入門Chef Solo INFRASTRUCTURE AS CODE



入門 Chef Solo

伊藤直也 著

2013-03-22版 達人出版会 発行

はじめに

近頃のクラウドの本格的普及もあってか、サーバー管理の自動化に注目が集まっています。Chef はそのツール/フレームワークのひとつです。

Chef への注目が集まっているにも関わらず Chef に関するある程度まとまった体系的 な情報はまだまだ不足している、というのが現状です。また Chef は実際には同類のツー ルに比べてシンプルで分かりやすいのですが、公式ドキュメントがあまりにしっかりと書 かれすぎていることもあって「はじめの一歩」としてどの辺りを知ればいいのか、つまり 「普通に使う分にはこの程度知っていれば OK」というのがどの辺りなのかを掴むのが難 しい……というのが筆者の個人的な印象です。

そこではじめの一歩にもちょうど良い、Chef のスタンドアロン版である Chef Solo の 基本的な部分に的を絞って解説した本を作ろうと思い立って書いたのが本書です。

基本的な部分に絞ったとはいえ、筆者が月間数万ユーザーの利用者がいるウェブサービ スに実践投入した経験に基づいて書きましたので、ここで得た知識を応用できる場面は広 いのではないかと思います。

実際に読みながら手を動かして試せるように、試験環境の構築の仕方なども解説してい ます。ぜひ、本書片手に Chef Solo をいじり倒し、遊んでみてください。

2013年3月伊藤直也

本書のポリシー

本書を構成するにあたっては以下の点をポリシーとしました。

- 一通り読めば Chef Solo でサーバー管理の基本操作は行えるまでの範囲をカバー する
- そこそこの時間で読破できるようにページ数を抑える
- 後から参照しやすいよう「○○したい場合」とユースケースベースの見出る
- 読み切りやすいよう一つの話題をブログ記事一つ程度の長さにする
- 気軽に買って読める程度の価格に抑える

システム環境

筆者のシステム環境を参考までに記します。

- 作業環境: OSX 10.8.2
- ・サーバー: Amazon EC2 + Amazon Linux
- 試験環境: Vagrant + CentOS 6.3
- Chef 11.4.0
- knife-solo 0.3.0 (github 版)

フィードバック

本書に関するフィードバックは以下にお寄せ下さい。

- Twitter : @naoya_ito
- ・メール: i.naoya@gmail.com

なお、Chef その他にまつわるテクニカルサポートは一切行っておりません。その類の お問い合わせに関しましては一切お答えいたしませんので予めご了承ください。

目次

はじめに		i
第1章	Chef とは何かを知りたい — Chef Overview	1
第2章	Chef Solo をインストールして試したい — Hello Chef!	8
第3章	nginx を Chef Solo で立ち上げたい	15
第4章	Chef Solo の試験環境を 3 分で用意する — Vagrant	22
第5章	リモートから chef-solo を実行する — knife-solo	28
第6章	レシピを作って実行する流れをおさらいしたい	34
第7章	サードパーティの Chef クックブックを使いたい	37
第8章	代表的なレシピのサンプルを見たい — td-agent のレシピを読む	41
第9章	パッケージをインストールする — Package	51
第 10 章	サービスを起動したい — Service と Notification	54
第 11 章	テンプレートから設定ファイルを配置したい — Template	57
第 12 章	ファイルやディレクトリを扱いたい — Cookbook File, Directory	61
第 13 章	ユーザーを作成したい — User, Group	64

This document is licensed to shinyaa31@gmail.com.

iii

第 14 章	git レポジトリからファイルを取ってくる — Git	66
第 15 章	任意のシェルスクリプトを実行したい — Execute, Script	68
第 16 章	その他の Resource	73
第 17 章	レシピ落ち穂拾い — run_list, ファイル分け, include_recipe	77
第 18 章	Resource を自分で定義したい — Definition	81
第 19 章	Attribute と Data Bag	84
第 20 章	ノードを役割ごとにグルーピングして管理したい — Role	88
第 21 章	サードパーティのクックブックを Bundler 風に管理したい — Berkshelf	91
第 22 章	Chef Server の様子を知りたい ― 概要からセットアップまで	96
第 23 章	どこまでを Chef でやるべきか	112
おわりに		117

第1章

Chef とは何かを知りたい — Chef Overview

Infrastructure as Code

Chef はサーバー設定や更新を自動化するツールです。より凝った言い方をすると「サー バー環境のメタデータを管理しノードの役割・状態を収束させるオペレーションフレーム ワーク」です。



図 1.1 Chef

サーバーの設定管理をやっていると誰もが一度はサーバー構成変更にまつわる「手順書」を書いたことがあると思います。まずは IP アドレスを設定して、ユーザーを作って、 nginx を rpm で入れて、mysql を入れて、LDAP を設定して……。ローカルのメモ、ある いは wiki なんかに残す。その手順書をグループウェアなどで他の開発者と共有して日々 の構成変更に対応するというのが日常だったという人も少なくないはず。

手順書になっているくらいだからほぼそれはルーチンワークなはずです。にも関わらず、サーバー増設や構成変更にあたっては毎回人手でそれを実行していた。どう考えても

非効率ですし、オペレーションミスの温床になるわけです。

Chef はその手順書に書かれたような作業をすべて自動化し、一度コードで記述してし まえばあとは全部コンピュータ任せにできるフレームワークを提供します。

Chef での「手順書のコード」は Ruby で書きます。

例えば nginx をインストールして立ち上げたいなら

```
package "nginx" do
   action :install
end
service "nginx" do
   supports :status => true, :restart => true, :reload => true
   action [ :enable, :start ]
end
```

と書いて対象ホストで Chef を実行する。

すると Redhat 系なら yum レポジトリから、Debian なら apt からあるいは OSX なら homebrew などからパッケージを探してきてインストールして、init スクリプト周り を調整する。これを全部 Chef がやってくれます。

これら手順のコードは「レシピ」と呼ばれます。レシピは Ruby のコードですから、 当然手元の Emacs や vim や Sublime Text…… 好きなエディタで編集することができ ます。

mysql も redis も memcached もあらゆるパッケージ、サーバーの状態に対してレシ ピを用意することで、管理を自動化できます。

できあがったレシピは git に commit してバージョン管理しましょう。github か何かで 同僚と共有すれば、「コードになったサーバーの状態」をみんなで共有しながら煮詰めて いくことができます。

Emacs や vim を使って Ruby でコードを書いて、git に push してあとは Chef に任せ る。何かと人手を介さずには成り立たなかったサーバーやインフラ周りを、Chef を利用 することでまるでアプリケーションのコードを書くかのように、コードによって支配する ことができるようになるのです。

インフラがソフトウェアの一部品であるかのようです。

"Infrastructure as Code" の時代がやってきました。

「サーバーの状態」を管理する

Chef は手順書をコードによって自動化するツールと述べましたが、より本質的には 「サーバーの状態を管理し収束させるためのフレームワーク」です。

例えばとあるサーバーを、サーバーが起動したまっさらな状態から **すべて** Chef で管理 することを想像してみてください。nginx をインストールし、mysql をインストールし、 ネットワークの設定を変更し……。レシピを一つ一つ書いていくと、いずれそれらレシピ 群がそのサーバーの状態そのものを表すコードになりますよね。

こうして蓄積されたレシピ群は、サーバーの今の状態つまりはメタデータを表現した コードです。サーバーにそのレシピを適用すれば、管理者である我々が「こういう状態で あるべき」と定義した状態にサーバー構成が変化する。

定義した「あるべき状態」のレシピと実際のサーバーが異なる状態にあるなら Chef を 動かして、レシピ適用する。すると、そのサーバーは定義された「あるべき状態 (State) に収束 (Convergence)」します。「ソフトウェアのインストールやサーバーの設定変更を 自動化する」というのが Chef の機能のわかりやすい説明ですが、より本質的に捉えるな らそれは「サーバーの状態を管理して、それをあるべき状態に収束させるフレームワー ク」ということです。

……とまあ堅いことを言いましたがとりあえずは「サーバー設定周りを自動化するツー ル」くらいに思っていて構いません。手を動かして慣れてくるうちに、自然とその本質的 概念みたいなものが分かってくるでしょう。

Chef は難しい?

なるほど Ruby でコードを書けば自動化できることは分かった。Chef の思想もなんと なく雰囲気は掴めた。でも自分は Ruby は得意じゃないとか、きちんとしたコードを書く のは自信がないという感想を持った方もいるかも。

「大丈夫だ、問題ない。」

かくいう筆者も Ruby は普段はそんなに書きませんし、今でも Ruby on Rails のこと などはさっぱりわかりません。それで困ることなくレシピをもりもり書いています。

Ruby を書くと言っても、package や template といった DSL を使うだけ。あとは ループや配列の操作くらいを知っていれば OK というのがその理由。Ruby に詳しい必要 はありません。

CFEngine、Puppet、Chef ······

この分野のツールは何も Chef だけではりません。古くは CFEngine、あるいは Puppet などもあります。特に Chef と比較されることの多い Puppet は広く利用されていて実績も十分です。

昨今は Puppet と Chef どちらを選ぶか、というのがこの分野の状況です。優劣はあま りないと思います。Chef の方が Puppet よりもわかりやすいという感想もちらほら耳に しますが、わかりやすさだけがツールの善し悪しではありません。

Chef の方が後発であることもあって、こちらの方が洗練されているとか、なんとなく 最近は Chef の方が話題になりやすい……というようなところがあると思います。

本書では書名通り Chef に絞って解説しますが、もし興味があれば Chef で一通り概念 を覚えた後 Puppet も触れてみると良いのではないでしょうか。Chef だけじゃないよ、 ということだけは覚えておいてください。

Chef の使われ方

Chef は実際どんなところで利用されているのでしょうか。

有名どころといえばやはり Facebook です。Facebook はデータセンター内を自転車 やスクーターで移動するなんて嘘のような本当の話がありますが、そのおそらく数万から 数十万台規模のサーバーを、Chef で管理していると言います。すごいですね。Chef を開 発している Opscode では Facebook から多数のフィードバックを受けて Chef を改良し ているそうです。

先日聞いた話によると、なんでも Amazon EC2 で 10,000 台 のインスタンスを瞬間的 に立ち上げて大規模計算を行い終わったらインスタンスを破棄するという試みがとあるプ ロジェクトで成されて、そこで Chef が使われたそうです。

2013 年 2 月に Amazon Web Services (以下 AWS) で「AWS OpsWorks」というサー ビスが始まりました。AWS OpsWorks は AWS 上の各種サービスを GUI でごりごり設 定していくと簡単に複数サーバーで構成されるシステムを立ち上げたりすることができる というサービスですが、ここでも Chef が使われています。

大規模な話ばかりになってしまいましたが、筆者は3台程度で構成されるウェブサービ

スに Chef を使っています。またサーバーが 1 台のみの環境でも Chef を使っています。 たとえ対象サーバーが小規模でも、インフラ周りをコードで書けるという利点はとても大 きいし 何より楽しいのでもう Chef なしでのサーバー管理は考えづらいです。決して大 げさな感想ではないと思います。

OSX の開発環境を Chef で状態管理することでマシン買い換え時の面倒くさい作業を なくすとか、あるいはチーム開発で使うときの各種ツールのバージョンを合わせる、なん てことをしている人たちもいます。

個人の開発環境から数万台のサーバー管理までを Chef はカバーする、ということで すね。

Chef Server と Chef Solo

ところで Chef と一言にいっても大きくは二つの利用形態があります。

ひとつは Chef Server + Chef Client という、クライアント/サーバーモデルにより大規 模環境を管理するもの。もうひとつはサーバー管理を必要としない単独のコマンドとして Chef を実行する Chef Solo です。Chef Solo は Chef Server + Chef Client のサブセッ トだと思ってください。(なお、Chef Server には Opscode 社が SaaS として提供して いる "Hosted Chef" というサービスとそうではない OSS による配布版があります。)

Chef Server では、Chef Client が HTTPS で Chef Server から必要な情報を GET し て、サーバーに登録された設定をクライアントが受け取って実行するというような PULL 型アーキテクチャを採ります。つまり、システム管理者は Chef Server に命令を投げる ことでそこにぶら下がるすべての Client の状態管理が行えるというもの。スケーラブル なアーキテクチャです。

……ただしちょっと気軽に試せそうな感じはしないですね。筆者のように数台のサー バーしか対象にしないとかあるいは開発者向けの単独マシンだけが対象という場合にはそ れだと大げさ。そういう場合に使われるのがスタンドアロン版の Chef Solo です。

Chef Solo はサーバーもクライアントも必要としない、ただのコマンドとして実装されています。このコマンドに引数でレシピなどを与えて実行することによって諸々が実行されます。

Chef Server と Chef Solo ではレシピの書き方などは一緒で、Chef Solo で覚えた概 念や書いたレシピはたとえ Chef Server に移行したとしてもそのまま利用することがで きます。

工夫次第では Chef Solo でも比較的規模の大きいシステムを管理することはできるそうで、PaaS プロバイダとして有名な Engine Yard はバックエンドで Chef Solo を使ってるという話を聞きました。

ただし、やはりシステム的なスケーラビリティはもちろん業務プロセス的なスケーラビ リティの観点その他からある程度以上の規模になったら Chef Server の導入を検討する べきだとは思います。

本書ではまずは Chef に慣れるということを主眼としていますので、Chef Solo に的を 絞って解説を進めていきます。

まずは標準 Resource の使い方だけ覚えれば OK

筆者がなんでこんな本を書こと思ったかという一つの理由でもありますが、Chef は比 較的シンプルで分かりやすいツールのはずなのに学習コストが少々お高め、という印象が あります。

その原因のひとつに、Cookbook、Resource、Attribute、Data Bags、Role、Environment、Provider、LWRP …… など多数の Chef 独自の概念が存在することが挙げられ ます。公式のドキュメントはいずれの項目についてもしっかりと書かれているのですが、 しっかりと書かれているが故に、とりあえず使ってみたいという程度の場合に、どれをど こまで覚える必要があるのかが把握しづらい。

実際には、筆者のような小規模な環境の使い方では Chef Solo の実行の仕方と標準で 提供されている Resource さえ分かればほぼ問題ありませんでした。「もうちょっとこん なことがしたいな」と思ったときに Google で検索して、必要であればその他の新しい概 念を理解するというので十分です。もちろん、より堅牢なレシピを作りたいとおもったら その他の概念も理解し使いこなす必要があるでしょう。ただし、はじめの一歩に限定して しまえばそれらは後回しで構いません。

概念の多さから来る学習コストの上昇カーブを極力抑えて、「とりあえずレシピを書い て Chef で遊べる・管理できる」レベルに最小限の投資で到達できるようにする、という のが本書のねらいです。前半に Chef Solo の実行の仕方、Resource の使い方に重きを 置いて解説して、知的負荷の下がったところでその他の概念についてさらっと触れる…… という構成をとりました。 では前置きはこの辺にして早速 Hello, World! からはじめましょう。

第2章

Chef Solo をインストールして試したい — Hello Chef!

では早速 Chef Solo を触っていきましょう。ここでは Chef Solo をインストールして Hello, World! を実行してみます。その後、パッケージのインストールを試してみましょ う。ついでに Chef 実行時の重要な考え方である冪等性 (べきとうせい) についても解説 します。

レシピ文法の詳細については後ほど詳しくみていくので、ここは「Chef Solo 実行まで の流れがどのようなものになるか」を意識して読み進めていってください。

なお、Hello, World! したいけどすぐには Chef の試験環境が用意できない、という方も 諦めずにとりあえず読み進めていってください。後ほど 第4章 で Vagrant という Chef の試験環境にうってつけなツールを紹介します。

Chef のインストール

Chef のインストールは簡単です。Omnibus Chef Packaging という Opscode が提供 している Chef 配布に便利なスクリプトを使うだけ。

\$ curl -L http://www.opscode.com/chef/install.sh | sudo bash

これでインストールされます。ここでは AWS 上に EC2 インスタンス、OS は Amazon Linux を新規に用意して適用しましたが問題なく入りました。

Chef は rubygems で gem パッケージとしても配布されているので

\$ gem install chef

でインストールすることも可能です。rvm や rbenv を利用したい場合はこちらでも良いでしょう。

レポジトリ (キッチン)、クックブック、レシピ

Chef では「コード化された手順書」あるいは「サーバーの状態」をレシピと呼ぶと述べましたが、もう二つほど言葉を覚えてください。

ファイルに対するディレクトリ、あるいはクラスに対する名前空間のように、特定のレシピに必要なデータやファイルをまとめる「クックブック」と呼ばれる入れ物があります。

そしてクックブック群を含む、Chef の実行に必要な一連のファイルをまとめる入れ物 もありそれが「レポジトリ」あるいは「キッチン」と呼ばれます。本書ではレポジトリと 呼んでいきます。

つまり、Chef では

• レポジトリ > クックブック > レシピ

という階層でレシピ群が管理されるということです。

レポジトリの作成

そういうわけで、Chef で「何かを始めよう」と思ったらまずはレポジトリを作ります。 ここでは OpsCode が github に公開しているレポジトリのひな形を使ってみましょう。

\$ git clone git://github.com/opscode/chef-repo.git

レポジトリは特定のシステムに一個くらいの粒度の大きさです。例えば全く異なる二つ のシステム A と B があったとして A と B にレポジトリを一個ずつくらいの単位。A の ために二つの Chef レポジトリを作るということはしません。

なお、この作業は 第 5 章で紹介する knife-solo に肩代わりさせることができるので OpsCode のひな形を使うのは今回限りです。

クックブックの作成 with knife

レポジトリの次は、クックブックの作成です。

Chef をインストールすると knife という、レポジトリを操作するためのツールがイン ストールされます。クックブックは knife コマンドを使って作成します。

knife でクックブックを作る前に、まずは knife の初回の設定をしておきましょう。

\$ knife configure

色々と質問事項が聞かれますがすべてデフォルトで構いません。初期化が完了すると ~/.chef/knife.rb に knife の設定ファイルが保存されます。 終わったら、knife を使ってクックブックを作成しましょう。

```
$ cd chef-repo
$ knife cookbook create hello -o cookbooks
```

cookbooks ディレクトリ内に hello というクックブックを作りました。

knife はこのように knife cookbook とか knife solo あるいは knife ec2 のよう なサブコマンドによってコマンドの動きが変わるツールです。git などと同じです。こ のサブコマンドは非常に多数あるのですが、その多くは Chef Server 環境でサーバー とクライアントを管理するためのコマンドです。Chef Solo 環境で利用するのは knife cookbook と後ほど解説する knife solo 程度になります。

レシピの編集

次はレシピを作ります。クックブックを作った時点でレシピファイルのひな形はできて いるのでそれを編集しましょう。

```
$ vi cookboks/hello/recipes/default.rb
```

