Chef Solo入門

Infrastructure as codeへようこそ！

先端技術研究センター

王 海東

2014/03/12

Chef

Agenda

1. 最近のインフラ技術
2. 必要なRuby知識
3. Chef Soloとは
4. インストールと環境設定
5. アーキテクトと構成要素
6. まとめ

Chef

最近のインフラ技術

Chef

ここ最近のインフラ系技術の流れがおもしろい
キーワードは

Infrastructure as code

Test-Driven Infrastructure

Immutable Infrastructure

Chef

「Infrastructure as Code」とは、

インフラをどのように構築し、維持するべきかという定義はDSLなどの自然言語と近い文法で記述され、ソースコードのように扱うことができます。つまり、インフラの構成管理をコードによって行えることになります。

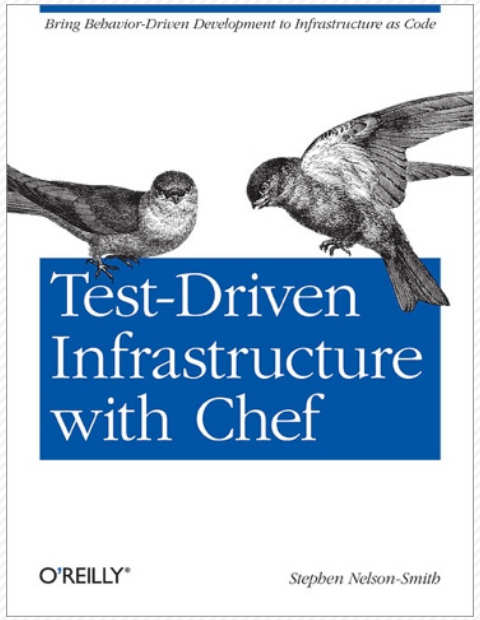
　プログラミングによる手順書を作成したり、実作業を人手に任せたりする必要がなく、プログラミング言語によるインフラの定義を作成し、後の動作はプログラムに任せて自動化できます。

Chef

「Test-Driven Infrastructure」とは、

インフラストラクチャーを対象とした自動テストを作成する方法です。

スクリプト化された変更をインフラストラクチャーに対して適用するたびに、このようなテストを実行すれば、その変更によって環境にエラーが取り込まれたとしても確実に素早いフィードバックを得ることができます。



ChefSpec

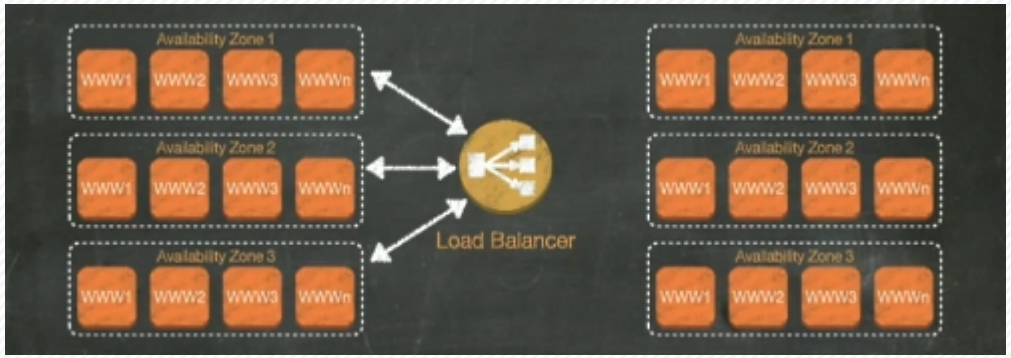
SERVERSPEC

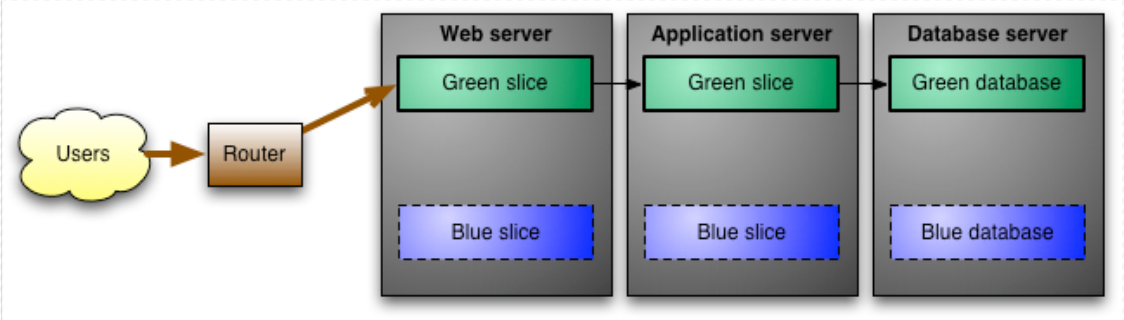
RSpec tests for your servers configured by Puppet, Chef or anything else.

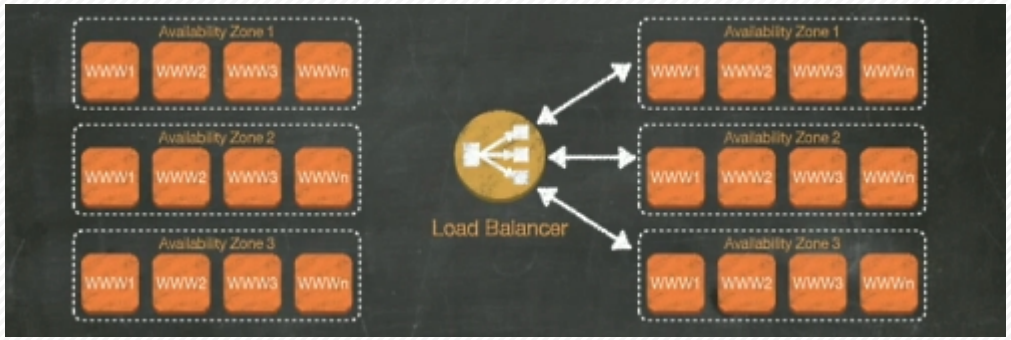
「Immutable Infrastructure」とは、

サーバを一度セットアップしたら二度と変更を加えないという運用スタイルのことを指します。

クラウド環境では、必要に応じてすぐにサーバを用意し、不要になったら簡単に破棄することができます。Immutable Infrastructure は、このようなクラウドの特性を活かす運用スタイルとして、注目されつつあります。

