LỜI CẢM ƠN

Trước khi trình bày nội dung chính của đồ án tốt nghiệp, em xin bày tỏ lòng biết ơn sâu sắc tới Thạc sỹ Nguyễn Nam Hưng - người đã tận tình hướng dẫn, chỉ bảo để em có thể hoàn thành đề tài này.

Em cũng xin bày tỏ lòng biết ơn chân thành tới toàn thể các thầy cô giáo trong khoa Công Nghệ Thông Tin, Trường Đại học Thủy Lợi đã dạy bảo em tận tình trong suốt quá trình học tập tại khoa.

Nhân dịp này em cũng xin được gửi lời cảm ơn chân thành tới gia đình, bạn bè đã luôn bên em, cổ vũ, động viên, giúp đỡ em trong suốt quá trình học tập và thực hiện đồ án tốt nghiệp.

Hà Nội, Ngày 2 tháng 1 năm 2014

Sinh viên

Dương Tiến Thuận

LỜI MỞ ĐẦU

Điện toán đám mây[1] không còn là một khái niệm mới ở Việt Nam cũng như trên thế giới. Mặc dù điện toán đám mây chỉ là một cách khác để cung cấp các tài nguyên máy tính, chứ không phải là một công nghệ mới, nhưng nó đã châm ngòi một cuộc cách mạng trong cách cung cấp thông tin và dịch vụ của các doanh nghiệp - tổ chức trên toàn thế giới.

Sự phát triển nhanh chóng và mạnh mẽ của điện toán đám mây đã đặt ra nhiều thách thức đối với các nhà cung cấp dịch vụ (cloud providers) cũng như các doanh nghiệp, tổ chức sử dụng nó. Áp lực đó không chỉ là cơ sở hạ tầng (hệ thống máy chủ, hệ thống mạng, trung tâm dữ liệu .v.v.) mà quan trọng hơn đó là yếu tố con người.

Đối với doanh nghiệp, điện đoán đám mây giúp giảm thiểu thời gian và chi phí đưa sản phẩm tới người dùng. Đối với quản trị hệ thống, điện toán đám mây mang lại khả năng mở rộng hệ thống một cách nhanh chóng, nhưng đồng thời nó cũng đặt ra thách thức trong việc quản lý hạ tầng máy chủ. Với số lượng hàng trăm server thì việc phải quản lý thủ công theo cách truyền thống là không hiệu quả và đem lại nguy cơ cao cho hệ thống. Từ yêu cầu thực tế đó, các framework cho việc tự động hóa hệ thống ra đời. Chúng cho

[1]Cloud Computing: https://en.wikipedia.org/wiki/Cloud_computing

phép người quản trị hệ thống có thể quản lý cấu hình của các máy chủ, đồng bộ hóa chúng với nhau, đồng thời đảm bảo việc nó không bị thay đổi sai bởi sự vô tình của con người. Không những thế, chúng còn cho phép người quản trị hệ thống triển khai ứng dụng một cách nhanh chóng tới nhiều môi trường và máy chủ khác nhau.

Các automation framework tiêu biểu được đề cập trong đồ án này bao gồm: Puppet, Chef và Ansible. Chúng đều là những sản phẩm mạnh mẽ và được sử dụng rộng rãi bởi nhiều doanh nghiệp, tổ chức lớn trên toàn thế giới.

Trong nội dung của đồ án này, em tập trung tìm hiểu cách sử dụng các automation framework và sau đó là ứng dụng chúng để tự động hóa các công việc liên quan đến hệ thống hạ tầng các sản phẩm công nghệ.

Hạ tầng sản phẩm công nghệ ở đây bao gồm:

- Hạ tầng máy chủ (Linux) cùng các cấu hình; các dịch vụ hệ thống như NTP, SSH, Cron, DHCP, DNS, .v.v.

- Các dịch vụ và ứng dụng phục vụ trực tiếp cho sản phẩm như webserver (apache, nginx), database (postgresql, mysql) .v.v.

Việc phân chia 2 tầng như trên chỉ mang tính chất tương đối để chúng ta có thể xác định cụ thể quyền hạn và trách nhiệm của 2 dạng công cụ sẽ được đề cập đến trong đồ án là công cụ triển khai sản phẩm (deployment tool) và công cụ quản lý cấu hình (configuration manager).

MỤC LỤC

DANH SÁCH HÌNH VẼ

DANH SÁCH BẢNG

DANH SÁCH MÃ NGUỒN

CHƯƠNG 1

TỔNG QUAN VỀ TỰ ĐỘNG HÓA HỆ THỐNG

1.1 Lý do cần tự động hóa hệ thống

Không phải ai cũng biết rằng, công việc của quản trị hệ thống thường xoay quanh một loạt các nhiệm vụ lặp đi lặp lại: cấu hình máy chủ, tạo người dùng, quản lý ứng dụng, dịch vụ và các chương trình chạy nền khác. Những công việc này thường được lặp đi lặp lại nhiều lần trong vòng đời của một máy chủ, từ lúc xây dựng hệ thống đến khi ngừng hoạt động. Nó bao gồm cả những việc như quản lý cấu hình trên nhiều cụm máy chủ khác nhau, đồng bộ hóa cấu hình giữa chúng theo chức năng; triển khai backend cho các sản phẩm; triển khai các sản phẩm trên các cụm máy chủ dịch vụ; dọn rác khi đĩa đầy, xóa log không dùng đến, xóa cache ... và đôi lúc là dọn "rác" của những người tiền nhiệm.

1.2 Các vấn đề nảy sinh

- Quản trị hệ thống họ cũng là con người, họ có những vấn đề của con người[1]. Điều nghiêm trọng nhất là quên và nhớ.

  Một người quản trị hệ thống cho dù giỏi đến đâu cũng có sẽ có lúc quên chính xác những gì mình làm ở trong hệ thống của mình. Nhất là khi rất nhiều người cùng quản trị một hệ thống, mà hệ thống đó lại không được quản lý tốt. Đó là sẽ một vũng lầy!

- Khi hệ thống đã là một vũng lầy, trên thực tế người ta sẽ đành chấp nhận nó. Nhưng sau một quá trình "chịu đựng", người ta sẽ tính đến chuyện đập đi làm lại, hay còn gọi là "migration". Cách giải quyết ở trong giai đoạn này thường chỉ mang tính chấp vá.

[1] http://en.wikipedia.org/wiki/Human_reliability

1.3 Cách giải quyết vấn đề

- **Chuẩn hóa quy trình làm việc**

  Điển hỉnh là các công ty Nhật, họ có một quy trình làm việc cực kì chặt chẽ, mọi thay đổi đều phải có quy trình và được kiểm soát nghiêm ngặt. Siết chặt các công đoạn làm việc, lịch sử thay đổi và công tác quản lý có thể sẽ hữu ích để giữ cho system không bị rối và luôn chạy ổn định.

- **Chấp nhận sống trong "vũng lầy"**.

  Một điều đáng ngạc nhiên là có rất nhiều công ty, tập đoàn từ công nghệ đến cả tài chính trên thế giới phải làm kiểu này. Hệ thống vẫn hoạt động được và vẫn kiếm ra của cải, rất khó để có thể đưa một thay đổi đáng kể nào vào hệ thống đó. Người ta lúc này buộc lòng phải chấp nhận sống trong "vũng lầy"

- **Tự động hóa hệ thống**

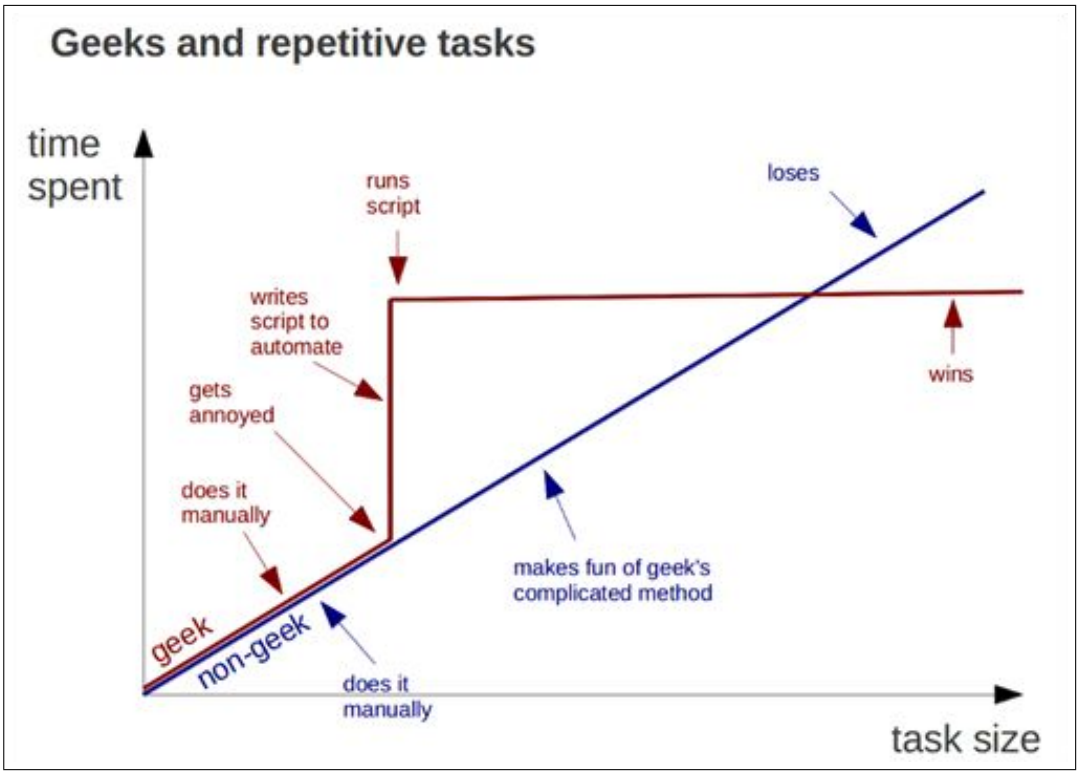
  Đây là hướng mà đồ án này nói tới. Tuy việc tự động hóa hệ thống này đã được ứng dụng rộng rãi từ lâu nhưng nó vẫn là tập hợp của những công nghệ bleeding-edge[2]. Vì thế theo quan điểm của người viết đồ án thì nó chỉ phù hợp với các công ty công nghệ mà thôi.

[2] http://en.wikipedia.org/wiki/Bleeding_edge_technology

1.4 Sự cần thiết của các công cụ quản lý

Như đã nói ở trên, người quản trị hệ thống với những công việc lặp đi lặp lại này, điều thường thấy là người quản trị hệ thống sẽ cố gắng để tự động hóa chúng bằng các kịch bản (script) và các công cụ. Điều này dẫn đến sự phát triển của các ứng dụng và kịch bản tùy chỉnh. Những kịch bản này thường phức tạp, không có tài liệu đi kèm, và chỉ hoạt động trên một số hệ thống nhất định.

Việc này không có gì độc đáo cả, và việc phát triển các kịch bản này chỉ là một phản ứng thông thường với mong muốn làm cho công việc dễ dàng hơn, tự động hóa những việc thủ công nhàm chán để có thêm thời gian dành cho những điều thú vị hơn.



Hình 1.1: Tự động hóa hệ thống bằng kịch bản

Có rất ít các kịch bản đã được phát triển theo cách đặc biệt này được xuất bản, có tài liệu hay sử dụng lại. Thêm vào đó, bản quyền đối với hầu hết những kịch bản kể trên đều thuộc về những người quản trị hệ thống đã viết ra nó, và thường nó sẽ bị bỏ lại khi họ rời bỏ tổ chức đó hoặc chuyển sang vị trí khác. Điều này dẫn đến việc cùng một công cụ cứ được làm đi làm lại hết lần này đến lần khác. Thậm chí, đôi lúc chỉ đơn giản là điều đó không phù hợp với cách làm việc của người tại nhiệm hoặc người tại nhiệm không thể nắm bắt được những gì người tiền nhiệm đã để lại.

Khi hệ thống được mở rộng, những ứng dụng và kịch bản tùy chỉnh kiểu này hiếm khi phù hợp với môi trường lớn, và họ thường phải gặp phải rất nhiều vấn đề với tính ổn định, sự linh hoạt và các tính năng. Trong môi trường đa nền tảng, những kịch bản như vậy có xu hướng chỉ phù hợp với một nền tảng nhất định, dẫn đến trong các tính huống thực sự cần thiết sẽ phải tạo ra các kịch bản sử dụng cho BSD, một cái khác cho Linux và thậm chí là một cái khác nữa cho Solaris. Thật là tốn công sức và thời gian khi phát triển cũng như duy trì các công cụ như vậy với hy vọng nó sẽ giảm thiểu những công việc nhiêu khê mà bạn phải làm.

Một cách tiếp cận khác đó là mua các công cụ điều hành và quản lý cấu hình như Opsware của HP, CONTROL-M của BMC, Tivoli của IBM hay Unicenter của CA. Những công cụ thương mại này thường gặp phải 2 vấn đề chính đó là giá cả và tính linh hoạt. Đặc biệt là giá cả, nó có thể nhanh chóng trở thành một vấn đề lớn khi bạn sử dụng nhiều hơn các nền tảng và hạ tầng, giá có thể lên rất cao. Trong những hệ thống lớn, chi phí cho giấy phép sử dụng những công cụ như thế có thể lên tới hàng triệu USD.

Tính linh hoạt cũng là một mối quan tâm chính. Các công cụ thương mại thường đóng mã nguồn và giới hạn các tính năng có sẵn của chúng, có nghĩa

là nếu bạn muốn mở rộng chúng để có thêm một tùy chỉnh nào đó phù hợp với hệ thống của bạn, bạn cần phải yêu cầu một tính năng mới, và đương nhiên nó có khả năng phải mất một thời gian chờ đợi và một số khoản chi phí liên quan. Với sự đa dạng về việc triển khai, các nền tảng, các cấu hình và các ứng dụng trong các tổ chức, thật hiếm khi phát hiện bất kỳ công cụ cung cấp khả năng hoàn toàn tùy chỉnh cho phù hợp với môi trường của bạn.

Có một giải pháp thay thế cho cả việc phát triển phần mềm tự thân và phần mềm thương mại. Đó là Phần Mềm Tự Do Mã Nguồn Mở [3]. Các công cụ quản lý cấu hình mã nguồn mở cung cấp cho các tổ chức, doanh nghiệp 2 lợi ích chính sau:

- Chúng đi kèm mã nguồn nên có thể mở rộng được
- Chúng là miễn phí!

Với các sản phẩm phần mềm nguồn mở, mã nguồn của công cụ trong tầm tay của bạn, cho phép bạn để phát triển cải tiến hoặc điều chỉnh của riêng bạn. Bạn không cần phải chờ đợi cho các nhà cung cấp để thực hiện các chức năng cần thiết hoặc trả tiền cho các tính năng mới hoặc thay đổi. Bạn cũng là một phần của một cộng đồng người sử dụng và phát triển những người chia sẻ một tầm nhìn cho sự phát triển của công cụ. Bạn và tổ chức của bạn có thể lần lượt đóng góp vào tầm nhìn đó. Kết hợp lại, bạn có thể định hình hướng đi của các công cụ bạn đang sử dụng, đem lại thêm sự linh hoạt tổ chức của bạn.

Giá cả là một trong những yếu tố quan trọng đối với việc mua bất kỳ công cụ. Với phần mềm tự do mã nguồn mở, nó không phải là một vấn đề. Bạn

[3]Free and Open Source Software (FOSS): https://en.wikipedia.org/wiki/FOSS

không phải trả bất cứ chi phí nào cho phần mềm, thêm vào đó bạn còn có được mã nguồn của nó.

