Chương 2. Tự động hóa các nhiệm vụ đơn giản

Như chúng ta đã đề cập trong chương trước, Ansible có thể được sử dụng cho cả hai, tạo và quản lý toàn bộ cơ sở hạ tầng, cũng như được tích hợp vào cơ sở hạ tầng đã hoạt động.

Trong chương này, chúng ta sẽ thấy:

- Playbook là gì và hoạt động như thế nào
- Cách tạo máy chủ web bằng Ansible
- Một cái nhìn cận cảnh về công cụ mẫu Jinja2

Nhưng trước tiên chúng ta sẽ nói về **YAML Ain't Markup Language** (**YAML**), một ngôn ngữ tuần tự hóa dữ liệu có thể đọc được của con người được sử dụng rộng rãi trong Ansible.

YAML

YAML, giống như nhiều ngôn ngữ tuần tự hóa dữ liệu khác (như JSON), có rất ít, các khái niệm cơ bản:

- Tuyên bố
- Danh sách
- mảng kết hợp

Một khai báo rất giống với một biến trong bất kỳ ngôn ngữ nào khác, đó là:

```
tên: 'Đây là tên'
```

Để tạo danh sách, chúng tôi sẽ phải sử dụng ' - ':

```
- 'mục1'
- 'mục2'
- 'mục3'
```

YAML sử dụng thụt đầu dòng để phân chia hợp lý cha mẹ với con cái. Vì vậy, nếu chúng ta muốn tạo các mảng kết hợp (còn được gọi là các đối tượng), chúng ta chỉ cần thêm một vết lõm:

```
mục:
  tên: Tên
  địa điểm: TheLocation
```

Rõ ràng, chúng ta có thể trộn chúng lại với nhau, đó là:

```
những người:
  - tên: Albert
    số: +1000000000
    đất nước: Hoa Kỳ
  - tên: David

    số: +44000000000
    quốc gia: Anh
```

Đó là những điều cơ bản của YAML. YAML có thể làm nhiều hơn nữa, nhưng bây giờ điều này là đủ.

Xin chào

Như chúng ta đã thấy trong chương trước, có thể sử dụng Ansible để tự động hóa các tác vụ đơn giản mà bạn có thể đã thực hiện hàng ngày.

Hãy bắt đầu bằng cách kiểm tra xem một máy từ xa có thể truy cập được không; nói cách khác, hãy bắt đầu bằng cách ping máy.

Cách đơn giản nhất để làm điều này là chạy như sau:

```
$ ansible all -i HOST, -m ping
```

Ở đây, `HOST` là một địa chỉ IP, **Tên miền đủ điều kiện** (**FQDN**) hoặc bí danh của một máy mà bạn có quyền truy cập SSH (bạn có thể sử dụng **Máy ảo dựa trên hạt nhân** (**KVM**), như chúng ta đã thấy trong chương trước).

tiền boa

Sau " `HOST` ", dấu phẩy là bắt buộc, vì nếu không, nó sẽ không được xem như một danh sách, mà là một chuỗi.

Trong trường hợp này, chúng tôi đã thực hiện nó với một máy ảo trên hệ thống của chúng tôi:

```
$ ansible all -i test01.fale.io, -m ping
```

Bạn sẽ nhận được một cái gì đó như thế này là kết quả:

```
test01.fale.io | THÀNH CÔNG => {
    "đã thay đổi": sai,
    "bóng bàn"
}
```

Bây giờ, hãy xem những gì chúng tôi đã làm và tại sao. Hãy bắt đầu từ sự giúp đỡ của Ansible. Để truy vấn nó, chúng ta có thể sử dụng lệnh sau:

```
$ ansible - trợ giúp
```

Để dễ đọc hơn, chúng tôi đã xóa tất cả đầu ra liên quan đến các tùy chọn mà chúng tôi chưa sử dụng:

```
Cách sử dụng: ansible <host-pattern> [tùy chọn]
Tùy chọn:
  -i HÀNG ĐẦU, --inventory-file = KHOẢN
                        chỉ định đường dẫn máy chủ kho
                        (mặc định = / etc / ansible / hosts) hoặc dấu
                        phẩy
                        danh sách máy chủ riêng biệt.
  -m MODULE_NAME, --module-name = MODULE_NAME
                        Tên mô-đun để thực thi (default = lệnh)
```

Vì vậy, những gì chúng tôi làm là:

1. Chúng tôi gọi Ansible.
2. Chúng tôi đã hướng dẫn Ansible chạy trên tất cả các máy chủ.
3. Chúng tôi đã chỉ định hàng tồn kho của chúng tôi (còn được gọi là danh sách các máy chủ lưu trữ).
4. Chúng tôi đã chỉ định mô-đun chúng tôi muốn chạy (`ping`).

Bây giờ chúng ta có thể ping máy chủ, hãy tạo `tiếng vang xin chào!`

```
$ ansible all -i test01.fale.io, -m shell -a '/ bin / echo xin
chào ansible!'
```

Bạn sẽ nhận được một cái gì đó như thế này là kết quả:

```
test01.fale.io | THÀNH CÔNG | RC = 0 >>
```

```
xin chào ansible!
```

Trong ví dụ này, chúng tôi đã sử dụng một tùy chọn bổ sung. Hãy kiểm tra trợ giúp để xem những gì nó làm:

```
Cách sử dụng: ansible <host-pattern> [tùy chọn]
Tùy chọn:
  -a MODULE_ARGS, --args = MODULE_ARGS đối số
                                mô-đun
```

Như bạn có thể đoán từ bối cảnh và tên, các `args` tùy chọn cho phép bạn vượt qua đối số bổ sung cho các mô-đun. Một số mô-đun (như `ping`) không hỗ trợ bất kỳ đối số nào, trong khi các mô-đun khác (như `shell`) sẽ yêu cầu đối số.

Làm việc với playbooks

Playbooks là một trong những tính năng cốt lõi của Ansible và cho Ansible biết những gì cần thực hiện. Chúng giống như một danh sách việc cần làm cho Ansible có chứa danh sách các nhiệm vụ; mỗi tác vụ bên trong liên kết đến một đoạn mã gọi là **mô-đun** . Playbook là các tệp YAML đơn giản, dễ đọc với con người, trong khi các mô-đun là một đoạn mã có thể được viết bằng bất kỳ ngôn ngữ nào với điều kiện đầu ra của nó phải ở định dạng JSON. Bạn có thể có nhiều nhiệm vụ được liệt kê trong một cuốn sách và những nhiệm vụ này sẽ được Ansible thực hiện một cách thanh thản. Bạn có thể nghĩ về playbooks như một biểu hiện tương đương trong Puppet, bang ở Salt hoặc sách dạy nấu ăn trong Chef; chúng cho phép bạn nhập danh sách các tác vụ hoặc lệnh bạn muốn thực hiện trên hệ thống từ xa.

Nghiên cứu giải phẫu của một cuốn sách

Playbook có thể có một danh sách các máy chủ từ xa, biến người dùng, tác vụ, trình xử lý, v.v. Bạn cũng có thể ghi đè hầu hết các cài đặt cấu hình thông qua một playbook. Chúng ta hãy bắt đầu xem xét giải phẫu của một cuốn sách.

Mục đích của playbook chúng ta sẽ xem xét bây giờ, là để đảm bảo rằng gói `httpd` được cài đặt và dịch vụ được **kích hoạt** và **bắt đầu** . Đây là nội dung của tệp `setup_apache.yaml` :

```
---
- máy chủ: tất cả
  remote_user: nhiệm
  vụ fale:
  - name: Đảm bảo gói HTTPd được cài đặt yum:
      tên: httpd
      nhà nước: hiện tại
      trở thành sự thật
  - name: Đảm bảo dịch vụ HTTPd được bật và chạy dịch vụ:
      tên: httpd
      bang: đã bắt đầu
      kích hoạt: True
    trở thành: Đúng
```

Tệp `setup_apache.yaml` là một ví dụ về playbook. Các tập tin bao gồm ba phần chính, như sau:

- `máy chủ` : Điều này liệt kê máy chủ hoặc nhóm máy chủ mà chúng tôi muốn chạy tác vụ. Trường chủ là bắt buộc và mọi playbook nên có nó. Nó báo cho Ansible biết máy chủ nào sẽ chạy các tác vụ được liệt kê. Khi được cung cấp một nhóm máy chủ, Ansible sẽ lấy nhóm máy chủ từ playbook và thử tìm nó trong một tệp kiểm kê. Nếu không có kết quả khớp, Ansible sẽ bỏ qua tất cả các nhiệm vụ cho nhóm máy chủ đó. Các `--list-host` tùy chọn cùng với playbook (`ansible-playbook <playbook> --list-host`) sẽ cho bạn biết chính xác những host PlayBook sẽ chạy chống lại.

- `remote_user` : Đây là một trong các tham số câu hình của Ansible (ví dụ: xem xét, `tom - remote_user`) cho Ansible sử dụng một người dùng cụ thể (trong trường hợp này là `tom`) trong khi đăng nhập vào hệ thống.
- `nhiệm vụ` : Cuối cùng, chúng ta đến với nhiệm vụ. Tất cả các sách vở nên chứa các nhiệm vụ. Nhiệm vụ là một danh sách các hành động bạn muốn thực hiện. Trường tác vụ chứa tên của tác vụ (nghĩa là văn bản trợ giúp cho người dùng về tác vụ), một mô-đun nên được thực thi và các đối số được yêu cầu cho mô-đun. Chúng ta hãy xem xét một nhiệm vụ được liệt kê trong playbook, như được hiển thị trong đoạn mã trước:

chú thích

Tất cả các ví dụ trong cuốn sách sẽ được thực thi trên CentOS, nhưng cùng một tập hợp các ví dụ với một vài thay đổi cũng sẽ hoạt động trên các bản phân phối khác.