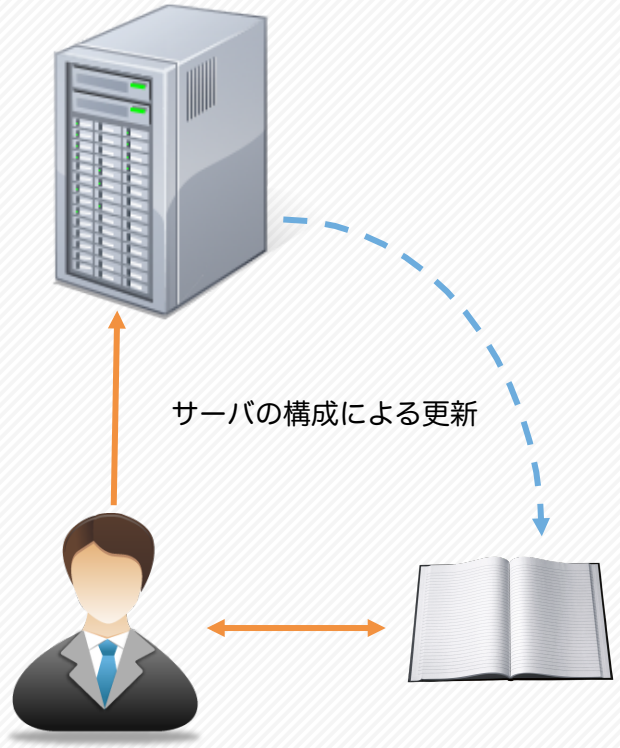






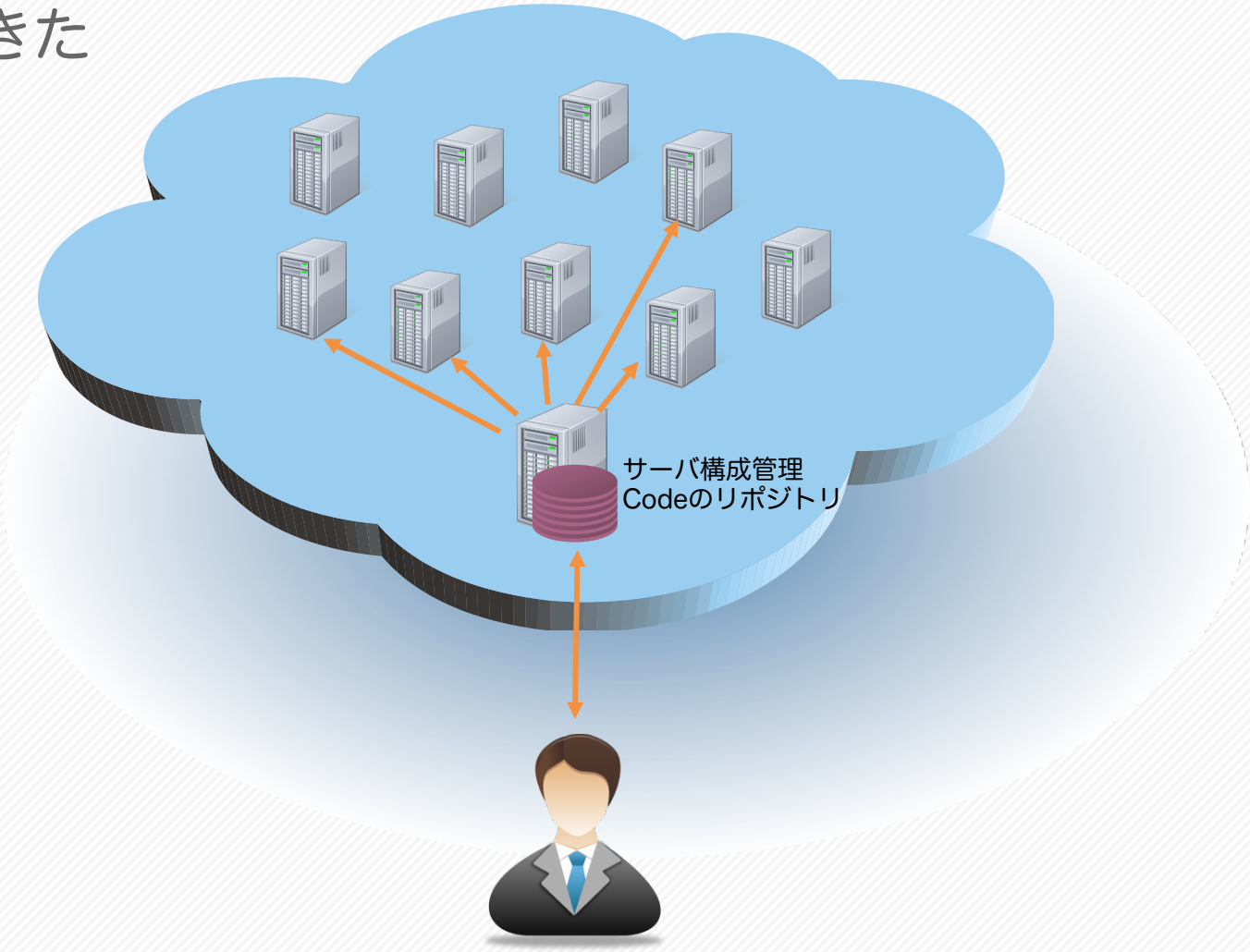
※ 図はMartin Fowler氏のブログ、「AWS re:Invent」2012にAmazon CTOのキーノートから参考

背景はCloudの技術が成熟してきた



仮想化

自動化



昔手順書をベースにして、
サーバを構築してから手順書を更新する。
2重作業になっている。

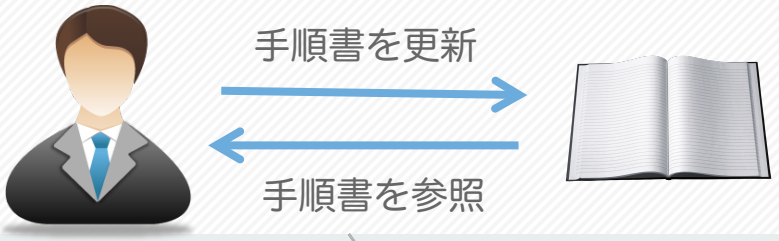
手順書をコード化にして、サーバ構築はコード実行による自動化にされる。

本来のインフラ作業

Apache

少数のサーバでは手作業を対応できるが、
数千台の規模なら、やはり無理！

手作業で操作ミスの可能性は高い！
手順書の作成、メンテナンスは面倒



1 電源を入れ、OSをインストール アカウント作成

2 必要なソフトをインストール、設定

3 アプリをデプロイ、設定

4 運用、監視 (ソフト、アプリの更新 サーバ入れ替え)