以下のようにログとして "Hello, Chef!" を出力するだけのレシピに仕上げます。

```
#
# Cookbook Name:: hello
# Recipe:: default
#
# Copyright 2013, YOUR_COMPANY_NAME
#
# All rights reserved - Do Not Redistribute
#
log "Hello, Chef!"
```

Chef Solo の実行

あとちょっとだけ設定が必要。Chef Solo 実行時に実行するレシピを記述する JSON ファイルを用意します。ここでは適当に localhost.json という名前を付けました。これ を chef-repo ディレクトリ直下にでも置いてください。

```
// localhost.json
{
    "run_list" : [
        "recipe[hello]"
    ]
}
```

それから Chef が利用するテンポラリディレクトリやクックブックのパスを指定する設 定ファイル。solo.rb という名前で書いてこれも chef-repo ディレクトリ直下に置いてく ださい。

```
# solo.rb
file_cache_path "/tmp/chef-solo"
cookbook_path ["/home/ec2-user/chef-repo/cookbooks"]
```

なお、繰り返しになりますがこの辺の設定ファイルをちまちま書く作業も knife-solo で 簡略化されますので今回限りです。

準備が整いました。作った二つの設定ファイルを指定して、chef-solo コマンドを実行します。Chef Solo はサーバーのあらゆるファイルを操作するという性質上、その実行に sudo が必要な点に注意してください。

\$ sudo chef-solo -c solo.rb -j ./localhost.json

以下のように、"Hello, Chef" が出力されれば正しく動いている証拠です。

```
$ sudo chef-solo -c solo.rb -j ./localhost.json
Starting Chef Client, version 11.4.0
Compiling Cookbooks...
Converging 1 resources
Recipe: hello::default
```

* log[Hello, Chef!] action write

Chef Client finished, 1 resources updated

以上のように Chef Solo は

- レシピを作って
- JSON ファイルで実行するレシピを指定して
- chef-solo コマンドでそれを実行する

という流れで使います。簡単ですね。

パッケージをインストールする

ログを出力するだけでは面白くないので、実際にサーバーの状態を変更してみましょう。試しに zsh を Chef Solo を使ってインストールしてみます。先ほどのレシピファイル (default.rb) に

```
package "zsh" do
    action :install
end
```

と追記して再び実行してみましょう。

```
$ sudo chef-solo -c solo.rb -j ./localhost.json
Starting Chef Client, version 11.4.0
Compiling Cookbooks...
Converging 2 resources
Recipe: hello::default
 * log[Hello, Chef!] action write
 * package[zsh] action install
 - install version 4.3.10-5.6.amzn1 of package zsh
Chef Client finished, 2 resources updated
```

ちゃんと zsh が入った旨が報告されていますね。確認してみましょう。

\$ rpm -qa | grep zsh
zsh-4.3.10-5.6.amzn1.x86_64

package "zsh" do ……の記述に一切 yum や apt そのほかの記述が入ってないこと に注目してください。にも関わらず zsh が入ったということは Chef がプラットフォー ム別の差異を吸収してくれているということです。

「zsh が入ったのは分かった。もう一回実行するとどうなるの?」と気になった方もいる でしょう。試しにもう一回実行してみましょう。

```
$ sudo chef-solo -c solo.rb -j ./localhost.json
Starting Chef Client, version 11.4.0
Compiling Cookbooks...
Converging 2 resources
Recipe: hello::default
 * log[Hello, Chef!] action write
 * package[zsh] action install (up to date)
Chef Client finished, 1 resources updated
```

少し出力が変化しました。すでにインストールされているよ、という報告ですね。既に インストールされているからといってエラーになったりしていない、ということを覚えて おいてください。

Ruby 力を発揮

複数のパッケージをまとめて入れてみましょう。ここは Ruby なので

```
%w{zsh gcc make readline-devel}.each do |pkg|
package pkg do
    action :install
end
end
```

とループで記述してしまいましょう。このレシピを実行すると zsh, gcc, make, readline-devel なんかがまとめて入ります。このように Ruby のコードを交えることで レシピを簡潔に記述することができます。

冪等性 (idempotence) とは

Chef のレシピにおいて重要な考え方として「冪等性 (べきとうせい) を保証する」とい うポリシーがあります。冪等性は数学やアルゴリズムなどで良く使う言葉で、その操作を 複数回行っても結果が同じであることをいう概念です。

Chef のレシピは何回実行しても結果は同一……つまり Chef を実行し終わった後の サーバーの状態は同一であることを保証するべき、ということです。

先ほど zsh のインストールのレシピを二回流したとき、二回目はインストールされず にスルーされました。そこでエラーが起こったりサーバーに別の影響を与えたり……と いったことは発生しません。冪等性が保証されるとはそういうことです。なんてことのな い概念のようですが、後々重要な話になってきますので頭の片隅にでも置いておいてくだ さい。

Resource とは

レシピを書く際に log や package という命令を使いました。この log や package は Ruby 組み込みの文法ではなく Chef が提供する DSL です。これらのレシピ内で使う、 サーバーの状態になにがしかの影響を与える命令を Chef では "Resource" と呼びます。 第1章 で「まずは Resource だけ覚えれば OK」と言ったあれですね。

今回は Chef のログを操作する Log と、パッケージの状態を記述する Package という Resource を使いました。いずれも Chef が標準で提供する Resource です。Resource は自分で定義することもできますし、第三者が作ったそれをインポートすることもできま す。できますが、基本的なことは標準で提供されている各種 Resource を利用すること でだいたい実現できます。

Resource の一覧は公式のドキュメントで見ることができます。まだ何がなんだか、という段階だとは思いますが一応 URL を紹介するだけしておきましょう。

http://docs.opscode.com/resource.html

第3章

nginx を Chef Solo で立ち上げたい

Hello, World! でおおまかな流れが掴めたところで、次はもう少し実践的なレシピを実行してみましょう。

Chef 以外の手段でのパッケージ操作や設定ファイルの変更などは一切行わずに、Chef Solo だけでウェブサーバーを立ち上げるところまでやってみます。Web サーバーの nginx をインストールしてサービスを起動し、設定ファイルも Chef Solo で配備するレ シピを書きます。

ー連の流れで 第2章 でも見た JSON ファイルのそもそもの役割や、クックブックの ディレクトリ構成についても触れていくとしましょう。

なお 第 2 章 に引き続き、今回も AWS 上の EC2 + Amazon Linux のサーバーを利用 しています。次の 第 4 章 で試験環境の構築方法を解説しますので、環境がない方はいま しばらく我慢して読み進めていってください。

レシピ

レポジトリ内にまずは nginx 用のクックブックを新規に作成します。knife を使うの でしたね。

\$ knife cookbook create nginx -o cookbooks

次にレシピを書きましょう。cookbooks/nginx/recipes/default.rb です。

```
package "nginx" do
   action :install
end
service "nginx" do
   supports :status => true, :restart => true, :reload => true
   action [ :enable, :start ]
end
```

```
template "nginx.conf" do
  path "/etc/nginx/nginx.conf"
  source "nginx.conf.erb"
  owner "root"
  group "root"
  mode 0644
  notifies :reload, 'service[nginx]'
end
```

前回も使った package に加え、今回は service と template という Resource を使っています。

Service はその名の通りサービスを操作する Resource。RedHat 系で言えば、 /etc/init.d 配下のスクリプトを使ってサービスを起動したり止めたり、あるいは chkconfig か何かで OS ブート時に起動するサービスを登録したりもしますね。そういった サービスの状態を、Chef レシピ上で記述したい場合に Service を使います。細かい文法 については先で解説しますのでここでは Resource の役割だけ覚えておいてください。 ま、何をやっているかは文法を知らなくても見ればだいたい分かるとは思います。

Template は今回のケースのように何かしらの設定ファイルを Chef でいじりたいとき に使う Resource です。後に見るように nginx.conf を erb テンプレートとして用意して おき、レシピにそれを使うよう書いておくとテンプレートが展開されて nginx.conf とし て配置されます。erb は Ruby の代表的なテンプレートエンジンです。

テンプレートファイル

その erb テンプレートは cookbooks/nginx/templates/default/nginx.conf.erb に置きます。内容は以下です。

```
user nginx;
worker_processes 1;
error_log /var/log/nginx/error.log;
pid /var/run/nginx.pid;
events {
   worker_connections 1024;
}
```

```
http {
                  /etc/nginx/mime.types;
    include
    default_type application/octet-stream;
    server {
                     <%= node['nginx']['port'] %>;
        listen
                     localhost;
        server_name
        location / {
                   /usr/share/nginx/html;
            root
            index index.html index.htm;
        }
    }
}
```

nginx.conf ほぼそのまんまですね。nginx の設定そのものは簡単のため最小限の設定 にとどめました。一点だけ

listen <%= node['nginx']['port'] %>;

という行に注目してください。<%= … %> は erb における変数展開用のタグで す。Chef Solo を実行するとこのテンプレートタグに指定されている変数、ここでは node['nginx']['port']というハッシュが展開されることになります。

今回はウェブサーバーがバインドする port を 80 に固定するのではなく Chef Solo 実 行時に指定するという例として、上記箇所にテンプレートタグを使いました。例えば port を 8080 番にしたくなったときに毎回テンプレートをいじりたくないでしょうし、サー バー A とサーバー B でバインドする port を変えたいなんて要件がある場合もあるでしょ う。そういう時にテンプレートと変数をうまく使うと良いのです。

この変数のことを Chef では "Attribute" と言います。Attribute について詳しくはまた 別途解説するとしましょう。

JSON ファイル (Node Object)

さてテンプレートに変数が使えるのは良いとして、その値はどこから指定するのでしょ うか。第2章 でも利用した JSON ファイルが正解です。以下のように JSON で書いて おくと先ほどのようにテンプレート上で node というハッシュを通じて、その値を取り出 すことができます。

```
{
    "nginx": {
        "port" : 80
    },
    "run_list":[
        "nginx"
    ]
}
```

このように Chef Solo の JSON ファイルは、Chef Solo 実行時に渡す変数の値やど のレシピを実行するか = run_list といった実行時に決めたい値を設定するファイルです。 Chef Server を使う場合はこれらの情報は Chef Server に格納されるのですが、Chef Solo ではその格納場所がないのでローカルに JSON ファイルとして書いておくように なっています。

Chef では管理対象のサーバーのことをノード (node) と呼びます。 そしてその文脈で より正しく言うなら、JSON ファイルに書かれているのはある特定の ノードの状態です。 JSON ファイルに書いているデータ構造は Node Object と呼ばれます。Node Object の プロパティとして、そのノードに適用するべきレシピを列挙し、またそのノードが持って いる変数 (Attribute) の値も列挙する。これが JSON ファイルの役割です。Chef は実行 時に、対応するノードの Node Object を受け取ってそのノードに適用するべきレシピや Attribute を判断します。

従って Node Object つまり JSON ファイルは、基本、Chef でいじりたい対象のノー ドごとにひとつ作ることになります。

クックブック内のディレクトリ

さて、nginx 配備のレシピの準備は整いました。Chef Solo を実行……の前にクック ブック内のディレクトリ構成を少し解説しておきましょう。knife cookbook create で作るディレクトリの中身ですね。

ert — — definitions/
\vdash — — files/
$ \ default /$
ert libraries/
ert metadata.rb
ert — — providers/
ert — — recipes/
│ └── default.rb
\vdash —— resources/
$^{ar{}}$ templates/
$^{ackslash }$ default/
$^{ackslash }$ nginx.conf.erb

と、こんな感じになっていますが現時点で覚えておけばいいのは recipes、templates、 files、attributes くらいですね。そのほかのディレクトリはそこまで使用頻度は高くあり ません。

- recipes はこれまで見た通りレシピファイルを格納するディレクトリ
- templates はテンプレートを格納するディレクトリ
- files はファイルを格納するディレクトリ
- attributes は変数のデフォルト値を設定したい場合に、そのファイルを格納する
 ディレクトリ

となっています。

ところで templates のサブディレクトリに default というディレクトリがあります。 default ではないディレクトリはどういう場合に作る & 使うのかというとレシピを作るに あたってクロスプラットフォーム対応つまり Debian でも Gentoo でも Ubuntu でも動 作するレシピを作りたい場合です。しばらくはそこまでは意識しないので、スルーしてお きましょう。

files は本書では初出かつまだ未使用ですが、先ほどの nginx.conf のようにレシピから 操作したいファイルのうち変数を使う必要ないものを置く場所です。今回は nginx.conf で変数を使ったので Template Resource を使いましたが、変数が必要ない場合は Cookbook File Resource を使い実際のファイルは files ディレクトリ以下に置くことになり ます。

attributes もまだ未使用ですが、テンプレートで使った Attribute (変数) のデフォルト 値を設定したい場合などに使います。こちらはしばらくは使わないので、そんなのもある んだなくらいに覚えておいてください。

Chef Solo を実行する

では、改めて Chef Solo を実行しましょう。対象サーバー上で

\$ sudo chef-solo -c solo.rb -j ./localhost.json

でしたね。

レシピに問題がなければ nginx が無事起ち上がっているはず。ブラウザから URL を入 れて確認してみましょう。ここでは EC2 を使っているのでインスタンスの public DNS にアクセス。無事 nginx のデフォルトの画面が出力されました。

Vagrant + CentOS での注意点

続く 第3章 で試験環境を簡単に構築できる Vagrant を紹介しますが、Vagrant を使って以上を試そうとすると幾つか落とし穴があるので捕捉しておきます。

Vagrant では、OS のイメージによってはデフォルトで iptables が有効になっていて 80 番ポート含むほとんどの外部ポートに localhost 以外からはアクセスできないように なっていることがあります。筆者が普段利用している CentOS 6.3 はそうでした。従っ てそのままではブラウザから nginx にアクセスできません。本当は良くないのですが /etc/init.d/iptables stop で iptables ごと止めておきましょう。Vagrant なのでセ キュリティ的な不安はありませんしね。(iptables の設定は複雑なのでここでは触れま せん。)

iptables をオフにする設定、これも Chef でやってあげると良いでしょう。Resource は service ですね。

```
service 'iptables' do
            action [:disable, :stop]
end
```

で、iptables が stop 扱いになりすべてのポートに到達可能になります。(ただしくれぐ れも本番環境へのレシピ適用時に同じことをしないように注意してください。)

もう一点。CentOS 6.3 の初期状態では yum の EPEL という、アドオンパッケージ の yum レポジトリを有効にしないと nginx が取得できません。CentOS 標準には nginx パッケージがないんですね。EPEL の有効化も Chef Solo でやりたいところですが、そ れは 第7章 で解説します。

第4章

Chef Solo の試験環境を3分で用意する — Vagrant

Chef を何度も実行するにはいつ壊しても問題ない OS 環境があると便利です。Vagrant を使うと手元の OS 内に簡単に仮想サーバーを立てたり壊したりすることができます。実 際やってみると本当に簡単なので驚くと思います。

Vagrant は x86 と AMD64/Intel64 の仮想化ツールである VirtualBox のフロントエンドツールで、vagrant コマンドで簡単に仮想マシンを立ち上げることができるものです。

Vagrant なら好きなタイミングで仮想サーバーをどんどん立てては壊すことができるの で、Chef Solo のレシピを試していくのに最適です。Chef のテスト環境として Vagrant を選ぶのは自然な流れのようで、筆者が参加した Chef コミュニティでも Vagrant に関す る話題が頻出していました。

ここでは Vagrant の導入の仕方と、Chef を利用するにあたってやっておくとよい設 定、それから Chef レシピを試行錯誤する際に便利なプラグインである sahara について 解説していきます。

Vagrant の導入

まずは Oracle VirtualBox をインストールしてください。VirtualBox のインストール はインストーラーの指示に従うだけです。



VirtualBox

search... ogin Preferences

Welcome to VirtualBox.org!

About Screenshots Downloads Documentation End-user docs Technical docs Contribute Community VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. Not only is VirtualBox an extremely feature rich, high performance product for enterprise customers, it is also the only professional solution that is freely available as Open Source Software under the terms of the GNU General Public License (GPL) version 2. See "About VirtualBox" for an introduction.

Presently, VirtualBox runs on Windows, Linux, Macintosh, and Solaris hosts and

図 4.1 VirtualBox

News Flash

- New February 28th, 2013 VirtualBox 4.2.8 released! Oracle today released VirtualBox 4.2.8, a maintenance release of VirtualBox 4.2 which improves stability and fixes regressions. See the ChangeLog for details.
 New September 13th, 2012 VirtualBox 4.2 released! Read the official press release for
- Read the official press release for more details. New December 19th, 2012
- VirtualBox 4.1.24 released! Oracle today released VirtualBox

VirtualBox がインストールできたら、次は Vagrant を入れます。Vagrant は rubygems に登録されているので

\$ gem install vagrant

でインストールできます。(システムワイドな ruby を使っている場合は sudo なりで root 権限を用いてください。) vagrant コマンドが使えるようになります。

仮想サーバーを立ち上げる

仮想サーバーの立ち上げはすべてコマンドラインから vagrant コマンドで行います。 ただし初回のみ、"Vagrant box" と呼ばれる OS のイメージをダウンロードしてくる必要 があります。

http://www.vagrantbox.es/ に Vagrant 用の OS のイメージが各種公開されています。 ここでは Linux は CentOS 6.3 のものを使います。ダウンロードは vagrant box add コマンドで実行します。 \$ vagrant box add base http://developer.nrel.gov/downloads/vagrant-boxes /CentOS-6.3-x86_64-v20130101.box

vagrant box add が完了したら仮想サーバーの起動。vagrant init && vagrant up するだけです。

適当なディレクトリを作ってその中で

\$ vagrant init

とすると ディレクトリ内に Vagrantfile という ruby で書かれた設定ファイルができま す。vagrant up の前にネットワークの設定を少しだけいじっておきましょう。

```
# Vagrantfile
Vagrant::Config.run do |config|
  config.vm.box = "base"
  config.vm.network :hostonly, "192.168.50.12"
  ...
```

config.vm.network :hostonly, "192.168.50.12" はホストオンリーネットワー クという、ホスト OS 側からゲスト OS にネットワークアクセスできる機能をアクティブ にして且つゲスト OS の IP を 192.168.50.12 にする設定です。

これで準備 ok です。

\$ vagrant up

でホストが起動します。

chef sandbox vagrant up [default] Importing base box 'base'... [default] Matching MAC address for NAT networking... [default] Clearing any previously set forwarded ports... [default] Fixed port collision for 22 => 2222. Now on port 2200. [default] Forwarding ports... [default] -- 22 => 2200 (adapter 1) [default] Creating shared folders metadata... [default] Clearing any previously set network interfaces... [default] Preparing network interfaces based on configuration... [default] Booting VM... [default] Waiting for VM to boot. This can take a few minutes. [default] VM booted and ready for use! [default] Configuring and enabling network interfaces... [default] Mounting shared folders... [default] -- v-root: /vagrant

☑ 4.2 vagrant up

簡単すぎて拍子抜けしたのではないでしょうか。以降は OS のイメージを取得する必要もありませんから、本当に 3 分で真っさらな仮想サーバーが手に入ってしまいます。

\$ vagrant ssh

とすれば ok。CentOS が仮想サーバーとして動いているはず。

ssh アクセスできるようにする

後ほど Chef Solo を knife-solo というツールで動かすことを考えて普通の方法、つまり

\$ ssh 192.168.50.12

で ssh できるようにしておきましょう。~/.ssh/config に

~/.ssh/config
Host 192.168.50.*
IdentityFile ~/.vagrant.d/insecure_private_key
User vagrant

と記載して、該当ネットワークでは Vagrant の秘密鍵をデフォルトで使うよう設定します。

ssh アクセスできるようにする方法その 2

直接 ~/.ssh/config をいじってもよいのですが、それも面倒な場合は

\$ vagrant ssh-config --host melody

とすると、その仮想サーバーへの ssh 設定を吐き出してくれますので

\$ vagrant ssh-config --host melody >> ~/.ssh/config

とすれば OK です。これで

```
$ ssh melody
```

でログインできるようになります。IP アドレスを意識しなくて良い分、こちらのほう が良いかもしれませんね。お好みでどうぞ。

停止と破壊

仮想サーバーを一次停止するには halt、壊してリセットするには destroy です

```
$ vagrant halt
$ vagrant destroy
```

なお、二回目以降の起動時は add box や init は要らず Vagrantfile があるディレクト リで vagrant up のみで ok です。

応用: OS のロールバックを可能にする — sahara

あっというまに仮想サーバーができあがってそれだけでも便利なのですが、sahara というプラグインを入れると OS の途中の状態を記憶しておいて何らかの変更をそこまでロールバックする、という神業 (!) が使えるようになります。

```
# sahara をインストール
$ vagrant gem install sahara
# sandbox モードを有効にする
$ vagrant sandbox on
# sandbox on した所までOSの状態を戻す!
$ vagrant sandbox rollback
# OSの状態変更を確定
$ vagrant sandbox commit
# sandbox モードを解除
$ vagrant sandbox off
```

Chef Solo を実行する前に sandbox モードにしておいて、rollback しながら動きを試 行錯誤するなどに使えますね!