Trong trường hợp trước, có hai nhiệm vụ. Các `tên` tham số đại diện cho những gì công việc đang làm và là `hiện nay` chủ yếu là để cải thiện khả năng đọc, như chúng ta sẽ thấy trong thời gian chạy playbook. Các `tên` tham số là không bắt buộc. Các `mô-đun` , `yum` và `dịch vụ` , có bộ thông số riêng. Hầu như tất cả các mô-đun có tham số `tên` (có các trường hợp ngoại lệ như mô-đun `gỡ lỗi`), cho biết các hành động được thực hiện trên thành phần nào. Hãy xem xét các thông số khác:

- Trong trường hợp của mô-đun `yum` , tham số `trạng thái` có giá trị mới nhất và nó chỉ ra rằng gói `httpd` mới nhất sẽ được cài đặt. Lệnh thực thi nhiều hay ít dịch sang `yum install httpd` .
- Trong kịch bản của mô-đun `dịch vụ` , tham số `trạng thái` có giá trị bắt đầu cho biết rằng dịch vụ `httpd` sẽ được khởi động và nó tạm dịch là `/etc/init.d/httpd bắt đầu`. Trong mô-đun này, chúng tôi cũng có tham số " `đã bật` " xác định liệu dịch vụ có nên khởi động khi khởi động hay không.
- Các `trở thành: True` tham số đại diện cho một thực tế rằng việc này cần được thực hiện với `sudo` truy cập. Nếu tệp `sudo` user không cho phép người dùng chạy lệnh cụ thể, thì playbook sẽ thất bại khi chạy.

chú thích

Bạn có thể có câu hỏi về lý do tại sao không có mô-đun gói nào chỉ ra kiến trúc bên trong và chạy `yum` , `apt` hoặc bất kỳ tùy chọn gói nào khác tùy thuộc vào kiến trúc của hệ thống. Ansible điền giá trị trình quản lý gói vào một biến có tên `ansible_pkg_manager` .

Nói chung, chúng ta cần nhớ rằng số lượng các gói có tên chung trên các hệ điều hành khác nhau là một tập hợp nhỏ của số lượng các gói thực sự có mặt. Ví dụ: `gói httpd` được gọi là `httpd` trong các hệ thống Red Hat và `apache2` trong các hệ thống dựa trên Debian. Chúng ta cũng cần nhớ rằng mỗi trình quản lý gói có một bộ tùy chọn riêng làm cho nó trở nên mạnh mẽ; kết quả là, sẽ hợp lý hơn khi sử dụng tên trình quản lý gói rõ ràng để toàn bộ tùy chọn có sẵn cho người dùng cuối viết playbook.

Chạy một cuốn sách

Bây giờ, đã đến lúc (vâng, cuối cùng!) Để chạy playbook. Để hướng dẫn Ansible thực thi một playbook thay vì một mô-đun, chúng ta sẽ phải sử dụng một lệnh khác (`ansible-playbooks`) có cú pháp rất giống với lệnh " `ansible` " mà chúng ta đã thấy:

```
$ ansible-playbook -i HOST, setup_apache.yaml
```

Như bạn có thể thấy, ngoài mẫu máy chủ (được chỉ định trong playbook) đã biến mất và tùy chọn mô-đun đã được thay thế bằng tên của playbook, không có gì thay đổi. Vì vậy, để thực thi lệnh này trên máy của tôi, lệnh chính xác là:

```
$ ansible-playbook -i test01.fale.io, setup_apache.yaml
```

Kết quả là như sau:

```
Chơi tất cả] ********************************************** ******
NHIỆM VỤ [thiết lập] ****** /
4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8 ******
```

```
ok: [test01.fale.io]


NHIỆM VỤ [Đảm bảo gói HTTPd đã được cài đặt] ****** / TÌM HIỂU đã thay
đổi: [test01.fale.io]

NHIỆM VỤ [Đảm bảo dịch vụ HTTPd được bật và chạy] *********** đã thay
đổi: [test01.fale.io]

CHƠI RECAP ****** TÌM HIỂU ******
test01.fale.io : ok = 3 đã thay đổi = 2 không thể truy cập = 0 thất bại =
0
```

Ồ Ví dụ đã làm việc. Bây giờ chúng ta hãy kiểm tra xem gói `httpd đã` được cài đặt và chạy và chạy trên máy chưa. Để kiểm tra xem HTTPd đã được cài đặt chưa, cách dễ nhất là hỏi `vòng / phút` :

```
$ vòng / phút | grep httpd
```

Nếu mọi thứ hoạt động đúng, bạn nên có một đầu ra như sau:

```
httpd-tools-2.4.6-40.el7.centos.x86_64
httpd-2.4.6-40.el7.centos.x86_64
```

Để xem trạng thái của dịch vụ, chúng tôi có thể hỏi `systemd` :

```
$ trạng thái systemctl httpd
```

Kết quả mong đợi là như sau:

```
httpd.service - Máy chủ HTTP Apache
  Đã tải: đã tải (/usr/lib/systemd/system/httpd.service; đã bật;
nhà cung cấp cài sẵn: bị vô hiệu hóa)
   Hoạt động: hoạt động (đang chạy) kể từ Thứ bảy 2016-05-07
trước
13:22:14 EDT; 7 phút
     Tài liệu: người đàn ông: httpd (8)
           người đàn ông: apachectl (8)
 Chính PID: 2214 (httpd)
   Trạng thái: "Tổng số yêu cầu: 0; Yêu cầu hiện tại / giây: 0; Hiện tại
lưu lượng truy cập: 0 B / giây "
   Cgroup: /system.slice/httpd.service
           -2214 / usr / sbin / httpd-GIỚI THIỆU
           -2215 / usr / sbin / httpd-GIỚI THIỆU
           -2216 / usr / sbin / httpd-GIỚI THIỆU
           -2217 / usr / sbin / httpd-GIỚI THIỆU
           -2218 / usr / sbin / httpd-GIỚI THIỆU
           -2219 / usr / sbin / httpd-GIỚI THIỆU
```

Trạng thái kết thúc, theo playbook, đã đạt được. Chúng ta hãy xem xét ngắn gọn chính xác những gì xảy ra trong quá trình chơi playbook:

```
Chơi tất cả] ********************************************** ******
```

Dòng này cho chúng tôi biết rằng một playbook sẽ bắt đầu ở đây và nó sẽ được thực thi trên các máy chủ " tất cả ":

```
NHIỆM VỤ [thiết lập] ****** /
4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8 ****** ok:
[test01.fale.io]
```

Các dòng TASK hiển thị tên của tác vụ (thiết lập trong trường hợp này) và tác dụng của chúng trên mỗi máy chủ. Đôi khi mọi người bị lẫn lộn bởi nhiệm vụ thiết lập . Trong thực tế, nếu bạn nhìn vào playbook, không có nhiệm vụ thiết lập . Điều này là do Ansible, trước khi thực hiện các tác vụ mà chúng tôi đã yêu cầu, sẽ cố gắng kết nối với máy và thu thập thông tin về nó có thể hữu ích sau này. Như bạn có thể thấy, tác vụ dẫn đến trạng thái ok màu xanh lá cây , vì vậy nó đã thành công và không có gì thay đổi trên máy chủ:

```
NHIỆM VỤ [Đảm bảo gói HTTPd đã được cài đặt] ****** / TÌM HIỂU
đã thay đổi: [test01.fale.io]
NHIỆM VỤ [Đảm bảo dịch vụ HTTPd được bật và chạy] ****** / 4
đã thay đổi: [test01.fale.io]
```

Hai trạng thái của nhiệm vụ là màu vàng và chính tả " thay đổi ". Điều này có nghĩa là những tác vụ đó đã được thực thi và đã thành công nhưng thực sự đã thay đổi một cái gì đó trên máy:

```
CHƠI RECAP ****** TÌM HIỂU ******* test01.fale.io: ok = 3 đã thay đổi =
2 không thể truy cập = 0 thất bại = 0
```

Những dòng cuối cùng là một tóm tắt về cách chơi đã đi. Bây giờ hãy chạy lại tác vụ và xem đầu ra sau khi cả hai tác vụ đã thực sự chạy:

```
Chơi tất cả] ********************************************** ******
NHIỆM VỤ [thiết lập] ****** /
4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8 ******
ok: [test01.fale.io]
NHIỆM VỤ [Đảm bảo gói HTTPd đã được cài đặt] ****** / TÌM HIỂU
ok: [test01.fale.io]
NHIỆM VỤ [Đảm bảo dịch vụ HTTPd được bật và chạy] ****** / 4
ok: [test01.fale.io]
CHƠI RECAP ****** TÌM HIỂU ******
test01.fale.io : ok = 3 thay đổi = 0 không thể truy cập = 0 thất bại =
0
```

Như bạn đã mong đợi, hai tác vụ được đề cập cho kết quả đầu ra là ok , điều đó có nghĩa là trạng thái mong muốn đã được đáp ứng trước khi chạy tác vụ. Điều quan trọng cần nhớ là nhiều tác vụ như nhiệm vụ **Thu thập** thông tin có được thông tin liên quan đến một thành phần cụ thể của hệ thống và không nhất thiết phải thay đổi bất cứ điều gì trên hệ thống; do đó, các tác vụ này không hiển thị đầu ra thay đổi trước đó.

Phần PLAY RECAP trong lần chạy đầu tiên và thứ hai được hiển thị như sau. Bạn sẽ thấy đầu ra sau trong lần chạy đầu tiên:

```
CHƠI RECAP ****** TÌM HIỂU ******
test01.fale.io : ok = 3 đã thay đổi = 2 không thể truy cập = 0 thất bại =
0
```

Bạn sẽ thấy đầu ra sau trong lần chạy thứ hai:

```
CHƠI RECAP ****** TÌM HIỂU ******
test01.fale.io : ok = 3 thay đổi = 0 không thể truy cập = 0 thất bại =
0
```

Như bạn có thể thấy, sự khác biệt là đầu ra của tác vụ đầu tiên `thay đổi = 2` , có nghĩa là trạng thái hệ thống thay đổi hai lần do hai tác vụ. Rất hữu ích khi xem xét đầu ra này, vì, nếu một hệ thống đạt được trạng thái mong muốn và sau đó bạn chạy playbook trên nó, thì đầu ra dự kiến sẽ được `thay đổi = 0` .

