

# 管理机密

## 培训目标

学完本节后，您应能够使用 Ansible Vault 加密敏感变量，并运行引用 Vault 加密变量文件的 playbook。

## 介绍 ANSIBLE VAULT

Ansible 可能需要访问密码或 API 密钥等敏感数据，以便能配置受管主机。通常，此信息可能以纯文本形式存储在清单变量或其他 Ansible 文件中。但若如此，任何有权访问 Ansible 文件的用户或存储这些 Ansible 文件的版本控制系统都能够访问此敏感数据。这显然存在安全风险。

使用 Ansible 随附的 Ansible Vault 可以加密和解密任何由 Ansible 使用的结构化数据文件。若要使用 Ansible Vault，可通过一个名为 **ansible-vault** 的命令行工具创建、编辑、加密、解密和查看文件。Ansible Vault 可以加密任何由 Ansible 使用的结构化数据文件。这可能包括清单变量、playbook 中含有的变量文件、在执行 playbook 时作为参数传递的变量文件，或者 Ansible 角色中定义的变量。



### 重要

Ansible Vault 并不实施自有的加密函数，而是使用外部 Python 工具集。文件通过利用 AES256 的对称加密（将密码用作机密密钥）加以保护。请注意，这种方式尚未得到第三方正式审核。

## 创建加密的文件

要创建新的加密文件，可使用 **ansible-vault create filename** 命令。该命令将提示输入新的 vault 密码，然后利用默认编辑器 **vi** 打开文件。您可以设置和导出 **EDITOR** 环境变量，通过设置和导出指定其他默认编辑器。例如，若要将默认编辑器设为 **nano**，可设置为 **export EDITOR=nano**。

```
[student@demo ~]$ ansible-vault create secret.yml  
New Vault password: redhat  
Confirm New Vault password: redhat
```

您可以使用 vault 密码文件来存储 vault 密码，而不是通过标准输入途径输入 vault 密码。您需要使用文件权限和其他方式来严密保护该文件。

```
[student@demo ~]$ ansible-vault create --vault-password-file=vault-pass secret.yml
```

在最近版本的 Ansible 中，用于保护文件的密文是 AES256，但利用较旧版本加密的文件可能依然使用 128 位 AES。

## 查看加密的文件

您可以使用 **ansible-vault view filename** 命令查看 Ansible Vault 加密的文件，而不必打开它进行编辑。

```
[student@demo ~]$ ansible-vault view secret1.yml
Vault password: secret
less 458 (POSIX regular expressions)
Copyright (C) 1984-2012 Mark Nudelman

less comes with NO WARRANTY, to the extent permitted by law.
For information about the terms of redistribution,
see the file named README in the less distribution.
Homepage: http://www.greenwoodsoftware.com/less
my_secret: "yJJvPqhsiusmmPPZdnjndkdnYNDjdj782meUZcw"
```

## 编辑现有的加密文件

要编辑现有的加密文件，Ansible Vault 提供了 **ansible-vault edit filename** 命令。此命令将文件解密为一个临时文件，并允许您编辑该文件。保存时，它将复制其内容并删除临时文件。

```
[student@demo ~]$ ansible-vault edit secret.yml
Vault password: redhat
```



### 注意

**edit** 子命令始终重写文件，因此只应在进行更改时使用它。这在文件保管在版本控制下时有影响。要查看文件的内容而不进行更改，始终应使用 **view** 子命令。

## 加密现有的文件

要加密已存在的文件，请使用 **ansible-vault encrypt filename** 命令。此命令可取多个欲加密文件的名称作为参数。

```
[student@demo ~]$ ansible-vault encrypt secret1.yml secret2.yml
New Vault password: redhat
Confirm New Vault password: redhat
Encryption successful
```

使用 **--output=OUTPUT\_FILE** 选项，可将加密文件保存为新的名称。您只能通过 **--output** 选项使用一个输入文件。

## 解密现有的文件

现有的加密文件可以通过 **ansible-vault decrypt filename** 命令永久解密。在解密单个文件时，可使用 **--output** 选项以其他名称保存解密的文件。

```
[student@demo ~]$ ansible-vault decrypt secret1.yml --output=secret1-decrypted.yml
Vault password: redhat
Decryption successful
```

## 更改加密文件的密码

可使用 **ansible-vault rekey filename** 命令更改加密文件的密码。此命令可一次性更新多个数据文件的密钥。它将提示提供原始密码和新密码。

```
[student@demo ~]$ ansible-vault rekey secret.yml  
Vault password: redhat  
New Vault password: RedHat  
Confirm New Vault password: RedHat  
Rekey successful
```