まとめ

Vagrant を利用するとローカルに簡単に仮想サーバーを vagrant up で立てて、 vagrant destroy で棄てるということが可能になります。Chef Solo のテストに最 適です。Chef を実行してなんかおかしくなったな、と思ったら迷わず棄ててしまえばよ いのです。がんがん使っていきましょう。

ちなみに Vagrant には Chef との連携機能がついていて、vagrant up 時に指定したレシピを実行させるなんてこともできます。それについては先で触れるとしましょう。

第5章

リモートから chef-solo を実行する knife-solo

ここまで Chef のレシピの実行は、サーバーにログインした上でレシピを編集 & chefsolo を実行するという手順を踏んできました。が、これはちょっと面倒ですね。なによ りローカルのエディタを使ってコードを編集するという使い方にならないのが痛い。

本格的なレシピの解説に入る前にこの面倒ごとを解消しておきましょう。

本書では、幾つかの方法のうちから筆者的に (いまのところ) ベストな選択肢だと言え る knife-solo をオススメしたいと思います。knife-solo は knife のプラグインで、Chef Solo を利用するにあたって便利な機能を knife に追加します。その中に手元のレシピを リモートに rsync で転送した後 chef-solo を実行し、その出力をストリームで送り返して くれるというズバリの機能があります。





では knife-solo の導入と使い方を見ていきましょう。

knife-solo の導入

knife-solo は rubygems です。

\$ gem install knife-solo

で入ります。例によって root 権限が必要な場合は sudo なりをつけてください。

knife-solo は開発が非常に活発です。本記事執筆時点での rubygems にあるもの、すなわち stable リリースは 0.2.0 ですが、後ほど説明する -o オプションなどは 0.3.0 から対応しています。

新しいものを使いたい場合は github から clone して rake install するなどしてください。

```
$ git clone git://github.com/matschaffer/knife-solo.git
$ cd knife-solo
$ rake install
```

本書は 0.3.0 を使いながら執筆しているので、 もし可能であれば 0.3.0 の導入をおす すめします。

knife-solo 0.3.0 を利用する場合は knife の設定ファイル ~/.chef/knife.rb に knife-solo が利用するテンポラリディレクトリのパスを設定するため、以下の一行を追加してください。

knife[:solo_path] = '/tmp/chef-solo'

knife-solo

knife-solo をインストールするだけで knife コマンドに chef-solo 向けのコマンドが 多数追加されます。

お目当てのレシピ転送&リモート実行は

\$ knife solo cook <host>

です。

よく使うコマンドやオプションとして以下のようなものがあります。

```
# <host> に chef-solo をインストールする
$ knife solo prepare <host>
$ knife solo prepare <user>@<host>
# <host> で chef-solo を実行
```

\$ knife solo cook <host>
run_list を個別に指定 (※ バージョン 0.3.0 から)
\$ knife solo cook <host> -o hello::default, nginx::default
<host> に転送したレシピ群を削除して掃除する
\$ knife solo clean <host>
新規 Chef レポジトリを作る
\$ knife solo init chef-repo

なお、knife-solo が ssh 経由で chef-solo を実行する都合上、ssh に使われるログイン ユーザーが sudo 且つパスワードなしで chef-solo を実行できる権限を持っている必要が あります。Vagrant や EC2 なら初期状態でログインユーザが sudo 権限持っているので 調整は必要ありません。

Vagrant の解説で Host Only Network を有効にしたのは、この knife-solo で ssh ログ インの必要があるから、なのでした。

knife-solo での solo.rb および JSON ファイルの扱い

knife-solo を使う場合、第2章や第3章で見た solo.rb や JSON ファイルの扱い方が少し変わります。

solo.rb は 0.2.0 では knife solo init で一緒に生成されます。これをそのまま使 えば良いでしょう。0.3.0 では、そもそも solo.rb は生成されず、solo.rb がなくても knife-solo が良い感じに調整してくれるようになったようです。

JSON ファイルは knife-solo では、knife solo prepare した際あるいは初めて knife solo cook した場合に Chef レポジトリの nodes ディレクトリ以下に該当 ノードのホスト名もしくは IP アドレスがファイル名になって生成されます。例え ば nodes/192.168.50.12.json のように。以降 knife-solo は knife solo cook にあ たって対象ノードの名前からこの JSON ファイルを選択します。よって、以降はこの JSON ファイルを編集するようにしてください。

なお、knife solo cook コマンドの -j オプションで任意の JSON ファイルを指定す ることもできます。
knife solo prepare で Chef Ready!

knife solo prepare コマンドはなかなか便利なコマンドで、指定したサーバーに Chef Solo を良い感じにインストールして実行可能になるよう調整してくれます。サー バー側に必要なのは ssh & sudo できるログインアカウントだけ。

Vagrant でも EC2 でもあるいは自前のサーバーでも、サーバーを立ち上げたら

\$ knife solo prepare <host>

とするだけで、以降 chef-solo が実行できるようになるわけです。

実は Vagrant には初期状態から Chef Solo がインストールされていますが、OS のイ メージによってはバージョンが古かったりもしますので、とりあえず初回は knife prepare を実行しておくと良いでしょう。

knife-solo で Chef Solo 環境を生成する

knife-solo には chef-solo に最適なレイアウトで Chef レポジトリを作ってくれる機能 もあります。筆者はレポジトリを作る時はいつも knife-solo 任せにしています。

<pre>\$ knife sol</pre>	o init chef-repo			
\$ ls -F che:	f-repo			
cookbooks/	data_bags/	nodes/	roles/	site-
cookbool	ks/			

作成したレポジトリは git で管理しましょう。

```
$ cd chef-repo
$ git init
$ git add .
$ git commit -m 'fist commit'
```

この後 cookbook を site-cookbooks 内に作るには

\$ knife cookbook create hello -o site-cookbooks

で OK ですね。

ここまで本書ではレポジトリを OpsCode のひながたから作成していましたが、以降は knife-solo で生成するようにしていきましょう。

knife solo init で作ったレポジトリのディレクトリ構成

以前 第3章 では knife が作成するクックブックのディレクトリ構成を見ました。ここで knife-solo が作成する Chef レポジトリのディレクトリ構成を見てみましょう。

```
% ls -Fl | cut -f14 -d " "
cookbooks/
data_bags/
nodes/
roles/
site-cookbooks/
```

という5つのディレクトリから構成されます。

cookbooks と site-cookbooks はいずれもクックブックを格納するディレクトリです。 cookbooks には、ダウンロードしてきたクックブックなど第三者が作ったものを入れる。 site-cookbooks に自分の作ったクックブックを入れる、というのが推奨される使い方の ようです。

```
nodes ディレクトリは先に述べた通りノードごとの JSON ファイルの格納場所。
data_bags と roles は当分使わないので現時点で覚える必要はありません。字面の通
り Chef の Data Bags の機能と Roles の機能を使いたいときに利用する場所です。
```

複数ホストへの knife solo の実行

Chef Solo での管理対象のノードが複数になると、knife-solo の実行を複数行うことになります。その場合は

\$ echo user@node1 user@node2 user@node3 | xargs -n 1 knife solo cook

として xargs を利用すれば ok です。ホスト名を毎回入力するのが面倒な場合は、シェ ルスクリプトなどにしてしまえばよいでしょう。

まとめ

knife-solo を使うとサーバーにいちいちログインすることなく、手元の作業環境から Chef Solo を実行させることができます。

また knife solo prepare で Chef 環境が整っていないサーバーもすぐに Chef Solo Ready な状態にすることが可能です。

knife solo init でレポジトリを作り、手元のマシンの好きなエディタでレシピを編 集、knife solo prepare でリモートの Chef Solo 環境を整えて、knife solo cook で実行……というのが一連の流れです。

第6章

レシピを作って実行する流れをおさらいし たい

ここまでで、Vagrant で試験環境を用意して Chef Solo のレシピを作って knife-solo で流すという一通りの作業の解説が終わりました。以降は落ち穂拾い的な話を少しして、 続けて本格的に Resource の解説に入ります。

いろいろとコマンドがでてきたので一度、普段の作業の流れをおさらいしておきましょう。なお、Vagrant と knife-solo はすでにインストール済みという前提です。

vagrant up

まずはレシピを適用するノードを Vagrant で用意。

```
$ cd /some/where
# 新規に仮想サーバを作るなら init して Vagrantfile を編集
$ vagrant init
$ vagrant ssh-config --host melody >> ~/.ssh/config
$ vagrant up
```

Chef レポジトリを作成

Chef レポジトリを作り、一緒に git 管理も始めてしまいましょう。

```
$ knife solo init chef-repo
$ cd chef-repo
$ git init
$ git add .
$ git commit -m 'first commit'
```

knife solo prepare

仮想サーバーのホスト名は "melody" 。knife-solo prepare で環境を整える。JSON ファイルが一緒にできるのでこれも commit しておきましょう。

```
$ knife solo prepare melody
$ git add nodes/melody.json
$ git commit -m 'add node json file'
```

クックブック作成 & レシピ編集

クックブックを作ってレシピをよしなに編集します。ここでは nginx の立ち上げだったとしましょう。JSON ファイルの編集もお忘れ無く。(よく忘れます。)

\$ knife cookbook create nginx -o site-cookbooks

で、あとはレシピや JSON ファイルを適当に編集する。

```
$ vi site-cookbooks/nginx/recipes/default.rb
$ vi nodes/melody.json
```

Chef Solo 実行

できあがったら、レシピをノードに適用してみましょう。

\$ knife solo cook melody

意図した通りに動作したなら git に commit

```
$ git add site-cookbooks/nginx
$ git commit -m 'Add nginx recipe'
```

Done! 簡単ですね。

レシピを育てる

一通りのフローは以上です。あとはこれを繰り返してクックブックやレシピを増やして いく形になります。

ちなみに Chef のレシピを作っていく過程では、たくさんの試行錯誤をすることになる と思います。例えば最初は template の類は既存の設定ファイルをそのままコピーしてい たところ、だんだんと汎用的な作りにするために Attribute を導入していったりとか。レ シピもクロスプラットフォームは気にせずに書いていたところを、その辺を意識するよう になったりとか。

レポジトリをバージョン管理しておくことでその辺りの試行錯誤が非常に楽になります し、Vagrant などで試験環境を構築しておくことで気軽に試せるようになります。

筆者もはじめはおそるおそる Chef を使っていたのですが、何度かレシピを流してるう ちにだんだんとそれが育ってきて、いまでは結構な蓄積になってきました。Chef のレシ ピを構築していく過程には、アプリケーションをバージョンアップさせて育てていくよう な楽しみが備わっている感じがします。がんがん試してどんどんレシピを充実させていき ましょう。

第7章

サードパーティの Chef クックブックを使 いたい

本書ではレシピは自分で書いて育てていくことを念頭に置いて解説を進めています。一 方、メジャーなソフトウェアのクックブックはサードパーティライブラリとして公開され ていることも少なくありません。

Opscode Community にはそういったサードパーティ Chef クックブックが集積され ています。knife コマンドを使うとこれらの公開されているクックブックを手元に簡単に インポートすることができます。

ここではその導入の仕方をみておきましょう。

事前の設定

Opscode Community からクックブックをインポートするためには Opscode Community にユーザー登録し、秘密鍵を取得する必要があります。Opscode Community の サイトから Sign Up を行い、ユーザー設定画面から "Get a new private key" を選択して 秘密鍵をダウンロードしてください。



図 7.1 鍵の取得

ダウンロードした秘密鍵 (ファイル名 <ユーザー名>.pem、例: naoya.pem) は ~/.chef/naoya.pem などに保存します。パーミッションはもちろん 600 です。

次に、knife の設定を行います。第 3 章 で knife configure で生成した ~/.chef/knife.rbを編集し

client_key	'/Users/naoya/.chef/naoya.pem'
$cookbook_path$	['./cookbooks ']

この二つの設定を行ってください。前者は先ほどダウンロードした秘密鍵のパスの指定、後者はインポートするクックブックを保存するディレクトリです。./cookbooks にしておくと、Chef レポジトリのルートディレクトリでインポートコマンドを実行したときに良い感じに cookbooks ディレクトリ内にそれが入ります。

yum の EPEL を yum クックブックで有効にする

例として CentOS の yum レポジトリの設定で、アドオンパッケージの EPEL を有効 にするクックブックを使ってみましょう。

OpsCode Community のクックブックは knife コマンドで取得可能です。

このとき、コマンドを発行する場合自分の Chef レポジトリが git で管理されていてか

つワーキングディレクトリがクリーンな状態 (コミットしていない変更がない状態) であ る必要があります。(この辺、knife が若干余計なお世話だという気もします……)

```
$ cd chef-repo
# git
$ git init
$ git add .
$ git commit -m 'blah blah'
```

さて、knife で yum クックブックを取得しましょう。

\$ knife cookbook site vendor yum

と打つと、cookbooks ディレクトリ以下に yum クックブックがダウンロードされま す。ついでに git add + commit も自動で行われます。

これで yum クックブックは既に使える状態になっています。yum クックブックの使い 方はレシピをみるなりドキュメントを読むなりしていください。ひとまず EPEL を有効 にするには JSON ファイルの run list に

```
{
    "run_list":[
        "yum::epel"
]
}
```

と書いて Chef::Solo を実行すれば OK です。これは yum クックブックの epel.rb レ シピを実行しろということですね。

これでいつも通り knife solo cook を流すと公開鍵そのほか yum レポジトリの設定 に必要な諸々が調整されて EPEL が有効化されます。試しに確認してみましょう。

```
$ vagrant ssh
$ yum repolist
...
epel Extra Packages for Enterprise Linux 8,425
```

EPEL が使えるようになっているのが分かります。yum クックブックは実際には /etc/yum.repos.d 以下に epel の設定ファイルを作るなどの調整を行っているようです。

自分で書くかサードパーティのものを使うか

というわけで、サードパーティクックブックの使い方でした。

さて、こういったサードパーティのものを積極的に使っていくべきかどうか。はっき り言ってその辺はポリシー次第なのでどちらがいいと断言できるものではありません。 Opscode Community のレシピはあくまでコミュニティから提供されているものなので、 すべてのクックブックに理想的な動作が保証されているわけではありませんし、かといっ てじゃあ全然安定していないのかと言われれば十分に利用できるレベルにあります。

全体として言えることは、サードパーティクックブックはその性格上、汎用性を上げる ためにわりとしっかりと書かれているものが多いということです。クロスプラットフォー ム周りなどが特に。一方、これから自分でレシピを書いてみるとわかるのですが、自分の 環境にある程度限定してしまえばレシピで書かなければいけない項目はかなり少なく、自 分で書いたものの方がわかりやすい事のほうが多いです。

rubygems なんかのように、車輪の再発明を避けてどんどん既存のものを使っていくべき……という性格のものとは少し正確が違うようには思います。

筆者の場合は、サードパーティクックブックはそれほど利用していません。ここで紹介 した EPEL の有効化のように自分で書くと面倒だけど、クックブックの導入自体は至極 簡単、というものぐらいしか使っていません。他のレシピはすべて自分で書いています。

少なくとも、基本的なレシピであれば問題なく読める・書ける程度に Chef が身につく まではサードパーティクックブックに頼った運用はしないほうが無難でしょう。サーバー 構成変更を自動化するという性格上あまりにそれがブラックボックスであるのは、好まし い状態とは言えませんね。

レシピの取り扱いに十分に習熟した頃合いにでも、自分たちの運用管理ポリシーと照合 しながらサードパーティのそれを導入するかどうかを一度検討してみてください。

第8章

代表的なレシピのサンプルを見たい td-agent のレシピを読む

ここまででレシピとは何で、どこに書くものか、あるいは作ったレシピをどうサーバー に適用するかまでは分かったと思います。

以降はより実践的なレシピが記述できるように各 Resource の詳細を見ていきたいと 思いますが、個別の解説に入るまえに、まずは overview 的に参考になりそうな実例を見 て、全体像を掴むところからはじめましょう。

サンプルに利用するのは td-agent というパッケージのクックブックです。以下の URL で公開されています。

treasure-data / chef-td-agent

td-agent は Fluentd というログ収集ソフトウェアの配布パッケージです。

本書で解説したい項目が多数利用されている良いサンプルということで、このレシピを 題材にしました。

td-agent のレシピを見てみよう

td-agent のクックブックを実行すると td-agent がインストールされた上、テンプレートとして用意された設定ファイルがサーバーに展開されます。よくある動きですね。