Tất nhiên, chúng ta đều biết chẳng bao giờ có một bữa ăn trưa miễn phí cả, không giống như phần mềm thương mại, phần mềm nguồn mở không đi kèm với bất kỳ sự hỗ trợ đảm bảo nào. Nói như thế không có nghĩa là sự hỗ trợ không có sẵn: Nhiều công cụ mã nguồn mở có cộng đồng lớn và rất năng động, ở đó các thành viên trả lời câu hỏi và cung cấp hỗ trợ thông qua các cơ chế như mailing list, diễn đàn, Wiki hay IRC.[4]

[4]Các công cụ mã nguồn mở, bao gồm cả Puppet hay Chef, cũng có tổ chức cung cấp các phiên bản thương mại hoặc hỗ trợ cho những công cụ này.

CHƯƠNG 2

CÁC CÔNG CỤ TỰ ĐỘNG HÓA HỆ THỐNG

Sự gia tăng của ảo hóa cùng với sức mạnh ngày càng tăng của các máy chủ chuẩn công nghiệp và sự sẵn có của điện toán đám mây đã dẫn đến một sự gia tăng đáng kể số lượng các máy chủ cần phải được quản lý trong và ngoài các doanh nghiệp, tổ chức. Trước kia, chúng ta có thể làm việc với hàng loạt các máy chủ vật lý hay truy cập trực tiếp vào các trung tâm dữ liệu nhưng giờ đây, chúng ta phải quản lý nhiều hơn rất nhiều các máy chủ và chúng lại còn được đặt ở rất nhiều các trung tập dữ liệu trên toàn thế giới.

Những trung tâm dữ liệu - đó chính là nơi các công cụ tổ chức và quản lý cấu hình vào cuộc. Trong nhiều trường hợp, chúng ta quản lý nhiều nhóm máy chủ giống hệt nhau, chạy các ứng dụng và dịch vụ giống hệt nhau. Chúng được triển khai trên các nền tảng ảo hóa trong các doanh nghiệp, tổ chức, hay ở trên hệ thống điện toán đám mây hoặc một máy chủ nào đó tại một trung tâm dữ liệu ở xa. Điều đó dẫn đến cần có một công cụ tự động hóa để có thể quản lý các cơ sở hạ tầng lớn và đang phát triển không ngừng này.

Puppet, Chef hay Ansible đã được xây dựng với suy nghĩ: đó là làm sao để có thể cấu hình và duy trì hàng chục, hàng trăm, thậm chí hàng ngàn máy chủ một cách dễ dàng và thuận tiện. Nói như vậy không có nghĩa rằng các doanh nghiệp - tổ chức nhỏ hơn sẽ không được hưởng lợi từ những công cụ này. Nói chung, các công cụ tự động hóa và sắp xếp công việc đều làm cho công việc dễ dàng hơn cho dù cơ sở hạ tầng lớn hay nhỏ.

Trong chương này chúng ta sẽ tìm hiểu thiết kế và chức năng của các công cụ tự động hóa hệ thống. Từ đó, đưa ra quyết định sử dụng công cụ cho phù hợp nhằm giải quyết bài toán trong thực tế mà chúng ta gặp phải.

2.1 Puppet

Puppet là một framework mã nguồn mở và là công cụ để quản lý cấu hình của hệ thống máy tính. Trong phần này, chúng ta sẽ tìm hiểu tổng quan về Puppet: cách thức nó hoạt động, cách nó quản lý cấu hình của hệ thống máy chủ, lý do puppet có thể chạy trên nhiều nền tảng khác nhau. Tiếp đó, chúng ta sẽ tìm hiểu kiến trúc của Puppet, cách Puppet thu thập và quản lý các gói dữ liệu cấu hình.

2.1.1 Tổng quan

Puppet là một công cụ quản lý cấu hình mã nguồn mở viết bằng Ruby, được sử dụng để quản lý cấu hình máy chủ và tự động hóa hệ thống trong các trung tâm dữ liệu của Google, Twitter, thị trường chứng khoán New York, và nhiều doanh nghiệp lớn khác. Puppet được phát triển đầu tiên bởi Puppet Labs, và hiện tại Puppet Labs cũng là người duy trì chính của dự án này. Puppet được dùng để quản lý có khi chỉ vài máy chủ nhưng cũng có khi lên tới 50.000 máy chủ, cùng với đó là đội quản trị hệ thống từ một người tới hàng trăm người.

Puppet là một công cụ để quản lý cấu hình và bảo trì hệ thống máy tính; ngôn ngữ cấu hình của nó rất đơn giản. Chúng ta chỉ cần chỉ cho Puppet thấy chúng ta muốn cấu hình máy tính của chúng ta như thế nào, nó sẽ thực hiện đúng những gì chúng ta muốn. Khi hệ thống có sự thay đổi chẳng hạn như một phiên bản cập nhật của gói phần mềm, thêm người dùng mới hay một cấu hình nào đó thay đổi, Puppet tự động cập nhật tất cả các máy chủ trong hệ thống đúng như cấu hình chúng ta muốn.

```
class ssh {
   package { ssh:
      ensure => installed
   }
   file { "/etc/ssh/sshd_config":
      source => 'puppet:///modules/ssh/sshd_config',
      ensure => present,
      require => Package[ssh]
   }
   service { sshd:
      ensure => running,
      require => [File["/etc/ssh/sshd_config"], Package[ssh]]
   }
}
```

Mã nguồn 2.1: Ví dụ về ngôn ngữ cấu hình của Puppet

Trong mã nguồn 2.1 phía trên, Puppet đảm bảo rằng gói phần mềm ***ssh*** được cài đặt và file cấu hình dịch vụ SSH [1] ***/etc/ssh/sshd_config*** của tất cả các máy chủ trong hệ thống mà Puppet quản lý đều có cùng nội dung mới file đã định trước; thêm vào đó, Puppet đảm bảo việc dịch vụ này luôn được chạy giúp người quản trị hệ thống có thể can thiệp thủ công khi cần thiết.

2.1.2 Kiến trúc hệ thống

Puppet được xây dựng với hai chế độ làm việc:

- **Chế độ client/server** Có một máy chủ trung tâm với một dịch vụ chạy nền kết nối đến các "agents" chạy độc lập trên các máy trạm.
- **Chế độ serverless** Chỉ có một tiến trình duy nhất thực hiện tất cả các công việc.

[1] Secure Shell: https://en.wikipedia.org/wiki/Secure_Shell

Để đảm bảo tính nhất quán giữa các chế độ, Puppet luôn có sự minh bạch trong các liên kết nội bộ của bản thân nó. Do đó, hai chế độ này sử dụng cùng một đường dẫn như nhau cho dù chúng có giao tiếp với nhau qua mạng hay không. Mỗi lệnh được thực thi cho dù lấy cấu hình từ bản nó hay một máy khác ở xa trong mạng thì chúng đều có một cách thực hiện như nhau. Tuy nhiên cũng phải lưu ý rằng, chế độ serverless là một phần trong trong mô hình client/server: tất cả các file cấu hình sẽ được đẩy xuống cho agents ở máy trạm xử lý, tại đây máy trạm chạy ở chế độ serverless sẽ làm việc trực tiếp với các tệp tin cấu hình và thực thi chúng. Phần này sẽ chỉ tập trung vào chế độ client/server bởi vì nó dễ hiểu hơn với các thành phân riêng biệt, nhưng hãy luôn nhớ rằng tất cả đều chạy ở chế độ serverless.

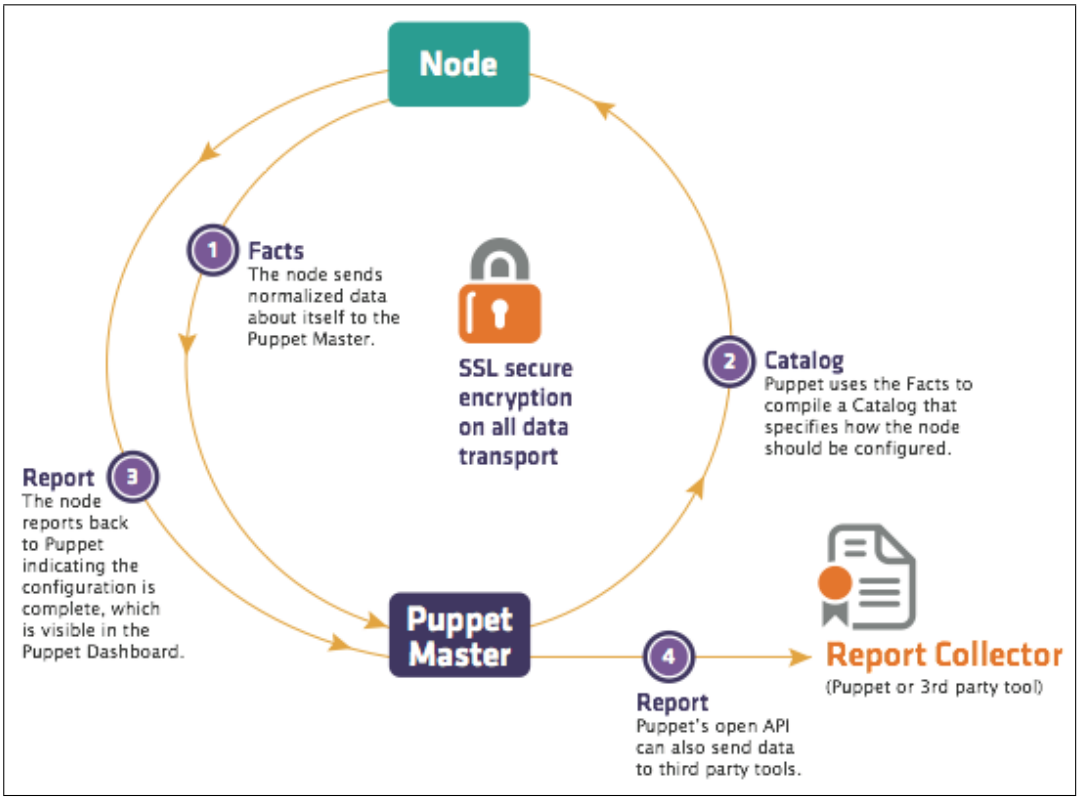
Một trong những lựa chọn ngay từ đầu trong kiến trúc của Puppet là máy trạm không nên truy cập trực tiếp (raw access) vào các module, thay vào đó, chúng lấy các cấu hình đã được chuẩn bị sẵn từ trước. Việc này cung cấp nhiều lợi ích:

- **Thứ nhất**, chung ta thực hiện được việc tối thiểu quyền hạn cần thiết. Trong đó mỗi máy chủ chỉ biết chính xác những gì nó cần phải biết (nó nên được cấu hình như thế nào), nhưng nó không biết (và không quan tâm) những máy trạm khác được cấu hình như thế nào.

- **Thứ hai**, chúng ta hoàn toàn có thể phân tách các quyền cần thiết để tạo ra một cấu hình (bao gồm cả quyền truy cập vào nơi lưu trữ dữ liệu trung tâm) mà nó sẽ được thực hiện dưới máy trạm.

- **Thứ ba**, chúng ta có thể chạy các máy trạm trong chế độ ngắt kết nối với máy chủ trung tâm, nhưng các cấu hình đã có của Puppet sẽ vẫn luôn được áp dụng. Nghĩa là, cho dù máy chủ trung tâm (puppet-master)

không còn hoạt động hoặc không có kết nối đến nó thì mỗi máy trạm vẫn có thể làm việc độc lập.



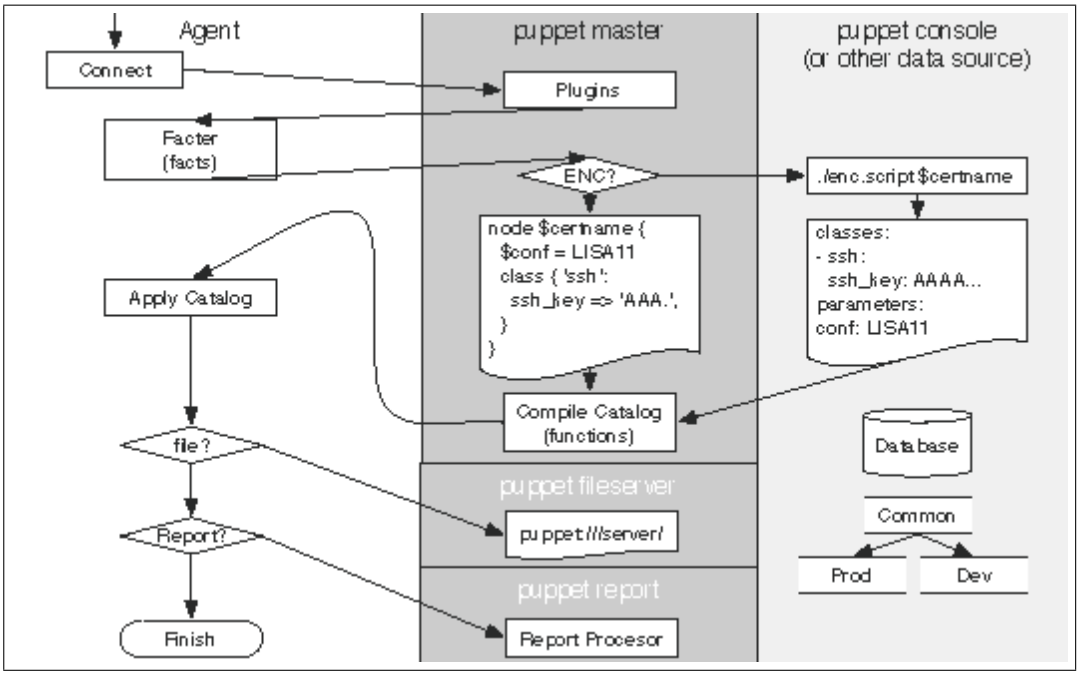
Hình 2.1: Biểu đồ luồng dữ liệu của Puppet

Với sự lựa chọn này, quy trình làm việc trở nên tương đối đơn giản

- Tiến trình trên máy trạm (Puppet agent) thu thập các thông tin về hệ thống mà nó đang làm việc, sau đó chuyển các thông tin này tới máy chủ trung tâm (Puppet Master)
- Tại máy chủ trung tâm, các thông tin đó cùng các module trên ổ đĩa cục bộ được biên dịch thành một cấu hình cho một máy chủ cụ thể và trả lại nó cho các tiến trình trên máy trạm.

- Các tiến trình trên máy trạm áp dụng những cấu hình cục bộ này và nó chỉ ảnh hưởng tới riêng máy trạm đó. Sau đó, các tập tin báo cáo được tạo ra rồi đưa kết quả về máy chủ trung tâm.

Vì thế, các agent có quyền truy cập vào thông tin riêng trên hệ thống của mình, các cấu hình của bản thân nó, cũng như các báo cáo nó tạo ra. Máy chủ trung tâm có bản sao của tất cả các dữ liệu này, cùng với quyền truy cập toàn bộ các module, cũng như bất kỳ cơ sở dữ liệu và dịch vụ nào khác dùng để biên dịch các cấu hình cần thiết.