Nếu bạn đang nghĩ đến từ **Idempotency** ở giai đoạn này, bạn hoàn toàn đúng và xứng đáng được vỗ lưng! Idempotency là một trong những nguyên lý chính của quản lý cấu hình. Wikipedia định nghĩa Idempotency là một hoạt động, nếu được áp dụng hai lần cho bất kỳ giá trị nào, sẽ cho kết quả tương tự như khi nó được áp dụng một lần. Các ví dụ sớm nhất về điều này mà bạn sẽ gặp trong thời thơ ấu của mình sẽ là các phép toán nhân trên số `1` , trong đó `1 * 1 = 1` mỗi lần.

Hầu hết các công cụ quản lý cấu hình đã áp dụng nguyên tắc này và áp dụng nó vào cơ sở hạ tầng. Trong một cơ sở hạ tầng lớn, rất nên theo dõi hoặc theo dõi số lượng nhiệm vụ thay đổi trong cơ sở hạ tầng của bạn và cảnh báo các nhiệm vụ liên quan nếu bạn tìm thấy sự kỳ lạ; điều này áp dụng cho bất kỳ công cụ quản lý cấu hình nói chung. Ở trạng thái lý tưởng, lần duy nhất bạn sẽ thấy các thay đổi là khi bạn giới thiệu một thay đổi mới dưới dạng bất kỳ thao tác **Tạo** , **Xóa** , **Cập nhật** hoặc **Xóa** (**CRUD**) nào trên các thành phần hệ thống khác nhau. Nếu bạn đang tự hỏi làm thế nào bạn có thể làm điều đó với Ansible, hãy tiếp tục đọc cuốn sách và cuối cùng bạn sẽ tìm thấy câu trả lời!

Hãy tiếp tục. Bạn cũng có thể đã viết các tác vụ trước như sau nhưng khi các tác vụ được chạy, từ góc độ của người dùng cuối, chúng khá dễ đọc (chúng tôi sẽ gọi tệp này là `setup_apache_no_com.yaml`):

```
---
- máy chủ: tất cả
  remote_user: fale
  nhiệm vụ:
  - yum:
      tên: httpd
      nhà nước: hiện tại
    trở thành sự thật
  - dịch vụ: tên:
      httpd
      bang: đã bắt đầu
      đã bật: Đúng
    trở thành sự thật
```

Hãy chạy lại playbook để phát hiện ra bất kỳ sự khác biệt nào trong đầu ra:

```
$ ansible-playbook -i test01.fale.io, setup_apache_no_com.yaml
```

Đầu ra sẽ là:

```
Chơi tất cả] ********************************************** ******
NHIỆM VỤ [thiết lập] ****** /
4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8 ******
ok: [test01.fale.io]


NHIỆM VỤ [yum] ****** / 4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8

****** / ok: [test01.fale.io]

NHIỆM VỤ [dịch vụ] ****** /

4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8 ***** ok:

[test01.fale.io]

CHƠI RECAP ****** TÌM HIỂU ******
```

```
test01.fale.io : ok = 3 thay đổi = 0 không thể truy cập = 0 thất bại =
0
```

Như bạn có thể thấy, sự khác biệt nằm ở khả năng đọc. Bất cứ nơi nào có thể, bạn nên giữ các tác vụ đơn giản nhất có thể (nguyên tắc **KISS** của **Keep It Simple St ngu**) để cho phép duy trì các tập lệnh của bạn trong thời gian dài.

Bây giờ chúng ta đã thấy làm thế nào bạn có thể viết một cuốn sách cơ bản và chạy nó với một máy chủ, hãy xem các tùy chọn khác sẽ giúp bạn trong khi chạy các cuốn sách.

Độ dài ansible

Một trong những tùy chọn đầu tiên mà bất cứ ai chọn là tùy chọn gỡ lỗi. Để hiểu những gì đang xảy ra khi bạn chạy playbook, bạn có thể chạy nó với tùy chọn **verbose** (`-v`). Mỗi `v` thêm sẽ cung cấp cho người dùng cuối với đầu ra gỡ lỗi nhiều hơn.

Chúng ta hãy xem một ví dụ về việc sử dụng gỡ lỗi playbook cho một tác vụ bằng các tùy chọn gỡ lỗi sau:

- Các `-v` tùy chọn cung cấp đầu ra mặc định, như trong các ví dụ trước.
- Các `-vv` tùy chọn bổ sung thêm một ít thông tin hơn, như trong ví dụ sau:

```
Sử dụng /etc/ansible/ansible.cfg làm tệp cấu hình

PLAYBOOK: setup_apache.yaml ****** / TÌM HIỂU 1 lượt chơi
trong setup_apache.yaml
Chơi tất cả] ********************************************** **

NHIỆM VỤ [thiết lập] ****** /
4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8
ok: [test01.fale.io]

NHIỆM VỤ [Đảm bảo gói HTTPd đã được cài đặt] ****** / TÌM HIỂU
đường dẫn tác vụ: /home/fale/setup_apache.yaml giáp
ok: [test01.fale.io] => {"đã thay đổi": sai, "tin nhắn": "",
"rc": 0, "kết quả": ["httpd-2.4.6-40.el7.centos.x86_64 cung cấp httpd
đã được cài đặt "]}
NHIỆM VỤ [Đảm bảo dịch vụ HTTPd được bật và chạy] **** đường
dẫn tác vụ: /home/fale/setup_apache.yaml:10
ok: [test01.fale.io] => {"đã thay đổi": sai, "đã bật": đúng,
"tên": "httpd", "bang": "đã bắt đầu"}

CHƠI RECAP ****** TÌM HIỂU test01.fale.io: ok = 3 thay đổi = 0
không thể truy cập = 0 thất bại = 0
```

- Các `-vvv` tùy chọn thêm một thông tin rất nhiều, như trong đoạn mã sau. Điều này cho thấy lệnh `ssh` Ansible sử dụng để tạo một tệp tạm thời trên máy chủ từ xa và chạy tập lệnh từ xa:

```
NHIỆM VỤ [Đảm bảo gói HTTPd đã được cài đặt] ****** TÌM HIỂU
đường dẫn tác vụ: /home/fale/setup_apache.yaml giáp
<test01.fale.io> THÀNH LẬP KẾT NỐI SSH CHO NGƯỜI DÙNG: fale
<test01.fale.io> SSH: EXEC ssh -C -q -o ControlMaster = auto -o
ControlPersist = 60s -o KbdInteractiveAuthentication = no -o
PreferredAuthentifying = gssapi-with-mic, gssapi-keyex
PasswordAuthentication = no -o User = fale -o ConnectTimeout = 10 -o
ControlPath = / home / fale / .ansible / cp / ansible-ssh-% C
test01.fale.io
```

```
'/ bin / sh -c' "'"' (ô 22 && mkdir -p "` echo $ HOME / .ansible / tmp /
ansible-tmp-1462644055.19-51001413558638 `" && echo "` echo
$ HOME / .ansible / tmp / ansible-tmp-1462644055.19-51001413558638 `"
) '"'" ''
<test01.fale.io> PUT / tmp / tmp9JSYiP ĐẾN /home/fale/.ansible/tmp/
ansible-tmp-1462644055.19-51001413558638 / năm
<test01.fale.io> SSH: EXEC sftp -b - -C -o ControlMaster = auto -o
ControlPersist = 60s -o KbdInteractiveAuthentication = no -o
PreferredAuthentuggest = gssapi-with-mic, gssapi-keyex, hostbasing, publi
ckey -o Mật khẩu xác thực = không -o Người dùng = fale -o ConnectTimeout
= 10
-o ControlPath = / home / fale / .ansible / cp / ansible-ssh-% C
'[test01.fale.io]'
<test01.fale.io> THÀNH LẬP KẾT NỐI SSH CHO NGƯỜI DÙNG: fale
<test01.fale.io> SSH: EXEC ssh -C -q -o ControlMaster = auto -o
ControlPersist = 60s -o KbdInteractiveAuthentication = no -o
PreferredAuthentuggest = gssapi-with-mic, gssapi-keyex, hostbasing, publi
ckey -o Mật khẩu xác thực = không -o Người dùng = fale -o ConnectTimeout
= 10
-o ControlPath = / home / fale / .ansible / cp / ansible-ssh-% C -tt
test01.fale.io '/ bin / sh -c' "'"' sudo -H -S -n -u root / bin / sh -c
'' '' '' '' '' '' '' '' '' '
TRỞ THÀNH THÀNH CÔNG-axnwopicemeccmdhnlmhawtwlysgfgjc; LANG = en_US.utf8
LC_ALL = en_US.utf8 LC_MESSAGES = en_US.utf8 / usr / bin / python -tt /
home /
fale / .ansible / tmp / ansible-tmp-1462644055.19-51001413558638 / yum;
rm
-rf "/home/fale/.ansible/tmp/
ansible-tmp-1462644055.19-51001413558638 / "> / dev / null
2> & 1 '"'" "" "" "" "" "" "" "" "" "
ok: [test01.fale.io] => {"đã thay đổi": sai, "lời mời":
{"module_args": {"conf_file": null, "eac_gpg_check": false,
"disablerepo": null, "enablerepo": null, "loại trừ": null,
"install_Vpoquery": true, "list": null, "name": ["httpd"], "state":
"hiện tại", "update_cache": false}, "module_name": "yum"}, "dir": "",
"RC": 0, "kết quả": ["httpd-2.4.6-40.el7.centos.x86_64 cung cấp
httpd đã được cài đặt"]}
```

Các biến trong playbooks

Đôi khi, điều quan trọng là `thiết lập` và `nhận` các biến trong một playbook.