それを実現しているのが以下のレシピです。基本に忠実に書かれているので、ざっと読 んだけでもだいたい何をしているのかは理解できると思います。各項目は後に見ていきま すので、まずは目を通していってください。

```
# Cookbook Name:: td-agent
# Recipe:: default
#
# Copyright 2011, Treasure Data, Inc.
#
```

#

```
group 'td-agent' do
  group_name 'td-agent'
             403
  gid
           [:create]
  action
end
user 'td-agent' do
  comment 'td-agent'
 uid
          403
 group 'td-agent'
          '/var/run/td-agent'
 home
  shell '/bin/false'
  password nil
  supports :manage_home => true
  action [:create, :manage]
end
directory '/etc/td-agent/' do
  owner 'td-agent'
  group 'td-agent'
 mode
        '0755'
  action :create
end
case node['platform']
when "ubuntu"
  dist = node['lsb']['codename']
  source = (dist == 'precise') ? "http://packages.treasure-data.com/
      precise/" : "http://packages.treasure-data.com/debian/"
  apt_repository "treasure-data" do
   uri source
    distribution dist
    components ["contrib"]
    action :add
  end
when "centos", "redhat"
  yum_repository "treasure-data" do
   url "http://packages.treasure-data.com/redhat/$basearch"
    action :add
  end
end
template "/etc/td-agent/td-agent.conf" do
 mode "0644"
```

```
source "td-agent.conf.erb"
end
package "td-agent" do
   options "-f --force-yes"
   action :upgrade
end
service "td-agent" do
   action [ :enable, :start ]
   subscribes :restart, resources(:template => "/etc/td-agent/td-agent.
        conf")
end
```

このレシピがやっているのは

- td-agent グループを作る
- td-agent ユーザーを作る
- ・/etc/td-agent/ ディレクトリを作る
- td-agent 配布用のサイトを apt ソースもしくは yum レポジトリに追加する
- td-agent の設定ファイルをテンプレートから作成して設置する
- td-agent パッケージをインストールする
- td-agent サービスを有効にする

という一連の流れです。

なお、td-agent のレシピは複数プラットフォームへのインストールを考慮してユー ザー、グループ、ディレクトリなどをレシピ内で作成していますが、このあたりは一般 的な RPM もしくは deb パッケージはそれらパッケージがユーザーの作成そのほかを実 行してくれます。従って、単に nginx を入れたいだけであれば package Resource でイ ンストールするのみで ok です。ここではあくまで「シンタックスの参考」としてのみ td-agent のレシピを紹介していることに注意してください。

各々詳細を見ていきましょう。

グループを作る

グループの状態は Group という Resource を使って記述します。ここでは gid 403、 グループ名が td-agent になるように設定されています。

ユーザーを作る

ユーザーの状態は User Resource で記述します。

```
user 'td-agent' do
  comment 'td-agent'
  uid 403
  group 'td-agent'
  home '/var/run/td-agent'
  shell '/bin/false'
  password nil
  supports :manage_home => true
  action [:create, :manage]
end
```

ここではユーザ名はもちろんのこと、コメントや uid、グループ、ホームディレクトリ やデフォルトのログインシェルなんかも設定されていますね。シェルを /bin/false にして パスワードは nil にすることでログインできないユーザーとしてユーザーを構築している ようです。

オプションで先に作っておいたグループ td-agent が指定されています。 **Chef のレシ** ピは上から順に逐次で実行されていくので、先に定義した処理はすでに実行されている前 提でレシピを書いていって構いません。

ディレクトリを作る

ディレクトリは Directory Resource で記述します。特に難しいところはないでしょう。

```
directory '/etc/td-agent/' do
  owner 'td-agent'
  group 'td-agent'
```

```
mode '0755'
action :create
end
```

ここでも owner, group に先に作ったユーザーとグループが使われていますね。

パッケージの配布サイトを登録する

次はパッケージの配布サイト (URL) を各 OS のパッケージシステムに登録する設定。 これを行うことで、後に package Resource で td-agent が該当 URL からダウンロード されてインストールされることになります。Debian 系と Redhat 系両方のディストリ ビューションに対応させているため、これまでのレシピとは異なり少しだけ込みいった記 述になっていると思います。

```
case node['platform']
when "ubuntu"
  dist = node['lsb']['codename']
  source = (dist == 'precise') ? "http://packages.treasure-data.com/
      precise/" : "http://packages.treasure-data.com/debian/"
  apt_repository "treasure-data" do
    uri source
    distribution dist
    components ["contrib"]
    action :add
  end
when "centos", "redhat"
  yum_repository "treasure-data" do
    url "http://packages.treasure-data.com/redhat/$basearch"
    action :add
  end
end
```

ここでは node ['platform'] という変数を見てその値を軸に処理を分岐させています ね。Ubuntu なら apt を、CentOS や RedHat なら yum を設定するわけです。node は テンプレートの所でもみた例の変数です。node からはこのように、Chef があらかじめシ ステムから抽出しておいてくれた値を取り出すこともできます。

そして分岐の中ではそれぞれ apt_repository と yum_repository の Resource を 使って配布 URL を設定しています。

Attribute と Ohai

ところでここまで変数 node のことを「変数」と呼んできましたが、これは Chef の用 語では Attribute と呼ばれます。テンプレートやレシピの中から参照できる様々な Key + Value の値を提供するのが Attribute という仕組み。

そして Attribute では先に見たとおり、Chef がシステムからあらかじめ抽出しておい てくれた値を取得することもできます。td-agent のレシピでは node ['platform'] など が参照されています。

では実際に Attribute で参照できる情報にはどういったものがあるか、つまりどんな キーでどんな情報が参照できるか。それは ohai というコマンドを実行すればわかり ます。

Ohai は Chef をインストールしたときに一緒にインストールされるライブラリで、シ ステム上の様々な値を抽出して JSON のデータ構造でそれを扱えるようにしてくれるも の。ohai コマンドはそのフロントエンドのツールです。そして Chef は Ohai を利用し て Attribute にサーバー環境の情報をセットしているわけです。

ohai の実行結果は非常に長いので少しだけ載せておきましょう。

```
$ ohai | head
{
    "languages": {
        "ruby": {
            "platform": "x86_64-linux",
            "version": "1.8.7",
            "release_date": "2011-06-30",
            "target": "x86_64-redhat-linux-gnu",
            "target_cpu": "x86_64",
```

Ruby のバージョンなんかが調べられそうですね。ちなみに

```
$ ohai platform
[
    "centos"
]
```

と、引数にキーを指定するだけで任意の値だけを取得することも可能です。

これら Ohai の収集する値を Chef で使いたい場合に node [:platform] などと書いて 取得するわけです。

Ohai はかなり細かい情報まで収集してきてくれるので非常に感心するところなのですが、実際それらの情報を収集する Ohai のコードは本当に泥……汗と涙の結晶のようなコードになっています。興味のある方はぜひソースを覗いてみましょう。

テンプレートから設定ファイルを作る

次はテンプレートから設定ファイルを作るレシピ。ここは <mark>第 3 章</mark> で見たものほぼ同様 ですね。

```
template "/etc/td-agent/td-agent.conf" do
  mode "0644"
  source "td-agent.conf.erb"
end
```

パッケージ td-agent をインストール

Package Resource を使って td-agent パッケージをインストールします。以前に述べ たように Package は該当プラットフォームのパッケージシステムが何なのかは勝手に調 べて実行してくれます。Debian なら apt、Redhat なら yum です。

td-agent は apt や yum のレポジトリには含まれていないサードパーティのパッケージ ですが、先に apt_repository もしくは yum_repository のところで配布 URL を設定 しているので、以下の記述だけで apt or yum で td-agent がインストールされるという 算段になっています。

```
package "td-agent" do
    options "-f --force-yes"
    action :upgrade
end
```

また、ここでは action が action :install ではなく action :upgrade になってい る点に注目してください。install の場合は既にパッケージがインストールされていれば 何も実行されません。upgrade の場合は、最新版があればそれにアップグレードします。

該当パッケージが最新版であることを維持したい場合は、アクションに upgrade を指定 すれば良いわけですね。

サービスを起動する

最後に td-agent のサービスを起動しています。アクションは[:enable, :start] と配列で二つのアクションを指定することでその両方が実行されます。:enable は OS 起動時のサービスとして登録するアクション、:start はその名の通り起動です。

ここで気になるのが subscribes という行ですね。これは、テンプレートファイルが更 新されていたら:restart アクションを実行しろ……という記述になります。td-agent に限らずサービスの類は設定が更新されたら常駐プロセスを活性化する必要があることが 多いのですが、それを実現するのが上記です。

Subscribe に関してはまた先で詳しく触れるとしましょう。

オフィシャルドキュメントを参照しよう

いかがでしたでしょうか。レシピの書き方がだいぶクリアになったのではないかと思 います。実際、よく使う Resource はここで挙げられたものがほとんどで、これらの Resource の使い方を覚えておけばやりたいことの大方は実現できると思います。

ここまでくればあとは公式の Resource のドキュメントである About Resources and Providers — Chef Docs を見ると良いでしょう。情報が多すぎて混乱する、なん てことはもうなくなっているはずです。ドキュメントでは標準リソースとそのオプション が一通り解説されていますので、レシピを作っていて分からなくなったらこのドキュメン トを見に行くと良いでしょう。筆者も Chef 利用時にいちばん多く参照しているのが、こ のドキュメントです。

なんで "Resource"?

最後にちょっと「お話」的な内容を。

この Chef の Resource、なぜ「リソース」という単語なのでしょうか。その答えは 第 1章 で触れた、Chef は「サーバーの状態を管理し収束させるためのフレームワークであ る」ということと関係があります。

Chef をサーバー自動化ツールという機能だけに着目して見た場合、レシピに書かれるのは自動化の手順でしかありません。

一方、本質的には Chef がやっていることは「サーバーの 状態を管理する」ということ だと述べました。つまり、この視点でみると、レシピには「ノードのあるべき状態」が書 かれているとみなせるわけです。nginx が必要なら「nginx がインストールされている」 という 状態をレシピに書く。バージョンを固定したいなら「バージョンはいくつ」とい う 状態を各書く。

管理するのはノードを構成する何かしらの要素の状態。その「何かしらの要素」こそが リソースですね。ノードを構成するパッケージ、サービス、設定ファイルといった各種リ ソースがあってレシピにはそれら **リソースの状態**を記述するんです。ゆえに、そのシン タックスを提供するものは "Resource" と呼ばれるのです。

更新したレシピにはノードの「新しいリソースの状態」が定義されています。Chef は、 そのレシピをまだ適用していないノードの状態つまりは「現在のリソース」を取得して 「新しいリソース」と比較しそれらの状態が異なっているなら、それらを新しい状態に「収 束させる」……そういうフレームワークです。この文脈でいくと、Chef のクックブック やレシピには「処理の順番を定義する」のではなく「サーバーが最終的にどうなっていて 欲しいか」という状態を定義する、という捉え方がより Chef らしい捉え方であるとも言 えます。

この辺りの Chef の思想などについては What Is Chef? というスライド (日本語) にま とめられているので、Chef の一通りの使い方に慣れてきたころにでも一度目を通してみ てください。

ちなみに本書では

```
template '/etc/nginx/nginx.conf' do
    owner 'root'
```

end

というレシピの記述における owner などの構文を "オプション" と呼んでいますが、 Chef の世界ではこれは本来 "Attribute" と呼ばれます。上記の場合 Template というリ ソースに属する性質としての owner …… つまり owner 属性 (Attribute) は root である、 という定義になります。リソースがあってその属性がある、という見方でいくとなぜオプ ション行を Attribute と呼ぶのかがわかりますね。ただし、Attribute と言うと Template やレシピで使えるの変数の仕組みである Attribute との区別がつきづらいので、以降もこ ちらは "オプション" と呼んでいくことにします。

第9章

パッケージをインストールする ―

Package

ここからしばらくはよく使う Resource の詳細についてみていきましょう。

Chef でパッケージをインストールするには Package Resource を使います。ここま で散々見てきましたね。おそらくこれが一番よく使う Resource ではないでしょうか。

Package

Package はパッケージの状態を記述するための Resource です。

と書きます。Package は実際の動作時にはプラットフォームのそれにあわせてパッ ケージシステムを選択してくれます。Redhat 系なら yum、Debian 系なら apt などなど。 複数をまとめて入れたいなら

```
%w{gcc make nginx mysql}.each do |pkg|
package pkg do
    action :install
end
end
```

と Ruby のシンタックスを巧く活かせば良いでしょう。

```
package "perl" do
    action :install
    version "5.10.1"
end
```

のように version 指定でバージョンを固定することができます。

アクション:installはパッケージをインストールするだけですが 第8章 で触れたように:upgradeにすると、レシピを複数回実行したとき既存のパッケージが古い場合には それを最新版に入れ替えます。

パッケージの削除は

```
package "perl" do
    action :remove
end
```

です。

パッケージを指定したファイルからインストールしたい場合もあるでしょう。その場 合は

```
package "tar" do
    action :install
    source "/tmp/tar-1.16.1-1.rpm"
    provider Chef::Provider::Package::Rpm
end
```

で入ります。Cookbook File Resource と組み合わせて、その rpm ファイルも Chef レポジトリにおいて管理しておきそれを Chef で入れるという合わせ技も可能です。これ についてはまた後ほど解説するとしましょう。

gem_package

Package Resource のサブリソース (とでも言えばいいのでしょうか) の gem_package を使うと rubygems の gem を扱うことができます。使い方は通常の Package と同じで

```
gem_package "rake" do
    action :install
end
```

です。これで rake がインストールされます。

rubygems を入れたい場合、gem コマンドのパスを明示的に指定したい場合がありま す。例えば 第8章 でも紹介した td-agent はそのプラグインは gem でインストールする

のですが、その際 td-agent 組み込みの gem コマンドを使う必要があります。システム にある素の gem コマンドを使ってしまうと、td-agent が使う ruby バイナリが期待する のとは違うパスに gem が入ってしまいます。

こういう場合は

```
gem_package 'fluent-plugin-extract_query_params' do
  gem_binary "/usr/lib64/fluent/ruby/bin/fluent-gem"
  version '0.0.2'
  action :upgrade
end
```

と gem_binary オプションでパスを指定します。

他にも細かなオプションが幾つかありますが、使用頻度は高くないでしょう。より詳し くはドキュメントを参照してください。



サービスを起動したい — Service と Notification

ウェブサーバーやデータベースのような「サービス」のパッケージを Package Resource で入れる場合、そのままではサービスの起動や OS 起動時への登録は行われません。 Service Resource を使うと、そのサービスの状態を記述することができます。

サービス系のパッケージには設定ファイルが含まれていることがよくあります。例えば nginx に対する nginx.conf など。このとき nginx.conf を更新したら nginx を reload も しくは restart するなどして設定を再読込させる必要があります。Notification (通知) と いう機能と Service をうまく組み合わせるとそれもちゃんと実現できます。

Service

以下は nginx を起動するレシピです。

```
service "nginx" do
    action [ :enable, :start ]
    supports :status => true, :restart => true, :reload => true
end
```

action 行でサービスを有効にしかつ起動させています。これで OS ブート時に nginx が起ち上がるようになります。実際には Redhat 系であれば /etc/init.d 以下のスクリプト が操作されて、期待通りの状態に設定されます。

続く supports の行ですが、これは他のリソースなどに「このサービスは status, restart, reload を使えるよ」ということを教えるためのオプション。なくても動きますが、指定できるならしておいたほうが良いでしょう。

例えば :restart => true が指定されていなかった場合、Chef はサービスの restart を stop + start で代用しようとします。各種サービスの init スクリプトは stop + start では吸収仕切れないなにがしかの動作を restart として定義している場合もあるので、

restart できるならそちらに任せる方が賢明です。

Notitifacation と Service の組み合わせ

Notification を使うと、冒頭で述べたようなほかの Resource の更新に合わせてサービ スを再起動するといった動作を記述することができます。

例えば nginx サービスを、nginx.conf が更新された場合に reload するには以下のよう に書くとよいでしょう。

```
service "nginx" do
   supports :status => true, :restart => true, :reload => true
   action [ :enable, :start ]
end
template "nginx.conf" do
   path "/etc/nginx/nginx.conf"
   source "nginx.conf.erb"
   owner "root"
   group "root"
   mode 0644
   notifies :reload, 'service[nginx]'
end
```

Template の記述に notifies :reload, 'service[nginx]'がありますね。これが Notification です。ほかの Resource から特定の Resource に Notification (通知) を送る ことで、任意のアクションをトリガすることができます。

notifies は第一引数にアクション、第二引数に resource_type [resource_name] という形式で記述します。resource_type には service や template といった Resource の種類、resource_name は自分でレシピ内に定義した Resource ですね。例えば service [nginx] や template [nginx.conf] といった形になります。

ここでは Service に通知を送ることで設定の再読込 (:reload アクション) をキックし ていますが、通知先に指定できるのは Service に限りません。

```
notifies :run, "execute[test-nagios-config]"
```

として Execute Resource を実行させる、なんてこともできます。とはいえ Notification を一番よく使うのはやはりサービスの活性化をしたいとき、ですね。

Notification のタイミング

なお、デフォルトでは Notification の実行は遅延します。notifies の行が実行された タイミングですぐ通知内容が実行されるのではなくて、いったんそれはキューに入れられ て、Chef 全体の実行の限りなく終盤で実行されることになります。これにより通知を送 る側はあまり順番を気にせずにただ通知を送ればよい、という仕組みになっています。

notifies の第三引数に : immediately を渡すと通知タイミングを即時に変えること ができますが、利用場面はそれほど多くないようには思います。

notifies :reload, 'service[nginx]', :immediately

Subscribe — Notification の逆方向の通知

Notification は何かしらの Resource にアクション「させたいとき」に利用しますがその逆、何かしらの Resource が何かを行ったときにアクション「したいとき」に利用するのが Subscribe です。第8章のtd-agentのレシピでも利用されていました。

```
service "td-agent" do
   supports :status => true, :restart => true, :reload => true
   action [ :enable, :start ]
   subscribes :restart, "template[td-agent.conf]"
end
```

template Resource を使って書いた td-agent.conf が更新されたら、サービスを restart しろという命令になっています。Notification の例では Template 側に notifies を記述 したのに対して、こちらでは Service 側に subscribes を書いていますね。実現される の動作は Notification と全く同じで、通知の向きが違っているだけです。

Notification と Subscribe はどちらを使っても構いません。その都度都度で可読性の高い方を選択すれば良いと思われます。

第11章

テンプレートから設定ファイルを配置した い — Template

次は Template の解説です。ここまで何度も例を見てきたので、今回はほぼおさらい という形になります。設定ファイルなどの外部ファイルを Chef 経由で配備するための Resource が Template です。