Hình 2.2: Luồng dữ liệu lưu chuyển
giữa các thành phần và tiến trình của Puppet

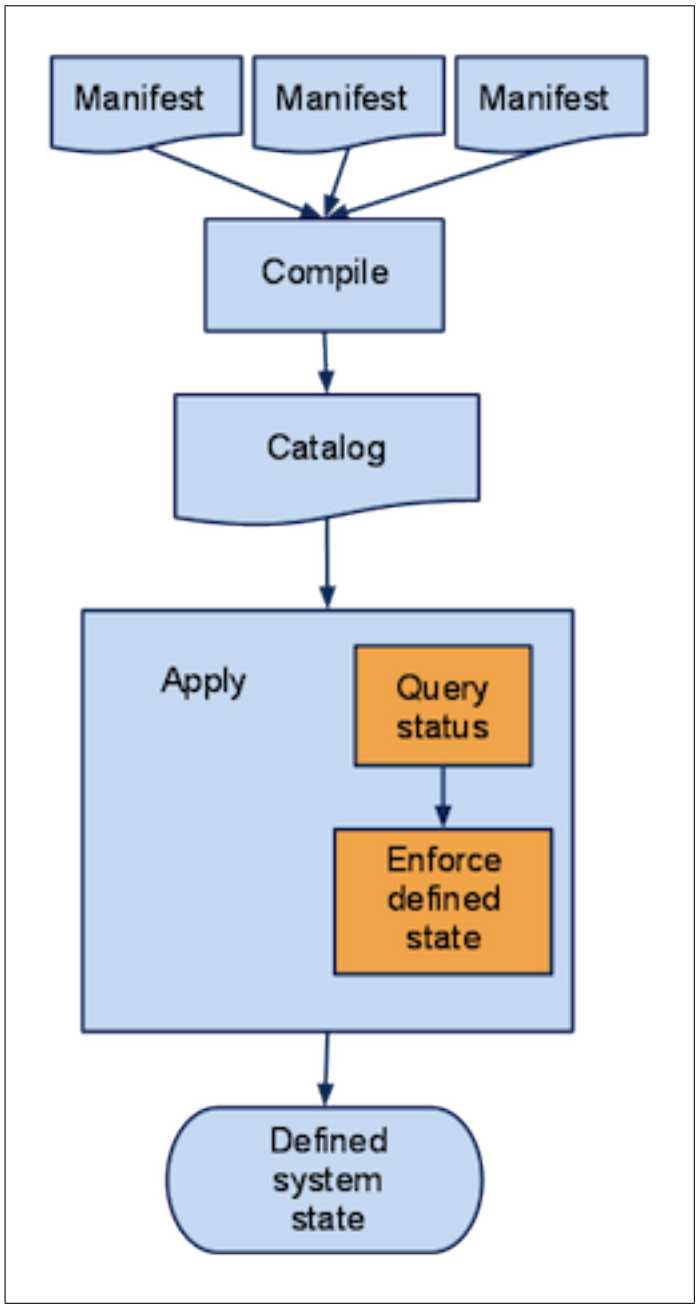
Ngoài các thành phần ở trong quy trình làm việc này, có rất nhiều loại dữ liệu được Puppet sử dụng cho các giao tiếp nội bộ của nó. Các loại dữ liệu rất quan trọng, bởi vì chúng hoàn thực hiện tất cả các thông tin liên lạc, đồng

thời chúng cũng cung cấp các giao diện công cộng cho những công cụ khác sử dụng hay làm việc với chúng.

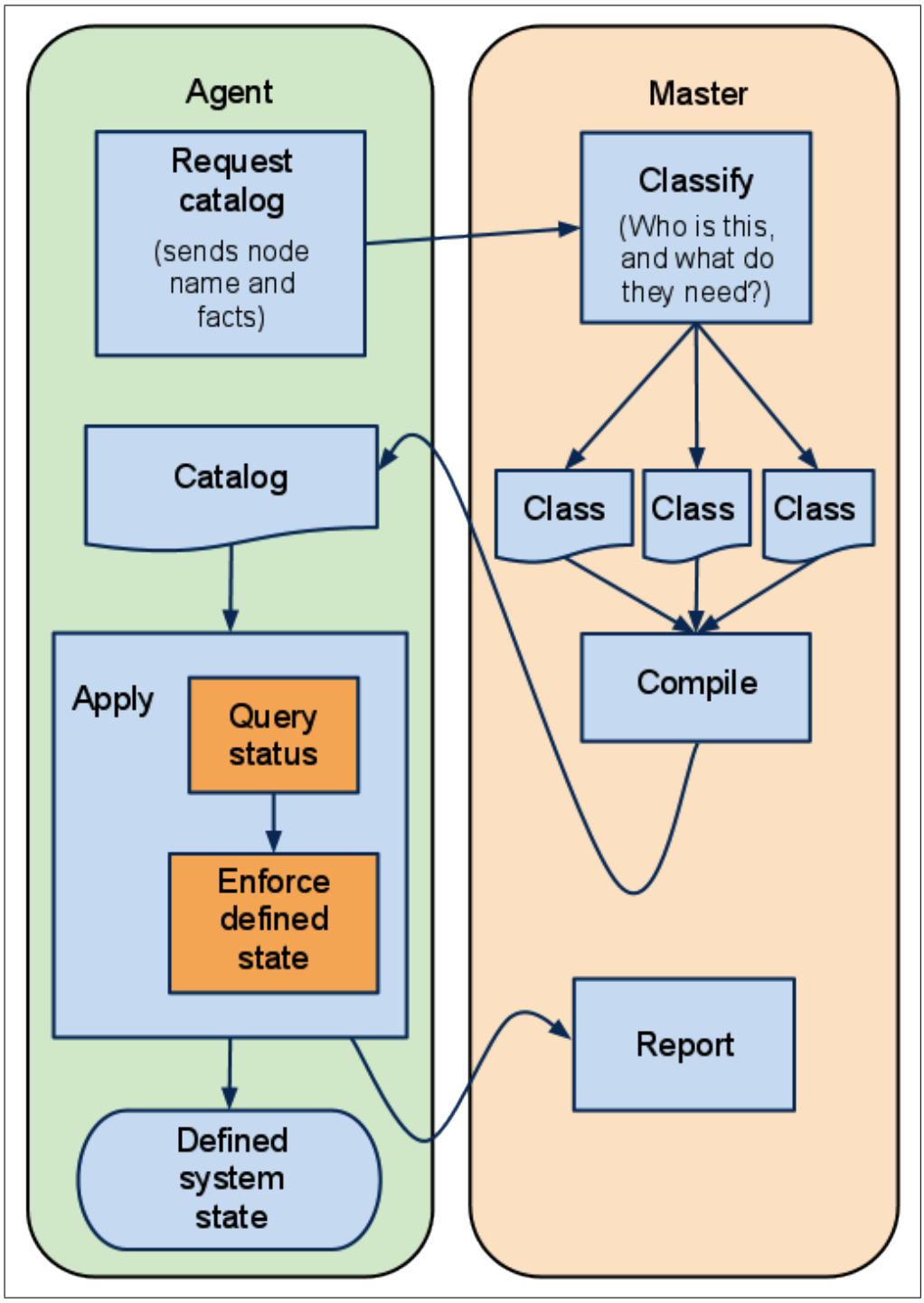
Các kiểu dữ liệu quan trọng nhất trong Puppet là:

- **Facts**: Thu thập các thông thin hệ thống trên mỗi máy trạm. Những thông tin này được dùng để biên dịch ra các cấu hình.
- **Manifest**: Các tập tin chứa ngôn ngữ cấu hình Puppet, chúng thường được tổ chức thành các bộ sưu tập được gọi là module.
- **Catalog**: Một đồ thị về các tài nguyên của máy chủ được quản lý và các rằng buộc giữa chúng.
- **Report**: Tập hợp tất cả các sự kiện được tạo ra trong suốt quá trình tạo ra các Catalog.

Ngoài Facts, Manifests, Catalogs, and Reports, Puppet còn hỗ trợ các kiểu dữ liệu như các tệp tin, các chứng chỉ (được dùng trong việc xác thực) cùng nhiều kiểu dữ liệu khác.



Hình 2.3: Cách Puppet biên dịch và thực hiện một manifest trong chế độ Serverless



Hình 2.4: Cách Puppet biên dịch và thực hiện một manifest trong chế độ Client-Server

2.1.3 Các thành phần chính

Các thành phần chính của Puppet bao gồm: Agent, Facter, ENC, Compiler, Transaction, RAL và Reporting

Agents

Thành phần đầu tiên mà chúng ta tiếp xúc khi sử dụng Puppet là tiến trình agent. Trong các phiên bản trước của puppet, tiến trình này được tách riêng thành một tiến trình riêng biệt có tên là ***puppetd***, nhưng trong phiên bản 2.6 trở đi, chúng được tối giản hóa và bây giờ chúng được gọi bằng lệnh ***puppet agent***, tương tự như cách làm của Git[2]. Các agent thực hiện rất ít các chức năng của riêng nó mà chủ yếu các công việc liên quan đến các cấu hình hay mã nguồn được thực hiện ở phía máy trạm như trong quy trình làm việc đã mô tả ở trên.

Facter

Thành phần tiếp theo sau agent là một công cụ bên ngoài gọi là **facter**, đó là một công cụ đơn giản được sử dụng để tìm kiếm và thu thập các thông tin về hệ thống nó đang chạy trên đó. Thông tin đó thường là hệ điều hành, địa chỉ IP, tên máy chủ, nhưng vì Facter là một công cụ có khả năng mở rộng nên rất nhiều các tổ chức thường thêm vào các plugin riêng họ để thu thập các thông tin khác mà họ cần. Các agent gửi những thông tin mà Facter đã thu thập được tới máy chủ trung tâm, nơi nó được tiếp quản và xử lý theo quy trình ở hình 2.1

External Node Classifier

Trên máy chủ trung tâm, thành phần đầu tiên chúng ta cần đề cập tới là External Node Classifier[3] hay ENC. ENC chấp nhận tên máy trạm hoặc trả

[2]Một hệ thống quản lý mã nguồn phân tán `http://git-scm.com/`

[3]Các lớp phân loại mở rộng

về các cấu trúc dữ liệu đơn giản có chứa các cấu hình cấp cao của máy chủ đó. ENC nói chung là một ứng dụng hay dịch vụ riêng biệt: điển hình là một vài sản phẩm mã nguồn mở như Puppet Dashboard hay Foreman; đôi khi nó được tích hợp vào hệ thống dữ liệu có sẵn chẳng hạn như máy chủ LDAP... Mục đích cơ bản của ENC là để xác định những gì mà các lớp chức năng thuộc về, kèm theo đó là những thông số cấu hình cho các lớp này. Ví dụ, một máy trạm có thể thuộc lớp ***debian*** hay ***webserver*** và chúng có tham số để báo rằng chúng ở trung tâm dữ liệu tại ***atlanta***

Lưu ý rằng như Puppet 2.7, ENC không phải là một thành phần bắt buộc, thay vào đó người dùng có thể trực tiếp chỉ định cấu hình tại mỗi nút của Puppet. ENC được hỗ trợ trong khoảng 2 năm sau khi Puppet ra đời, bởi vì những nhà phát triển nhận ra rằng các lớp về cơ bản là khác biệt với các việc cấu hình chúng. Và, nó sẽ có ý nghĩa hơn nhiều khi đưa những vấn đề này vào các công cụ riêng biệt thay vì mở rộng ngôn ngữ để hỗ trợ cả hai cơ sở này. ENC luôn luôn khuyến khích sử dụng, và tại một số nơi nó trở thành một thành phần cần thiết (lúc này Puppet sẽ xuất hiện với tư cách một công cụ hữu ích hơn là một gánh nặng).

Một khi máy chủ trung tâm nhận được thông tin phân loại theo lớp từ ENC và thông tin hệ thống từ facter (thông qua các agent), nó đóng gói tất cả các thông tin vào một đối tượng Node và chuyển nó vào các trình biên dịch (Compiler).

Compiler

Như đã đề cập ở trên, Puppet có một ngôn ngữ tùy chỉnh được xây dựng dành riêng cho việc cấu hình hệ thống. Trình biên dịch này bao gồm 3 khối: một bộ phân tích ngôn kiểu-Yacc[4] đã được tùy chỉnh; một nhóm các lớp học

[4]https://en.wikipedia.org/wiki/Yacc

sử dụng để tạo ra Cây cú pháp trừu tượng (Abstract Syntax Tree hay AST); sau cùng lớp biên dịch, nó xử lý các tương tác của tất cả các lớp, đồng thời cũng có chức năng như API và là một phần của hệ thống.

Transaction

Khi các Catalog đã được xây dựng thành công, nó được chuyển qua cho lớp Transaction. Trong một hệ thống mà máy trạm và máy chủ trung tâm được tách biệt, bộ phận này được chạy trên máy trạm, nó có nhiệm vụ kéo xuống các Catalog qua giao thức HTTP như Hình 2.2.

Lớp Transaction thực hiện một công việc tương đối đơn giản: tìm sự rằng buộc trong các mối quan hệ và đảm bảo các tài nguyên đấy được đồng bộ. Nó thực hiện một quá trình với 3 bước đơn giản: lấy trạng thái hiện tại của tài nguyên đó, so sánh nó với trạng thái mong muốn và thực hiện bất kì thay đổi cần thiết nào để sữa chữa sai lệnh.

Resource Abstraction Layer

Như đã biết, lớp Transcaction là rất quan trọng sự hoàn thành các công việc trong Puppet, nhưng thật ra, tất cả các công việc được thực sự thực hiện bởi lớp tài nguyên trừu tượng (Resource Abstraction Layer, viết tắt là RAL). Đây cũng là một trong các thành phần thú vị nhất của Puppet.

RAL là thành phần đầu tiên được tạo ra trong Puppet, hơn cả ngôn ngữ cấu hình, nó định nghĩa rõ ràng những gì con người có thể làm với nó. Công việc của RAL là xác định những gì là tài nguyên và cách các tài nguyên đó có thể thực hiện công việc của nó trên hệ thống. Sau đó ngôn ngữ Puppet được cụ thể hóa thành các mô hình bằng RAL. Bởi vì điều này nên nó là thành phần quan trọng nhất trong hệ thống, và khó khăn nhất để thay đổi. Có rất nhiều điều những nhà phát triển muốn khắc phục trong RAL, và họ đã thực hiện rất nhiều cải tiến quan trọng đối với nó trong nhiều năm qua (quan trọng

nhất là việc bổ sung các Providers), nhưng vẫn có rất nhiều việc phải làm với RAL trong dài hạn.

Như đã nói, lớp Transaction không thực sự ảnh hưởng trực tiếp đến hệ thống, mà nó dựa vào RAL để làm việc đó. Trong thực tế, providers là thành phần duy nhất của Puppet mà thực sự tác động vào hệ thống. Nếu lớp Transaction yêu cầu nội dung của một tập tin thì lớp Providers tìm kiếm và thu thập nó; nếu lớp Transaction xác định rằng nội dung của một tập tin nên được thay đổi thì lớp Providers sẽ thay đổi nó. Tuy nhiên phải lưu ý rằng, bản thân lớp Providers không bao giờ có quyền quyết định ảnh hưởng đến hệ thống mà chính lớp Transaction mới có quyền này, lớp Providers chỉ là người thực hiện công việc. Điều này cho phép các lớp Transaction kiểm soát hoàn toàn mà không yêu cầu phải hiểu bất cứ điều gì về các tập tin, người dùng, hoặc các gói. Sự phân chia rõ ràng này cho phép Puppet có sự mô phỏng đầy đủ nơi mà chúng ta có thể đảm bảo phần lớn hệ thống không bị ảnh hưởng.