在使用 vault 密码文件时，请使用 **--new-vault-password-file** 选项：

```
[student@demo ~]$ ansible-vault rekey \  
> --new-vault-password-file=NEW_VAULT_PASSWORD_FILE secret.yml
```

## PLAYBOOK 和 ANSIBLE VAULT

要运行可访问通过 Ansible Vault 加密的文件的 playbook，需要向 **ansible-playbook** 命令提供加密密码。如果不提供密码，playbook 将返回错误：

```
[student@demo ~]$ ansible-playbook site.yml  
ERROR: A vault password must be specified to decrypt vars/api_key.yml
```

要为 playbook 提供 vault 密码，可使用 **--vault-id** 选项。例如，要以交互方式提供 vault 密码，请使用下例中所示的 **--vault-id @prompt**：

```
[student@demo ~]$ ansible-playbook --vault-id @prompt site.yml  
Vault password (default): redhat
```



### 重要

如果您使用的是版本 2.4 之前的 Ansible 版本，则需要使用 **--ask-vault-pass** 选项以交互方式提供 vault 密码。如果 playbook 使用的所有 vault 加密文件都使用同一密码加密，则仍可使用该选项。

```
[student@demo ~]$ ansible-playbook --ask-vault-pass site.yml  
Vault password: redhat
```

此外，也可使用 **--vault-password-file** 选项指定以纯文本存储加密密码的文件。密码应当在该文件中存储为一行字符串。由于该文件包含敏感的纯文本密码，因此务必要通过文件权限和其他安全措施对其加以保护。

```
[student@demo ~]$ ansible-playbook --vault-password-file=vault-pw-file site.yml
```

您也可以使用 **ANSIBLE\_VAULT\_PASSWORD\_FILE** 环境变量，指定密码文件的默认位置。

**重要**

从 Ansible 2.4 开始，您可以通过 **ansible-playbook** 使用多个 Ansible Vault 密码。要使用多个密码，请将多个 **--vault-id** 或 **--vault-password-file** 选项传递给 **ansible-playbook** 命令。

```
[student@demo ~]$ ansible-playbook \
> --vault-id one@prompt --vault-id two@prompt site.yml
Vault password (one):
Vault password (two):
...output omitted...
```

**@prompt** 前面的 vault ID **one** 和 **two** 可以是任何字符，甚至可以完全省略它们。不过，如果在使用 **ansible-vault** 命令加密文件时使用 **--vault-id id** 选项，则在运行 **ansible-playbook** 时，将最先尝试匹配 ID 的密码。如果不匹配，接下来将尝试您提供的其他密码。没有 ID 的 vault ID **@prompt** 实际上是 **default@prompt** 的简写，这意味着提示输入 vault ID **default** 的密码。

## 变量文件管理的推荐做法

若要简化管理，务必要设置您的 Ansible 项目，使敏感变量和所有其他变量保存在相互独立的文件中。然后，包含敏感变量的文件可通过 **ansible-vault** 命令进行保护。

请记住，管理组变量和主机变量的首选方式是在 playbook 级别上创建目录。**group\_vars** 目录通常包含名称与它们所应用的主机组匹配的变量文件。**host\_vars** 目录通常包含名称与它们所应用的受管主机名称匹配的变量文件。

不过，除了使用 **group\_vars** 或 **host\_vars** 中的文件外，您也可对每一主机组或受管主机使用目录。这些目录而后可包含多个变量文件，它们都由该主机组或受管主机使用。例如，在 **playbook.yml** 的以下项目目录中，**webservers** 主机组的成员将使用 **group\_vars/webservers/vars** 文件中的变量，而 **demo.example.com** 将使用 **host\_vars/demo.example.com/vars** 和 **host\_vars/demo.example.com/vault** 中的变量：

```
.
├── ansible.cfg
├── group_vars
│   └── webservers
│       └── vars
└── host_vars
    └── demo.example.com
        ├── vars
        └── vault
└── inventory
└── playbook.yml
```

在这种情景中，其好处在于用于 **demo.example.com** 的大部分变量可以放在 **vars** 文件中，敏感变量则可单独放在 **vault** 文件中保密。然后，管理员可以使用 **ansible-vault** 加密 **vault** 文件，而将 **vars** 文件保留为纯文本。

在本例中，**host\_vars/demo.example.com** 目录内使用的文件名没有什么特别之处。该目录可以包含更多文件，一些由 Ansible Vault 加密，另一些则不加密。