Rất thường xuyên, bạn sẽ cần tự động hóa nhiều hoạt động tương tự. Trong những trường hợp đó, bạn sẽ muốn tạo một playbook duy nhất có thể được gọi với các biến khác nhau để đảm bảo khả năng sử dụng lại mã.

Một trường hợp khác mà các biến rất quan trọng là khi bạn có nhiều hơn một trung tâm dữ liệu và một số giá trị sẽ dành riêng cho trung tâm dữ liệu. Một ví dụ phổ biến là các máy chủ DNS. Hãy phân tích mã đơn giản sau đây sẽ giới thiệu cho chúng ta cách Ansible để đặt và nhận các biến:

```
---
```

```
- máy chủ: tất cả
  remote_user: nhiệm
  vụ fale:
  - name: Đặt biến 'name'
    set_fact:
      tên: Máy kiểm tra
  - name: Gỡ lỗi biến 'name':
      tin nhắn: '{{name}}'
```

Hãy chạy nó theo cách thông thường:

```
$ ansible-playbook -i test01.fale.io, biến.yaml
```

Bạn sẽ thấy kết quả sau:

```
Chơi tất cả] ********************************************** ****** / TÌM
NHIỆM VỤ [thiết lập] ****** /
4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8 ******
ok: [test01.fale.io]


NHIỆM VỤ [Đặt tên 'biến'] ****** / TÌM HIỂU ok: [test01.fale.io]

NHIỆM VỤ [Tên biến 'in'] ****** / TÌM HIỂU
     ok: [test01.fale.io] => {
         "tin nhắn": "Máy thử"
}


CHƠI RECAP ****** TÌM HIỂU ********** test01.fale.io: ok = 3 đã thay đổi
= 0 không thể truy cập = 0 thất bại = 0
```

Nếu chúng ta phân tích mã mà chúng ta vừa thực hiện, thì rõ ràng điều gì đang xảy ra. Chúng tôi thiết lập một biến (mà trong Ansible được gọi là `sự kiện`) và sau đó chúng ta in nó với `debug` chức năng.

tiền boa

Các biến phải luôn nằm giữa các trích dẫn khi bạn sử dụng phiên bản YAML mở rộng này.

Ansible cho phép bạn đặt các biến của mình theo nhiều cách khác nhau, nghĩa là bằng cách chuyển một tệp biến, khai báo nó trong một playbook, chuyển nó tới lệnh `ansible-playbook` bằng cách sử dụng `-e` / `--extra-vars` hoặc bằng cách khai báo nó trong một tệp kiểm kê (chúng ta sẽ thảo luận sâu hơn về vấn đề này trong chương tiếp theo).

Bây giờ là lúc bắt đầu sử dụng một số siêu dữ liệu mà Ansible thu được trong giai đoạn thiết lập. Hãy bắt đầu bằng cách xem dữ liệu được thu thập bởi Ansible. Để làm điều này, chúng tôi sẽ thực hiện:

```
$ ansible all -i HOST, -m setup
```

Trong trường hợp cụ thể của chúng tôi, điều này có nghĩa là thực hiện như sau:

```
$ ansible all -i test01.fale.io, -m thiết lập
```

Rõ ràng chúng ta có thể làm tương tự với một cuốn sách, nhưng cách này nhanh hơn. Ngoài ra, đối với trường hợp "thiết lập", bạn sẽ chỉ cần xem đầu ra trong quá trình phát triển để chắc chắn sử dụng đúng tên biến cho mục tiêu của mình.

Đầu ra sẽ giống như thế này:


```
test01.fale.io | THÀNH CÔNG => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "178,62.36.208",
            "10.16.0.7"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80 :: 601: e2ff: fef1: 1301"
        ],
        "ansible_arch architecture": "x86_64",
        "ansible_bios_date": "25/11/2016",
        "ansible_bios_version": "20160425",
        "ansible_cmdline": {
            "ro": đúng,
            "root": "LABEL = DOROOT"
        },
        "ansible_date_time": {
            "ngày": "2016-05-14",
            "ngày": "14",
            "kỷ nguyên": "1463244633",
            "giờ": "12",

          "Iso8601": "2016-05-14T16: 50: 33Z",
          "iso8601_basic": "20160514T125033231663",
          "iso8601_basic_short": "20160514T125033",
          "iso8601_micro": "2016-05-14T16: 50: 33,231770Z",
          "phút": "50",
          "tháng": "05",
          "Thứ hai": "33",
          "thời gian": "12:50:33",
          "tz": "EDT",
          "tz_offset": "-0400",
          "ngày trong tuần": "thứ bảy",
          "weekday_number": "6",
          "số tuần": "19",
          "năm": "2016"
        },
        "ansible_default_ipv4": {
            "địa chỉ": "178.62.36.208",
            "bí danh": "eth0",
            "phát sóng": "178.62.63.255",
            "cổng": "178.62.0.1",
            "giao diện": "eth0",
            "Macaddress": "04: 01: e2: F1: 13: 01",
            "mtu": 1500,
            "netmask": "255.255.192.0",
            "mạng": "178.62.0.0",
            "loại": "ether"
        },
```

```
        "ansible_default_ipv6": {},
        "ansible_devices": {
            "vda": {
                "người nắm giữ": [],
                "chủ nhà": "",
                "mô hình": null,
                "phân vùng": {
                    "vda1": {
                        "Các ngành": "41943040",
                        "phân vùng": 512,
                        "kích thước": "20,00 GB",
                        "bắt đầu": "2048"
                    }
                },
                "Có thể tháo rời": "0",
                "Xoay": "1",
                "calendaruler_mode": "",
                "ngành": "41947136",
                "phân vùng": "512",
                "kích thước": "20,00 GB",

                "support_discard": "0",
                "nhà cung cấp": "0x1af4"
            }
        },
        "ansible_distribution": "CentOS",
        "ansible_distribution_major_version": "7",
        "ansible_distribution_release": "Lõi",
        "ansible_distribution_version": "7.2.1511",
        "ansible_dns": {
            "máy chủ tên": [
                "8.8.8.8",
                "8.8.4.4"
            ]
        },
        "ansible_domain": "",
        "ansible_env": {
            "TRANG CHỦ": "/ nhà / fale",
            "LANG": "en_US.utf8",
            "LC_ALL": "en_US.utf8",
            "LC_MESSAGES": "en_US.utf8",
            "BÀI HỌC": "|| /usr/bin/lesspipe.sh% s",
            "LOGNAME": "nhạt nhẽo",
            "MAIL": "/ var / mail / fale",
            "PATH": "/ usr / local / bin: / usr / bin",
            "NKT": "/ nhà / fale",
            "CHIA SẺ": "/ bin / bash",
            "SHLVL": "2",
            "SSH_CLIENT": "86.187.141,39 37764 22",
            "SSH_CONNMENT": "86.187.141,39 37764 178.62.36.208 22",
```

```
        "SSH_TTY": "/ dev / pts / 0",
        "HẠN": "rxvt-unicode-256color",
        "NGƯỜI DÙNG": "nhạt nhẽo",
        "XDG_RUNTIME_DIR": "/ run / user / 1000",
        "XDG_SESSION_ID": "180",
        "_": "/ usr / bin / trăn"
    },
    "ansible_eth0": {
        "hoạt động": đúng,
        "thiết bị": "eth0",
        "ipv4": {
            "địa chỉ": "178.62.36.208",
            "phát sóng": "178.62.63.255",
            "netmask": "255.255.192.0",
            "mạng": "178.62.0.0"
        },
        "ipv4_secondaries": [
            {
                "địa chỉ": "10.16.0.7",
                "Phát sóng": "10.16.255.255",
                "netmask": "255.255.0.0",
                "mạng": "10.16.0.0"
            }
        ],
        "ipv6": [
            {
                "địa chỉ": "fe80 :: 601: e2ff: fef1: 1301",
                "tiền tố": "64",
                "phạm vi": "liên kết"
            }
        ],
        "Macaddress": "04: 01: e2: F1: 13: 01",
        "mô-đun": "virtio_net",
        "mtu": 1500,
        "pciid": "virtio0",
        "lăng nhăng": sai,
        "loại": "ether"
    },
    "ansible_eth1": {
        "hoạt động": sai,
        "thiết bị": "eth1",
        "Macaddress": "04: 01: e2: F1: 13: 02",
        "mô-đun": "virtio_net",
        "mtu": 1500,
        "pciid": "virtio1",
        "lăng nhăng": sai,
        "loại": "ether"
    },
    "ansible_fips": sai,
```

```
"ansible_form_factor": "Khác",
"ansible_fqdn": "kiểm tra",
"ansible_hostname": "kiểm tra",
"ansible_interfaces": [
    "lo",
    "eth1",
    "eth0"
],
"Ansible_kernel": "3.10.0-327.10.1.el7.x86_64",
"ansible_lo": {
    "hoạt động": đúng,
    "thiết bị": "lo",
    "ipv4": {
        "địa chỉ": "127.0.0.1",
        "phát sóng": "máy chủ",
        "netmask": "255.0.0.0",

        "mạng": "127.0.0.0"
    },
    "ipv6": [
        {
            "địa chỉ 1",
            "tiền tố": "128",
            "phạm vi": "máy chủ"
        }
    ],
    "mtu": 65536,
    "lăng nhăng": sai,
    "loại": "loopback"
},
"ansible_machine": "x86_64",
"ansible_machine_id": "fd8cf26e06e411e4a9d004010897bd01",
"ansible_memfree_mb": 6,
"ansible_memory_mb": {
    "Nocache": {
        "miễn phí": 381,
        "đã sử dụng": 108
    },
    "thực": {
        "miễn phí": 6,
        "tổng cộng": 489,
        "đã sử dụng": 483
    },
    "hoán đổi": {
        "lưu trữ": 0,
        "miễn phí": 0,
        "tổng cộng": 0,
        "đã sử dụng": 0
    }
},
```

```
        "ansible_memtotal_mb": 489,
        "ansible_mounts": [
            {
                "thiết bị": "/ dev / vda1",
                "fstype": "ext4",
                "gắn kết": "/",
                "tùy chọn": "rw, relatime, data = order",
                "size_av Available": 18368385024,
                "size_total": 21004894208,
                "uuid": "c5845b43-fe98-499a-bf31-4eccae14261b"
            }
        ],
        "ansible_nodename": "kiểm tra",
        "ansible_os_f Family": "RedHat",

          "ansible_pkg_mgr": "yum",
          "ansible_ Processor": [
              "Chính hãng",
               "CPU Intel (R) Xeon (R) E5-2630L v2 @ 2.40GHz"
          ],
          "ansible_ Processor_cores": 1,
          "ansible_ Processor_count": 1,
          "ansible_ Processor_threads_per_core": 1,
          "ansible_ Processor_vcpus": 1,
          "ansible_product_name": "Giọt nước",
          "ansible_product_serial": "NA",
          "ansible_product_uuid": "NA",
          "ansible_product_version": "20160415",
          "ansible_python_version": "2.7.5",
          "ansible_selinux": {
              "trạng thái": "bị vô hiệu hóa"
          },
          "ansible_service_mgr": "systemd",
          "ansible_ssh_host_key_dsa_public":
"AAAAB3NzaC1kc3MAAACBAPEf4dzeET6ukHemTASsamoRLxo2R8iHg5J1bYQUyuggtRKl
bRrHMtpQ8qN5CQNtp8J + 2Hq6 / JKiDF + cdxgOehf9b7F4araVvJxqx967RvLNBrMWXv7
/
4hi + efgXG9eejGoGQNAD66up /
fkLMd0L8fwSwmTJoZXwOxFwcbnxCZsFAAAAFQDgK7fka + 1AKjYZNFIfCB2b0ZitGQAAAI
ADeofiC5q + SLgEvkBCVELyJ + EVb6WHeHbVdrpE2GdnUr03R6MmmYhYZMijruS /
RCpzBLmi8juDkqAWy6Xqxd + DwixykntXPeUFS3F7LK5vNwFalaRltPwr4Azh + EeSUQ2Zz
2AdKx6zSqtLOD8ZMPkRDvz4WGHGmeR + i7UFsFDZdgAAAIEAy26Tx0jAlY3mEaTW9lQ9Do
GXgPBxsSX /
XqeLh5wBaBO6AJaIrs0dQJdNeHcMhFy0seVkOMN1SpeoBTJSoTOx15HAGsKsAcmnA5mcJ
eUZqptVR6JxROztHw3zQePQ3 /
V3KQzAN31tIm3PbKztlEZbXRUM7RV5WsdRHTb8rutENhY = ","
          ansible_ssh_host_key_ecdsa_public ":
"AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBPDXQ9rjgDmUKsEW
H4U2vg4iqtK + 75urlj9nwW + rN
tJnIcJ1hINdrWrZNpn4 = ",
          "ansible_ssh_host_key_rsa_public":
```

```
"AAAAB3NzaC1yc2EAAAADAQABAAABAQCwQx5EElH7FeD / agB /
gCJfBUEVhk44tldzdEzwc2IEbI59relTGNOU7soCCMcSH7nwlEbOOvmLa2R / YaXdHv /
cb1aXBC / wj /
m4ZHylBeF5qzECUkeaB3 + CT + hp8qHHApclFr2lm2CwZ +
YXjEyjJ3en4K3gLlIQyQjgE2F
57kmD1FVVDSJFvNTn + NQvb3DPppND + HKEeHwrJ0GgznoP62yobEgriAIBSGf // 0WHCO
/
9shEvauoRpPM + U9pU7lv637s7qyubIqyrs5fz3u34qBj8oCATOefRN1wsfJDeMG0D5ryI
6BI6t / eAi8BPr7VHJSQBk + buM9Jr1yoMQTEasq2J ",
        "ansible_swapfree_mb": 0,
        "ansible_swaptotal_mb": 0,
        "Ansible_system": "Linux",
        "ansible_system_vendor": "DigitalOcean",
        "ansible_uptime_seconds": 603067,
```



```
}
    "ansible_user_dir": "/ home / fale",
    "ansible_user_gecos": "",
    "ansible_user_gid": 1000,
    "ansible_user_id": "fale",
    "ansible_user_shell": "/ bin / bash",
    "ansible_user_uid": 1000,
    "ansible_userspace_arch architecture": "x86_64",
    "ansible_userspace_bits": "64",
    "ansible_virtualization_role": "máy chủ",
    "ansible_virtualization_type": "kvm",
    "module_setup": đúng
},
"đã thay đổi": sai
```