Reporting

Trong quá trình lớp transaction đi theo các đồ thị và sử dụng RAL để thay đổi các cấu hình của hệ thống, các bản báo cáo được tạo ra. Báo cáo này bao gồm hầu hết cac sự kiện được tạo ra bởi những thay đổi trong hệ thống. Những sự kiện này, chúng là phản ảnh toàn diện về những công việc đã thực hiện: mốc thời gian thay đổi, giá trị trước đó, giá trị mới và bất kì thông báo nào được tạo ra, và tất nhiên là việc thay đổi đó thành công hay thất bại.

Các sự kiện được bao bọc trong một đối tượng ResourceStatus được ánh xạ vào từng tài nguyên. Vì vậy, đối với một giao dịch nhất định, chúng ta biết tất cả các nguồn tài nguyên đã được sử dụng, tất cả các thay đổi đã xảy ra, cùng với tất cả các metadata mà chúng ta cần biết về thay đổi đó.

Sau khi giao dịch hoàn tất, một số số liệu cơ bản được tính toán và lưu trữ

trong báo cáo, và sau đó nó được gửi đến máy chủ trung tâm (nếu cấu hình). Với báo cáo đã gửi, quá trình cấu hình hoàn tất, các agent sẽ quay về trạng thái nghỉ hoặc là tự động kết thúc.

2.1.4 Kiến trúc hạ tầng

Trong mục này chúng ta sẽ tìm hiểu các thành phần hạ tầng của Puppet

Hệ thống các Plugin

Một trong những điều tuyệt vời của Puppet là nó có khả năng mở rộng rất tốt. Có ít nhất 12 loại mở rộng khác nhau của Puppet, có nghĩa nó có thể sư dụng cho bất kì ai. Ví dụ, chúng ta có thể bổ sung các tùy chỉnh trong các lĩnh vực sau:

- Các kiểu tài nguyên hay provider tùy chỉnh.
- Cách xử lý các báo cáo cũng như việc lưu trữ các báo cáo này vào cơ sở dữ liệu riêng.
- Bổ sung thêm các Indirector để tương tác với những cơ sở dữ liệu sẵn có.
- Các facter để thu thập và cung cấp thêm các thông tin về máy trạm.

Tuy nhiên, do tính chất phân tán tự nhiên của Puppet nên cần có một nơi để các agent có thể tải về các plugin này. Vì vậy, mỗi lần chạy Puppet, điều đầu tiên chúng ta cần làm là tải về tất cả các plugin và đặt chúng ở máy chủ trung tâm. Các plugin này bao gồm các loại tài nguyên mới, các thông tin mới cần thu thập, hoặc một cách xử lý báo cáo khác thường nào đó.

Điều này khiến cho việc nâng cấp các agent của Puppet mà không bao giờ ảnh hưởng tới những gói cốt lõi nhất. Điều này vô cùng quan trọng trong các hệ thống Puppet có độ tùy chỉnh cao.

Indirector

Indirector là một dạng của chuẩn Inversion of Control[5] (IoC) framework với khả năng mở rộng cao. IoC cho phép chúng ta tách biệt việc phát triển các chức năng với việc kiểm soát các chức năng mà bạn đang sử dụng. Trong trường hợp của Puppet, chúng cho phép nhiều plugin cùng cung cấp các chức năng chẳng hạn như việc trình biên dịch đưa thông tin qua HTTP hoặc tải nó khi nó đang chạy hay việc chuyển đổi giữa việc thay đổi cấu hình nhỏ hơn là thay đổi cả source code. Indirector là một dạng thể hiện đơn giản của IoC. Tất các lớp bắt tay nhau làm việc từ lớp này tới lớp khác thông qua Indirector bằng một giao diện chuẩn REST[6]. Việc này có nghĩa có thể chuyển các máy trạm chạy ở chế độ Serverless với đầu cuối HTTP để lấy cái danh mục về thay vì sử dụng một đầu cuối đã được biên dịch.

Hệ thống mạng

Một câu hỏi được đặt ra khi viết nguyên mẫu của Puppet năm 2004 là sẽ sử dụng XMLRPC hay SOAP, những nhà phát triển đã chọn XMLRPC. Tất nhiên là chúng vẫn làm việc được nhưng nó gặp những vấn đề mà tất cả những người khác đều gặp phải: Nó không có một chuẩn giao diện giữa các thành phần, nó có xu hướng phức tạp rất nhanh.

Trong phiên bản 0.25 phát hành năm 2008, những nhà phát triển đã bắt đầu quá trình chuyển đổi tất cả các kết nối mạng sang dạng REST, nhưng họ đã chọn con đường phức tạp hơn là chỉ thay đổi kết nốt mạng. Họ đã phát

[5] https://en.wikipedia.org/wiki/Inversion_of_control

[6] https://en.wikipedia.org/wiki/REST

triển Indirector như một bộ khung tiêu chuẩn trong việc giao tiếp giữa các thành phần của Puppet, tuy nhiên lúc này REST chỉ là một lựa chọn trong cài đặt. Phải mất tới 2 phiên bản họ mới có bản hỗ trợ đầy đủ REST, tuy vậy họ vẫn chưa thực sự hoàn thành việc chuyển đổi để sử dụng JSON thay vì YAML. Họ chuyển sang JSON vì 2 lý do chính: xử lý YAML với Ruby chậm hơn rất nhiều lần so với việc xử lý JSON; đồng thời hầu hết các website đã chuyển sang dùng JSON, việc sử dụng JSON có vẻ đơn giản hơn nhiều so với YAML.

Các phiên bản mới của Puppet đã dần loại bỏ hoàn toàn các XMLRPC

2.1.5 Cài đặt và sử dụng

Trong các phần trước chúng ta đã tìm hiểu cách thức hoạt động, kiến trúc hệ thống và hạ tầng của Puppet. Trong phần này, chúng ta sẽ tìm hiểu cơ bản về việc cài đặt và sử dụng Puppet trong môi trường Linux.

Cách cài đặt Puppet trên một số HĐH Linux phổ dụng

Puppet có thể được cài đặt và sử dụng trên rất nhiều các nền tảng khác nhau bao gồm:

- Red Hat Enterprise Linux (RHEL), CentOS, Fedora & Oracle Enterprise Linux
- Debian và Ubuntu
- Mandrake và Mandriva
- Gentoo
- Solaris và OpenSolaris

- MacOS X và MacOS X Server
- *BSD - Các hệ điều hành họ BSD
- AIX
- HP UX
- Microsoft Windows (tuy nhiên có một số hạn chế nhất định)

Trên một số nền tảng, Puppet có thể quản lý rất nhiều các loại cấu hình khác nhau bao gồm:

- Files
- Services
- Packages
- Users
- Groups
- Cron jobs
- SSH keys
- Nagios configuration

Đối với Puppet, các agent và master server có cách cài đặt giống nhau, mặc dù hầu hết các hệ điều hành và bản phân phối các gói được phân chia theo chức năng của master và agent thành các gói riêng biệt. Trên một số hệ điều hành và bản phân phối, bạn cũng sẽ cần phải cài đặt Ruby cùng các thư viện của nó và số gói bổ sung cần thiết khác. Hầu hết các hệ thống đóng gói

tốt sẽ có hầu hết các gói cần thiết, điển hình là Ruby - đó là yếu tố tiên quyết để có thể chạy được Puppet.

Trong nội dung hạn chế của đồ án này, chúng ta sẽ chỉ tìm hiểu cách cài đặt trên 2 bản phân phối Linux phổ biến nhất là RHEL và Debian (tương tự với CentOS hay Ubuntu). Chúng ta sử dụng repo của **Puppet Labs** để cài đặt Puppet vì ở đây là repo của nơi phát triển Puppet nên nó luôn đi kèm phiên bản mới cùng các bản vá bảo mật đi kèm.

- **Cách cài đặt Puppet trên RHEL**

Đối với máy chủ trung tâm:

```
# Install Puppet Labs repo
rpm -ivh http://yum.puppetlabs.com/puppetlabs-release-el-6.noarch.rpm
# Download puppet-server from Puppet Labs
yum install -y puppet-server
# Start puppet master
/etc/init.d/puppetmaster start
# Set Puppet Master to run on startup
puppet resource service puppetmaster ensure=running enable=true
```

Mã nguồn 2.2: Cách cài đặt puppet master trên RHEL

Đối với agent:

```
# Add the puppet labs repo
rpm -ivh http://yum.puppetlabs.com/puppetlabs-release-el-6.noarch.rpm
# Install the puppet agent
yum install -y puppet
# Set Puppet Agent to run on startup
puppet resource service puppet ensure=running enable=true
```

Mã nguồn 2.3: Cách cài đặt puppet agent trên RHEL

- **Cách cài đặt Puppet trên Debian**

Đối với máy chủ trung tâm:

```
# Download Puppet Labs repo
wget http://apt.puppetlabs.com/puppetlabs-release-wheezy.deb
# Install Puppet Labs repo
sudo dpkg -i puppetlabs-release-wheezy.deb
# Install puppetmaster package
sudo apt-get update
sudo apt-get install puppetmaster
# Set Puppet Master to run on startup
puppet resource service puppetmaster ensure=running enable=true
```

Mã nguồn 2.4: Cách cài đặt puppet master trên Debian

Đối với agent:

```
# Download Puppet Labs repo
wget http://apt.puppetlabs.com/puppetlabs-release-wheezy.deb
# Install Puppet Labs repo
sudo dpkg -i puppetlabs-release-wheezy.deb
# Install puppet package
sudo apt-get update
sudo apt-get install puppet
# Set Puppet Agent to run on startup
puppet resource service puppet ensure=running enable=true
```

Mã nguồn 2.5: Cách cài đặt puppet agent trên Debian

Cách cấu hình cơ bản hệ thống Puppet

Như đã nói ở trên, tuy các nền tảng khác nhau có các cách cài đặt khác nhau nhưng việc cấu hình trên các nền tảng đều tương tự nhau.

Các agent liên lạc với master qua các kênh SSL mã hóa, vì thế, đầu tiên chúng ta phải thiết lập kênh liên lạc cho các agent tới master.

```
# On the client machine, request a certificate to master:
puppet agent --server puppet --waitforcert 60 --test
# On the Puppetmaster machine, view a list of all certificates waiting to
    be signed:
puppet cert --list
#Sign the certificate for all clients
puppet cert --sign --all
```

Mã nguồn 2.6: Thiết lập kết nối qua kênh SSL

Chú ý: Puppet master phải mở firewall ở port 8140 để các agent có thể kết nối tới được.

Sau khi đã thiết lập các kênh liên lạc từ các agent tới master, chúng ta bắt đầu quá trình thiết lập cấu hình cho các node. Sẽ có rất nhiều các thiết lập phải cấu hình tùy theo nhu cầu hệ thống thực tế. Ví dụ dưới dây là một thiết lập cho **class sudo**

```
# /etc/puppet/manifests/classes/sudo.pp

class sudo {
  file { "/etc/sudoers":
    owner => "root",
    group => "root",
    mode => 440,
    source => "puppet://puppet.labs/files/sudoers"
  }
}
```

Mã nguồn 2.7: Cấu hình class sudo của Puppet

Các thiết đặt cùng cấu hình chi tiết hơn chúng ta có thể tham khảo trên trang chủ tài liệu của Puppet tại địa chỉ: http://docs.puppetlabs.com. Tài liệu ở đây khá đầy đủ và chi tiết.

Tóm lại

Puppet là một hệ thống các công cụ cả đơn giản lẫn phức tạp. Puppet là một framework điển hình để giải quyết các vấn đề liên quan đến cấu hình trong tự động hóa hệ thống. Tuy vậy nó là một ứng dụng đơn giản và dễ tiếp cận.

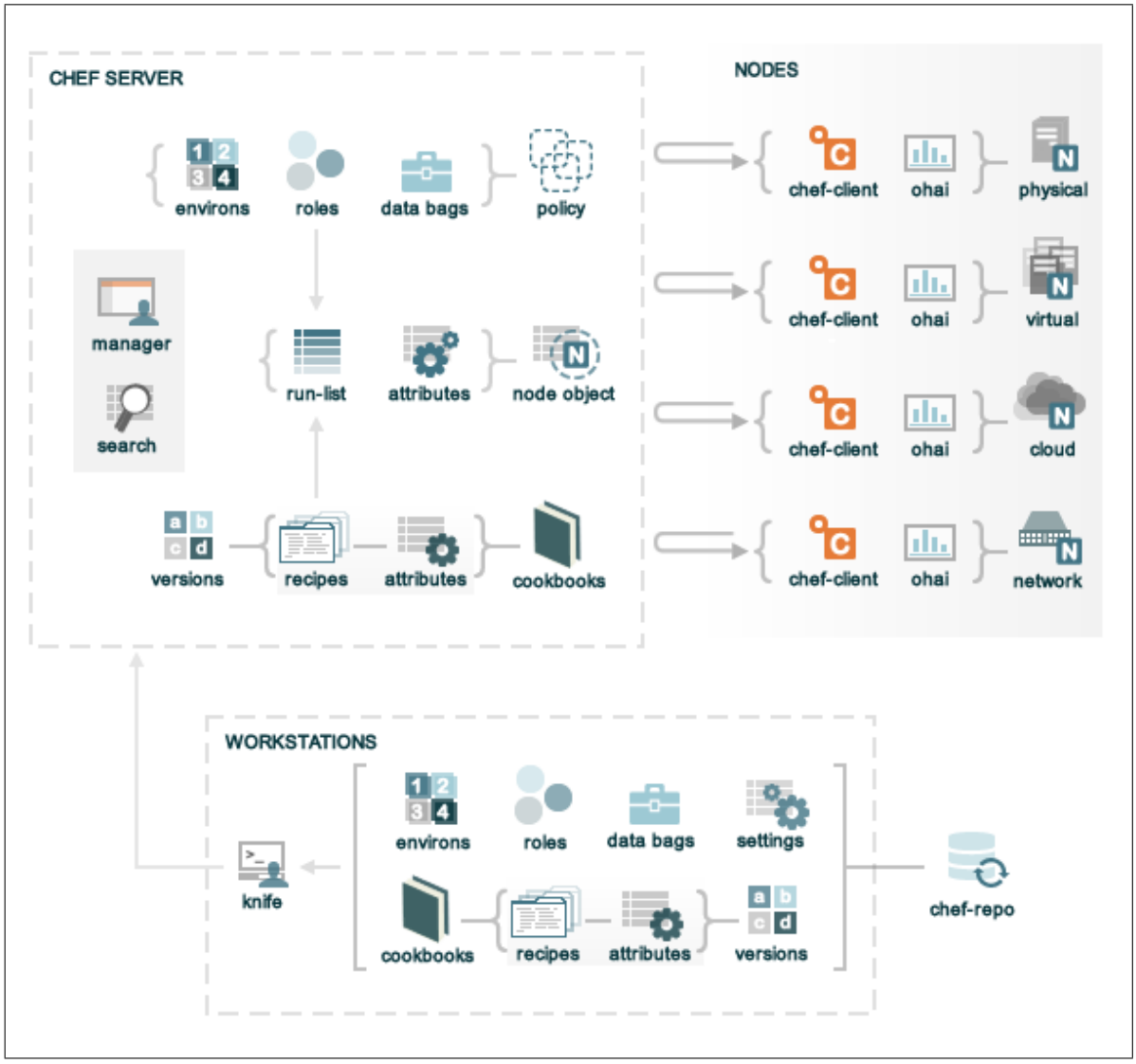
2.2 Chef

2.2.1 Tổng quan

Chef là một framework cho việc tự động hóa hệ thống và cơ sở hạ tầng điện của toán đám mây. Chef được dùng để triển khai các máy chủ hoặc các ứng dụng tới bất kỳ đâu: từ máy chủ vật lý tới máy chủ ảo hay máy chủ trên hệ thống điện toán đám mây, mà không bao giờ phải bận tâm về kích thước của cơ sở hạ tầng. Chef-client dựa trên các định nghĩa trừu tượng (được gọi là cookbooks và recipes) được viết bằng Ruby và được quản lý như mã nguồn. Mỗi định nghĩa mô tả cách một phần cụ thể của cơ sở hạ tầng nên được xây dựng hoặc quản lý. Chef-client sau đó áp dụng những định nghĩa này cho các máy chủ và các ứng dụng, theo quy định, dẫn đến một cơ sở hạ tầng hoàn toàn tự động. Khi một node mới được đưa vào hệ thống, điều duy nhất mà chef-client cần biết là có những cookbooks và recipes nào cần phải được áp dụng.