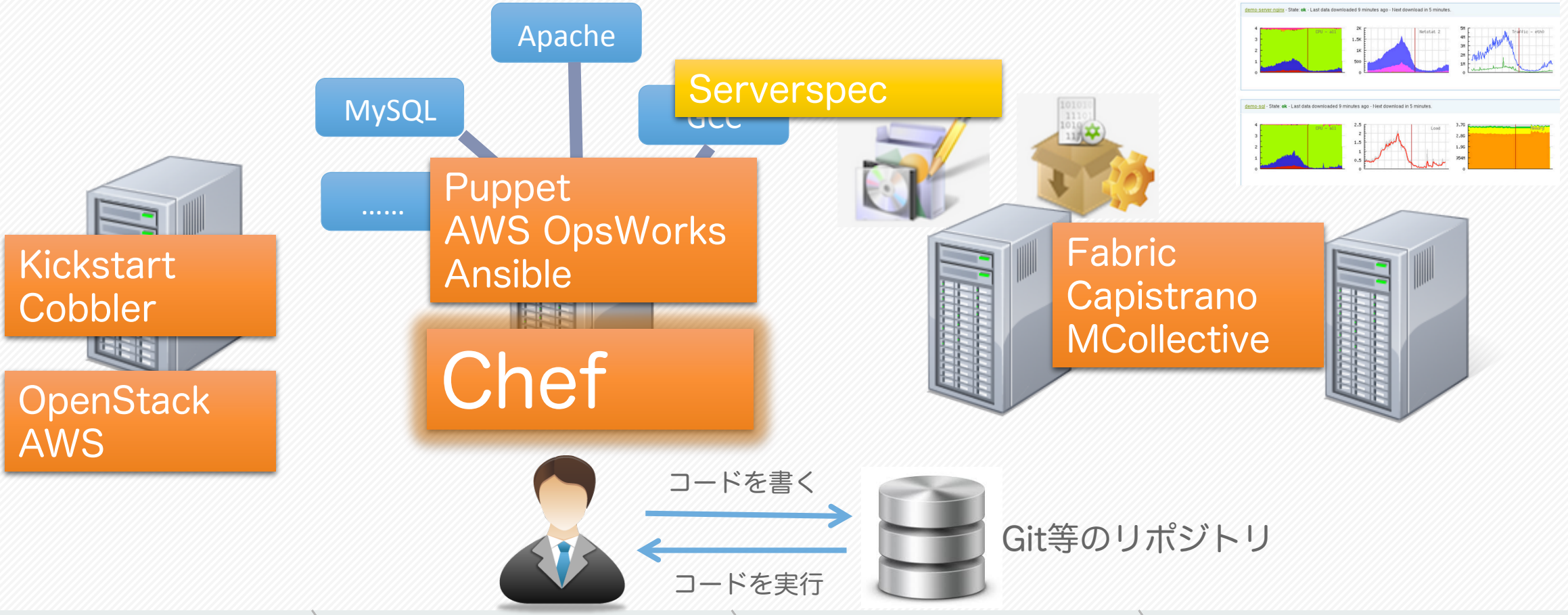
Chef

IT技術の本質は人を機械的な作業から解放すべき

サーバ構築を自動化したい

サーバ構築をテストしたい

今からのインフラ作業



1 電源を入れ、OSをインストール アカウント作成

2 必要なソフトをインストール、設定

3 アプリをデプロイ、設定

4 運用、監視 (ソフト、アプリの更新 サーバ入れ替え)

Chef

Chef を理解するための最低限のRuby知識

Chef

Rubyにおけるすべてのものはオブジェクトである

以下ではClassmethod社のブログを参考する
http://dev.classmethod.jp/server-side/chef-vagrant-ruby/

Rubyは括弧を省略して書く事ができる

```ruby
def hello name
  print "Hello World. #{name}\n"
end

# 定義したメソッドを呼び出す
hello "Hakamata"
```

=

```ruby
def hello(name)
  print("Hello World. #{name}\n")
end

# 定義したメソッドを呼び出す
hello("Hakamata")
```

Rubyは do や end を中括弧で書く事ができる

```ruby
count = 0
loop do
  break  if count == 3
  puts count
  count+=1
end
```

=

```ruby
count = 0
loop {
  break  if count == 3
  puts count
  count+=1
}
```

Chef

シンボル

```ruby
person = {:name => "Hakamata", :age => 35, :address=> "Tokyo"}
person = {name: "Hakamata", age:35, address:"Tokyo"} #この書き方でも同じ意味

# ハッシュから値を取り出して標準出力に出力する
puts person[:name]
```

キーワード引数

```ruby
def hello name:"Ruby"
  print "Hello World. #{name}\n"
end

# キーワード引数に代入
hello :name => "Hakamata"
```

ブロック付きメソッド（Chefのrecipeでよく使われている構文）

ブロックの呼出をyieldで宣言

```
def メソッド名 引数
  yield ブロック変数
end
```

ブロックを定義して渡す

```
メソッド名 メソッドの引数の変数名 do |ブロック変数|
  # yieldがある箇所に入れたい処理
end
```

```
def write_log
    File.open 'test.log', 'w' do |f|
        f.flock File::LOCK_EX

        yield f

        f.flock File::LOCK_UN
    end
end
```

```
write_log do |f|
    f.puts 'hello log'
end
```

ブロックを使った繰り返し

「3」もオブジェクトである

```
3.times do |i|
  # 3回だけ実行したい処理
  puts i
end
```

%w[make gcc openssl-devel] という部分は [“make”, “gcc”, “openssl-devel”] と同じ意味になる

```
animals = ["lion", "elephant", "giraffe"]
animals.each do |animal|
  puts animal
end
```

```
%w[make gcc openssl-devel].each do |name|
  package name do
    action :install
  end
end
```