Playbook 变量（与清单变量相对）也可通过 Ansible Vault 保护。敏感的 playbook 变量可以放在单独的文件中，此文件通过 Ansible Vault 加密，并通过 `vars_files` 指令包含在该 playbook 中。这很有用处，因为 playbook 变量的优先级高于清单变量。

如果您在 playbook 中使用多个 vault 密码，请确保为每个加密文件分配一个 vault ID，并在运行 playbook 时输入具有该 vault ID 的匹配密码。这可确保在解密 vault 加密文件时先选择正确的密码，这比强制 Ansible 尝试您提供的所有 vault 密码直至找到正确的密码要快。



### 参考文献

`ansible-playbook(1)` 和 `ansible-vault(1)` 帮助手册

### Vault &mdash; Ansible 文档

[https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_vault.html](https://docs.ansible.com/ansible/latest/user_guide/playbooks_vault.html)

### 变量和 Vault &mdash; Ansible 文档

[https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_best\\_practices.html#best-practices-for-variables-and-vaults](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#best-practices-for-variables-and-vaults)

## ► 指导练习

# 管理机密

在本练习中，您将使用 Ansible Vault 加密敏感变量以保护它们，然后运行使用这些变量的 playbook。

## 成果

您应能够：

- 使用加密文件中定义的变量来执行 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 data-secret start** 命令。此脚本确保 workstation 上已安装 Ansible，并创建供此练习使用的工作目录。此目录包含指向作为受管主机的 `servera.lab.example.com` 的清单文件，该受管主机是 `devservers` 组的成员。

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

- 1. 在 workstation 上，以 student 用户身份更改到 `/home/student/data-secret` 工作目录。

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

- 2. 编辑提供的加密文件 `secret.yml` 的内容。该文件可以使用 `redhat` 作为密码进行解密。取消注释 `username` 和 `pwhash` 变量条目。

- 2.1. 编辑加密文件 `/home/student/data-secret/secret.yml`。根据提示为 vault 提供密码 `redhat`。加密文件在默认编辑器 vim 中打开。

```
[student@workstation data-secret]$ ansible-vault edit secret.yml
Vault password: redhat
```

- 2.2. 取消注释两个变量条目，然后保存文件并退出编辑器。它们应如下所示：

```
username: ansibleuser1
pwhash: $6$jf...uxhP1
```

- 3. 创建一个名为 `/home/student/data-secret/create_users.yml` 的 playbook，它将使用 `/home/student/data-secret/secret.yml` 加密文件中定义的变量。

配置 playbook，以使用 `devservers` 主机组。以 `devops` 用户身份对远程受管主机运行此 playbook。配置该 playbook 以创建 `username` 变量定义的 `ansibleuser1` 用户。使用存储在 `pwhash` 变量中的密码散列设置用户的密码。

```

---
- name: create user accounts for all our servers
  hosts: devservers
  become: True
  remote_user: devops
  vars_files:
    - secret.yml
  tasks:
    - name: Creating user from secret.yml
      user:
        name: "{{ username }}"
        password: "{{ pwhash }}"

```

- 4. 使用 `ansible-playbook --syntax-check` 命令，验证 `create_users.yml` playbook 的语法。使用 `--ask-vault-pass` 选项，从而提示输入用于解密 `secret.yml` 的 vault 密码。先解决所有语法错误，再继续。

```
[student@workstation data-secret]$ ansible-playbook --syntax-check \
> --ask-vault-pass create_users.yml
Vault password (default): redhat

playbook: create_users.yml
```



#### 注意

您可以使用较新的 `--vault-id @prompt` 选项而不是使用 `--ask-vault-pass` 来执行此操作。

- 5. 创建名为 `vault-pass` 的密码文件，以用于 playbook 执行而不必要求输入密码。该文件必须包含纯文本 `redhat` 作为 vault 密码。将文件的权限更改为 `0600`。

```
[student@workstation data-secret]$ echo 'redhat' > vault-pass
[student@workstation data-secret]$ chmod 0600 vault-pass
```

- 6. 使用 `vault-pass` 执行 Ansible Playbook，以利用 `secret.yml` Ansible Vault 加密文件中存储为变量的密码在远程系统上创建 `ansibleuser1` 用户。

```
[student@workstation data-secret]$ ansible-playbook \
> --vault-password-file=vault-pass create_users.yml

PLAY [create user accounts for all our servers] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Creating users from secret.yml] ****
```