なお、Template はその名の通りテンプレートですから Attribute (変数) の値をテンプ レート内で展開したい場合に利用します。Attribute を一切利用しないのであればテンプ レートではなく、静的なファイルを操作する目的の Cookbook File Resource を使うと 良いでしょう。

Template

/etc/nginx/nginx.conf に置かれた設定ファイルをテンプレートで処理したい場合、 Template のシンタックスは以下のようになります。

```
template "nginx.conf" do
  path "/etc/nginx/nginx.conf"
  source "nginx.conf.erb"
  owner "root"
  group "root"
  mode 0644
end
```

テンプレートは templates/default/nginx.conf.erb が利用されます。 上記の記述はやや冗長で

```
template "/etc/nginx/nginx.conf" do
  source "nginx.conf.erb"
  owner "root"
  group "root"
```

第11章 テンプレートから設定ファイルを配置したい — Template

mode 0644 end

と書くこともできます。

更に省略して

```
template "/etc/nginx/nginx.conf" do
  owner "root"
  group "root"
  mode 0644
end
```

とそもそもソースファイルのパスを書かないで済ませることもできます。この場合 Chef の規約によって、templates/default/nginx.conf.erb が自動で選ばれることになり ます。

好みに応じて選択してください。

template で設定ファイル更新した際にサービスで設定を再読込させるには Notification を使うのでしたね。以下のように記述しましょう。

```
template "/etc/nginx/nginx.conf" do
  source "nginx.conf.erb"
  notifies :reload, 'service[nginx]'
end
```

テンプレート内では Attribute が使える

テンプレートの erb ファイル内では 第3章 で見たとおり <%= node[:attribute_name] %> という形式で変数展開できて、これにより Attribute の 値を埋め込むことができるのでしたね。変数の値は JSON ファイル内で指定することが できました。

```
{
    "nginx": {
        "port" : 80
    },
    "run_list":[
```

```
"nginx"
]
```

}

と定義された [nginx] [port] という値はテンプレートからは

<%= node['nginx']['port'] %>

と書くと取り出すことができます。ここではこのタグが "80" という値に置換されます。 また、第8章 で見たように Attribute には Ohai によって収集されたさまざまなシステ ム情報がデフォルトでセットされているのでした。それらの値にも同様に node 変数でア クセスできます。

<%= node[:platform] %>

ちなみに JSON ファイルに定義するような自分で定義した値は node['nginx']['port'] とキーを文字列で指定して、Ohai で取得するものは node[:plarform] とキーをシンボルで指定するのが通例のようです。

試しにテンプレートから Ohai で収集された情報にアクセスしてみましょう。

```
Platform: <%= node[:platform] %>
Ruby: <%= node[:languages][:ruby][:version] %>
IP Address: <%= node[:ipaddress] %>
```

こういう erb テンプレートを書いて、レシピ内では template Resource で

```
template "/tmp/template_test.txt" do
  source "template_test.txt.erb"
  mode 0644
end
```

と指定します。これを実行するとホストの /tmp/template_test.txt に

```
Platform: centos
Ruby: 1.8.7
IP Address: 10.0.2.15
```

という内容で展開されました。きちんと値が取れています。

テンプレートはまるっと既存のを持ってくるところから始 める

このように Template Resource を使うことで、変数を使って設定ファイルを配置する ことができます。

レシピを複数台以上のホストに定義するとなると、設定ファイル内にハードコードされ たホスト依存の値、例えば IP アドレスそのほかといったことが問題になる場合がありま す。そのような場合はテンプレート変数を使って Attribute でその値を差し替えてること で抽象化してしまえばよいでしょう。

ただ、そういった完璧な設定ファイルをいきなり用意しろ、といわれてもハードルが高 いことでしょう。基本的には Chef のレシピを書き始めた段階では設定ファイル系のテン プレートは既存のものあるいはパッケージを入れたときにのデフォルトのものをそのまま 丸っとコピーして使い始めるのが定石です。その後、各種設定を煮詰めながら徐々に配布 しやすい形態に変えていくと良いのです。ここでもやはり「レシピを育てる」という考え 方が重要です。

冒頭で "Attribute を一切使わないなら Cookbook File Resource を使うとよい" と述 べましたが、設定系のファイルはレシピを育てていく過程で「やっぱり Attribute を使い たくなった」ということが珍しくありません。よって良く書き換えるような設定ファイ ル、例えば nginx や mysql の設定ファイルあるいはシェルの rc ファイルといったものは Attribute を使っていなくてもはじめから Template で管理しておいても良いのではない かと思います。

第12章

ファイルやディレクトリを扱いたい — Cookbook File, Directory

Cookbook File を使うとクックブックに同梱したファイルを任意のパスへ転送して配置 することができます。第 11 章 で紹介した Template も同様の動きをしますが Template は変数を使う場合、Cookbook File は静的なファイルを扱いたい場合とで使い分けると 良いでしょう。

なお標準の Resource には "Cookbook File" ではなく "File" という Resource もあり ます。こちらは Cookbook File のように適当なファイルを転送するのではなく、ファイ ルを一から作成する場合の Resource です。使用頻度は高くありませんので、本書での 解説は割愛します。

ファイルが扱えるならディレクトリも扱いたいですよね。Directory を使うと、ディレクトリの削除や作成が可能です。

Cookbook File

Cookbook File の使い方は簡単で cookbook_file を使って

```
cookbook_file "/tmp/supervisor-3.0a12-2.el6.noarch.rpm" do
  mode 00644
end
```

と書くだけ。これでクックブックディレクトリ内の files/default/supervisor-3.0a12-2.el6.noarch.rpm というファイルが /tmp 以下に転送されます。転送先でファイル名を 変えたい場合などソースファイル名を明示的に指定したい場合は

```
cookbook_file "/tmp/supervisor -3.0a12-2.el6.noarch.rpm" do
  source "supervisor -3.0a12-2.el6.noarch.rpm"
  mode 00644
end
```

と source オプションを使います。他にも owner, group, path などのオプションがあ ります。効果は字面の通りです。詳しくはドキュメントをみてください。

ファイルを転送する際、ファイルが破損していたり改ざんされたファイルを転送してし まうリスクを軽減するためにチェックサムを使うことができます。

```
cookbook_file "/tmp/supervisor -3.0a12-2.el6.noarch.rpm" do
source "supervisor -3.0a12-2.el6.noarch.rpm"
mode 00644
checksum "012
f34db9e08f67e6060d7ab8d16c264b93cba82fb65b52090f0d342c406fbf7"
end
```

転送しようとしたファイルのチェックサムが合致しない場合、実行時エラーになり ます。

Chef のファイルチェックサムは SHA-256 が前提になっています。SHA-256 のファ イルチェックサムは shasum コマンドなどで生成すると良いでしょう。

\$ shasum -a 256 supervisor-3.0a12-2.el6.noarch.rpm

Directory

ディレクトリを操作するのは Directory Resource で。これは 第8章の td-agent の 例で見ましたね。

```
directory '/etc/td-agent/' do
  owner 'td-agent'
  group 'td-agent'
  mode '0755'
  action :create
end
```

アクションは : create もしくは : delete です。

Template や Cookbook File で扱うファイル置き場となるディレクトリが存在しない 場合に Chef が良い感じでディレクトリを作成してくれるかというと、そこまでは面倒み

てくれません。Direcotry を使って明示的にディレクトリを作成する必要があります。

パッケージをファイルからインストール

rpm パッケージが yum にない場合、あるいはオリジナルの rpm パッケージを使いた い場合は Cookbook File と Package を組み合わせるとそのインストールのレシピを書 くことができます。やり方を紹介しておきましょう。

```
rpmfile = "supervisor -3.0a12-2.el6.noarch.rpm"
cookbook_file "/tmp/#{rpmfile}" do
  mode 00644
  checksum "012
    f34db9e08f67e6060d7ab8d16c264b93cba82fb65b52090f0d342c406fbf7"
end
package "supervisor" do
  action :install
  source "/tmp/#{rpmfile}"
end
```

これだけです。ファイル名を二度書かなくて良いように ruby の変数に入れて、それを cookbook_file と package で利用しています。

rpm をファイルで入れる際、場合によってはその rpm に依存している別のパッケー ジが必要になることもあります。例えば上記の例に利用している supervisor は pythonmeld3 や python-setuptools に依存しています。これらの依存モジュールは Chef が良 い感じに解決してくれて、例えば Redhat 系なら yum レポジトリから取ってきてインス トールしてくれます。便利ですね。

第13章

ユーザーを作成したい — User, Group

ユーザーやグループの管理はそれぞれ User と Group Resource を使って記述します。

User

以下に例を示します。

```
user "fiorung" do
  comment "fiorung"
  home "/home/fiorung"
  shell "/bin/bash"
  password nil
  supports :manage_home => true
end
```

ほとんどのパラメータは字面通りで自明なので解説は不要でしょう。

User が取り得るアクションには :create のほかに :remove, :modify, :manage など があります。:modify と :manage はいずれも既存のユーザーを修正するためのアクショ ンですがそれぞれユーザーが存在しなかった場合の動きが異なります。:modify では例 外となってエラーになります。:manage はなにも起こりません。

supports :manage_home => true ここだけ解説しておきましょう。:manage_home => true にすると、ユーザーを新規作成したときにホームディレクトリを一緒に作るように指示することになります。

また supports :non_unique => true は新規ユーザー作成時にノンユニーク ID が 振られても構わない、という指示です。

Group

Group も簡単ですね。新規に xenoblade というグループを作ってそのメンバーに fiorung と dunban というユーザーを追加するには

```
group "xenoblade" do
  gid 999
  members ['fiorung', 'dunban']
  action :create
end
```

とします。

既存のグループにユーザーを追加したい場合は

```
group "xenoblade" do
    action :modify
    members [ 'shulk' ]
    append true
end
```

とすれば OK です。

第14章

git レポジトリからファイルを取ってくる — Git

何かしらのソフトウェアをインストールする場合、多くの場合は 第 9 章 の Package で対応できるのですが場合によっては github などで管理されている git レポジトリから ファイルを取ってきて利用したいという場合もあるでしょう。

その場合に使う Resource は Git です。

Git を使いたい場合、対象ノードにあらかじめ git がインストールされている必要があ りますので注意してください。必要なら git も Package でインストールしておくと良い でしょう。

Git

以下は oh-my-zsh という zsh の設定ファイル集を github から持ってきて設置する例 です。

```
git "/home/vagrant/.oh-my-zsh" do
  repository "git://github.com/robbyrussell/oh-my-zsh.git"
  reference "master"
  action :checkout
  user "fiorung"
  group "xenoblade"
end
```

概ね、見た通りの動きをします。/home/vagrant/.oh-my-zsh ディレクトリに、リモートの git レポジトリから取得してきたファイルが置かれます。

ここでは action に : checkcout を指定しています。この場合、git からファイルを チェックアウトするのは初回時のみで以降リモートレポジトリから取得するというような ことはありません。いうなれば Package のアクション : install に近しい動き。毎回必 要に応じてレポジトリを更新したい場合は : sync を指定します。
user と group は、取ってきたファイルのオーナーとグループを指定するものです。

git でとってきてインストールする

git でとってきて終わり、ではなくその後にとってきたファイルを使って何かしらのイ ンストール作業をしたい場合もあるでしょう。以下は ruby-build をインストールする拙 作レシピです。まだ解説していないリソース Script の bash を使っていますが、例によっ てやっていることは見ればわかると思います。

```
git "/tmp/ruby-build" do
  user node['user']['name']
  repository "git://github.com/sstephenson/ruby-build.git"
  reference "master"
  action :checkout
end
bash "install-rubybuild" do
  not_if 'which ruby-build'
  code <<-EOC
    cd /tmp/ruby-build
  ./install.sh
  EOC
end
```

git で 取得した ruby-build のインストーラを /tmp に置きます。その後、 bash でインス トーラーを起動する。ruby-build が既にインストールされている場合を考慮して not_if で何度も件の処理が実行されないように制御しています。

第15章

任意のシェルスクリプトを実行したい — Execute, Script

やりたいことが他の Resource を使ってもなかなか記述できない場合、任意のコマン ドやスクリプトを実行できる Execute や Script を利用することで解決できることが多い です。Execute, Script ではリソース内に定義したシェルスクリプトなどのスクリプトを root 権限で実行できるので、ほぼ全てのことが実現可能といっても過言ではありません。

ただし、Execute や Script は何でもできる反面ほとんど抽象化がなされていません。 冪等性は自分で保証してやる必要があるし、クロスプラットフォームなどの汎用性も同じ くです。もし Package や Git といったより抽象度の高い Resource を使って記述できる なら、そちらを使うべきです。

Execute、Script ともに任意のコマンドを実行する Resource ですが、bash や perl や ruby など指定したインタプリタを使いたいなら Script を使います。

Script (bash)

ここでは例として、Script Resource の一つである bash を使って、perlbrew をイン ストールするレシピをみていきましょう。

perlbrew をインストールするにはレシピにあるように curl を実行する必要があり ます。

% curl -kL http://install.perlbrew.pl | bash

このシェルスクリプトによるインストーラーの取り扱いが Package や Git では難しい ので、以下のレシピを書きました。

```
bash "install perlbrew" do
  user 'vagrant'
  group 'vagrant'
```

```
cwd '/home/vagrant'
environment "HOME" => '/home/vagrant'
code <<-EOC
curl -kL http://install.perlbrew.pl | bash
EOC
creates "/home/vagrant/perl5/perlbrew/bin/perlbrew"
end
```

スクリプトは特に指定がなくても基本 root 権限 (正確には chef-solo を実行したユー ザーの権限) で動きますが、user や group で指定のユーザーで動かすこともできます。 perlbrew はユーザーのホームディレクトリに入れるものなので、上記では vagrant ユー ザーとグループを指定しています。

cwd はカレントワーキングディレクトリを、environment は環境変数をそれぞれ設定 するオプションです。perlbrew をホームディレクトリにインストールさせるためには環 境変数 HOME がセットされている必要があります。

そして、code にセットしたスクリプトが実行されます。ここではヒアドキュメントで スクリプトを定義しています。

creates

この bash を使ったレシピでひときわ重要なのが creates の行です。creates は、このコマンドがこのファイルを作成するであろうことを指示し且つ既にそのファイルがある場合はこのコマンドを 実行しないことを指定します。

Script レシピは冪等性を保証しないと冒頭で述べた通り、特に何もしなければ Script レシピは **毎回実行されます。** つまり、この場合 perlbrew がインストールされていよう がされていまいが、何度もレシピが実行されてしまうということです。

そこで creates によって実行をガードすることで、既に「あるべき状態」になっているノードにそれ以上の操作は行わない、つまり既に perlbrew がインストールされているならインストールしない、ということを指示しているわけです。

ここでは perlbrew の実行ファイルがあるかどうかをインストールされた/されていない の判定に使っているわけですが、ここもまた Script Resource を使う場合の欠点であり ます。何かがインストールされている/そうではないの判定を独自に記述しなければいけ ないので何かとアドホックな書き方になりがちです。

not_if, only_if

creates はファイルの有無を見てコマンドの実行をガードしますが、より詳細に条件を 指定したいときは not_if や only_if を使います。

- not_if:指定した条件が 真でないならコマンドを実行する
- only_if:指定した条件が真のときのみコマンドを実行する

たとえば creates と同じことは

not_if { File.exists?("/home/vagrant/perl5/perlbrew/bin/perlbrew") }

と記述することもできます。

このとき not_if に与えた値によって条件判定の仕方が変わります。

- 文字列が与えられたとき:与えられたコマンドをインタプリタ (例 bash) で実行してその終了ステータスを判定
- Ruby のブロックが与えられたとき:与えられたコードブロックを ruby で解釈してその真偽値で判定

となります。例えば 第14章 で例に示した ruby-build のインストールでは

```
bash "install-rubybuild" do
not_if 'which ruby-build'
code <<-EOC
cd /tmp/ruby-build
./install.sh
EOC
end
```

と ruby-build の存在有無を which で確認してガードに使っていました。

なお、not_if や only_if ではその条件判定の処理の実行に使われるユーザーや cwd あるいは環境変数を指定することができます。

not_if <<-EOC, :user => 'vagrant', :environment => { 'HOME' => '/home/ vagrant' }

```
...
EOC
```

not_if, only_if は Script Resource に限定の要素ではなく、他の Resource にも利用することができます。もっぱらよく使うのは Script Resource で、ですけれども。

EC2 のマイクロインスタンスにスワップファイルを作る例

Script の使い方の例をもう少し挙げておきましょう。AWS EC2 のマイクロインスタ ンスはインスタンス起動直後にはスワップ領域が設定されていません。マイクロインス タンスを使うにあたってはスワップイメージを作ってスワップを有効化するのはもはや FAQ です。

以下、マイクロインスタンスならスワップを有効にした状態にしたい、という場合のレ シピです。

```
bash 'create swapfile' do
  code <<-EOC
    dd if=/dev/zero of=/swap.img bs=1M count=2048 &&
    chmod 600 /swap.img
    mkswap /swap.img
  EOC
  only_if { not node[:ec2].nil? and node[:ec2][:instance_type] == 't1.
      micro' }
  creates "/swap.img"
end
mount '/dev/null' do # swap file entry for fstab
  action :enable # cannot mount; only add to fstab
  device '/swap.img'
  fstype 'swap'
  only_if { not node[:ec2].nil? and node[:ec2][:instance_type] == 't1.
     micro' }
end
bash 'activate swap' do
  code 'swapon -ae'
  only_if "test `cat /proc/swaps | wc -l` -eq 1"
end
```

まずは dd や mkswap コマンドでスワップファイルを作ります。Ohai はノードが EC2

なのかどうか、また EC2 のインスタンスタイプが何なのかも収集してくれます (すごいですね)。その値を見てマイクロインスタンスならスワップを作ります。また、/swap.img があるかどうかもガード条件に加えます。

次は Mount Resource を使って fstab の設定をいじります。Mount を使って action :enable とすると任意のパスをマウントするように fstab を書き換えることができます。

そこまで終わったら最後に swapon -ae で、スワップファイルをマウントして有効化 します。ガード条件にいい方法が思いつかなかったので、実際のスワップエントリを調べ てひとつエントリがあったらスワップが有効になっているとみなし、それを条件にしてい ます。

このように Script タグを使うことで、スワップファイルを作って有効にするなんてい う芸当も実現できるはできるのですが、ここまで見たようにガード条件をうまく調整する 必要があるなど Execute, Script は両刃の刃です。日頃よりも慎重にレシピの調整をして いく必要があるでしょう。

第16章

その他の Resource

ここまでで Package や Service といった主要な Resource については一通り解説し ました。本書では解説していないそのほかの Resource については ドキュメント に一覧 がありますので、必要に応じてそちらも参照してください。

ここでは、取り扱わなかった Resource について、そのコードをドキュメントから引用 しつつ簡単に概要だけ紹介していきましょう。なお Windows 用 Resource など一部の ものは省略しています。