Chef

Chef Soloとは

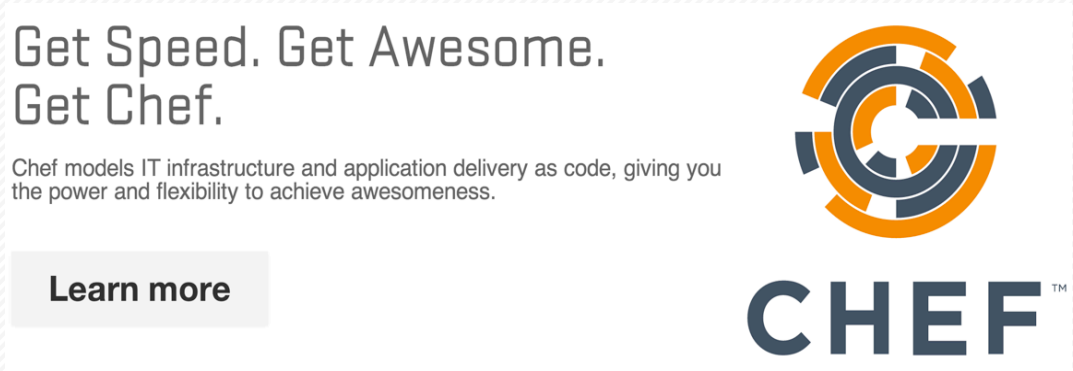
ちょっと待って、Chefはまだわからない

Chef

Chefの人気度ははるかにリードしている

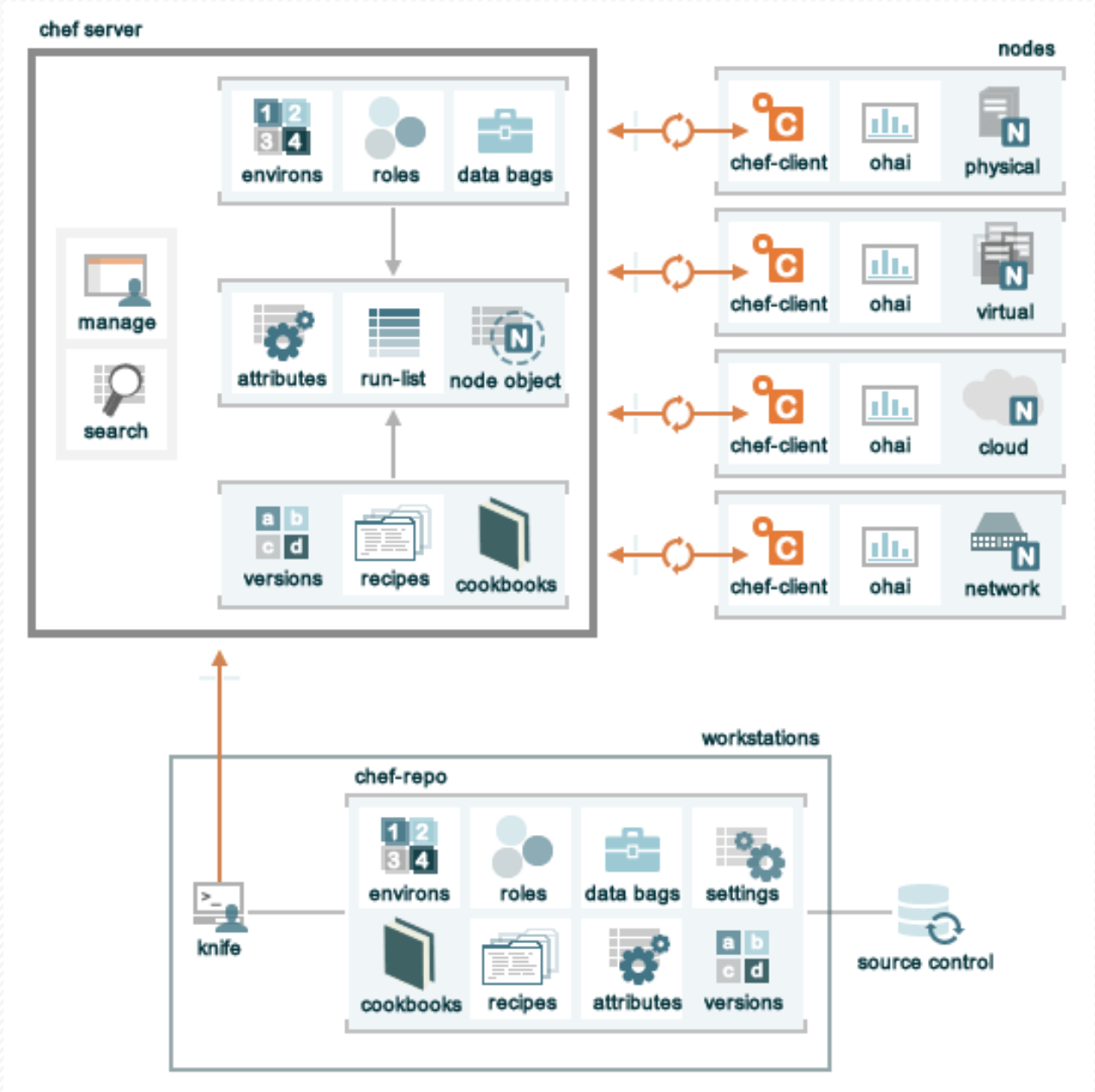
Chefとは

- Chef software(旧名：Opscode)が開発されたオープンソースのインフラ自動化ツールです。
- RubyのDSLで書きます。
- 複数のサーバにより構成されるシステムを運用管理するためのフレームワークです。
- システムが稼働するOSのディストリビューションによる差異を吸収し、同一の構成を複数の環境で再現することを容易にしています。
- 構成管理情報やノード(サーバ)のステータスを集約するChef-Serverと、ノードの状態を収束させるChef-Clientという構成と、単一のノードを構築するChef-Soloが利用できます。
- Chefのレシピ（Recipe）をコミュニティに公開され、誰でも再利用できます。インフラのノウハウを共有し、関連技術を促進します。



※ 図はchefオフィシャルサイトから参照

Chef

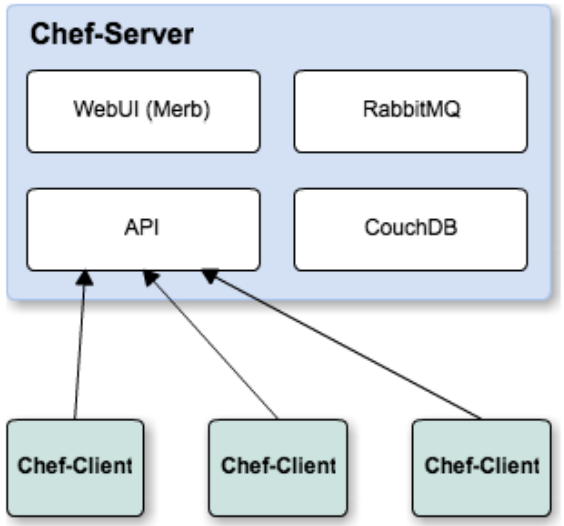


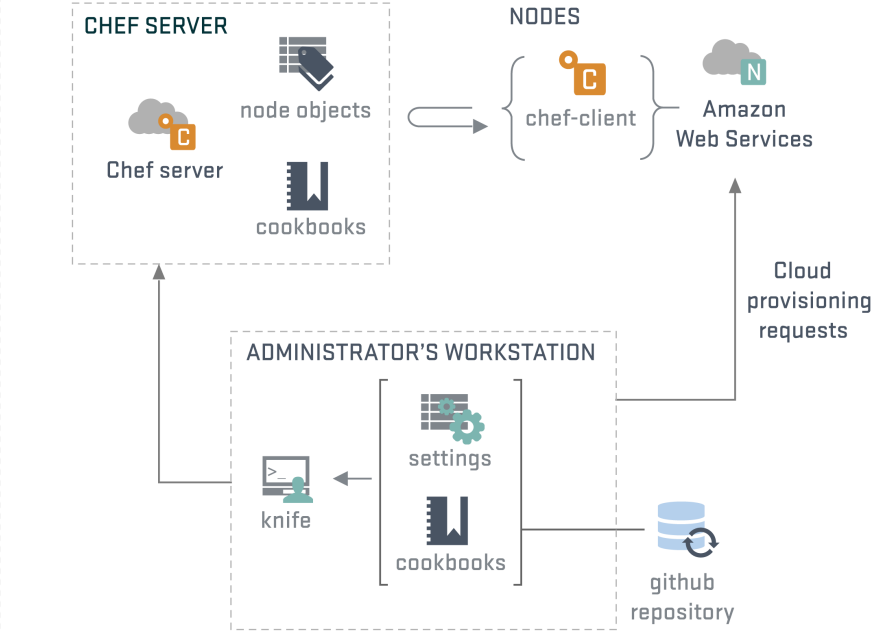
Client/Server構造

- 担当者は自分のマシンでcookbooksを作成
- Chef Serverにcookbooksを登録
- 作業対象（node）を選定
- 作業対象にChef Clientをインストール
- 作業対象でChef Clientを動かして、Chef Serverに問い合わせ、cookbooks等情報を参照し、自分の環境を構築、更新

※ 図はchefオフィシャルサイトから参照

Chef





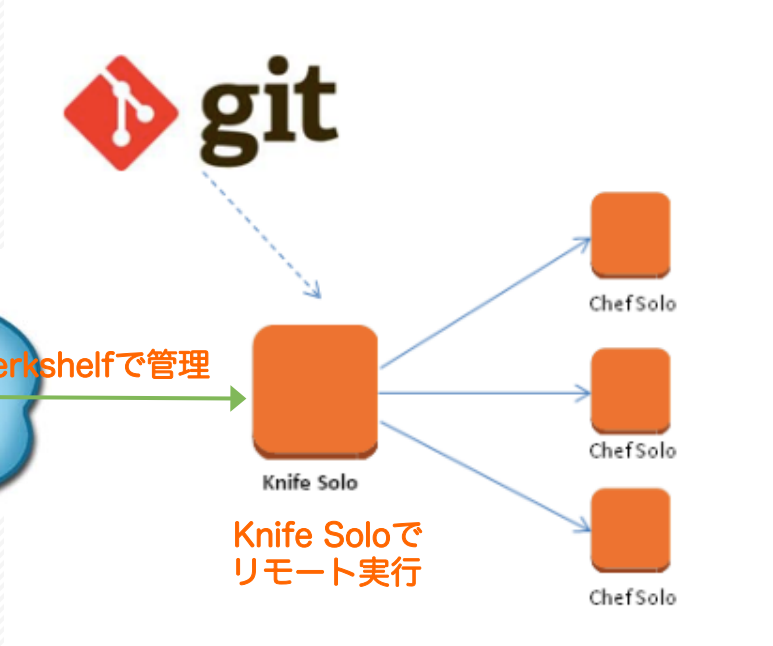
Client/Server構造

- Chef Serverは最新のミドルウェア（Erlang、CouchDB、RabbitMQ、Apache Solr等）を使用するので、簡単に構築できない
- Opscode社はChef Serverのクラウドサービスを提供
- AWSはChefを使用して、AWS OpsWorksサービスを提供
- 既存資産はChef Serverと連携、統合も難しい