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

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

- 7. 验证该 playbook 是否正确运行。用户 `ansibleuser1` 应该存在，并且在 `servera.lab.example.com` 上有正确的密码。通过使用 `ssh` 以该用户身份登录 `servera.lab.example.com` 来对此进行测试。`ansibleuser1` 的密码是 `redhat`。为了确保 SSH 仅尝试通过密码而不是 SSH 密钥进行身份验证，请在登录时使用 `-o PreferredAuthentications=password` 选项。

成功登录后，从 `servera` 注销。

```
[student@workstation data-secret]$ ssh -o PreferredAuthentications=password \
> ansibleuser1@servera.lab.example.com
ansibleuser1@servera.lab.example.com's password: redhat
Activate the web console with: systemctl enable --now cockpit.socket

[ansibleuser1@servera ~]$ exit
logout
Connection to servera.lab.example.com closed.
```

## 完成

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

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

本引导式练习到此结束。

# 管理事实

## 培训目标

学完本节后，您应能够使用 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。

## 描述 ANSIBLE 事实

Ansible 事实是 Ansible 在受管主机上自动检测到的变量。事实中含有与主机相关的信息，可以像 play 中的常规变量、条件、循环或依赖于从受管主机收集的值的任何其他语句那样使用。

为受管主机收集的一些事实可能包括

- 主机名称
- 内核版本
- 网络接口
- IP 地址
- 操作系统版本
- 各种环境变量
- CPU 数量
- 提供的或可用的内存
- 可用磁盘空间

借助事实，可以方便地检索受管主机的状态，并根据该状态确定要执行的操作。例如：

- 可以根据含有受管主机当前内核版本的事实运行条件任务，以此来重新启动服务器。
- 可以根据通过事实报告的可用内存来自定义 MySQL 配置文件。
- 可以根据事实的值设置配置文件中使用的 IPv4 地址。

通常，每个 play 在执行第一个任务之前会先自动运行 `setup` 模块来收集事实。这在 Ansible 2.3 中报告为 `Gathering Facts` 任务，或者在更早版本的 Ansible 中报告为 `setup`。默认情况下，Ansible 2.3 之前，`gather_facts` 指令将自动运行 `setup` 任务。如果启用了 `gather_facts`，则不会运行 `setup` 任务。 不需要具有在 play 中运行 `setup` 的任务。它通常会自动为您运行。

查看为受管主机收集的事实的一种方式是，运行一个收集事实并使用 `debug` 模块显示 `ansible_facts` 变量值的简短 playbook。

```
- name: Fact dump
hosts: all
tasks:
  - name: Print all facts
    debug:
      var: ansible_facts
```

运行该 playbook 时，事实将显示在作业输出中：

```
[user@demo ~]$ ansible-playbook facts.yml

PLAY [Fact dump] ****
TASK [Gathering Facts] ****
ok: [demo1.example.com]

TASK [Print all facts] ****
ok: [demo1.example.com] => {
    "ansible_facts": {
        "all_ipv4_addresses": [
            "172.25.250.10"
        ],
        "all_ipv6_addresses": [
            "fe80::5054:ff:fe00:fa0a"
        ],
        "ansible_local": {},
        "apparmor": {
            "status": "disabled"
        },
        "architecture": "x86_64",
        "bios_date": "01/01/2011",
        "bios_version": "0.5.1",
        "cmdline": {
            "BOOT_IMAGE": "/boot/vmlinuz-3.10.0-327.el7.x86_64",
            "LANG": "en_US.UTF-8",
            "console": "ttyS0,115200n8",
            "crashkernel": "auto",
            "net.ifnames": "0",
            "no_timer_check": true,
            "ro": true,
            "root": "UUID=2460ab6e-e869-4011-acae-31b2e8c05a3b"
        },
        ...output omitted...
    }
}
```

Playbook 以 JSON 格式显示 `ansible_facts` 变量的内容，作为变量的散列/字典。您可以浏览输出来查看收集了哪些事实，并查找您可能要在 play 中使用的事实。

下表显示了可能从受管节点收集的并可在 playbook 中使用的一些事实：

### Ansible 事实的示例

事实	变量
短主机名	<code>ansible_facts['hostname']</code>
完全限定的域名	<code>ansible_facts['fqdn']</code>
主要 IPv4 地址（基于路由）	<code>ansible_facts['default_ipv4']['address']</code>
所有网络接口的名称列表	<code>ansible_facts['interfaces']</code>

事实	变量
/dev/vda1 磁盘分区的大小	ansible_facts['devices']['vda']['partitions']['vda1']['size']
DNS 服务器列表	ansible_facts['dns']['nameservers']
当前运行的内核的版本	ansible_facts['kernel']