2.2.2 Kiến trúc hệ thống

Sơ đồ dưới đây cho thấy mối quan hệ giữa các yếu tố khác nhau của Chef, bao gồm các nút, máy chủ, và các máy trạm. Những yếu tố này làm việc cùng nhau để cung cấp các chef-client các thông tin và hướng dẫn giúp nó thực hiện các công việc của mình.



Hình 2.5: Mối quan hệ giữa các thành phần trong kiến trúc của Chef

Chef bao gồm ba yếu tố chính : một máy chủ trung tâm, một hay nhiều các nút, và ít nhất một máy trạm.

- Chef-server là máy chủ trung tâm. Nó phải luôn online để đảm bảo các chef-client có thể lấy được các cookbook và recipes để áp dụng.
- Các máy trạm là nơi mà từ đó các cookbook và recipes được viết ra, các chính sách được định nghĩa. Dữ liệu ở đây được đồng bộ với chef-repo

và cuối cùng tải lên server

- Mỗi nút trong hệ thống đều có một chef-client quản lý. Chef-client có nhiệm vụ tự động hóa các công việc mà mỗi nút được yêu cầu.

Cookbook là một yếu tố rất quan trọng và có thể coi là một thành phần riêng biệt (cùng với máy chủ, máy trạm và các nút). Nhìn chung, cookbook được viết ra và quản lý bởi các máy trạm, sau đó được chuyển lên server; từ server chúng được đẩy xuống các nút bằng chef-client trong mỗi lần chef-client thực thi.

2.2.3 Các thành phần chính

Phần dưới đây sẽ cho chúng ta biết chi tiết hơn về các thành phần đã kể trên

Các nút

Một nút trong hệ thống của Chef có thể là một máy chủ vật lý, máy chủ ảo hoặc là một máy chủ trên mây, hoặc một thiết bị mạng. Chúng được cấu hình và duy trì bởi chef-client.

Một nút dựa trên đám mây thường được lưu trữ trong một số dịch vụ dựa trên điện toán đám mây, chẳng hạn như Amazon Virtual Private Cloud, OpenStack, Rackspace, Google Compute Engine, Linode, hoặc Windows Azure. Knife là công cụ của người quản trị hệ thống[7], nó đã hỗ trợ sẵn các dịch vụ này. Knife có thể sử dụng các plugin để tạo ra các máy chủ trên hệ thống đám mây. Sau khi máy chủ đó được tạo ra, chef-client có thể được sử dụng để triển khai, cấu hình và duy trì những máy chủ này.

Một nút vật lý thường là một máy chủ hoặc một máy ảo, nhưng nó có thể là bất kỳ thiết bị gắn liền với hệ thống có khả năng gửi, nhận, và chuyển tiếp

[7]Trong kiến trúc của chef thì knife có nghĩa là người quản trị hệ thống

thông tin trên một kênh giao tiếp. Nói cách khác, một nút vật lý là bất kỳ thiết bị hoạt động nào gắn liền với hệ thống mà có thể chạy chef-client và cũng cho phép chef-client giao tiếp với máy chủ.

Các nút ảo thường mà các máy chủ ảo được tạo ra nhờ các phần mềm ảo hóa, cho dù vậy chúng có tính chất tương đương một nút vật lý.

Một nút mạng thường là các thiết bị mạng như swicth, router, VLAN hay những thứ tương tự mà có thể quản lý bằng chef-client.

Các thành phần quan trọng của nút bao gồm:

Chef-client

Chef-client là một agent chạy cục bộ trên tất cả các nút đã được đăng kí với máy chủ. Khi chef-client chạy, nó sẽ thực hiện tất cả các bước được yêu cầu để đưa nút vào trạng thái mong muốn. Những việc này bao gồm:

- Đăng ký và chứng thực các nút với máy chủ
- Xây dựng các đối tượng nút
- Đồng bộ hóa các cookbook
- Biên dịch các tập hợp tài nguyên bằng cách tải về cookbook cần thiết, bao gồm cả recipes, các thuộc tính, cùng tất cả phụ thuộc khác
- Thực hiện các hành động thích hợp và cần thiết để cấu hình các nút
- Tìm kiếm ngoại lệ và các thông báo, và xử lý mỗi khi có yêu cầu

Cặp khóa công khai RSA được sử dụng để xác thực các chef-client với máy chủ mỗi khi chef-client cần truy cập vào dữ liệu được lưu trữ trên máy chủ. Điều này ngăn cản bất kỳ nút nào truy cập dữ liệu mà nó không nên truy

cập và đồng thời cặp khóa này đảm bảo rằng chỉ có các nút đã được đăng ký với máy chủ có thể quản lý được các tài nguyên.

Ohai

Ohai là một công cụ được sử dụng để phát hiện các thuộc tính trên một nút, sau đó cung cấp những thuộc tính này cho chef-client tại mỗi lần chạy. Ohai là thành phần bắt buộc phải có của chef-client và nhất thiết phải có mặt trong một nút. Các loại thuộc tính mà Ohai thu thập bao gồm:

- Thông tin chi tiết về nền tảng
- Thông tin về mức sử dụng tài nguyên mạng
- Thông tin về mức sử dụng bộ nhớ
- Thông tin về mức sử dụng CPU
- Các thông tin về nhân hệ điều hành
- Tên của máy chủ
- Tên miền đầy đủ của máy chủ
- Chi tiết cấu hình khác

Những thuộc tính được thu thập bởi Ohai là các thuộc tính tự động, trong đó các thuộc tính được sử dụng bởi các chef-client để đảm bảo rằng những thuộc tính này vẫn không thay đổi sau khi các chef-client được thực hiện cấu hình các nút.

Máy trạm

Một máy trạm là một máy tính được cấu hình để chạy Knife. Nó cũng dùng để đồng bộ hóa với các chef-repo, cũng tương tác với một máy chủ duy

nhất. Vị trí của máy trạm là nơi mà từ đó người dùng sẽ làm hầu hết công việc của họ, bao gồm:

- Phát triển các cookbook và recipes (sử dụng ngôn ngữ lập trình Ruby)
- Đồng bộ hóa với chef-repo sử dụng hệ thống quản lý mã nguồn (Git hoặc SVN)
- Sử dụng Knife để tải lên cái item từ chef-repo lên server
- Cấu hình các chính sách của tổ chức, bao gồm cả việc định nghĩa các vai trò (roles) và môi trường, đồng thời đảm bảo việc dữ liệu quan trọng đã được lưu lại trong data bags
- Tương tác với các nút khi cần thiết, chẳng hạn thiết lập bootstrap cho một nút.

Knife

Knife là một công cụ dòng lệnh cung cấp một giao diện giữa các chef-repo cục bộ tại máy trạm và máy chủ. Knife giúp cho người dùng có thể quản lý:

- Các nút
- Cookbooks và recipes
- Roles
- Lưu trự dữ liệu JSON (data bags), bao gồm cả dữ liệu mã hóa
- Các thiết lập môi trường

- Cái tài nguyên của điện toán đám mây, bao gồm cả tài nguyên dự phòng
- Cài đặt các chef-client trên máy trạm quản lý
- Tìm kiếm các thông tin được index trên máy chủ

Chef-Repo

Chef-repo là nơi mà các đối tượng dữ liệu sau được lưu trữ:

- Cookbooks (bao gồm cả recipes, phiên bản, các thuộc tính, tài nguyên, nhà cung cấp, thư viện và các mẫu)
- Roles
- Databags
- Các thông số về môi trường làm việc
- Các file cấu hình (cho máy khách, máy trạm và máy chủ)

Chef-repo nằm trên một máy trạm và cần được đồng bộ hóa với hệ thống quản lý mã nguồn, chẳng hạn như git. Tất cả các dữ liệu trong chef-repo nên được đối xử như là mã nguồn.

Knife được sử dụng để upload ữu liệu từ chef-repo lên máy chủ. Sau khi tải lên, dữ liệu được các chef-client sử dụng để quản lý tất cả các nút đã được đăng kí, đồng thời đảm bảo tất cả những thứ như cookbooks, cái thống số môi trường, roles và các thiết lập khác được áp dụng cho nút đó một cách đứng đắn.

Con dao được sử dụng để upload dữ liệu đến máy chủ từ các đầu bếp-repo. Sau khi tải lên, dữ liệu được sử dụng bởi các đầu bếp-client để quản lý tất cả các nút đã được đăng ký với máy chủ và để đảm bảo rằng các sách dạy nấu

ăn đúng, môi trường, vai trò, và các thiết lập khác được áp dụng cho các nút một cách chính xác.

Máy chủ

Các máy chủ hoạt động như một trung tâm chứa dữ liệu cấu hình. Các máy chủ lưu trữ các cookbooks, chính sách sẽ được áp dụng cho các nút, và các metadata được quản lý bởi chef-client. Các nút sử dụng chef-client để yêu cầu máy chủ các thông tin chi tiết về cấu hình, chẳng hạn như recipes, templates hay các tệp tin. chef-client thực hiện các công việc của mình trên từng nút mà không phải là trên máy chủ. Cách tiếp cận theo hướng phân phối mở rộng này khá tương ứng với mô hình của Puppet.

Hosted Enterprise Chef là một phiên bản máy chủ của Chef. Hosted Enterprise Chef được xây dựng dựa trên điện toán đám mây, có khả năng mở rộng và luôn sẵn sàng 24x7/365. Hosted Enterprise Chef có thể quản lý bất kì máy chủ nào mà không yêu cầu nó phải được thiết lập và quản lý từ phía sau tường lửa.

Cookbooks

Cookbook là đơn vị cơ bản của Chef trong việc cấu hình và phân phối các chính sách. Mỗi cookbook định nghĩa một kịch bản, chẳng hạn như tất cả mọi thứ cần thiết để cài đặt và cấu hình MySQL, và sau đó nó chứa tất cả các thành phần được yêu cầu để hỗ trợ kịch bản đó, bao gồm:

- Các giá trị của thuộc tính được thiết lập trên các nút
- Định nghĩa sự cho phép tạo ra hoặc sử dụng lại tập hợp các tài nguyên
- Các thư viện dùng để mở rộng chef-client hoặc cung cấp sự trợ giúp bằng mã nguồn Ruby

- Recipes là cách thức xác định các tài nguyên cần thiết để quản lý và thứ tự của chúng khi áp dụng
- Các mẫu
- Các phiên bản
- Các siêu dữ liệu về cách thức (bao gồm cả các gói phụ thuộc), hạn chế phiên bản, nền tảng hỗ trợ hay những gì tương tự.

Chef-client sử dụng ngôn ngữ Ruby như một ngôn ngữ tham chiếu để tạo ra cookbook và xác định recipes, với DSL mở rộng cho các tài nguyên cụ thể. Chef-client cung cấp một tập hợp lý các tài nguyên, đủ để hỗ trợ rất nhiều các kịch bản tự động hóa cơ sở hạ tầng phổ biến nhất. Tuy nhiên, DSL này cũng có thể được mở rộng khi có thêm tài nguyên và khả năng được yêu cầu.

2.2.4 Cài đặt và sử dụng

Cách cài đặt Chef (server và client)

Hiện tại Chef-Server chỉ hỗ trợ gói cài đặt cho RHEL và Ubuntu. Chef-Client hỗ trợ đa nền tảng hơn hơn và có script cài đặt đi kèm.

- **Cách cài đặt Chef trên RHEL**

Đối với Chef-Server:

```
# Install chef-server
rpm -ivh https://opscode-omnibus-packages.s3.amazonaws.com/el/6/x86_64/
    chef-server-11.0.10-1.el6.x86_64.rpm
# Initial configuration for chef-server
chef-server-ctl reconfigure
```

Mã nguồn 2.8: Cách cài đặt chef-server trên RHEL

Đối với Chef-Client:

```
# Install chef-client
curl -L https://www.opscode.com/chef/install.sh | bash
```

Mã nguồn 2.9: Cách cài đặt chef-client trên RHEL

- **Cách cài đặt Chef trên Ubuntu**

Đối với Chef-Server:

```
# Download chef-server package
wget https://opscode-omnibus-packages.s3.amazonaws.com/ubuntu/12.04/
    x86_64/chef-server_11.0.10-1.ubuntu.12.04_amd64.deb
# Install chef-server package
sudo dpkg -i chef-server_11.0.10-1.ubuntu.12.04_amd64.deb
# Initial configuration for chef-server
sudo chef-server-ctl reconfigure
```

Mã nguồn 2.10: Cách cài đặt chef-server trên Ubuntu

Đối với Chef-Client:

```
# Install chef-client
curl -L https://www.opscode.com/chef/install.sh | bash
```

Mã nguồn 2.11: Cách cài đặt chef-client trên Ubuntu

Chú ý: Chef-Server cần mở cổng 443 để các Chef-Client có thể liên lạc được với nó.

- **Cách cài đặt Knife trên máy trạm**

Như đã nói trong kiến trúc của Chef, cần có một máy trạm chứa công cụ Knife và chef-repo để đưa các lệnh tới máy chủ và các nút.

Vì vậy chúng ta phải cài đặt chef-client trên máy trạm này, cài đặt công cụ knife và đồng bộ chef-repo.

```
# Install chef-client
curl -O -L http://www.opscode.com/chef/install.sh | bash

# Clone the Chef Repo skeleton directory to work in:
cd ~/Development
git clone https://github.com/opscode/chef-repo.git
```

Mã nguồn 2.12: Cách cài đặt knife và chef-repo

Cách cấu hình cơ bản hệ thống Chef

Các bước cấu hình của Chef phức tạp hơn so với Puppet khá nhiều, ở đây chúng ta chỉ giới thiệu một số bước cấu hình đơn giản. Chi tiết về các cấu hình còn lại có thể tham khảo trên trang tài liệu của dự án: http://docs.opscode.com

```
# Config knife
knife configure
...

# List chef-client (nodes)
knife client list

# Bootstrap first client server
knife bootstrap -u $USERNAME --sudo $FQDN_OF_CLIENT_SERVER
```

Mã nguồn 2.13: Các bước cấu hình hệ thống Chef

Tóm lại

Các nguyên tắc cơ bản quan trọng của Chef là chúng ta (những người sử dụng) là những người biết rõ nhất về những gì môi trường của bản thân, những gì cần làm, và làm thế nào để duy trì được nó. Chef-client được thiết kế để không phá vỡ những điều như vậy. Và chỉ có chúng ta cũng như đồng nghiệp của chúng ta mới hiểu rõ ràng các vấn đề về kĩ thuật cũng như phải làm gì để giải quyết chúng. Chỉ chúng ta mới có thể hiểu được các vấn đề về con người(trình độ kỹ năng, những kĩ năng kiểm soát, và các vấn đề nội bộ khác), những thứ mà duy nhất chỉ doanh nghiệp hay tổ chức của bạn biết được giải pháp.