※ 図はchefオフィシャルサイトから参照

それでは、Chef Soloとは

Chef Serverが無しで対象マーシンで
Chef Clientを実行する仕組みです。



データセンターのマシン、またクラウドVMなら、SSH登録でChef Clientを実行する必要なので、Knife Soloを併用します。

Gitは必須ではありません。

インストールと環境設定

ここからKnife soloとChef soloで環境構築を解説します。
まずChef soloとKnife soloの環境を構築します。

```
# https://github.com/sstephenson/rbenv
$ rbenv version
1.9.3-p194 (set by /home/wang/.rbenv/version)
```

Rubyで書かれますので、Rubyの環境は必須
1.9.3以上のバージョンが必要

```
$ gem i chef --no-ri --no-rdoc
```

Chef Clientを入れ（Knifeも入れました)

```
$ gem i knife-solo --no-ri --no-rdoc
```

Knife soloを入れ

```
$ gem i berkshelf --no-ri --no-rdoc
```

Cookbookを管理するツールBerkshelfを入れ

インストールと環境設定

早速Chef soloとKnife soloを試します。

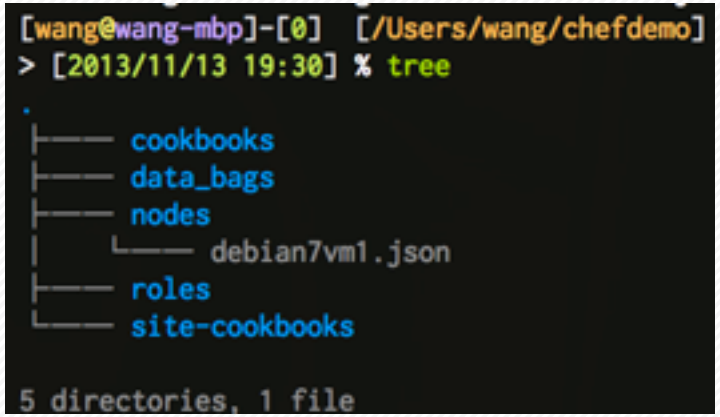
```
$ knife solo init chefdemo
```

Cookbookのscaffoldを生成

Cookbookを手動で作成できます。また、Berkshelfを利用して、http://community.opscode.com/cookbooksから既存のcookbookをダウンロードします。

```
$ cd chefdemo/
$ knife solo prepare wang@debian7vm1
```

Cookbookを対象サーバに適用する設定を準備




```
$ cat nodes/debian7vm1.json
{"run_list":[]}
```

何のcookbookも指定していない

```
$ knife solo cook wang@debian7vm1
```

実行

一体何を発生しましたか

```
$ knife solo prepare wang@debian7vm1
```

作業マシン

1. ssh wang@debian7vm1
2. chef soloをインストール

対象マシン

事前設定
- wangアカウントはdebian7vm1に事前作成済み
- SSH鍵認証が設定済み
- wangアカウントのsudoを有効化
- 頻繁にパワード入力を避けるため、ユーザのsudo設定にNOPASSWDを推奨

```
$ knife solo cook wang@debian7vm1
```

作業マシン

1. chefdemoをdebian7vm1に伝送
2. ssh wang@debian7vm1
3. chef soloを実行

対象マシン

Chef

Chef で何をできますか

- ✓ OSアカウント、グループの作成、設定
- ✓ OSシステム設定（ネットワーク等）
- ✓ ソフトウェアのインストール、アンインストール
- ✓ ファイル、ディレクトリを操作
- ✓ MySQL、Apacheのようなミドルウェアの設定ファイルを管理、更新
- ✓ 任意なスクリプト（Bash、Ruby、Python、Powershell）を実行
- ✓ OSサービスの起動、停止
- ✓ Ruby、Python等のモジュールをインストール（gem、easy_install）
- ✓ 非同期のジョブ（cron、batch）

Hello chef

```
[wang@wang-mbp]-[0]  [/Users/wang/chefdemo]
> [2013/11/13 20:07] % tree
.
├── cookbooks
├── data_bags
├── nodes
│   └── debian7vm1.json
├── roles
└── site-cookbooks

5 directories, 2 files
```

第三者が作成したcookbook
例えば、http://community.opscode.com/からダウンロードしたもの

自分で作ったcookbook

```
$ cd chefdemo/
[wang@wang-mbp]-[0]  [/Users/wang/chefdemo]
> [2013/11/13 20:31] % knife cookbook create hello -o site-cookbooks
** Creating cookbook hello
** Creating README for cookbook: hello
** Creating CHANGELOG for cookbook: hello
** Creating metadata for cookbook: hello
```

Cookbook helloを作成

Hello chef

```
[wang@wang-mbp]-[0]  [/Users/wang/chefdemo]
> [2013/11/13 20:39] % tree site-cookbooks/
site-cookbooks/
└── hello
    ├── CHANGELOG.md
    ├── README.md
    ├── attributes
    ├── definitions
    ├── files
    │   └── default
    ├── libraries
    ├── metadata.rb
    ├── providers
    ├── recipes
    │   └── default.rb
    ├── resources
    └── templates
        └── default

11 directories, 4 files
```

hello cookbookの構造(後ほど解説)

helloの操作をRubyで記述

```
[wang@wang-mbp]-[0]  [/Users/wang/chefdemo]
➤  [2013/11/13 20:45] % cat site-cookbooks/hello/recipes/default.rb
# 「Hello Chef」と表示する
log "Hello Chef"

# 列挙されたパッケージを全部インストールする
%w{curl wget}.each do |pkg|
  package pkg do
    action :install
  end
end
```

Hello chef

```
[wang@wang-mbp]-[0]  [/Users/wang/chefdemo]
> [2013/11/13 20:50] % cat nodes/debian7vm1.json
{"run_list":["recipe[hello]"]}
[wang@wang-mbp]-[0]  [/Users/wang/chefdemo]
> [2013/11/13 20:53] % knife solo cook wang@debian7vm1
Running Chef on debian7vm1...
Checking Chef version...
Uploading the kitchen...
Generating solo config...
Running Chef...
Starting Chef Client, version 11.6.0
Compiling Cookbooks...
Converging 3 resources
Recipe: hello::default
  * log[Hello Chef] action write

  * package[curl] action install (up to date)
  * package[wget] action install (up to date)
Chef Client finished, 1 resources updated
```