Cron

Cron は crontab を取り扱うための Resource です。Cron を使って定期バッチ処理の 状態を定義しておくと Chef がいい感じに crontab 周りを設定してあるべき状態に収束さ せてくれます。

```
cron "name_of_cron_entry" do
hour 8
weekday 6
mailto admin@opscode.com
action :create
end
```

Deploy

Deploy は git などの scm からのデプロイを行う Resource です。先に紹介した Git は Deploy に含まれる Resource のひとつのようです。

File

先に 第 12 章 で解説した Cookbook File はクックブック内に置いたファイルをノード へ転送するものでしたが、File はノード上のファイルを直接扱いたいときに利用します。

```
file "/tmp/something" do
  owner "root"
  group "root"
  mode 00755
  action :create
end
```

content オプションを使って任意の文字列をファイル内に書き込むこともできます。

http_request

ノードを調整するにあたって、そのノードからどこかしらの URL へ HTTP リクエスト を飛ばしたいということもあるかもしれません。その場合は http_request を使います。

```
http_request "please_delete_me" do
    url "http://www.opscode.com/some_page"
    action :get
end
```

Ifconfig

ネットワークインタフェースの状態を調整するのが Ifconfig です。IP アドレスの設定時などに利用することができます。

```
ifconfig "192.186.0.1" do
   device "eth0"
end
```

Link

ファイルやディレクトリを扱う Resource は解説済みですが、シンボリックリンクや ハードリンクを取り扱うことももちろん可能です。

```
link "/tmp/passwd" do
   to "/etc/passwd"
   link_type :hard
end
```

Mount

ファイルシステムのマウントを管理するには Mount を使います。実際には fstab のエ ントリなどが、Mount Resource によって調整されることになります。

```
mount "/mnt/volume1" do
  device "volume1"
  device_type :label
  fstype "xfs"
  options "rw"
end
```

Route

ルーティングテーブルの状態を管理するための Resource は Route です。

```
route "10.0.1.10/32" do
  gateway "10.0.0.20"
  device "eth1"
end
```

ruby_block

ruby_block は任意の Ruby のコードを実行するための Resource です。以下は Chef の設定ファイルを再読込するためのレシピの例です。

```
ruby_block "reload_client_config" do
    block do
        Chef::Config.from_file("/etc/chef/client.rb")
    end
    action :create
end
```

筆者はあまり Ruby Block を使ったことがないので、どう使うのが効果的かはまだ計り かねます。すみません。

まとめ

以上、そのほかの Resources の紹介でした。

ここまでに紹介した Resource の組み合わせで、おそらく実現したいことのほとんど はレシピとして記述できるはずです。もちろん、サードパーティのクックブックを取り込 んだりあるいは自分で Resource を定義するなどして拡張していくのも可能です。

Chef のインストールからはじめて一通りの Resource の使い方をここまでで解説し終 えました。おそらく、最初から読んでいただいた読者のみなさんは Chef がだいたいど のようなツールで、どういう風に使えるのかは既に頭に入ったのではないかと思います。 Chef Solo を利用する限りにおいては、この時点での解説で 80% ~ 90% のやりたいこ とは満たせるはずです。

第17章

レシピ落ち穂拾い — run_list, ファイル分 け, include recipe

レシピに関してここまでに解説してこなかった幾つかの点を落ち穂拾い的に取り合って いくとしましょう。

JSON ファイル (Node Object) への run_list の書き方について。それとも関係ある、 レシピのファイル分けについて。default.rb 以外のレシピをどう書いてどう実行するの か。それから他のレシピ内容を取り込む include_recipe について解説します。

run_list の書き方

これまでの例では JSON ファイルの run_list は

```
{
    "run_list":[
        "nginx",
        "apache2"
]
}
```

と書いてきました。この記述では

- nginx クックブックの recipes/default.rb
- apache2 クックブックの recipes/default.rb

が実行されます。

{

以下のように書くこともできます。

```
"run_list":[
"nginx::default",
"apache2",
```

```
"apache2::mod_ssl",
]
}
```

この場合

- nginx クックブックの recipes/default.rb
- apache2 クックブックの recipes/default.rb
- apache2 クックブックの recipes/mod_ssl.rb

がそれぞれ実行されます。

run_list に渡しているものがレシピであることを明示するため

```
{
    "run_list":[
        "recipe[nginx::default]",
        "recipe[apache2]",
        "recipe[apache2::mod_ssl]",
    ]
}
```

とも書けます。

「run_list にレシピ以外のものも渡せるの?」と思った方、鋭い。後ほど解説しますが、 適用したいレシピをグルーピングするのに Role という機能が使えて、run_list には Role を渡すこともできます。

```
{
    "run_list":[
        "recipe[yum::epel]",
        "role[webserver]",
    ]
}
```

レシピのファイル分け

run_list には apache::default で default.rb が、apache::mod_ssl で mod_ssl.rb が実行されると書きました。つまり、

apache2/		
recipes/		
default.rb		
mod_ssl.rb		

と一つのクックブック内にレシピファイルを二つ以上用意することもできる、というこ とです。

ではレシピファイルを複数用意したいのはどういった場合でしょうか。「これこれこう いうルールでファイルを分割しなければならない」という明示的なポリシーは Chef には ないようですが Opscode Community で配布されているクックブックを見ると、なんと なくの方針がわかると思います。

例えば apache2 は default.rb に Apache 本体のインストールそのほかの構成が定義 されていて、mod_ssl や mod_setenvif といった拡張モジュールに関しては mod_ssl.rb や mod_setenvif.rb に定義されています。このようにファイル分けがなされていると、 run_list を書くユーザーは「Apache 本体は使いたいけど mod_ssl は不要」といった Node Object を定義することができて便利です。

Opscode Community のクックブックを使ったり、レシピをより汎用的に書いていこ うとしたときに、おそらくこういったファイル分けを意識する必要が出てくることで しょう。

include_recipe

あるレシピから、別のレシピの内容をそのまま取り込みたいということがあります。そ の場合 include_recipe を使ってレシピ中に

```
include_recipe "bulid-essential::default"
```

などと書きます。上記は build-essential というレシピを取り込むための記述。もちろん build-essential クックブックが cookbook path の通ったディレクトリに配置されて いる必要があります。

普段レシピを書いていて include_recipe が使いたいというケースは多くはないと思い ますが、これもやはりサードパーティのクックブックを利用したりする場合に、ものに よっては include_recipe することを前提に書かれているものがあって、使う必要に迫ら れるかもしれません。

なお include_recipe に似た概念として、レシピ内に他のクックブックとの依存関係を 記述する dependency というシンタックスもありますがこちらは Chef Solo では利用し ません。

第18章

Resource を自分で定義したい — Definition

レシピを書いていると、自分で Resource を定義して使い回したいということが出て くると思います。もちろん、Chef はその方法を提供します。

自分で Resource を定義する方法は幾つかあるのですが、一番簡単なのは Definition という仕組みを使う方法です。以下 Definition での Resource の定義の仕方を解説します。

Definition で cpanm の Resource を定義する

試しに Perl の CPAN モジュールを cpanm を使ってインストールする Resource を 作ってみました。cpanm は Ruby でいうところの gem みたいなものです。 レシピ内に

cpanm 'Path::Tiny'

と書くと Path::Tiny という Perl モジュールがインストールされて

```
cpanm 'Path::Tiny' do
  force true
end
```

と書くと force インストールつまりテストに失敗した場合などでも強制的にインストー ルが行われる、という Resource を作ってみましょう。

Definition を使って Resource を定義する場合、クックブックの definitions ディレ クトリに Ruby のコードを置きます。以下は definitions/cpanm.rb の内容です。

```
define :cpanm, :force => nil do
    bash "install -#{params[:name]}" do
    user node['user']['name']
```

```
cwd node['user']['home']
    environment "HOME" => node['user']['home']
    if params[:force]
      code <<-EOC
        source ~/perl5/perlbrew/etc/bashrc
        cpanm --force #{params[:name]}
      EOC
    else
      code <<-EOC
        source ~/perl5/perlbrew/etc/bashrc
        cpanm #{params[:name]}
      EOC
    end
    not_if <<-EOC, :user => node['user']['name'], :environment => { '
        HOME ' => node['user']['home'] }
      source ~/perl5/perlbrew/etc/bashrc && perl -m#{params[:name]} -e
          .
    EOC
  end
end
```

DSL である define 構文を使って、Resource : cpanm を定義しています。中身で実際 にやっているのは Script (bash) Resource を使って cpanm コマンドを実行する一連の処 理です。

cpanm 'Path::Tiny' と書いたときの Path::Tiny という引数は、Definition には params[:name] というハッシュに格納されて渡ってきます。その値を使って Script Resource を構成しています。not_if を使ったガードにより冪等性を保証するあたりは いわずもがなですね。

このように definitions ディレクトリ内に

```
define :resource_name, :parameter => :argument do
    ...
end
```

という構文により自分で定義した Resource を、同クックブック内で利用することが できます。

より汎用的なレシピを書きたい場合は …… LWRP

Definition を使って Resource を定義するのは非常にお手軽ですが、より汎用的な Resource、特にクロスプラットフォームや第三者に公開することまでを見据えたものを 書きたい場合は、Resource 定義に必要な各種機能がまとまったフレームワークである Lightweight Resource and Providers (LWRP) を使って書くことが推奨されています。

LWRP はそれだけでもそれなりの規模のフレームワークです。本書では解説しませんので、必要に応じてドキュメントを参照してください。

第19章

Attribute と Data Bag

レシピやテンプレート内で動的に扱いたい値は Attribute を使うことで定義することが できました。node[:platform] などで取得できる値ですね。

ここまで Attribute は単なる変数、といった程度の解説しかしてきませんでしたが、 より本質的にみるならそれは "Attribute (属性)" という名前の通り、基本的にノードや Resource の **属性**に関して、固定的に扱うのではないもの、を定義する概念です。

一方、システムに追加されるべきユーザーの各種データなどはノードの「属性」 **でも** ないし Resource の「属性」 **でもない**データと見ることができます。こういったドメイ ンモデル的なデータは Chef のデータ管理の仕組みである Data Bag を使うほうが扱いや すい。

ここでは Attribute を落ち穂拾い的にみつつ、Data Bag の使い方も見ていくとしましょう。

Attribute

Attribute は例えばテンプレートの中で <%= node['nginx']['port'] => と書いたり、あるいはレシピの中で node[:platform] として取得するのでした。

node[:platform] は Ohai が収集したそのノードの属性ですが、JSON ファイルの Node Object への記述を通して Attribute を決めることもできましたね。

```
{
    "nginx" : {
        "port" : 80
    },
    "run_list":[
        "yum::epel",
        "recipe[setup]"
    ]
}
```

以上はここまでに解説してきた方法ですが、Attribute はそのデフォルト値をあらかじ めて決めておくこともできます。クックブック内のディレクトリの attributes ディレ クトリに default.rb という名前でファイルを作り

```
default["apache"]["dir"] = "/etc/apache2"
default["apache"]["listen_ports"] = [ "80", "443" ]
```

と定義すると、テンプレートやレシピから node["apache"]["dir"] と記述してこれ らの値を取り出すことができます。

Attribute は、

- 初期値をあらかじめクックブック内に定義しておく
- Role 毎に値を定義しておく (Role については後ほど解説)
- JSON ファイルでノード毎に値を定義しておく

と幾つかの方法で定義することができますが、それぞれ同じキーで値を定義した場合 attributes ディレクトリに定義されたデフォルト値よりも、Role、Role よりも JSON ファイルに記述された Node Object に定義された値の方が優先されます。

この優先順位を巧く利用して、普段は初期値に任せつつ、特定のノードのみ別の値で Attribute を上書きするという使い方が可能です。

Attribute の使いどころ

冒頭での繰り返しになりますが Attribute はその名の通り「属性」です。Chef は「ノードの状態を管理し収束させるフレームワーク」という話がありました。そしてノードや、 ノードをの状態構成する要素である Package や Service といったものをそれぞれ「リ ソース」と見立てた場合に、その各リソースのプロパティに相当するのが Attribute (属性) です。

従って Attribute は

- ノード (というリソース) のプロパティである IP アドレスを定義する
- ウェブサーバー (というリソース) のプロパティであるバインドポートを定義する

といった具合にリソースに属した性質、つまり「属性」を扱うのに向いた機能である、

といえます。

Data Bag

ー方、よりグローバルなドメインモデル的なデータは Data Bag を使って扱うと良いで しょう。Chef で複数台のノードを扱っている場合に、それらのノード全てに複数のユー ザーを追加したい……より Chef らしくいうと各ノードに複数ユーザーがいる状態を定義 したい、という場合それらユーザーのデータが、Data Bag で扱うべき類のデータです。 (LDAP や Active Directory で共有するリソースを思い浮かべる方もいると思いますが、 まさにそういう感じだと思ってください。)

Data Bag はクックブック単位ではなく、レポジトリ全体にグローバルなスコープの データです。ここではシステムに二人のユーザーを定義する例を示しましょう。

Data Bag で扱うデータは <レポジトリ>/data_bags ディレクトリに置きます。ここ では users というサブディレクトリを切って、その中にユーザーデータのオブジェクト を定義する JSON ファイルを作ります。

それぞれのデータオブジェクトは以下のように JSON で定義します。

```
// dunban.json
{
    "id" : "dunban",
    "username" : "dunban",
    "home" : "/home/dunban",
    "shell" : "/bin/bash"
}
// fiorung.json
{
    "id" : "fiorung",
    "username" : "fiorung",
```

```
"home" : "/home/fiorung",
"shell" : "/bin/bash"
}
```

これで Data Bag へのデータの格納は完了です。これらのデータを取得して、実際に ユーザーを定義するレシピを他に用意しましょう。例えば login_users というクック ブックを用意してそのレシピに

```
# login_users/recipes/default.rb
data_ids = data_bag('users')
data_ids.each do |id|
    u = data_bag_item('users', id)
    user u['username'] do
    home u['home']
    shell u['shell']
    end
end
```

と書きます。

data_bag('users') で data_bags/users ディレクトリに置かれたユーザーデータオ ブジェクトの id 一覧が取得できます。data_bag_item('users', id) で、指定した ID のオブジェクトを Data Bag から取り出すことができます。

あとはいつも通り User Resource を使って、そのユーザーを追加……つまりリソースの状態を定義しているわけですね。

このように Data Bag は Chef をデータベースのようなものと見立てて、そこにデータ を登録してレシピから検索するのに使える機能です。

なお、Data Bag にはパスワードや鍵などのデータを暗号化して格納しセキュアに扱う方法も用意されています。その機能を Chef Solo で使う方法は Chef Solo encrypted data bags などを参照してください。

第20章

ノードを役割ごとにグルーピングして管理 したい — Role

Chef Solo で管理してくるサーバーが増えてくると、Node Object を定義するファ イルである JSON ファイル数がそれに応じて増えてきます。Node Object に定義した run_list には実行するレシピが列挙されていますが、ノード数が多くなってくるとそれ を管理するのが面倒になってきます。例えば同一のレシピを適用している 5 つのノード あったとき、新しいレシピを追加したくなった場合にどうするか。5 回書く、というのは DRY でないしナンセンスですね。

そこで run_list や Attribute をノードの役割ごとにグルーピングできる Role の出番です。

Role

Chef Solo の場合 Role は roles ディレクトリ配下に Role 毎にファイルを作って、その中に Role の状態を例によって JSON で記入します。

例えばウェブサーバー用のノードの Role を roles/webserver.json に作ってみましょう。

```
{
    "name": "webserver",
    "default_attributes": {},
    "override_attributes": {},
    "json_class": "Chef::Role",
    "description": "",
    "chef_type": "role",
    "run_list": [
        "recipe[yum::epel]",
        "recipe[nginx]",
        "recipe[sysstat]"
]
```

と書きます。run_list のところに注目してください。Role 内にレシピが列挙されて いますね。つまり webserver とい Role は yum::epel, nginx, sysstat レシピを run_list として持ちなさいという定義になります。

Role を定義したら、あとは使うだけ。Node Object の run_list にはレシピだけではな く Role も指定することができます。ノード melody の Node Object である JSON ファ イル nodes/melody.json に

```
{
    "run_list":[
        "role[webserver]"
]
}
```

}

と書いて knife solo cook melody を実行すると、Role に定義された run_list が ノード melody に対して適用されます。

なお Node Object の run_list にはレシピと Role を混ぜて書くことも可能ですし、 Role を複数当てはめることもできます。

```
{
    "run_list":[
        "recipe[mkswap]",
        "role[webserver]",
        "role[database]"
]
}
```

Role に Attribute を定義する

Role の JSON ファイルを見てお分かりの通り、Role でグルーピングできるのは run_list だけではありません。Attribute も Role 毎に定義することができます。

```
default_attributes "apache2" => {
    "listen_ports" => [ "80", "443" ]
}
```

Role を使うと複数ノードの状態管理が論理的かつすっきりとしますね。積極的に使っていきましょう。

第21章

サードパーティのクックブックを

Bundler 風に管理したい — Berkshelf

先に 第7章 でみたように、knife cookbook site コマンドで Opscode Community で公開されているクックブックをリモートからインポートしてくることができたのでした。

この Opscode Community のクックブックは Ruby で言うところの gem のようなものです。そして gem に同じく、複数のクックブックを扱うようになってくると管理がやや煩雑になってきます。

gem の場合 Bundler を使うことで、それらの問題を解決することができますね。もは や Rubyist にとって Bundler は必要不可欠のツールになっています。

Chef のサードパーティのクックブックを Bundler のように管理したい、というときに 使えるのが Berkshelf というツールです。Berkshelf は rubygems として公開されてい ます。特に Chef を利用するにあたって Berkshelf の利用は必須ではないのですが、覚え ておくと役に立つと思いますので簡単に紹介しておきましょう。

Berkshelfの使い方

Bundler では gem を Gemfile にリストアップして bundle コマンドで一括でインストール & 管理することができました。Berkshelf の場合も同様に Berksfile にクックブックをリストアップして berks コマンドで一括インストール、です。

早速使ってみましょう。まずは Chef のレポジトリに移動します。

\$ cd chef-repo

Berkshelf をインストールしましょう。ここは折角なので Berkshelf を Bundler で入れましょう。

Gemfile
source :rubygems
gem 'berkshelf'

という Gemfile を作り

\$ bundle --path vendor/bundle

で vendor/ 以下に Berkshelf に必要な gem が全てはいります。vendor ディレクトリ は .gitignore に入れておくと良いでしょう。

Berkshelf の準備は整ったので、Berksfile というファイルを作り中に利用したいクッ クブック一覧をリストアップします。(事前に Opscode Community の鍵の設定が必要な のは 第 7 章 で述べた通りです) Berksfile のシンタックスは Gemfile とよく似ています。

Berksfile
site :opscode
cookbook 'yum'
cookbook 'nginx'

以上で準備は完了。あとは berks コマンドを実行してクックブックを持ってきましょう。Bundler で入れたので bundle exec berks ですね。

```
$ bundle exec berks --path cookbooks
Using yum (2.1.0)
Using nginx (1.3.0)
Using bluepill (2.2.0)
Using rsyslog (1.5.0)
Installing runit (0.16.2) from site: 'http://cookbooks.opscode.com/api/
v1/cookbooks'
Using build-essential (1.3.4)
Using ohai (1.1.8)
```

--path で指定したクックブックディレクトリ内に、必要なクックブックが全てダウン ロードされてきます。yum や nginx に依存しているクックブックも一緒に取得できてい るのが分かりますね。

あとはいつも通り Node Object なり Role なりの run_list に recipe[yum] や recipe[nginx] を設定してクックブックを使うだけ!