Như bạn có thể thấy, từ danh sách tùy chọn khổng lồ này, bạn có thể thu được một lượng thông tin khổng lồ và bạn có thể sử dụng chúng như bất kỳ biến nào khác. Hãy in tên hệ điều hành và phiên bản. Để làm như vậy, chúng ta có thể tạo một playbook mới có tên `setup_variables.yaml` với nội dung sau:

```
---
- máy chủ: tất cả
  remote_user: fale
  nhiệm vụ:
```

```
  - tên: In hệ điều hành và gỡ
    lỗi phiên bản:
      tin nhắn: '{{ansible_distribution}} {{
ansible_distribution_version}} '
```

Chạy nó với những điều sau đây:

```
$ ansible-playbook -itest01.fale.io, setup_variables.yaml
```

Điều này sẽ cung cấp cho chúng tôi đầu ra sau đây:

```
Chơi tất cả] ********************************************** ******
NHIỆM VỤ [thiết lập] ****** /
4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8 ******
ok: [test01.fale.io]


NHIỆM VỤ [In HĐH và phiên bản] ****** / TÌM HIỂU
ok: [test01.fale.io] => {
    "tin nhắn": "CentOS 7.2.1511"

}

CHƠI RECAP ****** TÌM HIỂU ******
test01.fale.io : ok = 2 đã thay đổi = 0 không thể truy cập = 0 thất bại =
0
```

Như bạn có thể thấy, nó đã in tên và phiên bản hệ điều hành, như mong đợi. Ngoài các phương thức đã thấy trước đây, cũng có thể truyền một biến bằng cách sử dụng đối số dòng lệnh. Trên thực tế, nếu chúng ta tìm kiếm sự trợ giúp của Ansible, chúng ta sẽ nhận thấy những điều sau:

```
-e EXTRA_VARS, --extra-vars = EXTRA_VARS
đặt các biến bổ sung là key = value hoặc YAML / JSON
```

Các dòng tương tự cũng có trong lệnh `ansible-playbook` . Chúng ta hãy tạo một cuốn sách nhỏ gọi là `cli_variabled.yaml` với nội dung sau:

```
---
- máy chủ: tất cả
  remote_user: nhiệm
  vụ fale:
  - name: Gỡ lỗi biến 'name':
      tin nhắn: '{{name}}'
```

Thực hiện nó với những điều sau đây:

```
$ ansible-playbook -i test01.fale.io, cli_variables.yaml -e
'name = test01'
```

Chúng tôi sẽ nhận được như sau:

```
Chơi tất cả] ********************************************** ******
NHIỆM VỤ [thiết lập] ****** /
4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8 ******
ok: [test01.fale.io]
NHIỆM VỤ [In biến 'tên'] ****** / TÌM HIỂU
ok: [test01.fale.io] => {
```

```
    "tin nhắn": "test01"
}
CHƠI RECAP ****** TÌM HIỂU ******
test01.fale.io : ok = 2 đã thay đổi = 0 không thể truy cập = 0 thất bại =
0
```

Trong trường hợp chúng tôi quên thêm tham số bổ sung để chỉ định biến, chúng tôi sẽ thực hiện nó dưới dạng:

```
$ ansible-playbook -i test01.fale.io, cli_variables.yaml
```

Chúng tôi đã nhận được đầu ra sau đây:

```
Chơi tất cả] ********************************************** ******
NHIỆM VỤ [thiết lập] ****** /
4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8 ******
ok: [test01.fale.io]

NHIỆM VỤ [In biến 'tên'] ****** / TÌM HIỂU

gây tử vong: [test01.fale.io]: KHÔNG THỂ! => {"fail": true, "dir":

"'name' không xác định"}


KHÔNG CÓ NHIỀU GIỜ NÀO CÒN HẤP DẪN ****** / TÌM HIỂU * để thử lại, sử

dụng: --limit @ cli_variables.retry

CHƠI RECAP ****** TÌM HIỂU ******
test01.fale.io : ok = 1 đã thay đổi = 0 không thể truy cập = 0 thất bại =
1
```

Bây giờ chúng ta đã học được những điều cơ bản của playbooks, hãy tạo một máy chủ web từ đầu bằng cách sử dụng chúng. Để làm như vậy, hãy bắt đầu lại từ đầu, tạo người dùng Ansible và sau đó tiến lên từ đó.

Tạo người dùng Ansible

Khi bạn tạo một máy (hoặc thuê một máy từ bất kỳ công ty lưu trữ nào), nó chỉ đến với người dùng `root` . Hãy bắt đầu tạo một playbook để đảm bảo rằng người dùng Ansible được tạo, nó có thể truy cập bằng khóa SSH và có thể thực hiện các hành động thay mặt cho những người dùng khác (`sudo`) mà không cần hỏi mật khẩu. Tôi thường gọi playbook này là `Firstrun.yaml` vì tôi thực thi nó ngay khi tạo ra một máy mới, nhưng sau đó, tôi không sử dụng nó vì nó sử dụng người dùng root mà tôi vô hiệu hóa vì lý do bảo mật. Kịch bản của chúng tôi sẽ trông giống như sau:

```
---
- máy chủ: tất cả
  người dùng:
  tác vụ gốc:
  - name: Đảm bảo người dùng tồn tại
    người dùng tồn tại:
      tên: ansible
      nhà nước: hiện tại
      bình luận: Ansible
  - name: Đảm bảo người dùng có thể chấp nhận khóa
    SSH ủy quyền_key:
      người dùng: ansible
      khóa: https://github.com/fale.keys
```

```
    nhà nước: hiện tại
  - name: Đảm bảo người dùng ansible là sudoer không yêu cầu
mật khẩu
    đường truyền:
      mệnh: / etc / sudoers
      nhà nước: hiện tại
      regrec: '^ ansible ALL \ ='
      dòng: 'ansible ALL = (ALL) NOPASSWD: ALL'
      xác thực: 'visudo -cf% s'
```

Trước khi chạy nó, hãy nhìn vào nó một chút. Chúng tôi đã sử dụng ba mô-đun khác nhau (`người sử dụng` , `authorized_key` , và `lineinfile`) mà chúng ta chưa bao giờ thấy. Các `sử dụng` mô-đun, như tên gọi gợi ý, cho phép chúng tôi để đảm bảo người dùng có mặt (hoặc không có).