Ý tưởng ở đây là bạn nắm rõ trong tay những gì sẽ xảy ra với hệ thống của bạn, và cách như thế nào để làm nó hoạt động hiệu quả. Tuy nhiên, một cá nhân rất khó có thể nắm bắt được mọi thứ về một vấn đề phức tạp, cũng như các bước để giải quyết chúng. Vấn đề cũng tương tự với những công cụ này. Chef hỗ trợ chúng ta quản lý các cơ sở hạ tầng. Chef có thể giúp giải quyết các vấn đề phức tạp. Chef có một cộng đồng rộng lớn, nơi mọi người có rất nhiều các kinh nghiệm trong việc giải quyết các vấn đề phức tạp và họ có thể cung cấp các kiến thức hay hỗ trợ chúng trong lĩnh vực mà tổ chức chúng ta chưa có kinh nghiệm. Cộng đồng cũng với Chef có thể giúp doanh nghiệp hay tổ chức của chúng ta giải quyết các vấn đề phức tạp.

2.3 Ansible

Ansible là một giải pháp đơn giản được sử dụng trong việc tự động hóa hàng loạt các công việc liên quan đến hạ tầng CNTT như tự động cấu hình, tự động triển khai phần mềm, và nhiều công việc khác nữa. Trong mô hình của Ansible, hạ tầng CNTT của bạn được nhìn góc độ là một kiến trúc tổng thể của các thành phần có liên quan thay vì chỉ quản lý một hệ thống tại một điểm riêng lẻ. Nó không sử dụng các agent hoặc thêm vào các lớp tùy chọn bảo mật bổ sung, do đó việc triển khai Ansible vô cùng đơn giản. Một điều quan trọng nữa đó là ngôn ngữ cấu hình mà nó sử dụng rất đơn giản (được gọi là playbooks), nó cho phép mô tả những công việc tự động bằng tiếng Anh đơn thuần thay vì viết một điều gì đó phức tạp bằng một ngôn ngữ lập trình nào đó. Bằng cách sử dụng Ansible, chúng ta sẽ thực hiện việc tự động hóa hàng loạt nhanh hơn, thậm chí nó có thể đạt tới những điều mà ta chưa từng thấy trước đó.

2.3.1 Tổng quan

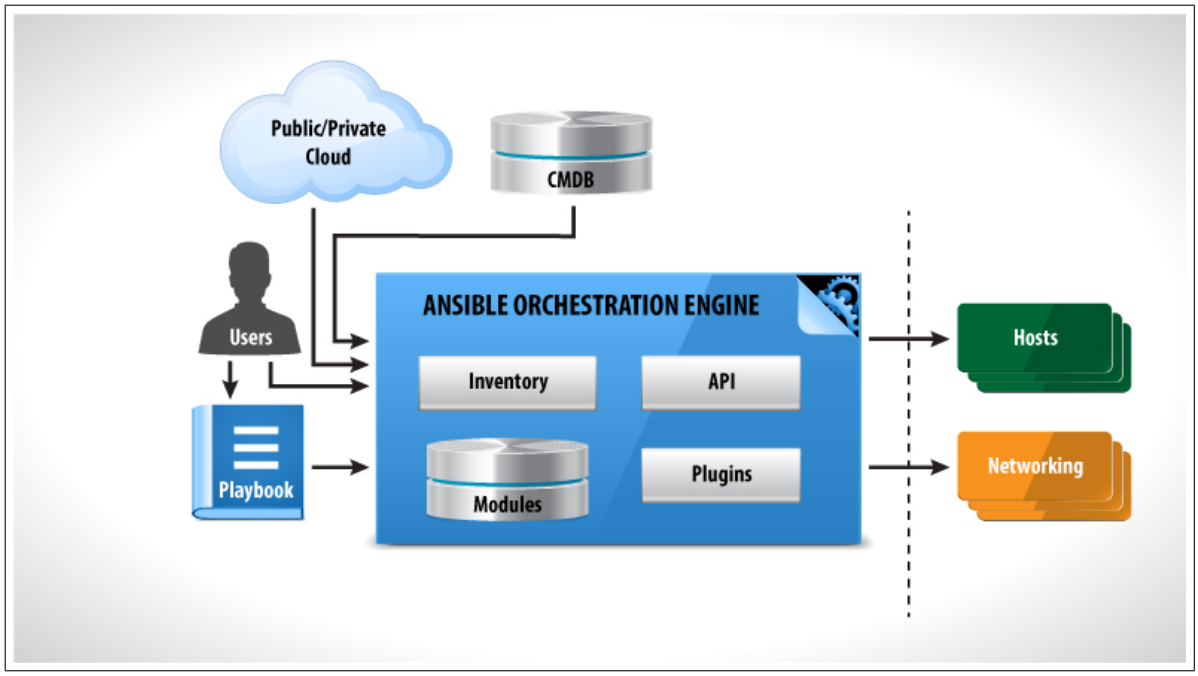
Ansible là một phần mềm mã nguồn mở dùng trong việc quản lý cấu hình của hạ tầng CNTT, triển khai sản phẩm công nghệ, cùng như điều phối các hoạt động tự động hóa khác. Với chỉ một công cụ duy nhất, nó đem lại những hiệu quả lớn trước hàng loạt các thách thức về tự động hóa. Ansible cung cấp sự thay thế hiệu các chức năng cốt lõi vốn có trong các giải pháp tự động hóa khác, đồng thời tìm kiếm lời giải cho những vấn đề chưa được giải quyết. Bao gồm sự phối hợp rõ ràng về các quy trình làm việc phức tạp và sự thống nhất về cấu hình của hệ điều hành và phần mềm ứng dụng triển khai.

Ansible tìm cách giữ cho những mô tả về các quy trình công việc dễ hiểu

và có thể được thực hiện nhanh chóng. Điều đó đồng nghĩa với việc những người mới sử dụng Ansible có thể nhanh chóng hòa nhập vào các dự án mới, đồng thời dễ dàng hiểu được những công việc cho dù họ có tham gia vào dự án muộn hơn. Không chỉ vậy, Ansible luôn tìm cách tạo ra những công cụ thật mạnh mẽ cho những chuyên gia, nhưng bình đẳng cho tất cả các cấp độ kỹ năng của người sử dụng. Từ đó, rút ngắn thời gian đưa sản phẩm ra thị trường; giảm thiểu các lỗi có thể xảy ra do sự thay đổi cấu hình của hạ tầng CNTT.

Ansible được thiết kế nhỏ gọn, tiện dụng, an toàn, và có độ tin cậy cao, không mất nhiều công sức trong việc học sử dụng cho dù là quản trị viên, nhà phát triển hay nhà quản lý.

2.3.2 Kiến trúc hệ thống



Hình 2.6: Kiến trúc hệ thống của Ansible

Một trong những khác biệt chính giữa Ansible với những sản phẩm cùng loại chính là kiến trúc của nó

- Mặc định, Ansible quản lý các máy trạm thông qua giao thức SSH. Nó sử dụng một thư viện được gọi là **paramiko** (được viết bằng lập trình Python[8]) hoặc sử dụng ngay OpenSSH của hệ điều hành.

- Ansible có thể truyền tải theo khác nhau, các phương thức là có thể thay đổi được. Ví dụ: Mặc dù **0mq** - một phương thức truyền tải tăng tốc (accelerated transport) được đưa ra nhưng Ansible hỗ trợ cả phương thức không sử dụng mạng.

- Ansible không yêu cầu quyền root[9] để truy cập mà nó có thể cấu hình để dùng sudo[10] trong các trường hợp cần thiết.

- Ansible không yêu cầu một khóa SSH cụ thể hay một người dùng riêng. Nó có thể làm việc với bất cứ người sử dụng được cung cấp, nghĩa là Ansible tôn trọng quyền truy cập của hệ điều hành của bạn.

- Khi có yêu cầu, Ansible sẽ chuyển các module cần thiết tới các nút điều khiển, sau đó chạy chúng từ xa với các thông tin người dùng đã được cung cấp và không để lại được bất cứ cài đặt gì trên các nút này.

- Ansible không yêu cầu bất kỳ phần mềm máy chủ được chạy từ một máy quản lý, nó chỉ yêu cầu các thông tin đăng nhập của người dùng mà thôi.

[8] https://en.wikipedia.org/wiki/Python_(programming_language)

[9] account có quyền tuyệt đối với hệ thống trên Linux, BSD hay Solaris

[10] https://en.wikipedia.org/wiki/Sudo

- Ansible không yêu cầu bất kì một phần mềm agent nào được chạy trên cái nút điều khiển.
- Ansible không cần mở thêm bất cứ một cổng nào ngoài SSH cũng như không yêu cầu phải có hạ tầng PKI để bảo trì.
- Những người có quyền truy cập vào máy chủ điều khiển (hoặc máy chủ điều khiển mã nguồn) không thể xóa hay thay đổi các nội dung các máy trạm (hoặc ra lệnh cho chúng chạy một lệnh nào đó) mà không có cũng có thông tin về hệ thống đó.
- Khi không còn quản lý nữa, Ansible không dùng đến bất cứ tài nguyên nào của những máy trạm.

Những đặc điểm trên kết hợp với nhau làm cho Ansible trở nên lý tưởng cho các môi trường bảo mật hoặc hiệu suất cao, nơi có những lo ngại về sự ổn định hoặc việc thay đổi thường xuyên của các agent. Những thuộc tính trên hầu hết đều hữu ích trong các lĩnh vực về máy tính.

Ansible là sự thiết kế đồng nhất giữa kinh nghiệm của sử dụng và phương pháp tiếp cận. Ansible được thiết kế để làm cho việc cấu hình và xử lý các hạ tầng CNTT cũng chỉ đơn giản như việc đọc hoặc viết cấu hình, thậm chí đối với những người chưa qua đào tạo về việc đọc chúng.

Mặc dù Ansible có thể thực hiện hầu hết các loại nhiệm vụ tự động hóa, Ansible không giống với ngôn ngữ lập trình phần mềm, nó chỉ là các mô tả cơ bản về trạng và tiến trình. Hơn nữa, nó cố gắng để giải quyết nhiều vấn đề chồng chéo của tự động hóa hệ thống từ một framework duy nhất với mục tiêu giảm thiểu thời gian và chi phí để học và hiểu cũng như gắn kết nhiều framework với nhau.

Với các phương pháp truyền thống khác, người dùng thường phải kết hợp nhiều công cụ với nhau có để bao quát hết những điều cơ bản trong quản lý hệ thống CNTT và cấu hình phần mềm, bao gồm:

- Một công cụ dùng quản lý cấu hình, nó dùng để làm việc với các hệ điều hành cơ bản, mô tả các trạng thái mong muốn của một hệ thống, nhưng không phải quá trình để đưa nó vào trạng thái đó.
- Một công cụ triển khai, dùng để đưa sản phẩm phần mềm lên hệ thống, nó tập trung vào quá trình thực hiện.
- Một công cụ dùng để thực thi, cho các tác vụ thực thi tức thời - những thứ mà không phù hợp với mô hình trước đây, chẳng hạn như restart hàng loạt các máy chủ.

Ansible đã gộp tất cả các yếu tó đó thành một công cụ duy nhất, đồng thời cung cấp các khả năng và đặc điểm cho phép thực hiện triển khai những phần mềm nhiều lớp và các quy trình làm việc phức tạp.

2.3.3 Mô hình sắp xếp công việc

Để có thể hiểu được mô hình sắp xếp công việc (Modeling Orchestration Workflows) của Ansible, chúng ta lấy xem xét một ví dụ cụ thể về một ứng dụng web 3 lớp truyền thống bao gồm các thành phần:

- Các máy chủ ứng dụng
- Các máy chủ cơ sở dữ liệu
- Các máy chủ nội dung

- Hệ thống cân bằng tải
- Hệ thống giám sát được kết nối với hệ thống cảnh báo như một dịch vụ thông báo
- Một hệ thống tích hợp liên tục[11]

Theo như ví dụ trên, Ansible có thể xử lý một cách đơn giản theo mô hình như sau:

1. Tư vấn cách cấu hình và cài đặt kho chứa cho thông tin về các máy chủ tham gia.
2. Cấu hình hệ điều hành cơ sở trên tất cả các máy và thực hiện các công việc sao cho tất cả chúng có cùng một trạng thái mong muốn.
3. Xác định thành phần của các máy chủ ứng dụng web để cập nhật.
4. Đưa ra tín hiệu báo cho hệ thống giám sát chuyển các servers vào trạng thái offline.
5. Đưa ra tín hiệu báo cho hệ thống cân bằng tải tách các máy chủ ứng dụng ra khỏi hệ thống
6. Triển khai và cập nhật các máy chủ ứng dụng web
7. Đưa ra tín hiệu báo cho hệ thống cân bằng tải đưa các máy chủ ứng dụng này trở lại hệ thống.
8. Đưa ra tín hiệu báo cho hệ thống giám sát tiếp tục giám sát các sự cố trên các máy chủ này.

[11] Continuous integration: https://en.wikipedia.org/wiki/Continuous_integration

9. Lặp đi lặp lại quá trình này cho tới khi tất cả các máy chủ ứng dụng còn lại được cập nhật.

10. Lặp lại các quá trình cập nhật lần lượt này cho các lớp khác như máy chủ cơ sở dữ liệu hay máy chủ nội dung.

11. Gửi thư điện thử thông báo và giải trình khi quá trình cập nhật hoàn thành.

2.3.4 Các thành phần chính

Modules

Các module của Ansible là nguồn tài nguyên được phân phối cho các nút ở xa để yêu cầu chúng thực hiện một số nhiệm vụ hoặc đưa nút đó một vào một trạng thái thích hợp theo yêu cầu. Ansible luôn đi theo triết lý "batteries included", vì vậy chúng ta có rất nhiều module tuyệt vời dành cho hầu hết các công việc lõi của ngành CNTT. Điều này có nghĩa là những module này luôn được cập nhật và chúng ta không cần phải cố gắng tìm kiếm cách để cho nó làm việc trên nền tảng của chúng ta. Chúng ta có thể tưởng tượng những module này là các thành phần của một bộ công cụ hữu ích với các công cụ quản lý hệ thống và các playbooks như một hướng dẫn để xây dựng một hệ thống bằng những công cụ này.

Với một số lượng khá lớn module, Ansible có khả năng dễ dàng triển khai những workloads tới một loạt các công nghệ ảo hóa và các dịch điện toán đám mây công cộng hay trên các môi trường tiền điện toán đám mây như VMware, OpenStack, Amazon Web Services EC2 (AWS), Eucalyptus Cloud, KVM, và CloudStack. Những máy chủ có thể được triển khai từ hình ảnh hệ điều hành

cơ sở mà không có bất kỳ sửa đổi, đồng thời được cấu hình đầy đủ chỉ trong một bước.

Không chỉ vậy, hiện nay Ansible được sử dụng rộng rãi trong việc triển khai các hệ thống BigData, các hệ thống lưu trữ và phân tích môi trường bao gồm những nền tảng quan trọng như Hadoop, Riak, và Aerospike.

Trong những môi trường có nhiều máy chủ chưa được cấu hình nhưng chúng phải được cấu hình trước mà không cần bất kì một chương trình agent quản lý. Khách hàng yêu cầu một công cụ điều khiển đơn giản và dễ dàng chỉnh sửa các chính sách cho cả hai việc triển khai và cập nhật các cụm clusters.