Berkshelf と Vagrant の連携

このようにとても便利な Berkshelf ですが、Berkshelf には Vagrant と連携する機能 があります。これを利用すると Berkshelf で Vagrant の初期化から Chef Solo の実行ま でを一括で管理することができます。

簡単に例を見てみましょう。

Berkshelf + Vagrant を使って Chef する場合 Chef レポジトリを berks cookbook コマンドで作ります。(Berkshelf ではレポジトリ = クックブックディレクトリひとつ、という構成のディレクトリレイアウトが想定されているようです。)

見てのとおりクックブックに必要なファイルの他、Berksfile や Vagrantfile が生成されます。

Gemfile には Vagrant の gem 定義が書かれています。

```
% cd sandbox
% bundle
```

で gem をインストール。

次に、Berksfile を編集。

```
site :opscode
metadata
cookbook 'yum'
cookbook 'nginx'
```

あとは Attribute や run_list の編集ですが、Node Object は例の JSON ファイルでは なく Vagrantfile 内に記述します。Vagrantfile 内に

```
config.vm.provision :chef_solo do |chef|
chef.json = {
    :mysql => {
        :server_root_password => 'rootpass',
        :server_debian_password => 'debpass',
        :server_repl_password => 'replpass'
    }
    chef.run_list = [
        "recipe[sandbox::default]",
        "recipe[yum::epel]",
        "recipe[yum::nginx]"
]
```

という記述があります。ここに Attribute や run_list を定義します。Vagrant には Chef と連携する機能がもともとついていて Berkshelf はそれをうまく利用しているんですね。 これで準備完了

% bundle exec vagrant up

で、仮想サーバーの起動と同時に Chef Solo が走り、Berksfile に書かれたクックブッ クがリモートから取得されてレシピが適用されます。

サーバー起動後に Chef を再度走らせたいなというときは

% bundle exec vagrant provision

を実行すると Vagrantfile に記述された Node Object の内容に合わせて Chef Solo が 動きます。

まとめ

以上、Berkshelf の解説でした。

• Berkshelf を使うと Opscode Community のクックブックを Bundler のように管

理できる

• Berkshelf の Vagrant 連携機能で、Vagrant とのより密な連携も可能

後者の Vagrant との連携部分に関しては利用するかどうか検討の必要があると思いま すが、前者の機能を単体で使う分には特に迷う部分はないと思います。Opscode からの クックブックが多数になってきたら Berkshelf の導入を検討してみてください。

第22章

Chef Server の様子を知りたい — 概要か らセットアップまで

小規模なシステムであれば Chef Solo だけでも管理は可能ですが、ある程度以上の規 模では Chef Server の導入も検討すべきです。Chef Server について検討するためには、 Chef Server がどんなものなのかを知っておく必要がありますね。

ここでは Chef Server のあらましと、Vagrant のマルチ VM の機能を使って試験環境 を構築する方法までを解説します。

Chef Server を使った場合、Chef Solo とはどの辺りが違うのかに着目して見ていくとしましょう。

Chef Server のアーキテクチャ

Chef Solo は管理対象のサーバーで chef-solo コマンドを叩いてローカルに保存され たレシピを実行するコマンドでした。knife-solo を使うとリモートへのレシピ転送と ssh 経由でのコマンド実行を通じて、あたかも手元のマシンからリモートホストを直接管理し ているかのようでしたが、裏側では、やはりレシピがノードのローカルにあってそれを chef-solo が実行しているという構図は変わりません。

一方 Chef Server を導入した場合は、Chef Server がノード一覧やレシピ、Role といった、Chef Solo でレポジトリに保存していた情報などをサーバーサイドのデータベースとして管理します。そして各ノードは、そのノードにインストールされた Chef Client を通じて Chef Server からそれらのデータを取得し、必要ならレシピを実行し自分自身の状態を変更します。

Chef Server と Chef Client は HTTPS 通信で JSON をやりとりして通信を行います。 つまり、Chef Server と Chef Client のやりとりは JSON over HTTPS で、Chef Client → Chef Server への PULL 型のアーキテクチャになっています。Chef Server にぶら下 がっている Chef Client は、定期的に Chef Server にリクエストを送ることによって状 態の更新がないかを監視し、自分自身をあるべき状態に保つように働きます。



図 22.1 Chef Server

Chef Server では Chef Client からの HTTP リクエストを待ち受けるために nginx と Erlang で書かれた erchef というサービスが起動しています。nginx は Client からリク エストを受け取ると、バックエンドに待ち構えている erchef ヘリクエストを転送すると いう、いわゆる Reverse Proxy の役割を果たしています。また、Chef Server ではレシ ピや run_list を含む構成管理用の各種情報を保存するためのデータベースなどのミドル ウェアも動きます。

システム管理者は手元のマシン (Chef Server の文脈で "Work Station" と呼ばれます) などから knife コマンドで Chef-Server に情報を送ります。例えばクックブックや Data Bag をアップロードしたり、ノード一覧にノードを追加したり削除したり。それら情報 は Chef Server のデータベースに保存されます。

このように Chef Server のデータベースに保存された情報を Chef Client が HTTPS で取得してくるのが全体のアーキテクチャで、サーバーサイドのデータベースを活用する 分、Chef Solo 単体では実現できない各種機能が提供されることになります。

特に重要なのは各種インフラ情報の検索性です。Ohai が収集する様々なノードの情報 を軸に、Chef Server に登録された全ノードを検索することができます。

これを応用して例えばロードバランサーと Chef Server を連携させて、システムに新 しいウェブサーバー用のノードが追加されたらロードバランサーに追加する、あるいは その逆で削除する……なんてことが自動化できるようになります。それ以外にも、例え ばハードウェアの情報や役割に基づいた検索……「メモリが 4GB 以上のノード一覧が欲 しい」ですとか、「データベースのマスター」といったメタデータでの検索が可能になり ます。

大規模なインフラ環境では、どのサーバーがどういう役割でどのような状態でいるかと いう情報を管理するだけでも一苦労なため、サーバー一覧を検索・管理するためのウェブ アプリケーションを独自に開発したりします。筆者が過去に携わったシステムにはいずれ もそういったアプリケーションがありました。こういったサーバー管理ツールのデータ ベースを Chef Server のそれと共通化することで Chef でのノードの状態管理や追加削 除に合わせてサーバー管理ツールからフィードされる情報をも動的に更新することができ るでしょう。Chef Server はそういった外部ツールと連携するためのサーバーサイド API も提供します。

ノード群に配布する必要がある各種ファイル……例えばパッケージファイルや tar ボー ルなどをサーバーに蓄積しておくことができたり、あるいはブラウザからノードを管理す る UI が提供されたり……などといった機能も Chef Server は提供します。

このように Chef Server には中規模以上でのシステム運用を見据えた、Chef Solo に はない各種機能が搭載されています。

本書では解説しませんが Opscode がホスティングする Chef Server の SaaS を Chef Client のサーバーに設定することで、完全に Chef Server の運用をアウトソースするこ とも可能です。Facebook はこのサービスを利用しているとのことです。

Chef Server 環境化での各種操作例

Chef Server を利用する場合、各ノードの状態管理は基本的に Chef Server とのやり とりによって行うことになるのは前述の通りです。(Chef Solo では knife コマンドはレ ポジトリを作成する程度にしか利用してきませんでしたが、Chef Server 環境では knife が Server や各ノードを管理するための中心的役割を果たします。)

例えばシステムに存在するノード一覧を得るには knife client list でサーバーー 覧を問い合わせます。

```
$ knife client list
chef-validator
chef-webui
node01
node02
```

クックブック、レシピは手元で作成してできあがったものを、サーバーにアップロード して、ローカルではなくサーバーに保存します。

\$ knife cookbook upload -a -o ~/chef-repo/cookbooks Uploading sample [0.1.0] Uploaded all cookbooks. \$ knife cookbook list sample 0.1.0

出力に [0.1.0] とバージョンタグがついていることからも分かる通り、アップロードしたコードはバージョン別に管理することができます。あるバージョンをコードフリーズしてデグレードを回避したり、あるいは緊急時の過去のバージョンへの切り戻しといった用途に利用することが可能です。

run_list を含む Node Object や Role Object もやはりサーバー側で管理されます。例 えば以下は、node01 という名前のノードの Node Object の run_list にレシピを追加す る例です。

```
$ knife node run_list add node01 'recipe[sample]'
node01:
  run_list: recipe[sample]
```

こうして Chef Server に、ノード一覧やクックブックや Node Object 等を登録してい くと、Chef Server がシステム上のあらゆるリソースの状態を知っている、という状態に なります。

管理される対象である各ノードで chef-client を実行すると Chef が走ります。

```
$ sudo chef-client
```

Chef Client は Chef-Server に問い合わせが行って各種データを取得しそれに基づき Chef を実行することで自分自身の構成をあるべき状態まで収束させます。

Chef Client はデーモン化することで Chef Server を定期的に polling させることができます。

```
$ sudo chef-client -d
```

これにより、管理者は Chef Client に Chef を走らせることをまったく気にせず Server の操作をおこなうだけでシステム全体の更新を行うことができます。

各ノードがいつ Chef Client を実行してアップデートされたかは、knife status で確認します。

```
$ knife status
4 minutes ago, node02, node02, 10.0.2.15, centos 6.3.
4 minutes ago, node01, node01, 10.0.2.15, centos 6.3.
4 minutes ago, node03, node03, 10.0.2.15, centos 6.3.
```

Chef Server なら細かな条件を指定してノードを検索できる、と述べました。knife search を利用します。

FQDN が "node" で始まるノードを検索
\$ knife search node "fqdn:node*"
Platform が CentOS のノードを検索
\$ knife search node "platform:centos"
CentOS で Ruby 1.8 系のノードを検索
knife search node "platform:centos AND languages_ruby_version:1.8*"

knife ssh コマンドを利用すると任意のノードに ssh でコマンドを発行することがで きますが、ノードの指定には knife search と同じクエリが利用できます。

\$ knife ssh node "fqdn:node*" "uptime" node03 06:35:22 up 1:47, 1 user, load average: 0.00, 0.00, 0.00 node01 06:35:22 up 1:48, 1 user, load average: 0.00, 0.00, 0.00 node02 06:35:22 up 1:47, 1 user, load average: 0.00, 0.00, 0.00 # 各ノードで chef-client を実行 \$ knife ssh "fqdn:node*" "sudo chef-client" # 各ノードで chef-client を demonize \$ knife ssh "fqdn:node*" "sudo chef-client -d" -x vagrant

以上は Chef Server の機能の一部です。

いかがでしょう、たしかにサーバーが 1,2 台では必要なさそうだけれども、もしそれ が十数台、数百台だったらどうでしょう。これらの機能が活躍する場面を容易に想像でき

るのではないでしょうか。

Vagrant のマルチ VM による環境構築

Chef Server を自分自身で触って検証してみることを想定して、試験環境の構築方法を 簡単に解説しておきましょう。

Server を検証するなら複数台の試験用ホストがあると便利です。Vagrant で一つ一つ ホストを立ち上げていっても良いのですが、ここは Vagratn のマルチ VM 機能を使うと 良いでしょう。一つの Vagrantfile から複数の仮想サーバーを立ち上げることができます。 Vagrantfile に

```
Vagrant::Config.run do |config|
 config.vm.define :chef_server do |cfg|
    cfg.vm.box = "base"
   cfg.vm.network :hostonly, "192.168.30.10"
    cfg.vm.host_name = "chef_server"
 end
 config.vm.define :workstation do |cfg|
    cfg.vm.box = "base"
    cfg.vm.network :hostonly, "192.168.30.20"
    cfg.vm.host_name = "workstation"
 end
 config.vm.define :node01 do |cfg|
    cfg.vm.box = "base"
    cfg.vm.network : hostonly, "192.168.30.21"
    cfg.vm.host_name = "node01"
 end
 config.vm.define :node02 do |cfg|
    cfg.vm.box = "base"
    cfg.vm.network :hostonly, "192.168.30.22"
   cfg.vm.host_name = "node02"
 end
 config.vm.define :node03 do |cfg|
    cfg.vm.box = "base"
   cfg.vm.network :hostonly, "192.168.30.23"
    cfg.vm.host_name = "node03"
 end
```

end

と複数サーバーの設定を記述してやってから

\$ vagrant up

で全仮想サーバーが順次起動します。もちろん指定した IP アドレスでホスト同士が通信することが可能です。終了はいつもどおり vagrant halt ですべてが停止します。

ここでは例として5つのホストを作成しています。Chef Server と、Chef Server に knife でコマンドを投げるだけの作業用ホストである Work Station、試験用のノード3つ です。ログイン後に対象を区別しやすいように、Vagrantfile 内でホスト名の設定も行っ ています。起動状態は vagrant status で確認可能です。

```
$ vagrant status
Current VM states:
chef_server running
workstation running
node01 running
node02 running
node03 running
```

ssh は vagrant ssh コマンドの引数に VM 名を与えます。

\$ vagrant ssh chef_server
\$ vagrant ssh node01

ssh-config の書き出しも同様です。

```
$ vagrant ssh-config chef_server >> ~/.ssh/config
$ vagrant ssh-config node01 >> ~/.ssh/config
```

Vagrant + CentOS 6.3 を利用する場合、環境構築中に躓かないようホストの iptables をあらかじめ stop しておきましょう。(前にも書きましたが決してこれを本番サーバーで やらないこと!!) 筆者はこれではまりました。

```
$ sudo /etc/init.d/iptables stop
```
Chef Server のインストール

本稿執筆時点での Chef のバージョンは 11 です。以前までのバージョン 10 とは導入 方法が色々と変わっていますので知見がある方はその点に注意してください。

Chef Server をインストールするには、例によって Opscode Omnibus Packaging を 利用するのが楽で良いでしょう。http://www.opscode.com/chef/install/ で、Chef Server 用の各種ディストリビューション向けのパッケージをダウンロードすることができるので それを利用します。

Chef Client と Chef Server でインストール方法が異なっている点に注意してください。

nstall Chef		
oose to install either Chef C	lient or Chef Serve	r and follow the instructions be
ownload option	ns	
Chef Client	Chef Server	
nstalling the Chef :	Server	
Select the kind of system you The versions listed have been	would like to install n tested and are sup	the Chef Server on.
Enterprise Linux \$		
6 *		
x86_64 ‡		
Downloads You can install manually by d	ownloading the pack ual installation, pleas	kage below after you have se we read the wiki.
11.0.6 ‡		
chef-server-11.0.6-1.el6.x	86_64.rpm	
nstructions		

図 22.2 Chef Server のダウンロード

CentOS の場合は rpm パッケージをダウンロードしてインストールする方式です。

\$ sudo rpm -Uvh chef-server-11.0.6-1.el6.x86_64.rpm

パッケージでのインストールが完了したら、chef-server-ctl で Chef Server を設定 します。

\$ sudo chef-server-ctl reconfigure

これで Server が起動し、nginx が HTTPS 443 ポートをバインドします。(この chef-server-ctl reconfigure は、Chef Server に必要な各種ミドルウェアの構成を 行うのですがその構成もまた Chef でやっているのがなかなか面白いところです。)

なお、Chef Server はサーバーに FQDN を割り当てて名前解決できる状態を前提にしているので、Vagrant で VM を立ち上げたまっさらな状態などの環境では /etc/hosts をいじるなどの必要があります。もしくは /etc/chef-server/chef-server.rb に以下のような設定ファイルを置いてから chef-server-ctl reconfigure を再実行してください。

```
# /etc/chef-server/chef-server.rb
# サーバーのIPは192.168.20.20を想定
server_name = "192.168.20.20"
api_fqdn server_name
nginx['url'] = "https://#{server_name}"
nginx['server_name'] = server_name
lb['fqdn'] = server_name
bookshelf['vip'] = server_name
```

なお、Chef の各種ファイルは /ops ディレクトリ以下にインストールされます。

knife の設定

次に knife configure で knife を設定します。

ここで、"Chef Client" についてもう少し詳細に説明しておきましょう。Chef における "Client" とはノードに限ったことではなく、Chef の API と通信できるすべての端末を指 す言葉です。その文脈で言うと knife も Chef Client の一つになります。そして、Chef Server は自分自身の API と通信できる端末をデジタル署名によって識別します。

任意の Chef Client がはじめて Chef Server と通信する場合、デジタル署名がありま せん。そこで初回のみ、chef-validator という Client の登録作業のみを専門にする Client から証明書を借りて Chef Client の登録作業を行います。この登録作業が完了すると、以 降は Client 側で作成したデジタル署名によって通信が行われるようになります。 rpm パッケージで Chef Server をインストールすると /etc/chef-server 以下に

- chef-validator.pem : chef-validator 用のデジタル署名ファイル
- admin.pem:管理用 Client のデジタル署名ファイル (この Client 署名のみ、 chef-validator を通さずに予め作成されている)

が設置されています。knife configure では knife にこれらのデジタル署名のパスを 正しく設定する必要があります。

その knife ですが、まずはサーバと同一ホストの knife を動くようにし、その次に Work Station 用の Client デジタル署名を生成して Work Station に転送する、という手順で セットアップします。

まずは /etc/chef-server 以下の pem ファイルをホームディレクトリに移します。

```
$ sudo cp /etc/chef-server/chef-validator.pem ~/.chef/validation.pem
$ sudo cp /etc/chef-server/admin.pem ~/.chef/
$ sudo chown -R vagrant:vagrant ~/.chef/*.pem
```

次に 'knife configure を実行して初期化を行います。なお、本稿執筆時点では knife は まだ Chef 11 を前提としていないようで各種設定項目をデフォルトのまま利用した場合 うまく動きません。サーバー URL、各種秘密鍵のパスを環境に合わせて入力します。

```
$ knife configure
Where should I put the config file? [/home/vagrant/.chef/knife.rb]
Please enter the chef server URL: [http://chef_server:4000] https
    ://192.168.30.10
Please enter an existing username or clientname for the API: [vagrant]
    admin
Please enter the validation clientname: [chef-validator]
Please enter the location of the validation key: [/etc/chef/validation.
    pem] ~/.chef/validation.pem
Please enter the path to a chef repository (or leave blank):
```