cookbook helloを適用

想定通りに実行しました！
ログはファイルに出力できます。
何か不具合が起きる場合、-vで詳細情報を出力します。

Chef

Demo

SSH鍵認証が設定済み

localhost
Knife Solo
Berkshelf

172.16.0.148
ユーザtest追加
ディレクトリ作成
nginxインストール
設定ファイルのテンプレートを適用

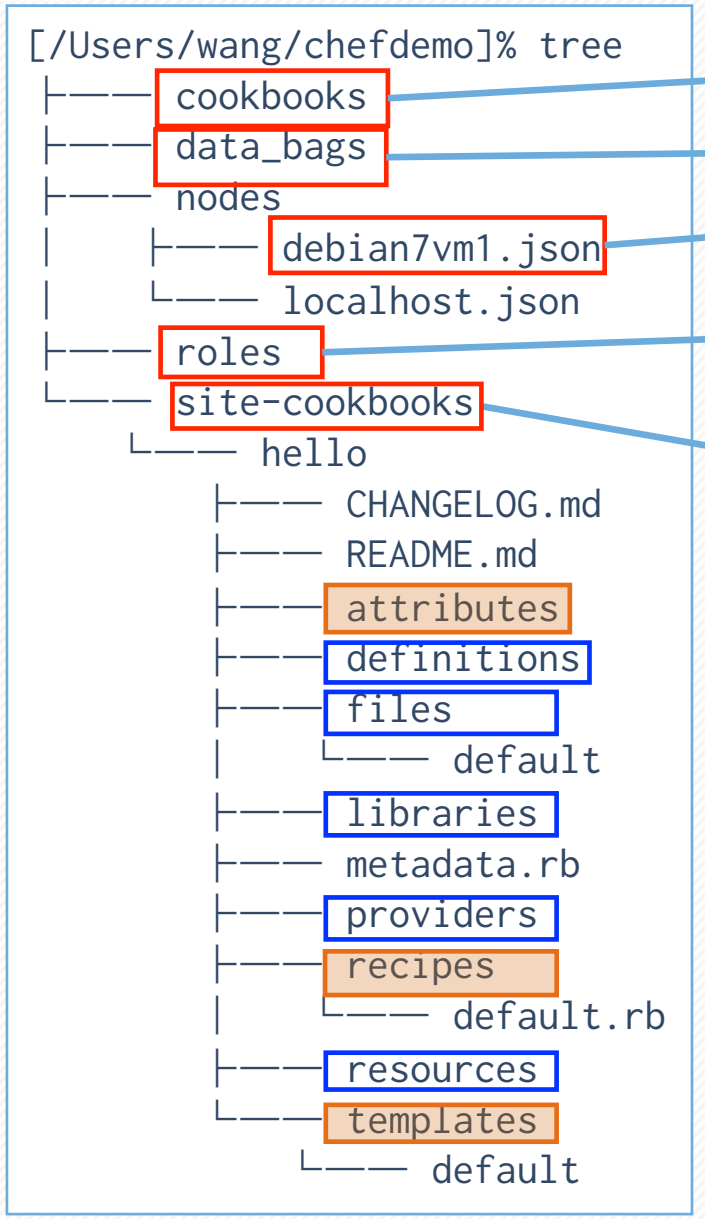
ちょっと迷いました。一体Chef soloは何のものですか？

Chef

Javaプログラマーから見てみましょう

Chef solo	Ant
指定したマシンを構築	指定したJavaアプリをビルド
どう構築しますか？ nodes/debian7vm1.json	どうビルドしますか？　build.xml
何を使って実行しますか。 knife solo	何を使って実行しますか。　antコマンド
構築内容は？ site-cookbooks/hello/recipes/default.rb	ビルド内容は？ build.xmlに<project>、<task>定義
設定情報はどこ？ site-cooks/{attributes, templates}/** roles/** data_bags/**	設定情報は？ build.properties

アーキテクトと構成要素



- ライブラリ的なcookbook(Berkshelfでopscode communityからインストールできる)
- JSON形式でパスワード情報などの変数を置く
- ノードごとの設定（変数、実行recipe）
- ノードの役割を定義（ノードの用途によってグルーピングする。例えは、開発ノード、単体テストノード、結合テストノード、本番ノード）
- 自分で作ったcookbook

- 変数のデフォールト値
- 自分で作ったリソース（ユーザ関数、Antのtackと類似）
- Cookbook内で利用されるファイル（JDKのインストールファイル等）
- Chefの機能を拡張するためのRubyコード
- リソースに対する処理を実行するための設定ファイル
- Recipe本体（Ant build.xmlの<project>要素）
- Recipe内で使われる設定対象を定義するための設定ファイル
- Cookbook内で利用されるテンプレートファイル（erb、http.conf等）

用語

```
[/Users/wang/chefdemo]% tree
├── cookbooks
├── data_bags
├── nodes
│   ├── debian7vm1.json
│   └── localhost.json
├── roles
└── site-cookbooks
    └── hello
        ├── CHANGELOG.md
        ├── README.md
        ├── attributes
        ├── definitions
        ├── files
        │   └── default
        ├── libraries
        ├── metadata.rb
        ├── providers
        ├── recipes
        │   └── default.rb
        ├── resources
        └── templates
            └── default
```

Repository(キチン、料理を作る場所)

- Chef動作に必要なファイル群
- 複数のCookbook、RoleとNodeの設定を含む

Cookbook(料理を作る本)

- Recipeをグループにまとめるもの
- 空でもいい

Recipe(レシピ)

- 複数のResourceで構成される構築手順

Resource

- 設定の最小単位（build.xmlにtaskと類似）
- ファイルやパッケージ単位の設定

Attributes

- 変数のデフォルト値を設定
- RecipeとTemplateに変数を利用できる

Templates

- 設定ファイルのerbテンプレート(JSPと類似)

Knife(料理を作る)

Chef

用語

- Node
  - Chefで管理されるサーバ
- Role
  - ノードの役割を定義
  - 使うRecipeとか変数とかをここで設定する（Abstractクラスみたい）
- Chef solo
  - サーバ無しでChefを利用するコマンド
  - Chef soloを動かしたノード自身を操作対象となる
- Knife solo
  - 対象ノードにレポジトリを転送してChef soloを動かす

Resource：設定の最小単位

テンプレートファイル

```
template "#{node['myjava']['home']}/.javarc" do
    source 'config/javarc.erb'
    owner  node['myjava']['user']
    group  node['myjava']['group']
    mode   '0644'

    variables(
        :author => 'Haidong Wang'
    )

    not_if { File.exists?("#{node['myjava']['rcfile_path']}")}
end
```

パッケージインストール

```
%w(curl unzip tree wget nkf ctags).each do |pkg|
    package pkg do
        action :install
    end
end
```

Bashスクリプト実行

```
bash "add javarc to bashrc" do
    user "wang"
    cwd node["myjava"]["home"]
    code <<-EOH
        if ! grep '.javarc' #{node['myjava']['home']}/.bashrc > /dev/null 2>&1; then
            echo "[[ -e #{node['myjava']['rcfile_path']} ]] && source #{node['myjava']['rcfile_path']}" >> #{node['myjava']['home']}/.bashrc
        fi

        source #{node['myjava']['rcfile_path']}
    EOH

    only_if do File.exists?("#{node['myjava']['rcfile_path']}") end
end
```

Chef

Recipe

- Recipeに書いたResourceが上から順に実行される。
- 実際にRubyのコードで手順DSLを記述する。
- 共通変数（グローバル）などをAttributesに定義する。
- 標準なResourceは以下のとおり
  - http://docs.opscode.com/resource.html

- 自動化を実現するため、Recipeは冪等性(べきとうせい)を持つ必要
  - 同じ操作を複数回で行っても結果が同じである性質のこと
  - 自分のBashスクリプトを工夫する必要

Attributes

site-cookbooks/mysql/attributes/default.rb

```
default["mysql"]["package_name"] = "mysql5-server"
```

site-cookbooks/mysql/recipes/default.rb

```
package node["mysql"]["package_name"] do
    action :install
end
```

```
package "mysql5-server" do
    action :install
end
```