Các `authorized_key` mô-đun cho phép chúng tôi để đảm bảo rằng một chìa khóa SSH nhất định có thể được sử dụng để đăng nhập như một người dùng cụ thể trên máy tính đó. Mô-đun này sẽ không thay thế tất cả các khóa SSH đã được bật cho người dùng đó, nhưng sẽ chỉ thêm (hoặc xóa) khóa được chỉ định. Nếu bạn muốn thay đổi hành vi này, bạn có thể sử dụng tùy chọn *độc quyền* , cho phép bạn xóa tất cả các khóa SSH không được chỉ định trong bước này.

Các `lineinfile` mô-đun cho phép chúng ta thay đổi nội dung của một tập tin. Nó hoạt động theo cách rất giống với **sed** (trình chỉnh sửa luồng), trong đó bạn chỉ định biểu thức chính quy sẽ được sử dụng để khớp với dòng và sau đó chỉ định dòng mới sẽ được sử dụng để thay thế dòng phù hợp. Nếu không có dòng nào khớp, dòng được thêm vào cuối tập tin. Bây giờ hãy chạy nó với:

```
$ ansible-playbook -i test01.fale.io, Firstrun.yaml
```

Điều này sẽ cho chúng ta kết quả như sau:

```
Chơi tất cả] ********************************************** ******
NHIỆM VỤ [thiết lập] ****** /
4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8 ******
ok: [test01.fale.io]


NHIỆM VỤ [Đảm bảo người dùng có thể tồn tại] ****** / TÌM HIỂU đã thay
đổi: [test01.fale.io]

NHIỆM VỤ [Đảm bảo người dùng có thể chấp nhận khóa SSH] ****** / TÌM
HIỂU đã thay đổi: [test01.fale.io]

NHIỆM VỤ [Đảm bảo người dùng anisble là sudoer không cần mật khẩu] * đã
thay đổi: [test01.fale.io]

CHƠI RECAP ****** TÌM HIỂU ******
test01.fale.io : ok = 4 đã thay đổi = 3 không thể truy cập = 0 thất bại =
0
```

Cấu hình một máy chủ cơ bản

Sau khi chúng tôi tạo người dùng cho Ansible với các đặc quyền cần thiết, chúng tôi có thể tiếp tục thực hiện một số thay đổi nhỏ khác cho HĐH. Để làm cho rõ ràng hơn, chúng ta sẽ xem từng hành động được thực hiện và sau đó chúng ta sẽ xem xét toàn bộ vở kịch.

Kích hoạt EPEL

EPEL là kho lưu trữ quan trọng nhất cho Enterprise Linux và nó chứa rất nhiều gói bổ sung. Đây cũng là kho lưu trữ an toàn vì không có gói nào trong EPEL sẽ xung đột với các gói trong kho lưu trữ cơ sở. Để kích hoạt EPEL trong RHEL / CentOS 7, chỉ cần cài đặt gói `phát hành epel` là đủ . Để làm như vậy trong Ansible, chúng tôi sẽ sử dụng:

```
- tên: Đảm bảo EPEL được bật
  yum:
    tên: phát hành epel
    trạng thái: hiện
  tại trở thành: Đúng
```

Như bạn có thể thấy, chúng tôi đã sử dụng mô-đun `yum` , như chúng tôi đã làm trong một trong những ví dụ đầu tiên của chương, chỉ định tên của gói và chúng tôi muốn nó có mặt.

Cài đặt các ràng buộc Python cho SELinux

Vì Ansible được viết bằng Python và chủ yếu sử dụng các liên kết Python để hoạt động trên hệ điều hành, nên chúng tôi sẽ cần cài đặt các liên kết Python cho SELinux:

```
- tên: Đảm bảo libselinux-python có mặt yum:
    tên: libselinux-trăn
    trạng thái: hiện
  tại trở thành: Đúng
- tên: Đảm bảo libsemanage-python có mặt yum:
    tên: libsemanage-python
    trạng thái: hiện
  tại trở thành: Đúng
```

tiền boa

Điều này có thể được viết theo cách ngắn hơn, bằng cách sử dụng một chu kỳ, nhưng chúng ta sẽ xem cách làm như vậy trong chương tiếp theo.

Nâng cấp tất cả các gói đã cài đặt

Để nâng cấp tất cả các gói đã cài đặt, chúng tôi sẽ cần sử dụng lại mô-đun `yum` , nhưng với một tham số khác, trên thực tế, chúng tôi sẽ sử dụng:

```
-  Tên: Đảm bảo chúng tôi có phiên bản cuối cùng của
  mọi gói yum:
    Tên: "*"
    trạng thái: mới
  nhất trở thành:
  Đúng
```

Như bạn có thể thấy, chúng ta đã xác định " `*` " là tên gói (điều này tượng trưng cho một ký tự đại diện để phù hợp với tất cả các gói cài đặt) và `nhà nước` là `mới nhất` . Điều này sẽ nâng cấp tất cả các gói đã cài đặt lên phiên bản mới nhất hiện có.

Nếu bạn còn nhớ, khi chúng tôi nói về trạng thái " `hiện tại` ", chúng tôi đã nói rằng nó sẽ cài đặt phiên bản có sẵn cuối cùng. Vậy sự khác biệt giữa " `hiện tại` " và " `mới nhất` " là gì? Present sẽ cài đặt phiên bản mới nhất nếu gói chưa được cài đặt, trong khi nếu gói đã được cài đặt (không có vấn đề gì với phiên bản), nó sẽ tiếp tục mà không thực hiện bất kỳ thay đổi nào. Mới nhất sẽ cài đặt phiên bản mới nhất nếu gói chưa được cài đặt, trong khi nếu gói đã được cài đặt sẽ kiểm tra xem có phiên bản mới hơn không và nếu có, Ansible sẽ cập nhật gói.

Đảm bảo rằng NTP được cài đặt, định cấu hình và chạy

Để đảm bảo NTP có mặt, chúng tôi sử dụng mô đun `yum` :

```
- tên: Đảm bảo NTP được cài đặt
  yum:
    tên: ntp
    trạng thái: hiện
  tại trở thành: Đúng
```

Bây giờ chúng tôi biết rằng NTP đã được cài đặt, chúng tôi nên đảm bảo rằng máy chủ đang sử dụng `múi giờ` mà chúng tôi muốn. Để làm như vậy, chúng ta sẽ tạo ra một liên kết tượng trưng trong `/ etc / localtime` rằng sẽ trỏ đến muốn `zoneinfo` file:

```
- name: Đảm bảo múi giờ được đặt thành tệp
  UTC:
    src: / usr / share / areainfo / GMT
    mệnh: / etc / giờ địa phương
    trạng thái:
  liên kết trở
  thành: Đúng
```

Như bạn có thể thấy, chúng tôi đã sử dụng mô-đun `tệp` để nói với Ansible, chỉ định rằng nó cần phải là một liên kết (`trạng thái: liên kết`).

Để hoàn tất cấu hình NTP, chúng tôi cần khởi động dịch vụ `ntpd` và đảm bảo rằng nó sẽ chạy ở mọi khởi động, do đó:

```
- name: Đảm bảo dịch vụ NTP đang chạy và kích hoạt dịch
  vụ:
    tên: ntpd
    bang: đã bắt đầu
    đã bật: Đúng
  trở thành sự thật
```

Đảm bảo rằng FirewallD có mặt và được kích hoạt

Như bạn có thể tưởng tượng, bước đầu tiên là đảm bảo rằng FirewallD được cài đặt:

```
- tên: Đảm bảo FirewallD được cài đặt
  yum:
    tên: tường lửa
    trạng thái: hiện
  tại trở thành: Đúng
```

Vì chúng tôi muốn chắc chắn rằng, khi chúng tôi kích hoạt FirewallD, chúng tôi sẽ không mất kết nối SSH, chúng tôi đảm bảo rằng lưu lượng SSH luôn có thể đi qua nó:

```
- name: Đảm bảo SSH có thể vượt qua tường
  lửa tường lửa:
    dịch vụ: ssh
    trạng thái: đã bật
    vĩnh viễn: Đúng
    Ngay lập tức: Đúng
  trở thành: Đúng
```

Để làm như vậy, chúng tôi đã sử dụng mô-đun `tường lửa` . Mô-đun này sẽ lấy các tham số rất giống với tham số mà bảng điều khiển `tường lửa-cmd` sẽ sử dụng. Bạn sẽ phải chỉ định dịch vụ được phép vượt qua tường lửa, cho dù bạn muốn áp dụng quy tắc này ngay lập tức và liệu bạn có muốn quy tắc này là vĩnh viễn để sau khi khởi động lại quy tắc vẫn sẽ xuất hiện hay không.

tiền boa

Bạn có thể chỉ định tên dịch vụ (chẳng hạn như ' `ssh` ') bằng cách sử dụng tham số `dịch vụ` hoặc bạn có thể chỉ định cổng (chẳng hạn như '`22 / tcp` ') bằng tham số `cổng` .