### 注意

请记住，如果变量的值为散列/字典，则可使用两种语法来检索该值。从上表中举两个例子：

- ansible\_facts['default\_ipv4']['address'] 也可以写成  
ansible\_facts.default\_ipv4.address
- ansible\_facts['dns']['nameservers'] 也可以写成  
ansible\_facts.dns.nameservers

在 playbook 中使用事实时，Ansible 将事实的变量名动态替换为对应的值：

```
---
- hosts: all
  tasks:
    - name: Prints various Ansible facts
      debug:
        msg: >
          The default IPv4 address of {{ ansible_fqdn }}
          is {{ ansible_facts.default_ipv4.address }}
```

下列输出演示了 Ansible 如何查询受管节点，并且动态使用系统信息来更新变量。也可使用事实来创建符合特定标准的动态主机组。

```
[user@demo ~]$ ansible-playbook playbook.yml
PLAY ****
TASK [Gathering Facts] ****
ok: [demo1.example.com]

TASK [Prints various Ansible facts] ****
ok: [demo1.example.com] => {
    "msg": "The default IPv4 address of demo1.example.com is
           172.25.250.10"
}

PLAY RECAP ****
demo1.example.com : ok=2      changed=0      unreachable=0      failed=0
```

## ANSIBLE 事实作为变量注入

在 Ansible 2.5 之前，事实是作为前缀为字符串 `ansible_` 的单个变量注入，而不是作为 `ansible_facts` 变量的一部分注入。例如，`ansible_facts['distribution']` 事实会被称为 `ansible_distribution`。

许多较旧的 playbook 仍然使用作为变量注入的事实，而不是在 `ansible_facts` 变量下创建命名空间的新语法。您可以使用临时命令来运行 `setup` 模块，以此形式显示所有事实的值。以下示例中使用一个临时命令在受管主机 `demo1.example.com` 上运行 `setup` 模块：

```
[user@demo ~]$ ansible demo1.example.com -m setup
demo1.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.25.250.10"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::5054:ff:fe00:fa0a"
        ],
        "ansible_apparmor": {
            "status": "disabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "01/01/2011",
        "ansible_bios_version": "0.5.1",
        "ansible_cmdline": {
            "BOOT_IMAGE": "/boot/vmlinuz-3.10.0-327.el7.x86_64",
            "LANG": "en_US.UTF-8",
            "console": "ttyS0,115200n8",
            "crashkernel": "auto",
            "net.ifnames": "0",
            "no_timer_check": true,
            "ro": true,
            "root": "UUID=2460ab6e-e869-4011-acae-31b2e8c05a3b"
        }
    ...
    ...output omitted...
}
```

下表比较了旧的和新的事实名称。

### 选定的 Ansible 事实名称的比较

ANSIBLE_FACTS 形式	旧事实变量形式
<code>ansible_facts['hostname']</code>	<code>ansible_hostname</code>
<code>ansible_facts['fqdn']</code>	<code>ansible_fqdn</code>
<code>ansible_facts['default_ipv4']['address']</code>	<code>ansible_default_ipv4['address']</code>
<code>ansible_facts['interfaces']</code>	<code>ansible_interfaces</code>
<code>ansible_facts['devices']['vda']['partitions']['vda1']['size']</code>	<code>ansible_devices['vda']['partitions']['vda1']['size']</code>

ANSIBLE_FACTS 形式	旧事实变量形式
ansible_facts['dns'] ['nameservers']	ansible_dns['nameservers']
ansible_facts['kernel']	ansible_kernel

**重要**

目前，Ansible 同时识别新的事实命名系统（使用 `ansible_facts`）和旧的 2.5 前“作为单独变量注入的事实”命名系统。

将 Ansible 配置文件的 **[default]** 部分中的 `inject_facts_as_vars` 参数设置为 `false`，可关闭旧命名系统。默认设置目前为 `true`。

`inject_facts_as_vars` 的默认值在 Ansible 的未来版本中可能会更改为 `false`。如果设置为 `false`，则只能使用新的 `ansible_facts.*` 命名系统引用 Ansible 事实。在这种情况下，尝试通过旧命名空间引用事实将导致以下错误：

```
...output omitted...
TASK [Show me the facts] *****
fatal: [demo.example.com]: FAILED! => {"msg": "The task includes an option
with an undefined variable. The error was: 'ansible_distribution' is
undefined\n\nThe error appears to have been in
'/home/student/demo/playbook.yml': line 5, column 7, but may\nbe elsewhere in
the file depending on the exact syntax problem.\n\nThe offending line appears
to be:\n\n  tasks:\n    - name: Show me the facts\n          ^ here\n"}
...output omitted...
```