Các khu vực khác bên ngoài lĩnh vực BigData cũng có thể tận dụng lợi thế này. Các tính chất này cũng làm cho Ansible hấp dẫn cho hệ thống giám sát lưu lượng truy cập cao tổ chức, nơi Ansible trả lại tất cả tài nguyên CPU có sẵn cho các môi trường máy tính khi không sử dụng. Không có việc lãng phí tài nguyên bộ nhớ hoặc CPU, cũng không cần phải có một daemon để duy trì hoạt động.

Để có thể tìm hiểu chi tiết hơn về chức năng của các module này trong Ansible, chúng ta có thể tham khảo trang tài liệu của Ansible Works. Tại địa chỉ `http://www.ansibleworks.com/docs/modules.html`

Plugins

Có rất nhiều điểm tích hợp có thể được sử dụng để mở rộng Ansible, bao gồm:

- Những module có thể chạy cục bộ hoặc từ xa để cấu hình các ứng dụng, các dịch vụ hoặc các tham số cho hệ điều hành.

- Các phương thức mới ghi nhập các callback cho các chương trình kiểm

soát hoặc báo cáo

- Tích hợp với các nguồn dữ liệu bên ngoài
- Các thông tin dữ liệu của Inventory từ hệ thống CMDB hoặc đám mây
- Các phương thức chuyển vận mới mà từ đó Ansible có thể giao tiếp với các máy mà mình quản lý

Cac đơn vị công việc trong ansible được đảm bảo bởi các module, đó là những chương trình nhỏ chạy trên máy chủ ở xa để đảm bảo rằng máy chủ đó được đưa vào trạng thái chỉ định phù hợp. Các module này có thể viết bằng ngôn ngữ bất kì chẳng hạn như Python, Perl, Ruby hay bash script. Vì vậy, người dùng có thể sử dụng công cụ yêu thích của họ để tạo ra các module cho bản thân mình hay những người khác.

Mặc định, các module lõi đã được tạo sẵn, có nghĩa là chúng giúp hệ thống có được một trạng thái mong muốn, và nếu không có thay đổi trạng thái được thực hiện, chúng không làm gì ảnh hưởng tới hệ thống cả. Ansible cũng làm cho nó trở nên dễ dàng để mô hình hóa các quá trình được không tạo sẵn, và cũng làm cho nó có thể chỉ đơn giản là chỉ cần đẩy ra kịch bản đơn giản và chạy chúng như mong muốn. Sử dụng các module có sẵn là thích hơn, tuy nhiên, nhưng điều quan trọng là để có thể để mô hình bất kỳ loại quá trình nào chứ không chỉ một mà rơi vào những giới hạn của chúng. Ansible luôn thực tế trong vấn đề này.

Playbooks

Trong Ansible, mọi công việc từ cấu hình hệ điều hành cơ sở đến mô hình hóa quá trình cập nhật hay thực thi ngay lập tức các công việc trên một máy chủ ở xa đều sử dụng chung một công cụ. Những cấu hình cụ thể được viết ra

trong Ansible được gọi là "Playbooks". Đó một dạng mã mà cả con người và máy tính đều có thể phân tích được (một dạng mã YAML[12]), đồng thời làm cho việc hợp tác với những chương trình khác dễ dàng hơn. Và quan trọng nhất là dễ đọc và hiểu đối với cả những người không phải là nhà phát triển.

Playbooks làm cho việc cấu hình hệ thống hay triển khai hệ thống trên nhiều máy trở lên đơn giản hơn bất kỳ công cụ nào đã có. Playbooks rất phù hợp trong việc triển khai các ứng dụng phức tạp.

Playbooks có thể khai báo cấu hình, nhưng nó cũng có thể dàn xếp các bước bất kì của một quá trình thủ công, thậm chí là cả những bước khác nhau khi mà chúng phải nhảy qua nhảy lại giữa một tập hợp các máy chủ với các nhiệm vụ khác nhau. Playbooks có thể thực hiện các công việc một cách đồng bộ hoặc không đồng bộ.

Trong khi bạn có thể chạy chính ansible cho các nhiệm vụ đột xuất, playbooks có thể được giữ trong các hệ thống kiểm soát mã nguồn và được sử dụng để đẩy đi các cấu hình hoặc đảm bảo cấu hình của hệ thống từ xa được giữ trong spec của nó.

Playbooks được thể hiện trong định dạng YAML với cú pháp được tối thiểu hóa. Những nhà phát triển Ansible đã hết sức cố gắng để làm cho nó không giống một ngôn ngữ lập trình hay script mà là giống một dạng mô hình cấu hình hay một quá trình.

Mỗi playbooks bao gồm một hay nhiều **play** trong danh sách

Mục tiêu của mỗi play là ánh xạ một nhóm các máy chủ với một số vai trò xác định, đại điện cho những điều mà Ansible gọi là nhiệm vụ (tasks). Ở mức độ cơ bản, một nhiệm vụ không gì hơn là gọi đến một module của Ansible

Bằng cách gộp nhiều plays, Playbooks có thể dàn xếp việc triển khai trên

[12] http://www.ansibleworks.com/docs/YAMLSyntax.html

nhiều máy, chạy một số bước nhất định trên nhóm máy chủ web, sau đó một số khác trên nhóm máy cơ sở dữ liệu, sau đó lại là một nhóm lệnh khác trên các máy chủ web ... cứ vậy cho đến khi hoàn thành.

"plays" ít hoặc nhiều có những nét tương đồng với những môn thể thao. Chúng ta có có thể có khá nhiều plays ảnh hưởng đến hệ thống của chúng ta theo nhiều cách khác nhau. Điều đó nghĩa là nếu chúng ta định nghĩa ra một mô hình hoặc một trạng thái, chúng ta có thể chạy nhiều plays khác nhau tại các thời điểm khác nhau.

Dưới đây là một ví dụ về một playbooks chỉ chứa 1 plays:

```
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: pkg=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running
    service: name=httpd state=started
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

Mã nguồn 2.14: Ví dụ về playbooks của Ansible

Trong playbooks trên, chúng ta khai báo một cụm **webservers** ở cổng 80 và có số client tối đa là 200, sử dụng user root để điều khiển. Playbooks cũng thiết lập một số công việc và trạng thái mong muốn với nó:

- Đảm bảo chắc chắn rằng Apache httpd server là phiên bản mới nhất
- Đảm bảo config của Apache **/etc/httpd.conf** giống như template có sẵn ở **/srv/httpd.j2**. Đồng thời thiết lập thông báo khi Apache restart.
- Đảm bảo rằng Apache đã được khởi động

2.3.5 Cách cài đặt và sử dụng

Ansible có thể cài đặt và chạy trên bất cứ máy tính nào có cài đặt Python 2.6. Hầu hết các nền tảng đều hỗ trợ mặc định Python trong hệ thống bao gồm cả RHEL, CentOS, Debian, Ubuntu hoặc bất cứ bản phân phối Linux nào hỗ trợ Python; Mac OS X, các hệ điều hành thuộc họ BSD ...

Cách cài đặt Ansible trên máy điều khiển

• Cài đặt từ mã nguồn

Ansible là rất dễ dàng để cài đặt từ mã nguồn, nó không yêu cầu sử dụng quyền root và cũng không có bất cứ phần mềm nào được sử dụng để cài Ansible ngoài chính nó. Không cần phải có daemon hoặc cơ sở dữ liệu nào cho việc cài đặt. Vì thế, nhiều người trong cộng đồng của Ansible luôn sử dụng phiên bản đang được phát triển để họ có thể tận dụng lợi thế của các tính năng mới khi chúng đang được thực hiện, và cũng dễ dàng đóng góp vào dự án. Đi theo các phiên bản đang phát triển luôn là cách nhanh và dễ dàng nhất để theo đuổi một dự án phần mềm nguồn mở.

```
$ git clone git://github.com/ansible/ansible.git
$ cd ./ansible
$ source ./hacking/env-setup
```

Mã nguồn 2.15: Cài đặt Ansible từ mã nguồn

Nếu máy điều khiển chưa được cài đặt **pip** thì chúng ta cần phải cài đặt pip, sau đó cài thêm một số moudule python cần thiết cho Ansible

```
# Install pip tool
$ sudo easy_install pip
# Install python modules
$ sudo pip install paramiko PyYAML jinja2 httplib2
```

Mã nguồn 2.16: Cài đặt các module cần thiết cho Ansible bằng pip

- **Cài đặt thông qua Yum trên RHEL, CentOS**

Ansible luôn có sẵn trong kho phần mềm EPEL[13] 6 của Fedora, tuy nhiên các dòng sử dụng trình quản lý gói rpm đều có thể sử dụng được.

Ansible có thể chạy được trên cả các phiên bản trước đó EL5 chứa Python 2.4 hoặc cao hơn.

```
# Install EPEL repo
$ rpm -ivh https://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release
   -6-8.noarch.rpm
# Install ansible with dependencies modules
$ sudo yum install ansible
```

Mã nguồn 2.17: Cài đặt Ansible từ mã nguồn

Chúng ta cũng có thể tự build gói rpm cho chính mình. Từ thư mục chủ của mã nguồn Ansible, chúng ta gõ lệnh **make rpm** để build gói rpm mà chúng ta có thể cài đặt bằng yum; để build được bạn cần cần cái gói ***rpm-build***, ***make***, ***python2-devel***.

```
$ git clone git://github.com/ansible/ansible.git
$ cd ./ansible
$ make rpm
$ sudo rpm -Uvh ~/rpmbuild/ansible-*.noarch.rpm
```

Mã nguồn 2.18: Tự build gói rpm của Ansible

[13] https://fedoraproject.org/wiki/EPEL

- **Cài đặt thông qua Apt trên Debian, Ubuntu**

Với distro Ubuntu, Ansible có thể được cài đặt thông qua hệ thống PPA[14] theo các bước sau:

```
$ sudo add-apt-repository ppa:rquillo/ansible
$ sudo apt-get update
$ sudo apt-get install ansible
```

Mã nguồn 2.19: Cài đặt Ansible trên Ubuntu thông qua PPA

- **Cài đặt thông qua pip**

Ansible cũng có thể cài đặt thông qua **pip** - Python package manager.

```
# Install pip - python package manager
$ sudo easy_install pip
# Install Ansible
$ sudo pip install ansible
```

Mã nguồn 2.20: Cài đặt Ansible thông qua pip

Cấu hình và làm quen với Ansible

Ở phía trên chúng ta đã tìm hiểu cách cài đặt Ansible thông qua một số cách thông dụng, trong phần này chúng ta sẽ tìm hiểu các cấu hình và một số thao tác cơ bản với Ansible.

Để có thể bắt đầu làm việc với cái máy cần điều khiển, chúng ta cần cấu hình các Inventory cho Ansible.

Các thông số về máy chủ cũng như các cụm máy chủ được Ansible lưu trữ ở **/etc/ansible/hosts** với format giống file ini.

```
mail.example.com

[webservers]
foo.example.com
bar.example.com
```

[14]https://launchpad.net/ rquillo/+archive/ansible

```
[dbservers]
one.example.com
two.example.com
three.example.com
```

Mã nguồn 2.21: Cấu hình Inventory của Ansible

Chi tiết về cách cấu hình inventory, chúng ta có thể xem tại đây[15]

Chúng ta có thể sử dụng Ansible để thao tác quản lý file trên máy chủ, quản lý các gói phần mềm, quản lý người dùng, quản lý các dịch vụ và triển khai các sản phẩm từ hệ thống quản lý mã nguồn ...

```
# File transfer
$ ansible atlanta -m copy -a "src=/etc/hostsdest=/tmp/hosts"
$ ansible webservers -m file -a "dest=/srv/foo/b.txtmode=600owner=mdehaan
    group=mdehaan"

# Package Manager
$ ansible webservers -m yum -a "name=acmestate=installed"
$ ansible webservers -m yum -a "name=acmestate=removed"

# User & Group
$ ansible all -m user -a "name=foopassword=<cryptedpasswordhere>"
$ ansible all -m user -a "name=foostate=absent"

# Deploy from source control
$ ansible webservers -m git -a "repo=git://foo.example.org/repo.gitdest=/
    srv/myappversion=HEAD"