Bây giờ chúng tôi đã cài đặt FirewallD và chúng tôi chắc chắn rằng kết nối SSH của chúng tôi sẽ tồn tại, chúng tôi có thể kích hoạt nó như bất kỳ dịch vụ nào khác:

```
- tên: Đảm bảo FirewallD đang chạy
  dịch vụ:
    tên: tường lửa
    bang: đã bắt đầu
    kích hoạt: True
  trở thành: Đúng
```

Thêm một MOTD tùy chỉnh

Để thêm MOTD, chúng ta sẽ cần một mẫu giống nhau cho tất cả các máy chủ và một tác vụ để sử dụng mẫu.

Tôi thấy rất hữu ích khi thêm một bộ GTVT cho mọi máy chủ. Nó thậm chí còn hữu ích hơn nếu bạn sử dụng Ansible, bởi vì bạn có thể sử dụng nó để cảnh báo người dùng của bạn rằng những thay đổi đối với hệ thống có thể bị ghi đè bởi Ansible. Mẫu thông thường của tôi được gọi là '`motd`' và có nội dung này:

```
                Hệ thống này được quản lý bởi Ansible
 Mọi thay đổi được thực hiện trên hệ thống này đều có thể được ghi đè
                            bởi Ansible

HĐH: {{ansible_distribution}} {{ansible_distribution_version}}
Tên máy chủ: {{Invent_hostname}}
địa chỉ eth0: {{ansible_eth0.ipv4.address}} Tất cả
các kết nối được theo dõi và ghi lại
    Ngắt kết nối NGAY LẬP TỨC nếu bạn không phải là người dùng được ủy
    quyền
```

Đây là một mẫu `jinja2` và nó cho phép chúng tôi sử dụng mọi biến được đặt trong các `vở kịch` . Điều này cũng cho phép chúng ta sử dụng cú pháp phức tạp cho các điều kiện và chu trình mà chúng ta sẽ thấy sau trong chương này. Để điền một tệp từ một mẫu trong Ansible, chúng ta sẽ cần sử dụng:

```
- name: Đảm bảo tệp MOTD có mặt và mẫu được cập nhật:
    src: motd
    mệnh: / etc / motd
    chủ sở hữu: root
    nhóm: gốc
    chế độ: 0644
  trở thành: Đúng
```

Mô-đun mẫu cho phép chúng tôi chỉ định một tệp cục bộ (`src`) sẽ được `jinja2` giải thích và đầu ra của thao tác này sẽ được lưu trên máy từ xa trong một đường dẫn cụ thể (`mệnh`), được sở hữu bởi một người dùng (`chủ sở hữu`) cụ thể và nhóm (`nhóm`) và có chế độ truy cập cụ thể (`chế độ`).

Thay đổi tên máy chủ

Để giữ cho mọi thứ đơn giản, một cách tôi thấy hữu ích là đặt tên máy chủ của máy thành một cái gì đó có ý nghĩa. Để làm như vậy, chúng ta có thể sử dụng một mô-đun Ansible rất đơn giản được gọi là `tên máy chủ` :

```
- name: Đảm bảo tên máy chủ giống với tên máy chủ kho:
    tên: "{{Invent_hostname}}" trở
  thành: Đúng
```

Xem lại và chạy playbook

Đặt mọi thứ lại với nhau, bây giờ chúng ta có playbook sau (được gọi là `common_t Nhiệm.yaml` để đơn giản):

```
---
- máy chủ: tất cả
  remote_user: nhiệm vụ
  ansible:
  - tên: Đảm bảo EPEL được bật yum:
      tên: phát hành epel
      trạng thái: hiện
    tại trở thành: Đúng
  - tên: Đảm bảo libselinux-python có mặt yum:
      tên: libselinux-trăn
      trạng thái: hiện
    tại trở thành: Đúng
  - tên: Đảm bảo libsemanage-python có mặt yum:
      tên: libsemanage-python
      trạng thái: hiện
    tại trở thành: Đúng
  - tên: Đảm bảo chúng tôi có phiên bản cuối cùng của mọi
    gói yum:
      Tên: "*"
      trạng thái: mới
    nhất trở thành: Đúng
  - tên: Đảm bảo NTP được cài đặt
    yum:
      tên: ntp
      trạng thái: hiện
    tại trở thành: Đúng
  - name: Đảm bảo múi giờ được đặt thành tệp
    UTC:
      src: / usr / share / areainfo / GMT
      mệnh: / etc / giờ địa phương
      trạng thái: liên
    kết trở thành:
    Đúng
  - name: Đảm bảo dịch vụ NTP đang chạy và kích hoạt dịch
    vụ:
      tên: ntpd
      bang: đã bắt đầu
      kích hoạt: True
    trở thành: Đúng
```

```
  - tên: Đảm bảo Tường lửa được cài đặt

    yum
      tên: tường lửa
      nhà nước: hiện tại
    trở thành sự thật
  - tên: Đảm bảo FirewallD đang chạy dịch
    vụ:
      tên: tường lửa
      bang: đã bắt đầu
      kích hoạt: True trở
    thành: Đúng
  - name: Đảm bảo SSH có thể vượt qua tường lửa
    tường lửa:
      dịch vụ: ssh
      trạng thái: đã bật
      vĩnh viễn: Đúng
      Ngay lập tức: Đúng
    trở thành: Đúng
  - name: Đảm bảo tệp MOTD có mặt và mẫu được cập nhật:
      src: motd
      mệnh: / etc / motd
      chủ sở hữu: root
      nhóm: gốc
      chế độ: 0644 trở
    thành: Đúng
  - name: Đảm bảo tên máy chủ giống với tên máy chủ kho:
      tên: "{{Invent_hostname}}" trở thành:
    Đúng
```

Vì `playbook` này khá phức tạp, chúng tôi có thể chạy như sau:

`$ ansible-playbook common_t Nhiệm.yaml --list-task`

Điều này yêu cầu Ansible in tất cả các tác vụ ở dạng ngắn hơn để chúng ta có thể nhanh chóng xem những nhiệm vụ mà một `cuốn sách` thực hiện. Đầu ra phải giống như sau:

```
playbook: common_t Nhiệm.yaml
  chơi số 1 (tất cả): tất cả TAG: []
    nhiệm vụ:
      Đảm bảo EPEL được bật TAGS: []
      Đảm bảo libselinux-python có mặt TAGS: []
      Đảm bảo libsemanage-python có mặt TAGS: []
      Đảm bảo chúng tôi có phiên bản cuối cùng của mỗi gói TAGS: []
      Đảm bảo NTP được cài đặt TAGS: []
      Đảm bảo múi giờ được đặt thành TAG UTC: []
      Đảm bảo dịch vụ NTP đang chạy và bật TAGS: []

      Đảm bảo FirewallD được cài đặt TAGS: []
      Đảm bảo FirewallD đang chạy TAGS: []
      Đảm bảo SSH có thể vượt qua TAGS tường lửa: []
      Đảm bảo tệp MOTD có mặt và được cập nhật TAGS: []
      Đảm bảo tên máy chủ giống với TAGS hàng tồn kho: []
```

Bây giờ chúng ta có thể chạy `playbook` với các mục sau:

```
$ ansible-playbook -itest01.fale.io, common_t Nhiệm.yaml
```

Chúng tôi sẽ nhận được đầu ra sau đây:

```
Chơi tất cả] ********************************************** ******

NHIỆM VỤ [thiết lập] ****** / 4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8
******
ok: [test01.fale.io]

NHIỆM VỤ [Đảm bảo EPEL được bật] ****** / TÌM HIỂU
đã thay đổi: [test01.fale.io]

NHIỆM VỤ [Đảm bảo libselinux-python có mặt] ****** / TÌM HIỂU
ok: [test01.fale.io]

NHIỆM VỤ [Esure libsemanage-python có mặt] ****** / TÌM HIỂU
ok: [test01.fale.io]

NHIỆM VỤ [Đảm bảo chúng tôi có phiên bản cuối cùng của mọi gói] ****** / TÌM
đã thay đổi: [test01.fale.io]

NHIỆM VỤ [Đảm bảo NTP đã được cài đặt] ****** / TÌM KIẾM
ok: [test01.fale.io]

NHIỆM VỤ [Đảm bảo múi giờ được đặt thành UTC] ****** / TÌM HIỂU
đã thay đổi: [test01.fale.io]

NHIỆM VỤ [Đảm bảo dịch vụ NTP đang chạy và được bật] ****** / TÌM
đã thay đổi: [test01.fale.io]

NHIỆM VỤ [Đảm bảo Tường lửa được cài đặt] ****** / TÌM HIỂU
ok: [test01.fale.io]

NHIỆM VỤ [Đảm bảo Tường lửa đang chạy] ****** / TÌM HIỂU
đã thay đổi: [test01.fale.io]

NHIỆM VỤ [Đảm bảo SSH có thể vượt qua tường lửa] ****** / TÌM HIỂU
ok: [test01.fale.io]

NHIỆM VỤ [Đảm bảo rằng tệp MOTD có mặt và được cập nhật] ****** TÌM
HIỂU đã thay đổi: [test01.fale.io]

NHIỆM VỤ [Đảm bảo tên máy chủ giống với hàng tồn kho] ********** đã
thay đổi: [test01.fale.io]

CHƠI RECAP ****** TÌM HIỂU ******
test01.fale.io : ok = 9 đã thay đổi = 7 không thể truy cập = 0 thất bại =
0
```

Cài đặt và cấu hình máy chủ web

Bây giờ chúng tôi đã thực hiện một số thay đổi chung cho hệ điều hành, hãy chuyển sang thực sự tạo một máy chủ web. Chúng tôi đang chia hai giai đoạn đó để có thể chia sẻ giai đoạn đầu tiên giữa mọi máy và chỉ

áp dụng giai đoạn thứ hai cho các máy chủ web.