## 关闭事实收集

有时，您不想为 play 收集事实。这样做的原因有几个。可能是您不准备使用任何事实，并且希望加快 play 速度或减小 play 在受管主机上造成的负载。可能是受管主机因为某种原因而无法运行 `setup` 模块，或者需要安装一些必备软件后再收集事实。

要为 play 禁用事实收集，可将 `gather_facts` 关键字设置为 `no`：

```
---
- name: This play gathers no facts automatically
  hosts: large_farm
  gather_facts: no
```

即使为 play 设置了 `gather_facts: no`，您也可以随时通过运行使用 `setup` 模块的任务来手动收集事实：

```
tasks:
  - name: Manually gather facts
    setup:
...output omitted...
```

## 创建自定义事实

管理员可以创建自定义事实，将其本地存储在每个受管主机上。这些事实整合到 `setup` 模块在受管主机上运行时收集的标准事实列表中。它们让受管主机能够向 Ansible 提供任意变量，以用于调整 play 的行为。

自定义事实可以在静态文件中定义，格式可为 INI 文件或采用 JSON。它们也可以是生成 JSON 输出的可执行脚本，如同动态清单脚本一样。

借助自定义事实，管理员可以为受管主机定义特定的值，供 play 用于填充配置文件或有条件地运行任务。动态自定义事实允许在 play 运行时以编程方式确定这些事实的值，甚至还可以确定提供哪些事实。

默认情况下，`setup` 模块从各受管主机的 `/etc/ansible/facts.d` 目录下的文件和脚本中加载自定义事实。各个文件或脚本的名称必须以 `.fact` 结尾才能被使用。动态自定义事实脚本必须输出 JSON 格式的事实，而且必须是可执行文件。

以下是采用 INI 格式编写的静态自定义事实文件。INI 格式的自定义事实文件包含由一个部分定义的顶层值，后跟用于待定义的事实的键值对：

```
[packages]
web_package = httpd
db_package = mariadb-server

[users]
user1 = joe
user2 = jane
```

同样的事实可能以 JSON 格式提供。以下 JSON 事实等同于以上示例中 INI 格式指定的事实。JSON 数据可以存储在静态文本文件中，或者通过可执行脚本输出到标准输出：

```
{
  "packages": {
    "web_package": "httpd",
    "db_package": "mariadb-server"
  },
  "users": {
    "user1": "joe",
    "user2": "jane"
  }
}
```



### 注意

自定义事实文件不能采用 playbook 那样的 YAML 格式。JSON 格式是最为接近的等效格式。

自定义事实由 `setup` 模块存储在 `ansible_facts.ansible_local` 变量中。

事实按照定义它们的文件的名称来整理。例如，假设前面的自定义事实由受管主机上保存为 `/etc/ansible/facts.d/custom.fact` 的文件生成。在这种情况下，`ansible_facts.ansible_local['custom']['users']['user1']` 的值为 `joe`。

您可以利用临时命令在受管主机上运行 `setup` 模块来检查自定义事实的结构。

```
[user@demo ~]$ ansible demo1.example.com -m setup
demo1.example.com | SUCCESS => {
    "ansible_facts": {
        ...output omitted...
        "ansible_local": {
            "custom": {
                "packages": {
                    "db_package": "mariadb-server",
                    "web_package": "httpd"
                },
                "users": {
                    "user1": "joe",
                    "user2": "jane"
                }
            }
        },
        ...output omitted...
    },
    "changed": false
}
```

自定义事实的使用方式与 playbook 中的默认事实相同：

```
[user@demo ~]$ cat playbook.yml
---
- hosts: all
  tasks:
    - name: Prints various Ansible facts
      debug:
        msg: >
              The package to install on {{ ansible_facts['fqdn'] }}
              is {{ ansible_facts['ansible_local']['custom']['packages'] }}[ 'web_package' ] }

[user@demo ~]$ ansible-playbook playbook.yml
PLAY ****
TASK [Gathering Facts] ****
ok: [demo1.example.com]

TASK [Prints various Ansible facts] ****
ok: [demo1.example.com] => {
    "msg": "The package to install on demo1.example.com is httpd"
}

PLAY RECAP ****
demo1.example.com : ok=2     changed=0     unreachable=0    failed=0
```