# Services
$ ansible webservers -m service -a "name=httpdstate=started"
$ ansible webservers -m service -a "name=httpdstate=stopped"
```

Mã nguồn 2.22: Ansible thực hiện các chức năng thông qua dòng lệnh

[15] http://www.ansibleworks.com/docs/intro_inventory.html

Trên đây là một số làm quen cơ bản với Ansible, chúng ta có thể đọc thêm nhiều ví dụ hơn trong các tài liệu khác của Ansible trong các địa chỉ dưới đây:

- http://www.ansibleworks.com/docs/playbooks_best_practices.html
- http://www.ansibleworks.com/docs/modules.html
- https://github.com/ansible/ansible-examples

Tóm lại

Ansible với một tập hợp của các công cụ đơn giản và gọn nhẹ nhưng đồng thời lại rất mạnh mẽ. Ansible có thể bao quát hết các vấn đề của tự động hóa. Việc học sử dụng hay cấu hình Ansible rất dễ dàng cho dù đối với cả những người chưa từng tiếp xúc với nó. Nó xứng đáng là một công cụ hàng đầu về tự động hóa cho các quản trị hệ thống.

2.4 Tổng kết chương

Với những đặc điểm và kiến trúc đã được nêu trong các mục ở trên, dễ dàng nhận thấy, Puppet hay Chef sẽ hấp dẫn các nhà phát triển hay những hệ thống phát triển có định hướng, Ansible được dùng nhiều hơn cho các nhu cầu của quản trị hệ thống. Giao diện đơn giản và tính hữu dụng của Ansible rất phù hợp những suy nghĩ của người quản trị hệ thống. Và trong một hệ thống với rất nhiều các hệ điều hành họ Linux và Unix, việc triển khai hệ thống sử dụng Ansible trở trên rất dễ dàng và nhanh chóng.

Puppet là framework trưởng thành nhất và cũng gần gũi nhất từ quan điểm khả năng sử dụng, mặc dù Puppet yêu cầu phải có nền tảng kiến thức vững chắc về ngôn ngữ lập trình Ruby. Puppet không ở dạng streamlined như Ansible. Puppet là phương án an toàn nhất đối với các môi trường không đồng nhất, tuy vậy Ansible cũng có thể tốt và phù hợp hơn trong một hệ thống có cơ sở hạ tầng lớn hơn hoặc phức tạp hơn.

Chef khá ổn định và được thiết kế tốt nhưng những tính năng của nó vẫn chưa đạt đến độ chín như của Puppet. Chef có thể gây ra một số khó khăn trong việc sử dụng nó đối với các quản trị hệ thống có ít kinh nghiệm về lập trình, tuy vậy nó cũng có thể phù hợp một cách tự nhiên cho những quản trị hệ thống có đầu óc lập trình.

Bảng 2.1: Bảng so sánh ưu nhược điểm của các công cụ tự động hóa

	Puppet	**Chef**	**Ansible**
Ưu điểm	• Các module được viết bằng ngôn ngữ Ruby	• Cookbooks và recipes có thể tận dụng toàn bộ sức mạnh của ngôn ngữ Ruby	• Các module có thể viết bằng gần như tất cả các ngôn ngữ
	• Push commands cho phép bạn kích hoạt thay đổi ngay lập tức	• Tập trung JSON dựa trên "túi dữ liệu" cho phép các kịch bản thu thập các biên trong thời gian chạy	• Không cần agent để có thể quản lý các client
	• Giao diện web cho phép xử lý báo cáo, kiểm kê và quản lý nút theo thời gian thực	• Giao diện web cho phép bạn tìm kiếm các nút và inventory, xem hành động của nút, và gán Cookbooks, roles, nodes	• Giao diện web cho phép bạn cấu hình người dùng, các nhóm, và các thông tin cần thu thập, và áp dụng Playbooks tới các Inventory

	• Chi tiết và báo cáo chuyên sâu về tình trạng của agnet và cấu hình của nút đó		• Cực kì đơn giản để cài đặt và sử dụng
Nhược điểm	• Yêu cầu phải học Puppet DSL hoặc Ruby • Quá trình cài đặt thiếu việc kiểm tra lỗi và báo cáo lỗi	• Yêu cầu phải có sự hiểu biết nhất định về lập trình Ruby • Hiện tại vẫn thiếu chức năng push command • Tài liệu đôi khi còn mơ hồ	• Không hỗ trợ các máy chủ Windows • Giao diện người dùng web không tự động triển khai cùng Ansible; Các inventory phải nhập vào thủ công
Chi phí	• Phiên bản miễn phí mã nguồn mở • Phiên bản dành cho doanh nghiệp Puppet Enterprise có giá 100$/năm cho mỗi máy	• Phiên bản miễn phí mã nguồn mở • Phiên bản dành cho doanh nghiệp Enterprise Chef: miễn phí cho 5 máy, 120$/tháng cho 20 máy, 300$/tháng cho 50 máy và 600$/tháng cho 100 máy	• Phiên bản miễn phí mã nguồn mở • Phiên bản dành cho doanh nghiệp AWX miễn phí cho 10 máy, 100$ hoặ 250$ một máy một năm tùy theo dịch vụ hỗ trợ

CHƯƠNG 3

TRIỂN KHAI THỰC NGHIỆM

Trong các chương trước, chúng ta đã lần lượt đi tìm hiểu các khái niệm về tự động hóa hệ thống, sau đó là tìm hiểu một loạt các framework tiêu biểu, ở chương này chúng ta sẽ sử dụng Ansible để ứng dụng trong giải quyết một số công việc liên quan đến tự động hóa hệ thống.

Người viết đồ án lựa chọn Ansible vì lý do các lý do sau:

- Ansible rất đơn giản để học và sử dụng.
- Ansible không yêu cầu phải đi kèm một hệ thống phần mềm phức tạp để triển khai.
- Ansible sử dụng ngôn ngữ Python - ngôn ngữ lập trình mà người viết đồ án khá thành thạo.
- Ansbile phù hợp với tư duy của người quản trị hệ thống. Bản thân người viết đồ án cũng là một quản trị hệ thống linux có kinh nghiệm.

3.1 Đặt bài toán

Như chúng ta đã biết, trong tự động hóa hệ thống có rất nhiều các vấn đề cần phải giải quyết. Với nội dung giới hạn của đồ án tốt nghiệp, người viết chỉ có thể triển khai giải quyết một số giới bài toán dựa trên những gì đã tìm hiểu được.

Dưới đây là một bài toán mà người viết lựa chọn để đưa vào phần triển khai thực nghiệm của mình:

"Viết công cụ tự động tạo ra một máy chủ trên nền điện toán đám mây Google Compute Engine (GCE)[1]. Sau đó tự động cài đặt và cấu hình hệ thống LAMP[2]; cùng với đó là tự động triển khai CMS Wordpress phiên bản mới nhất lên trên máy chủ vừa tạo."

3.2 Phân tích

Với bài toán trên, chúng ta có thể phân tích ra những công việc cần phải làm như sau:

1. Tạo một máy chủ ảo trên nền tảng Google Compute Engine.
2. Cài đặt LAMP stack và cấu hình các thông số cần thiết chuẩn bị cho việc triển khai Wordpress
3. Triển khai Wordpress lên hệ thống máy chủ vừa tạo.

Chúng ta sẽ phân tích chi tiết hơn vào từng công việc ở trên.

[1] https://cloud.google.com/products/compute-engine/

[2] https://en.wikipedia.org/wiki/LAMP_stack

3.2.1 Tạo máy chủ trên hệ thống GCE

Ansible những phiên bản gần đây đã hỗ trợ sẵn module dành cho các dịch của Google Compute Engine bao gồm: gce, gce_lb, gce_net, gce_pd

- **Module gce**[3]: Dùng cho việc khởi tạo và xóa bỏ một ***instances*** trên hệ thống GCE
- **Module gce_lb**[4]: Dùng để tạo ra hoặc xóa bỏ một ***loadbalancer*** hay ***httphealthcheck*** trên hệ thống GCE
- **Module gce_net**[5]: Dùng để tạo ra hoặc xóa bỏ các ***networks*** và ***firewall rules*** trên hệ thống GCE.
- **Module gce_pd**[6]: Dùng để tạo mới hoặc xóa bỏ một ***persistent disk***

Với mục đích của chúng ta là tạo ra một instance trên hệ thống GCE, chúng ta sẽ sử dụng module ***gce***

Các thông số cần thiết để tạo một máy chủ trên GCE gồm có:

- **image**: debian-7 hoặc centos-6 (ở đây chúng ta sẽ dùng centos-6)
- **instance_names**: tên của các máy chủ cần tạo
- **machine_type**: loại máy chủ cần tạo
- **zone**: trung tâm dữ liệu nơi lưu trữ máy chủ

[3] http://www.ansibleworks.com/docs/modules.html#gce
[4] http://www.ansibleworks.com/docs/modules.html#gce_lb
[5] http://www.ansibleworks.com/docs/modules.html#gce_net
[6] http://www.ansibleworks.com/docs/modules.html#gce_pd

Tất cả các module của GCE trong Ansible đều sử dụng ***apache-libcloud***, tuy nhiên chỉ từ version 0.13.3 trở nên thì module này mới hỗ trợ các dịch vụ của GCE.

3.2.2 Cài đặt và cấu hình LAMP stack

LAMP stack của chúng ta bao gồm:

- Phần mềm máy chủ web nginx
- Phần mềm cơ sở dữ liệu mysql
- PHP

Vì chúng ta sử dụng CentOS 6 nên chúng sẽ cần phải có thêm repo EPEL để có thể cài đặt một số gói cần thiết cho hệ thống.

Ansible thao tác để cài đặt gói trên máy chủ khá đơn giản, với CentOS, chúng ta sử dụng module yum[7] để làm việc này.

Các gói cần thiết để cài đặt LAMP stack bao gồm:

- nginx
- mysql-server
- php, php-fpm, php-enchant, php-IDNA_Convert, php-mbstring, php-mysql, php-PHPMailer, php-process, php-simplepie, php-xml
- Một số gọi phụ trợ cần thiết: MySQL-python, libselinux-python, libsemanage-python

[7] http://www.ansibleworks.com/docs/modules.html#yum

Các công việc để cài đặt và cấu hình LAMP chuẩn bị cho triển khai CMS Wordpress bao gồm

- Cài đặt máy chủ cơ sở dữ liệu mysql-server
- Cấu hình SELinux cho phép mysql chạy tại bất kì cổng nào
- Tạo cấu hình cho MySQL từ template
- Khởi chạy dịch vụ mysqld
- Cài đặt máy chủ web nginx
- Copy cấu hình đã dựng sẵn của nginx dành riêng cho wordpress lên máy chủ
- Cài đặt cái gói php cần thiết
- Disable default pool của php-fpm
- Copy file cấu hình của php-fpm lên máy chủ
- Restart php-fpm
- Restart nginx
- Cấu hình firewall cho phép truy cập vào webserver

3.2.3 Triển khai CMS Wordpress

Danh sách các công việc cần làm để triển khai CMS Wordpress

- Download mã nguồn của Wordpress
- Extract mã nguồn này vào thư mục của webserver (đã cấu hình từ trước)
- Tạo group và người dùng *wordpress* dùng cho việc chạy wordpress
- Tạo ra một bản ngẫu nhiêu các thông số cho file cấu hình
- Tạo cơ sở dữ liệu
- Tạo người dùng và gán quyền cho cơ sở dữ liệu
- Copy file config của wordpress lên máy chủ
- Chmod lại toàn bộ thư mục mã nguồn tại thư mục root của webserver

Công việc triển khai coi như hoàn tất. Lúc này người dùng vào trang cài đặt của Wordpress và hoàn thành nốt bước cuối cùng.

3.3 Triển khai thực tế

Để triển khai việc tự động hóa bài toán mà chúng ta đã đặt ra, chúng ta sử dụng một playbook có cấu trúc như sau:

```
hosts                    # inventory file
group_vars/              # Variables
   all                   #
   gce                   #
site.yml                 # master playbook
roles/                   #
   common/               #
      files/             #
         epel.repo       # EPEL repo
         iptables-save   # iptables rules
         RPM-GPG-KEY-EPEL-6 # EPEL GPG key
      handlers/          #
         main.yml        #
      tasks/             #
         main.yml        #
   gce/                  # GCE group
      tasks/             #
         main.yml        #
   mysql/                # mysql group
      handlers/          #
         main.yml        #
      tasks/             #
         main.yml        #
      templates/         #
         my.cnf.j2       # MySQL config file
   nginx/                # nginx group
      handlers/          #
         main.yml        #
      tasks/             #
         main.yml        #
```

```
    templates/          #
        default.conf    # config template
  php-fpm/              # php-fpm group
    handlers/           #
        main.yml        #
    tasks/              #
        main.yml        #
    templates/          #
        wordpress.conf  # config template
  wordpress/            # wordpress group
    tasks/              #
        main.yml        #
    templates/          #
        wp-config.php   # Wordpress config file
```

Mã nguồn 3.1: Cấu trúc của playbook dùng để giải quyết bài toán đặt ra

Trong mã nguồn trên, playbook chứa những danh sách những nhiệm vụ cần phải thực hiện, các template cho các cấu hình của từng roles, các handlers hay files ...

Chi tiết nội dung của các mã nguồn, chúng ta có thể tham khảo đĩa CD đi kèm. Ở đây chúng ta chỉ ví dụ cách thực hiện của task gce để tạo ra một instance trên hệ thống GCE.

```
[gce]
127.0.0.1
```

Mã nguồn 3.2: Inventory file - hosts

Ansible sẽ dùng module gce để giao tiếp với hệ thống GCE nên hosts file chỉ cần lấy địa chỉ localhost là đủ.

```
---
- name: Create GCE instances
  hosts: gce
  gather_facts: no
```

```
  roles:
    - gce

- name: Install Wordpress, MySQL, Nginx, and PHP-FPM on GCE instances
  hosts: wordpress-server
  remote_user: mrtux
  sudo: yes

  roles:
    - common
    - mysql
    - nginx
    - php-fpm
    - wordpress
```

Mã nguồn 3.3: Nội dung của site.yml

Master playbook site.yml bao gồm 2 task chính là tạo instance và cài đặt - triển khai LAMP stack cùng Wordpress

```
---
# This playbook will create GCE instances

- name: Launch instances
  local_action:
    module: gce
    instance_names: "{{ names }}"
    machine_type: "{{ type }}"
    image: "{{ image }}"
    zone: "{{ zone }}"
  register: gce

- name: Wait for SSH to be available
  local_action:
    module: wait_for
    host: "{{ item.public_ip }}"
    port: 22
```

```
    delay: 10
    timeout: 60
    state: started
  with_items: gce.instance_data

- name: Add public ip to inventory for another tasks
  add_host: name={{ item.public_ip }} groups=wordpress-server
  with_items: gce.instance_data
```

Mã nguồn 3.4: Nội dung của roles/gce/tasks/main.yml

Khởi tạo instance bằng module gce với các tham số đã được khai báo. Sau đó chờ đến khi instance này hoàn tất quá trình, Ansible lấy thông tin về địa chỉ public ip của máy chủ này đưa vào inventory cho các task sau sử dụng.

Để có thể sử dụng được module gce, chúng ta cần có module apache-libcloud phiên bản mới nhất, cùng với đó là private key trên hệ thống Google Cloud Platform. Chi tiết về cách cấu hình có thể tham khảo ở địa chỉ sau:

- https://github.com/ansible/ansible/blob/devel/test/gce_tests.py
- http://docs.saltstack.com/topics/cloud/gce.html

```
---
# Which version of Wordpress to deploy
wp_version: 3.8
wp_sha256sum:
    c419de49816b6483ab387567222898c02e8acd6ce64d6466a38f2fc05ebefb85

# These are the Wordpress database settings
wp_db_name: wordpress
wp_db_user: wordpress
wp_db_password: Th1sIsMyS3cr3tP@sswordF0rW0rdpress
```

```
# You shouldn't need to change this.
mysql_port: 3306

# This is used for the nginx server configuration, but # access to the
# Wordpress site is not restricted by a # named host.
server_hostname: ansible-demo-site
```

Mã nguồn 3.5: Nội dung của group_vars/all

```
---
# The variables file used by the playbooks for launch GCE instances

names: demo
type: n1-standard-1
#image: debian-7
image: centos-6
zone: us-central1-a
#zone:us-central1-b
#zone:europe-west1-a
#zone:europe-west1-b
#persistent_boot_disk: yes
```

Mã nguồn 3.6: Nội dung của group_vars/gce

Không quá khó để có thể hiểu nội dung của những file cấu hình đã post ở trên, đúng như những gì chúng ta đã thấy trước đâu, ngôn ngữ cấu hình của Ansible cực kì đơn giản và dễ hiểu. Quá trình thực hiện playbook này trên máy của người viết khoảng 5 phút với điều kiện internet tốt.

KẾT LUẬN

Tự động hóa hệ thống tuy không phải một vấn đề mới nhưng luôn là vấn đề đau đầu của những người quản trị hệ thống. Nhưng không giống như trước kia, ngày nay, chúng ta có thể sử dụng rất nhiều các công cụ để làm việc đó thay vì phải làm thủ công bằng các kịch bản truyền thống. Puppet, Chef hay Ansible là những công cụ cực kì hữu dụng mà bất cứ một nhà quản trị hệ thống nào hiện nay đều nên biết và sử dụng chúng trong công việc của mình.

Những kết quả của nghiên cứu này có thể là chưa sâu và bao quát hết được các tính năng của các công cụ đã giới thiệu. Nhưng đồ án này mang lại cái nhìn tổng quan và bao quát cho những ai chưa từng tiếp xúc với tự động hóa hệ thống, để họ có thể hiểu và áp dụng được nó vào thực tế công việc của từng người.

Người viết đồ án với mục đích là nghiên cứu để có thể triển khai trên thực tế các hoạt động tự động hóa cho hạ tầng các sản phẩm công nghệ đã học hỏi và rút ra được một số những kinh nghiệm quý cho bản thân. Trong tương lai, người viết sẽ cố gắng để hiểu sâu và thử nghiệm nhiều hơn nữa các công cụ này trong công việc hiện tại, nhằm nâng cao hiệu quả công việc của bản thân cũng như đồng nghiệp.

TÀI LIỆU THAM KHẢO

[1] James Turnbull and Jeffrey McCune, *"Pro Puppet"*, Apress, pp. 1-5, May 2011.

[2] Amy Brown and Greg Wilson, *"The Architecture Of Open Source Applications, Volume II"*, Lulu.com, pp. 309-322, May 2008.

[3] Grace Walker, *"Cloud computing fundamentals - A different way to deliver computer resources"*, IBM developerWorks, December 2010.

[4] Peter Mell and Timothy Grance, *"The NIST Definition of Cloud Computing"*, NIST Special Publication 800-145, Apr 2012.

[5] Puppet Labs, *"Puppet Labs Documentation"*,
URL: http://docs.puppetlabs.com/

[6] Opscode, *"Chef Documents"*,
URL: http://docs.opscode.com/

[7] Ansible Works, *"Ansible Documents"*,
URL: http://www.ansibleworks.com/docs/

HẾT