ここでは

- server URL: https://192.168.30.10
- clientname : admin
- validation key: /etc/chef-server/chef-validator.pem

を設定しました。これでこの knife は Client の証明書である ~/.chef/admin.pem を 使って通信を行えるようになります。試しに使ってみましょう。

\$ knife client list
chef-validator
chef-webui

Work Station のセットアップ

knife をサーバーにログインしないと利用できないのでは本末転倒ですね。管理用の作 業環境を想定した Work Station からも knife が使えるように設定していきましょう。

新しい Chef Client をセットアップする場合、まずはデジタル署名の生成と Client の Server への登録が必要です。これも knife で行うことになります。サーバーにセット アップした knife を使いましょう。

knife client create を実行するとエディタでメタデータの編集を要求されるので、 EDITOR 環境変数をセットしておきます。knife client create へのオプション -a は 管理権限を持ったクライアントの作成、-f は署名ファイルの出力先です。Vagrant を 使っている場合仮想サーバーの /vagrant は共有ディレクトリとして mount されていて他 仮想サーバーへのファイルの転送に便利です。そこに出力してしまいましょう。

```
$ export EDITOR=vi
$ knife client create workstation -a -f /vagrant/workstation.pem
```

Work Station の knife にも validation.pem を持たせておくと後々便利です。そちらも コピーしておきましょう。

```
$ cp ~/.chef/validation.pem /vagrant
```

ここから、Work Station 側での作業になります。まずは Work Station に Chef Solo の時にも利用した Opscode Omnibus Packaging のインストールスクリプトを流して Chef を入れます。

```
$ vagrant ssh workstation
$ curl -L https://www.opscode.com/chef/install.sh | sudo bash
```

```
$ chef-client -v
Chef: 11.4.0
```

あとはサーバーでやったのと同様に knife を設定します。

```
$ mkdir ~/.chef
$ cp /vagrant/workstation.pem ~/.chef
$ cp /vagrant/validation.pem ~/.chef
$ knife configure
WARNING: No knife configuration file found
Where should I put the config file? [/home/vagrant/.chef/knife.rb]
Please enter the chef server URL: [http://workstation:4000] https
    ://192.168.30.10
Please enter an existing username or clientname for the API: [vagrant]
    workstation
Please enter the validation clientname: [chef-validator]
Please enter the location of the validation key: [/etc/chef/validation.
    pem] ~/.chef/validation.pem
Please enter the path to a chef repository (or leave blank):
```

Server URL および先ほど生成、コピーした鍵のパスを指定します。

ただしく設定できていれば knife client list で一覧が出力されて、新しい Client として workstation が登録されているはず。

```
$ knife client list
chef-validator
chef-webui
workstation
```

Chef Client のインストール

Chef Server と Work Station が完了したらあとは、Client です。管理対象のノード側 を Chef Client としてセットアップします。

Client 側への Chef のインストールも Opscode Omnibus Packaging のインストール スクリプトを流せば OK です。

```
$ curl -L https://www.opscode.com/chef/install.sh | sudo bash
```

```
$ chef-client -v
```

Chef: 11.4.0

次にこの Client が先ほど立ち上げた Chef Server のノードとしてぶらさがっても良い ノードであるということを保証するために、chef-validator の鍵をコピーして保存します。 サーバーの /etc/chef-server/chef-validator.pem をノードの /etc/chef/validation.pem に 保存してください。Work Station のセットアップでコピーした validation.pem を使って も結構です。

\$ mkdir -p /etc/chef
\$ sudo cp /vagrant/validation.pem /etc/chef/validation.pem

これで準備は完了です。

ノードで chef-client を実行して、本ノードをサーバーに登録しましょう。--server でサーバー URL を指定し、-N でノード名を指定します。(このオプションを省略したい 場合は /etc/chef/client.rb を設定すると良いです。詳しくはマニュアルを見て下さい。)

\$ sudo chef-client --server https://192.168.20.20 -N node01

なお、この初回の Client 実行のタイミングで /etc/chef/client.pem にこのクライアン トのデジタル署名が生成されて、以降の通信は (validation.pem ではなく) この鍵が使わ れます。

Chef Server にノードが登録されているかどうか Work Station の knife で確認してみましょう。

```
# Work Staion で
$ knife client list
chef-validator
chef-webui
node01
workstation
```

レシピを実行させる

だいぶ長くなりましたが、以上で Chef Server および Work Station 、そしてノードの セットアップと Server への登録は完了です。(これでも以前のバージョンに比較すると

かなり楽になったそうです。)

試しにレシピを実行させてみましょう。Template リソースのところで解説した、Ohai を取得するテンプレートを配置する例を実践してみます。Work Station 上でクックブッ クを作成して Server にアップロードし、Client 側でそれを適用してみましょう。

```
# クックブックの作成
$ knife cookbook create sample -o ~/chef-repo/cookbooks
# レシピの作成
$ vi ~/chef-repo/cookbooks/sample/recipes/default.rb
template "/tmp/chef-sample.txt" do
 mode 0644
end
# テンプレートの作成
$ vi ~/chef-repo/cookbooks/sample/templates/default/chef-sample.txt.erb
Platform: <%= node[:platform] %>
Ruby: <%= node[:languages][:ruby][:version] %>
IP Address: <%= node[:ipaddress] %>
   クックブックのアップロード
#
$ knife cookbook upload -a -o ~/chef-repo/cookbooks
Uploading sample
                     [0.1.0]
Uploaded all cookbooks.
# Node Object の run_list にレシピ追加
$ knife node run_list add node01 'recipe[sample]'
node01:
 run_list: recipe[sample]
# Node Object をエディタで編集したい場合
$ knife node edit node01
```

これでサーバーへの必要情報の登録は完了しました。 ノード側で chef-client を実行してファイルの中身を確認してみましょう。

```
$ sudo chef-client --server https://192.168.20.20 -N node01
$ cat /tmp/chef-sample.tx
Platform: centos
Ruby: 1.8.7
IP Address: 10.0.2.15
```

うまくできましたね。

knife bootstrap によるノードのセットアップ

Chef Solo では knife solo prepare でノードを簡単に設定できたのに、Chef Server ではずいぶんと面倒だと感じた方も多いと思います。Chef Server はその性格上セット アップがやや煩雑になる (それでもずいぶん簡単になったんです) のはいたしかたないと ころではありますが、ノードは knife bootstrap コマンドを使うともっと簡単に済ませ ることができます。

Work Station から \$ knife bootstrap 192.168.30.21 -x vagrant --sudo -N node01

これだけで、新しく立ち上げたホスト (まだ Chef すら入ってないホストでも) が Chef Ready になり、Chef Server に登録されます。

knife bootstrap は

- Chef のインストール
- /etc/chef ディレクトリの作成
- chef-validator 用の鍵のコピーと設置
- chef-client の設定ファイルでる client.rb の設置
- chef-client を実行しての Chef Server への登録

までを一気通関で行ってくれます。先ほどの面倒な手続きは必要ありません。knife bootstrap を使うと client.rb を良い感じに設定まで面倒みてくれるので、chef-client 実 行時に --server や -N といったオプションが不要になります。可能なら knife bootstrap でのセットアップを試みたほうがよいかもしれませんね。

なお、Vagrant で立ち上げた仮想サーバーを knife bootstrap でセットアップする場合

- 対象サーバーへ ssh ログインできるようにホスト OS の ~/.vagrant/insecure_private_key を Work Station の ~/.ssh/id_rsa にコ ピーしておく
- 仮想サーバーに予め入っている古いバージョンの Chef がセットアップの邪魔になるのでどけおく。/usr/bin 以下の chef-* ファイルをどこかに mv しておけば OK

• Work Station の /etc/hosts に各ノードの IP アドレスを登録しておく

必要があります。

最後の一点は knife ssh がホストへの通信を FQDN で行おうとするための対策です。

```
# /etc/hosts
...
192.168.30.21 node01
192.168.30.22 node02
192.168.30.23 node03
```

まとめ

以上、Chef Server のあらましと試験環境セットアップ方法でした。

Chef Server のサーバーサイド API を軸にした管理機構はとても強力です。特にその 検索機能は自動化はもちろん、管理コスト負担を軽減するのに大いに貢献してくれること でしょう。

セットアップはだいぶ簡単になったとはいえ、やはり Chef Solo に比べると色々と面倒 ですね。小規模な環境で使うのはためらうこところかもしれません。本書がまずは Chef の使い方を覚えるのに Chef Solo からはじめたかったのはこの辺りもひとつの理由です。

自分たちの管理するサーバーの規模と比較し Chef Server によって得られるメリット が大きいようであれば、ぜひ Server も導入を検討してみてください。

参考文献

• 並河祐貴, 中島弘貴, 吉田晃典, 桑野章弘, Chef 入門, Software Design 10 月号, 2012

第23章

どこまでを Chef でやるべきか

入門 Chef Solo もそろそろ終わりです。

最後に、実際に Chef を使うにあたってどこからどこまでを Chef あるいは Chef Solo で管理していくべきなのかについて筆者なりの考えを述べておきたいと思います。

サーバー管理は可能な限り Chef でやる

Chef でサーバーを管理するにあたっては、その作業全般を可能な限り Chef を使って 行うようにしましょう。例外的にでも、ssh でログインして直接パッケージをインストー ルしたりといったことは 一切しません。

なぜなら何度か触れたとおり Chef は「ノードの状態を管理し、そのノードをあるべき 状態に収束させるフレームワーク」だからです。ノードの状態を管理するためには、ある ノードにまつわるすべての状態はレシピに書かれていなければなりません。例外的にでも ローカルでの構成変更を行ってしまうと、レシピに書かれていない状態が対象ノードに含 まれることになってしまいます。

もちろん、レシピの調整中にうまくいかないことがあって調査する、なんてときは ssh ログインしてログを見て回ったりはします。しかしそれはあくまで調査であって、構成変 更のためのログインではありません。

すべてを Chef で構成管理しようとすると、自作のレシピがまだ少ない頃は何をやるに しても時間がかかるとは思います。しかし、レシピを書きためていくうちにだんだんとそ の様子が逆転して、いつしか手元のレシピをいじって構成変更するほうが効率的かつ精神 衛生上もすっきりする……というタイミングがくるでしょう。

用途ごとにレポジトリを用意すべき? いいえ

すべてのサーバーを Chef で管理するとして、サーバーは役割ごとに構成が全く異なり ます。ウェブサーバー、アプリケーションサーバー、データベース、バッチ処理用のサー バー…… etc

Chef のレポジトリはそれぞれの役割ごとに作成して管理するべきでしょうか。いいえ、 そこは Role を使いましょう。レポジトリは一個に保ち、クックブックをなるべく他用途 とも使い回せる形で書いておき、Role 内の run_list や Attribute の組み合わせによって それぞれの役割ごとの状態を定義しましょう。

なお、対象システムのサービスが異なる場合はその限りではありません。例えば同一企 業でブログサービスと EC サービスを展開していた場合にそれらのレポジトリを一緒にす べきか別にすべきかは、それぞれのサービスの裏側のシステムをどこまで共通化している かに依るでしょう。独立性が高ければ Chef のレポジトリも独立させるべきですし、そう でなければそれらも Role でうまく管理できるのではないかと思います。

複数ノードへの Chef Solo の実行

Chef Solo で複数以上のサーバーを管理するとなると、knife-solo の実行を複数ホストに実行する必要が出てきます。その場合は 第5章 でも解説した通り

\$ echo user@node1 user@node2 user@node3 | xargs -n 1 knife solo cook

とシェル上でまとめてバッチで実行してしまえば良いでしょう。

他にも knife-solo を使わずにデプロイツールでリモートホストの chef-solo を実行す る方法などもあります。定番は Ruby の capistrano というツールと連携させる方法で す。capistrano はもともと Rails アプリケーションのデプロイ用ツールですが、ローカ ルのマシンから複数ホストへ ssh で任意のコマンドを実行させることができるツールな ので、こういった目的にも利用することができます。

Chef Solo と Chef Server

このようにシェルや何かしらのデプロイツール + Chef Solo を使えば Chef Server な しでも複数のノードを管理することは可能です。

ただし可能は可能ですが、対象サーバーがある程度以上の規模になったらそこはもう素 直に Chef Server の導入を検討するほうが良いだろうと思われます。

Chef Server を利用する場合、Chef Client をインストールした各ノードが Chef Server

に登録されたデータを取得する PULL 型のアーキテクチャを採ります。管理者による構 成変更は、基本 knife コマンドで Chef Server に通知・登録する形になります。各ノード にインストールされた Chef Clinet は定期実行モードで定期的にサーバーの状態を監視さ せることができ、Client は Chef Server の様子をみて、自分自身の状態が更新されるべ きと判断したら Chef を流し、状態をあるべき姿に収束させます。スケーラブルなアーキ テクチャですし、なにより系全体が Chef の思想 — あるべき状態への収束 — という思惑 で連携するのでエレガントです。

Chef Server には、例えばノードー覧の取得や管理、レシピー覧や run_list の管理、 Data Bag の登録と検索、クックブックのバージョン管理といった Chef Solo にはない サーバーとしての機能、重要な役割が搭載されています。単に Chef Solo を複数台に展 開したという以上の価値が Chef Server にはあります。

デプロイツールとの棲み分け

ノードの状態にまつわることはすべて Chef で賄うと述べましたが、アプリケーション のデプロイまでを Chef でやるかはおそらく議論の分かれるところでしょう。ここでいう アプリケーションとは nginx や MySQL といったパッケージ化されたソフトウェアのこ とではなく、自分たちで書いた Rails アプリケーションといったコードのことです。

Chef のリソースには Deploy というリソースがあって、これを使うとどうやらアプリ ケーションのデプロイも Chef 任せにすることができるようです。これを使ってデプロイ を行うという手もあるでしょう。

しかし、アプリケーションのデプロイにはより細かな制御、あるいは緊急時のロール バックの機能といった固有の要求事項があることが多くあります。この辺りは、やはりそ れに特化したツール capistrano や Cinnamon 等に任せたほうがよい……という考え方 も強くあるでしょう。

筆者はいまのところ後者の考えを採用して、アプリケーションの領域は Cinnamon で 直接デプロイしそれ以外の部分を Chef で管理するようにしています。

ここはどちらでやるべきという主張を前に思考停止せずに、対象のシステムにはどちら の方法が向いているかを冷静に判断して選択していきましょう。

筆者が Chef を使い始めた理由

筆者は現在複数のウェブサービスを個人で運用していて、そのすべてのサーバーの状態 管理に Chef を使っています。対象のシステムはいずれも小規模なので今のところ Chef Solo だけで間に合っています。

しかし、サービスを立ち上げてしばらくは Chef は利用していませんでした。

サーバーはすべて AWS で運用しています。AWS にはインスタンスのスナップショットを任意のタイミングで取得できる機能があり、これを使うとサーバー状態のバックアップや複製を簡単に行うことができます。例えばサーバーを増やしたいと思ったらスナップショットを取ってそこから別のインスタンスを立ち上げれば OK ですし、大きな変更前に構成をバックアップしておきたい場合にもスナップショット任せです。

このスナップショット機能をはじめとする AWS の各種サービス利用すると、そうでは ない場合に比較して運用が非常に楽になりました。それだけでも十分に楽だったので、こ れ以上の自動化は必要無いかな……なんて思っていました。

ところがスナップショットを多数とって管理しているうちに、どのスナップショットが どういった状態で保存されたものなのかを管理するのが大変になってきました。また、イ ンスタンスを複製して切り替えを行う場合、スナップショットを取った時点から新しいイ ンスタンスを立ち上げるまでの差分の間の構成変更分は、手で反映する必要があり、その ためにどういった構成変更を行ったかは記憶しておくかメモしておくかする必要がありま した。場合によってはそれを忘れて、新しく立ち上げたインスタンスが一部デグレードし ているなんてこともありました。

こうしてだんだんとスナップショットだけでの構成管理が大変になってきたところで、 Chef を導入してみることにしました。はじめは Chef を「ただの自動ビルドツール」と 思って利用していたのですが、使い続けるうちにこれはまさに「ノード状態を管理するた めのフレームワーク」であると理解し、スナップショット地獄で悩まされていた自分の問 題に対するベストな解決方法だなと感じるようになりました。すべての状態はコードに記 載されているので記憶に頼る必要はありませんし、履歴は git でバージョン管理されてい ます。インスタンスを立ち上げたらとりあえずレシピを適用しておけば、思ったとおりの 状態にノードが収束してくれます。こんなに良いことはないですね。

以来、ノードを構成しているすべてのリソースを Chef で管理するようになり、新規に

サービスを立ち上げる場合もすべて Chef で賄うようになりました。

おわりに

入門 Chef Solo、いかがでしたでしょうか。

筆者は過去に、1,000 台以上の規模のオンプレミスのサーバー管理を行った経験があ ります。それは楽しさもあったと同時に非常に面倒で苦痛も伴う仕事でした。はじめて Chef を触ったときに「これがあればあの頃の手間暇の多くをなかったことにできる」と 感じ、多くのエンジニアのみなさんにこのツールを知ってもらいたいと思うようになりま した。

Chef は 2 年ほど前から先進的なユーザーの間で話題になっていました。それが昨年 2012 年中頃からはより広範囲で話題に上るようになって来たように思います。2013 年 の今年は、クラウド (laaS/PaaS) がより本格的に普及する年になる勢いです。その勢い に後押しされる形で Chef のようなフレームワークに注目が集まっているということなの かもしれないなと思います。

2月に Engine Yard さんのオフィスで Chef の勉強会をやるというので、Engine Yard の @yando さんから、何か発表して欲しいと声をかけていただきました。Vagrant と knife-solo の簡単なプレゼンテーションを行いました。その場で Chef ユーザーのみなさ んとコミュニケーションをしているうちに、Chef の実行の仕方といったごくごく最初の 部分で躓いている人がたくさんいることに気がつきました。あちこちで断片的になってし まっている Chef の情報を、一度まとまった形にして配布する必要があるのではないか、 と感じました。

前々から一度、電子書籍という形で書籍を出版してみたいと思っていたのも後押しして、今回こうして Chef Solo 本を書くにいたりました。

本書をきっかけに Chef に興味を持っていただけたなら、それに勝る喜びはありません。 最後に不躾なお願いではありますが、もし本書をお読みいただき何かしらを感じていた だけたなら、Amazon.co.jp のレビューやブログにでも、その感想を投稿していただける と嬉しいです。

入門 Chef Solo

2013年3月22日v1.0.0版発行

著 者 伊藤直也

発行所 達人出版会

(C) 2013 Naoya Ito