Attributesに定義された変数はNodes、Rolesにオーバーライドされる可能

ここで優先順位を確認しましょう。
http://docs.opscode.com/chef_overview_attributes.html

Templates

Chefから配布された設定ファイルをノードごとに動的に生成する

- eRubyで書く。<%= %>と囲うことで、設定ファイルにRubyのコードを埋め込める

```
template "#{node['myjava']['home']}/hadoop/etc/hadoop/#{hadoop_config_file}" do
    source "hadoop/#{hadoop_config_file}.erb"
    owner  node['myjava']['user']
    group  node['myjava']['group']
    mode   '0644'

    variables(
        :author           =>    "Haidong Wang",
        :this_ip          =>    "#{node['ipaddress']}",
        :hadoop_tmp       =>    "#{node['myjava']['home']}/hadoop/tmp",
        :hadoop_name_dir  =>    "#{node['myjava']['home']}/hadoop/dfs/name",
        :hadoop_data_dir  =>    "#{node['myjava']['home']}/hadoop/dfs/data",
        :hdfs_port        =>    "54310",
        :hdfs_rep_num     =>    "1"
    )

    not_if { File.exists?("#{node['myjava']['home']}/hadoop/etc/hadoop/#{hadoop_config_file}")}
end
```

```
<configuration>

    <property>
        <name>hadoop.tmp.dir</name>
        <value><%= @hadoop_tmp %></value>
        <description>A base directory for temporary directories.</description>
    </property>

    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://<%= @this_ip %>:<%= @hdfs_port %></value>
    </property>


</configuration>
```

Cookbook

knife cookbook createでcookbookを作成
- Site-cookbooks以下に置くのが慣習

```
$ knife cookbook create chefdemo -o site-cookbooks
```

Cookbookは二種類がある
1. インタネットで他人が公開しているもの（Berkshelfで管理しやすい）
2. スクラッチで作ったもの

まずRecipeを書く

```
$ vim site-cookbooks/chefdemo/recipes/default.rb
```

```
log 'message' do
    message "hello chef for tcserver setup"
    level :info
end

%w(curl unzip tree wget nkf ctags).each do |pkg|
    package pkg do
        action :install
    end
end
```

Cookbook

すべてのcookbookを一から書く必要はありません

- Chef開発元がGithubで公開してるcookbook集が存在
  - https://github.com/opscode-cookbooks
  - 公開されているcookbookをダウンロードするには、Opscode Communityにユーザ登録し、秘密鍵をダウンロードしておく必要があります。秘密鍵をDownloadしたら、~/.chef/username.pemにパーミッション600で保存しておきましょう
- Knifeコマンドで取ってこれる
  - cookbooksに置くのが慣習

```
$ knife cookbook site install apt -o cookbooks
$ knife cookbook site install users -o cookbooks
```

```
[wang@wang-mbp]-[0]  [/Users/wang/chefdemo]
> [2013/11/14 01:59] % cat .chef/knife.rb
cookbook_path ["cookbooks", "site-cookbooks"]
node_path     "nodes"
role_path     "roles"
data_bag_path "data_bags"
#encrypted_data_bag_secret "data_bag_key"
client_key    "/Users/wang/.chef/wanghaidong.pem"

knife[:berkshelf_path] = "cookbooks"
```

もちろんGitを使えます

```
$ git clone https://github.com/opscode-cookbooks/build-essential.git
$ git clone https://github.com/opscode-cookbooks/openssl.git
```

Cookbook

Berkshelfを使うと、サードパーティのcookbookをBundler風に扱う事ができて、任意のgithubからダウンロード出来たり、バージョン指定したりすることが出来るようになります。

- 設定フィアルBerksfileを編集
  - 利用するcookbookを指定
- berksコマンドで取ってこれる
  - cookbooksに置くのが慣習

```
[wang@wang-mbp]-[0]  [/Users/wang/chefdemo]
> [2013/11/14 02:02] % cat Berksfile
site:opscode

cookbook 'apt'
cookbook 'users'

[wang@wang-mbp]-[0]  [/Users/wang/chefdemo]
> [2013/11/14 02:02] % berks install --path cookbooks/
```

```
[wang@wang-mbp]-[0]  [/Users/wang/chefdemo]
> [2013/11/14 02:03] % ls -l cookbooks/
total 0
drwxr-xr-x 13 wang 442 11 14 02:03 apt
drwxr-xr-x 10 wang 340 11 14 02:03 users
```

Nodes

ノードごとの設定はjsonファイルに定義されます。

```
[wang@wang-mbp]-[0]  [/Users/wang/chefdemo]
> [2013/11/14 02:14] % cat nodes/debian7vm1.json
{"run_list":["recipe[hello]"]}
```

site-cookbooks/hello/recipes/default.rbを実行

debian7vm1.jsonに変数も定義できます。
default.rb以外のレシピを::で引用します。
例えば：
site-cookbooks/hello/recipes/hadoop.rbは
{“run_list” : [“recipe[hello::hadoop]”]}で呼びます。

RolesとEnviroments

開発環境と本番環境の設定を分けたい場合、Enviromentを利用します。

environments/development.json

```
name "it_env"
description "IT test environment"
run_list {
  "recipe[log_conf]",
  "recipe[it_dataload]",
}
```

nodes/apserver.json

```
{
  "environment": "development",
  "run_list":[
    "recipe[hello]"
  ]
}
```

サーバの役割による構築したい場合、Roleを利用します。

roles/it_env.json

```
name  “dbserver"
description "IT database server"
run_list {
  "recipe[mysql_conf]",
  "recipe[it_dataload]",
}
```

nodes/it_server.json

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

Chef

data_bags

- 暗号化した変数を定義します（データベースのパスワード等）。
- Chef serverとChef clientのやりとりの設定を定義します。

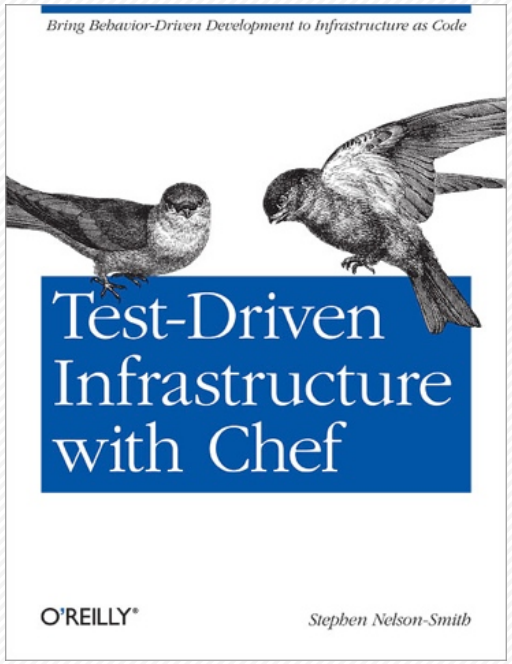
参考情報

伊藤直也さんの電子書籍がベスト

- 入門Chef Solo – Infrastructure as code
- http://www.amazon.co.jp/dp/B00BSPH158/

http://docs.opscode.com/

https://github.com/opscode-cookbooks



事例

Facebook improves business efficiency and flexibility with Chef.

> "Chef provided an automation solution flexible enough to bend to our scale dynamics without requiring us to change our workflow."
>
> — Phil Dibowitz, Production Engineer, Facebook

FacebookはChefの最大事例です。

最新のChefはFacebookとChef Softwareが一緒に実装されます。
Facebookのデータセンター構成が複雑すぎですので、独自の
Chefバージョンを開発しました。

Chef

事例

amazon web services™

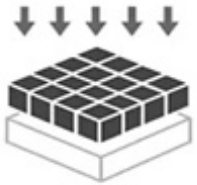
AWS OpsWorks

An integrated DevOps application management solution

A **stack** represents the compute infrastructure and applications that you want to manage together.

A **layer** defines how to set up and configure a set of instances and related resources.