Đối với giai đoạn thứ hai này, chúng tôi sẽ tạo ra một playbook mới gọi là `webserver.yaml` với nội dung sau:

```
---
- máy chủ: tất cả
  remote_user: nhiệm vụ
  ansible:
  - name: Đảm bảo gói HTTPd được cài đặt yum:
      tên: httpd
      trạng thái: hiện
    tại trở thành: Đúng
  - name: Đảm bảo dịch vụ HTTPd được bật và chạy dịch vụ:
      tên: httpd
      bang: đã bắt đầu
      kích hoạt: True
    trở thành: Đúng
  - name: Đảm bảo HTTP có thể vượt qua tường
    lửa tường lửa:
      dịch vụ: http
      trạng thái: đã bật
      vĩnh viễn: Đúng
      Ngay lập tức: Đúng
    trở thành: Đúng
  - tên: Đảm bảo HTTPS có thể vượt qua tường
    lửa tường lửa:
      dịch vụ: https
      trạng thái: đã bật
      vĩnh viễn: Đúng
      Ngay lập tức: Đúng
    trở thành: Đúng
```

Như bạn có thể thấy, hai nhiệm vụ đầu tiên giống như các nhiệm vụ trong ví dụ ở đầu chương này và hai nhiệm vụ cuối cùng được sử dụng để hướng dẫn FirewallD cho phép lưu lượng truy cập HTTP và HTTPS.

Hãy chạy đoạn script này như sau:

```
$ ansible-playbook -i test01.fale.io, webserver.yaml
```

Điều này dẫn đến kết quả như sau:

```
Chơi tất cả] ********************************************* ******
NHIỆM VỤ [thiết lập] ****** / 4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8
******
ok: [test01.fale.io]


NHIỆM VỤ [Đảm bảo gói HTTPd đã được cài đặt] ****** / TÌM HIỂU đã thay
đổi: [test01.fale.io]

NHIỆM VỤ [Đảm bảo dịch vụ HTTPd được bật và chạy] *********** đã thay
đổi: [test01.fale.io]
```

```
NHIỆM VỤ [Đảm bảo HTTP có thể vượt qua tường lửa] ****** / TÌM HIỂU đã
thay đổi: [test01.fale.io]

NHIỆM VỤ [Đảm bảo HTTPS có thể vượt qua tường lửa] ****** / TÌM HIỂU đã
thay đổi: [test01.fale.io]

CHƠI RECAP ****** TÌM HIỂU ******
test01.fale.io : ok = 5 thay đổi = 4 không thể truy cập = 0 thất bại =
0
```

Bây giờ chúng ta có một máy chủ web, hãy xuất bản một trang web tĩnh nhỏ một trang.

Xuất bản một trang web

Vì trang web của chúng tôi sẽ là một trang web đơn giản, đơn giản, chúng tôi có thể dễ dàng tạo và xuất bản nó bằng một tác vụ Ansible duy nhất. Để làm cho trang này thú vị hơn một chút, chúng tôi sẽ tạo nó từ một mẫu sẽ được Ansible điền với một ít dữ liệu về máy. Kịch bản để xuất bản nó sẽ được gọi là `triển khai_website.yaml` và sẽ có nội dung sau:

```
---
- máy chủ: tất cả
  remote_user: nhiệm vụ
  ansible:
  - tên: Đảm bảo trang web có mặt và cập nhật mẫu:
      src: index.html.j2
      mệnh: /var/www/html/index.html
      chủ sở hữu: root
      nhóm: gốc
      chế độ: 0644
    trở thành: Đúng
```

Hãy bắt đầu với một mẫu đơn giản mà chúng ta sẽ gọi `index.html.j2` :

```
<html>
    <cơ thể>
        <h1> Xin chào thế giới! </ h1>
    </ cơ thể>
</ html>
```

Bây giờ chúng tôi có thể kiểm tra việc triển khai trang web của mình bằng cách chạy như sau:

```
$ ansible-playbook -i test01.fale.io, triển khai_website.yaml
```

Chúng ta sẽ nhận được đầu ra sau:

```
Chơi tất cả] ********************************************** ******
NHIỆM VỤ [thiết lập] ****** /
4/4/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8/8 ******
ok: [test01.fale.io]


NHIỆM VỤ [Đảm bảo trang web có mặt và được cập nhật] ****** / TÌM
HIỂU đã thay đổi: [test01.fale.io]
```

```
CHƠI RECAP ****** TÌM HIỂU ******
test01.fale.io : ok = 2 đã thay đổi = 1 không thể truy cập = 0 thất bại =
0
```

Nếu bây giờ bạn truy cập IP / FQDN của máy kiểm tra trong trình duyệt của mình, bạn sẽ tìm thấy "Hello World!" trang.

Mẫu Jinja2

Jinja2 là một công cụ mẫu được sử dụng rộng rãi và đầy đủ tính năng cho Python. Hãy xem xét một số cú pháp sẽ giúp chúng ta với Ansible. Đoạn này không muốn thay thế cho tài liệu chính thức, nhưng mục tiêu của nó là dạy cho bạn một số thành phần mà bạn sẽ thấy rất hữu ích khi sử dụng chúng với Ansible.

Biến

Như chúng ta đã thấy, chúng ta có thể in nội dung biến chỉ bằng cú pháp ' `{{VARIABLE_NAME}}` '. Nếu chúng tôi muốn in chỉ một phần tử của một mảng, chúng tôi có thể sử dụng ' `{{ARRAY_NAME ['KEY']}}` ' và nếu chúng tôi muốn in một thuộc tính của một đối tượng, chúng tôi có thể sử dụng ' `{{OBJECT_NAME.PROPERTY_NAME}}` '.

Vì vậy, chúng tôi có thể cải thiện trang tĩnh trước đó theo cách sau:

```
<html>
    <cơ thể>
        <h1> Xin chào thế giới! </ h1>
        <p> Trang này được tạo trên {{ansible_date_time.date
}}. </ p>
    </ cơ thể>
</ html>
```

Bộ lọc

Thỉnh thoảng, chúng tôi có thể muốn thay đổi kiểu chuỗi một chút, mà không cần viết mã cụ thể cho chuỗi đó, ví dụ, chúng tôi có thể muốn viết hoa một số văn bản. Để làm như vậy, chúng tôi có thể sử dụng một trong các bộ lọc của Jinja2, chẳng hạn như: ' `{{VARIABLE_NAME | viết hoa}}` '. Có nhiều bộ lọc có sẵn cho Jinja2 và bạn có thể tìm thấy danh sách đầy đủ tại: http://jinja.pocoo.org/docs/dev/temsheet/#builtin-filters .

Điều kiện

Một điều bạn có thể thường thấy hữu ích trong một công cụ mẫu là khả năng in các chuỗi khác nhau tùy thuộc vào nội dung (hoặc sự tồn tại) của chuỗi. Vì vậy, chúng tôi có thể cải thiện trang web tĩnh của mình theo cách sau:

```
<html>
    <cơ thể>
        <h1> Xin chào thế giới! </ h1>
        <p> Trang này được tạo trên {{ansible_date_time.date
}}. </ p>
{% nếu ansible_eth0.active == True%}
        <p> eth0 địa chỉ {{ansible_eth0.ipv4.address}}. </ p>
{% endif%}
    </ cơ thể>
</ html>
```

Như bạn có thể thấy, chúng tôi đã thêm khả năng in địa chỉ IPv4 chính cho kết nối `eth0` , nếu kết nối đang `hoạt động` . Với các điều kiện chúng ta cũng có thể sử dụng các bài kiểm tra.

chú thích

Để biết danh sách đầy đủ, vui lòng tham khảo: http://jinja.pocoo.org/docs/dev/temsheet/#builtin-tests .

Vì vậy, để có được kết quả tương tự, chúng tôi cũng có thể viết như sau:

```
<html>
    <cơ thể>
        <h1> Xin chào thế giới! </ h1>
        <p> Trang này được tạo trên {{ansible_date_time.date
}}. </ p>
{% nếu ansible_eth0.active bằng True%}
        <p> eth0 địa chỉ {{ansible_eth0.ipv4.address}}. </ p>
{% endif%}
    </ cơ thể>
</ html>
```

Có rất nhiều bài kiểm tra khác nhau sẽ thực sự giúp bạn tạo ra các mẫu dễ đọc, hiệu quả.

Chu kỳ

Các `jinja2` mẫu hệ thống cũng cung cấp khả năng để tạo ra chu kỳ. Hãy thêm một tính năng vào trang của chúng tôi sẽ in địa chỉ mạng IPv4 chính cho mỗi thiết bị thay vì chỉ `eth0` . Sau đó chúng tôi sẽ có mã sau đây:

```
<html>
    <cơ thể>
        <h1> Xin chào thế giới! </ h1>
        <p> Trang này được tạo trên {{ansible_date_time.date
}}. </ p>
        <p> Máy này có thể đạt được trên các địa chỉ IP sau
</ p>
        <ul>
  {% cho địa chỉ trong ansible_all_ipv4_addresses%}
             <li> {{địa chỉ}} </ li>
{% kết thúc%}
        </ ul>
    </ cơ thể>
</ html>
```

Như bạn có thể thấy, cú pháp cho các chu kỳ là quen thuộc nếu bạn đã biết Python.

Một vài trang trên Jinja2 templating không thể thay thế cho tài liệu chính thức. Trong thực tế, các mẫu Jinja2 mạnh hơn nhiều so với những gì chúng ta đã thấy ở đây. Mục tiêu ở đây chỉ là cung cấp cho bạn các mẫu Jinja2 cơ bản thường được sử dụng nhất trong Ansible.

Tóm lược

Trong chương này, chúng tôi đã bắt đầu xem xét YAML và xem playbook là gì, cách thức hoạt động và cách sử dụng nó để tạo một máy chủ web (và triển khai cho trang web tĩnh của bạn). Chúng tôi cũng đã thấy nhiều mô-đun Ansible như mô-đun người dùng, yum, dịch vụ, FirewalID, lineinfile và mẫu. Cuối chương, chúng tôi tập trung vào các mẫu.

Trong chương tiếp theo, chúng ta sẽ nói về hàng tồn kho để có thể dễ dàng quản lý nhiều máy.