## 使用魔法变量

一些变量并非事实或通过 `setup` 模块配置，但也由 Ansible 自动设置。这些魔法变量也可用于获取与特定受管主机相关的信息。

最常用的有四个：

#### hostvars

包含受管主机的变量，可以用于获取另一台受管主机的变量的值。如果还没有为受管主机收集事实，则它不会包含该主机的事实。

#### group\_names

列出当前受管主机所属的所有组。

#### groups

列出清单中的所有组和主机。

#### inventory\_hostname

包含清单中配置的当前受管主机的主机名称。这可能因为各种原因而与事实报告的主机名称不同。

另外还有许多其他的“魔法变量”。有关更多信息，请参见 [https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_variables.html#variable-precedence-where-should-i-put-a-variable](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable)。若要深入了解它们的值，一个途径是使用 `debug` 模块报告特定主机的 `hostvars` 变量的内容：

```
[user@demo ~]$ ansible localhost -m debug -a 'var=hostvars["localhost"]'
localhost | SUCCESS => {
    "hostvars[\"localhost\"]": {
        "ansible_check_mode": false,
        "ansible_connection": "local",
        "ansible_diff_mode": false,
        "ansible_facts": {},
        "ansibleforks": 5,
        "ansible_inventory_sources": [
            "/home/student/demo/inventory"
        ],
        "ansible_playbook_python": "/usr/bin/python2",
        "ansible_python_interpreter": "/usr/bin/python2",
        "ansible_verbose": 0,
        "ansible_version": {
            "full": "2.7.0",
            "major": 2,
            "minor": 7,
            "revision": 0,
            "string": "2.7.0"
        },
        "group_names": [],
        "groups": {
            "all": [
                "serverb.lab.example.com"
            ],
            "ungrouped": [],
            "webservers": [
                "serverb.lab.example.com"
            ]
        },
        "inventory_hostname": "localhost",
        "inventory_hostname_short": "localhost",
        "omit": "__omit_place_holder__18d132963728b2cbf7143dd49dc4bf5745fe5ec3"
    }
}
```

```
    "playbook_dir": "/home/student/demo"
}
}
```



#### 参考文献

##### **setup - 收集关于远程主机的事实 &mdash; Ansible 文档**

[https://docs.ansible.com/ansible/latest/modules/setup\\_module.html](https://docs.ansible.com/ansible/latest/modules/setup_module.html)

##### **本地事实 (Facts.d) &mdash; 变量 &mdash; Ansible 文档**

[https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_variables.html#local-facts-facts-d](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#local-facts-facts-d)

## ► 指导练习

# 管理事实

在本练习中，您将从受管主机收集 Ansible 事实并在 play 中使用它们。

## 成果

您应能够：

- 从主机收集事实。
- 创建使用所收集事实的任务。

## 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-facts start** 命令。该脚本将创建工作目录 **data-facts**，并为它填充 Ansible 配置文件和主机清单。

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

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

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

- 2. Ansible **setup** 模块从系统检索事实。运行一个临时命令，检索 **webserver** 组中所有服务器的事实。输出中将以 JSON 格式显示为 **servera.lab.example.com** 收集的所有事实。检查显示的部分变量。

```
[student@workstation data-facts]$ ansible webserver -m setup
...output omitted...
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.25.250.10"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::2937:3aa3:ea8d:d3b1"
        ],
    },
...output omitted...
```

- 3. 在 workstation 上，创建名为 **/home/student/data-facts/custom.fact** 的事实文件。该事实文件定义要在 **servera** 上安装的软件包和启动的服务。该文件应当如下所示：

```
[general]
package = httpd
service = httpd
state = started
enabled = true
```

- 4. 创建 `setup_facts.yml` playbook，以生成 `/etc/ansible/facts.d` 远程目录并将 `custom факт` 文件保存到该目录。

```
---
- name: Install remote facts
  hosts: webserver
  vars:
    remote_dir: /etc/ansible/facts.d
    facts_file: custom.fact
  tasks:
    - name: Create the remote directory
      file:
        state: directory
        recurse: yes
        path: "{{ remote_dir }}"
    - name: Install the new facts
      copy:
        src: "{{ facts_file }}"
        dest: "{{ remote_dir }}"
```

- 5. 使用 `setup` 模块运行一个临时命令。搜索输出中的 `ansible_local` 部分。这时应当没有任何自定义事实。

```
[student@workstation data-facts]$ ansible webserver -m setup
servera.lab.example.com | SUCCESS => {
  "ansible_facts": {
    ...output omitted...
    "ansible_local": {}
    ...output omitted...
  },
  "changed": false
}
```

