open
firewalld:
  service: "{{ rule }}"
  permanent: true
  immediate: true
  state: enabled
```

- 7. 创建一个新 play，它将查询 Web 服务来确保一切都配置妥当。它应当在 `localhost` 上运行。鉴于该 Ansible 事实，Ansible 不必更改身份，因此可将 `become` 模块设为 `false`。您可以使用 `uri` 模块来检查 URL。对于此任务，检查状态代码是否为 200，以确认 `servera.lab.example.com` 上的 Web 服务器正在运行并且配置正确。

```
- name: Verify the Apache service
hosts: localhost
become: false
tasks:
- name: Ensure the webserver is reachable
  uri:
    url: http://servera.lab.example.com
    status_code: 200
```

- 8. 完成时，该 playbook 应当如下所示。检查 playbook，并确认两个 play 是否都正确。

```
---
- name: Deploy and start Apache HTTPD service
hosts: webserver
vars:
  web_pkg: httpd
  firewall_pkg: firewalld
  web_service: httpd
  firewall_service: firewalld
  python_pkg: python3-PyMySQL
  rule: http

tasks:
- name: Required packages are installed and up to date
  yum:
    name:
      - "{{ web_pkg }}"
      - "{{ firewall_pkg }}"
      - "{{ python_pkg }}"
    state: latest
```

```

- name: The {{ firewall_service }} service is started and enabled
  service:
    name: "{{ firewall_service }}"
    enabled: true
    state: started

- name: The {{ web_service }} service is started and enabled
  service:
    name: "{{ web_service }}"
    enabled: true
    state: started

- name: Web content is in place
  copy:
    content: "Example web content"
    dest: /var/www/html/index.html

- name: The firewall port for {{ rule }} is open
  firewalld:
    service: "{{ rule }}"
    permanent: true
    immediate: true
    state: enabled

- name: Verify the Apache service
  hosts: localhost
  become: false
  tasks:
    - name: Ensure the webserver is reachable
      uri:
        url: http://servera.lab.example.com
        status_code: 200

```

- 9. 在运行 playbook 之前, 请使用 **ansible-playbook --syntax-check** 命令验证其语法。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 data-variables]$ ansible-playbook --syntax-check playbook.yml

playbook: playbook.yml
```

- 10. 使用 **ansible-playbook** 命令运行 playbook。观察输出结果, Ansible 将首先安装软件包, 再启动并启用其服务, 然后确保 Web 服务器可被访问。

```
[student@workstation data-variables]$ ansible-playbook playbook.yml

PLAY [Deploy and start Apache HTTPD service] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Required packages are installed and up to date] ****
changed: [servera.lab.example.com]
```

```
TASK [The firewalld service is started and enabled] ****
ok: [servera.lab.example.com]

TASK [The httpd service is started and enabled] ****
changed: [servera.lab.example.com]

TASK [Web content is in place] ****
changed: [servera.lab.example.com]

TASK [The firewall port for http is open] ****
changed: [servera.lab.example.com]

PLAY [Verify the Apache service] ****

TASK [Gathering Facts] ****
ok: [localhost]

TASK [Ensure the webserver is reachable] ****
ok: [localhost]

PLAY RECAP ****
localhost : ok=2    changed=0    unreachable=0    failed=0
servera.lab.example.com : ok=6    changed=4    unreachable=0    failed=0
```

完成

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

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

本引导式练习到此结束。