- 6. 运行该 playbook 前，通过运行 `ansible-playbook --syntax-check` 来验证其语法是否正确。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-facts]$ ansible-playbook --syntax-check setup_facts.yml
playbook: setup_facts.yml
```

▶ 7. 运行 `setup_facts.yml` playbook。

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

PLAY [Install remote facts] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Create the remote directory] ****
changed: [servera.lab.example.com]

TASK [Install the new facts] ****
changed: [servera.lab.example.com]

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

▶ 8. 现在可以创建主 playbook，以同时使用默认事实和用户事实来配置 `servera`。在接下来的几步中，您要向 playbook 文件添加内容。创建 `playbook.yml` playbook，使其包含以下内容：

```
---
- name: Install Apache and starts the service
  hosts: webserver
```

▶ 9. 继续编辑 `playbook.yml` 文件，创建安装 `httpd` 软件包的第一项任务。将用户事实用作软件包名称。

```
tasks:
  - name: Install the required package
    yum:
      name: "{{ ansible_facts['ansible_local']['custom']['general']['package'] }}"
      state: latest
```

▶ 10. 创建另一项任务，该任务使用自定义事实来启动 `httpd` 服务。

```
- name: Start the service
  service:
    name: "{{ ansible_facts['ansible_local']['custom']['general']['service'] }}"
    state: "{{ ansible_facts['ansible_local']['custom']['general']['state'] }}"
    enabled: "{{ ansible_facts['ansible_local']['custom']['general']['enabled'] }}"
```

- 11. 完成所有任务时，完整的 playbook 应当如下所示：检查该 playbook，确保所有任务都已定义。

```

---
- name: Install Apache and starts the service
  hosts: webserver

  tasks:
    - name: Install the required package
      yum:
        name: "{{ ansible_facts['ansible_local']['custom']['general']['package'] }}"
        state: latest

    - name: Start the service
      service:
        name: "{{ ansible_facts['ansible_local']['custom']['general']['service'] }}"
        state: "{{ ansible_facts['ansible_local']['custom']['general']['state'] }}"
        enabled: "{{ ansible_facts['ansible_local']['custom']['general']['enabled'] }}"

```

- 12. 在运行 playbook 前，先使用临时命令验证 httpd 服务目前尚未在 servera 上运行。

```
[student@workstation data-facts]$ ansible servera.lab.example.com -m command \
> -a 'systemctl status httpd'
servera.lab.example.com | FAILED | rc=4 >>
Unit httpd.service could not be found.non-zero return code
```

- 13. 通过运行 **ansible-playbook --syntax-check**，验证 playbook 的语法。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-facts]$ ansible-playbook --syntax-check playbook.yml

playbook: playbook.yml
```

- 14. 使用 **ansible-playbook** 命令运行该 playbook。观察输出结果，Ansible 将先安装软件包，然后启用服务。

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

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

TASK [Install the required package] ****
changed: [servera.lab.example.com]

TASK [Start the service] ****
```

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

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

- 15. 使用一个临时命令来执行 **systemctl**, 以确定 servera 现在是否在运行 httpd 服务。

```
[student@workstation data-facts]$ ansible servera.lab.example.com -m command \
> -a 'systemctl status httpd'
servera.lab.example.com | CHANGED | rc=0 >>
● httpd.service - The Apache HTTP Server
  Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset:
disabled)
  Active: active (running) since Mon 2019-05-27 07:50:55 EDT; 50s ago
    Docs: man:httpd.service(8)
 Main PID: 11603 (httpd)
   Status: "Running, listening on: port 80"
     Tasks: 213 (limit: 4956)
    Memory: 24.1M
      CGroup: /system.slice/httpd.service
...output omitted...
```

## 完成

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

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

本引导式练习到此结束。

## ▶ 开放研究实验

# 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
<b>firewall_pkg</b>	<b>firewalld</b>
<b>firewall_svc</b>	<b>firewalld</b>
<b>web_pkg</b>	<b>httpd</b>
<b>web_svc</b>	<b>httpd</b>
<b>ssl_pkg</b>	<b>mod_ssl</b>
<b>httpdconf_src</b>	<b>files/httpd.conf</b>
<b>httpdconf_dest</b>	<b>/etc/httpd/conf/httpd.conf</b>

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

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。
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。
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。
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.
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。
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。
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。
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。
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。
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**。
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。
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。

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。
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。

## 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
```

本实验到此结束。