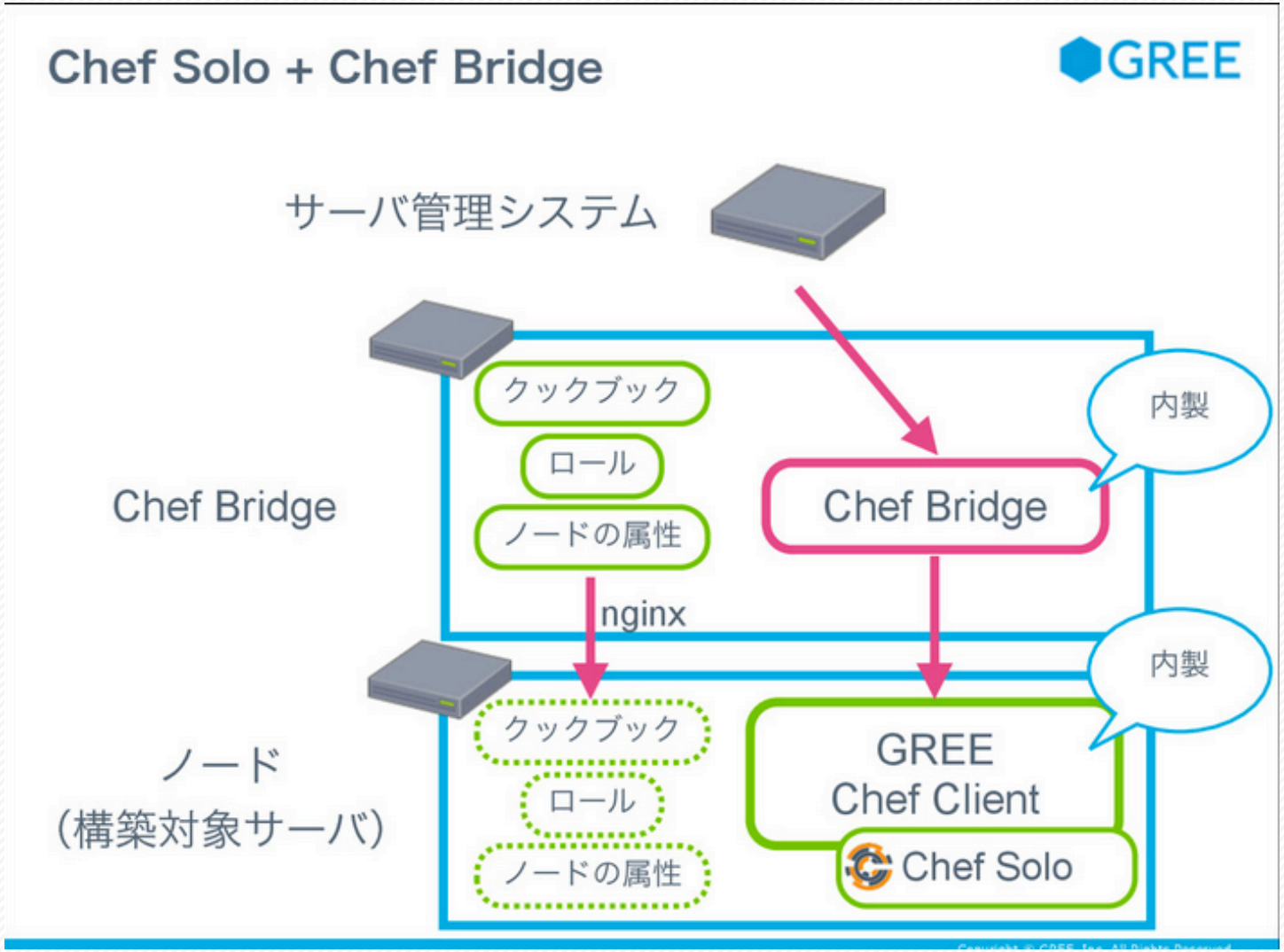
Decide how to scale: manually, with **24/7** instances, or automatically, with **load-based** or **time-based** instances.



Then deploy your app to specific instances and customize the deployment with Chef recipes.

AWS OpsWorks は、ロードバランサーからデータベースまでのアプリケーション全体を DevOps ユーザーが簡単にモデル化および管理できるようにするアプリケーション管理サービスです。Ruby、Node.JS、PHP、Java などの一般的なテクノロジー用のテンプレートから始めることも、Chef のレシピを使用して独自に構築し、ソフトウェアパッケージをインストールしてスクリプト化できるタスクを実行することもできます。

事例



デブサミ2014
グリーにおけるChef導入事例
～既存の資産を活かし新しい技術を導入する～
（荒井良太〔グリー〕）

Chef

まとめ

- Chefのレシピを書くのは簡単ではありません。
  - 開発が活発化にしていますので、日々進化しています。
  - Roles、Attributes、NodesとRecipesの関係をわかりにくいです。
  - Attributesのオーバーライドは改善余地が大きいです。

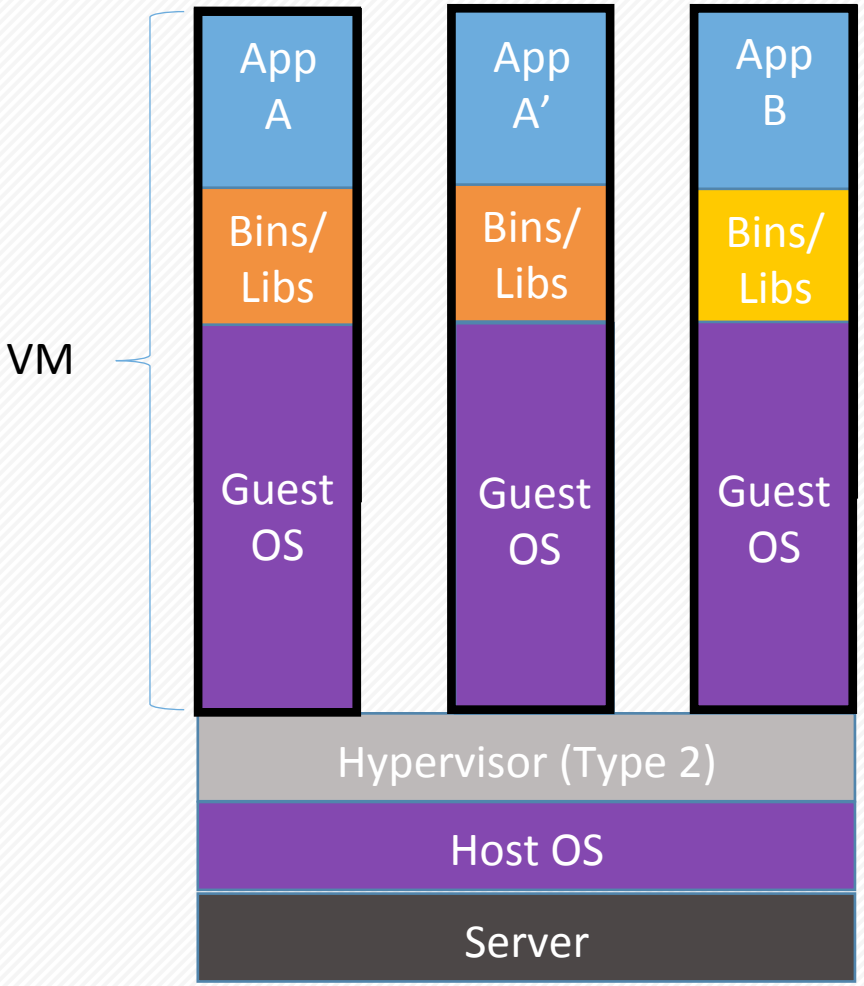
- 使うメリット
  - 異なるOSが混在している環境でも効果を発揮
    - 大規模な本番運用で有利
  - 開発環境で素早く統一できます
    - Vagrantを併用すれば、もっと効果を出るはず
  - 自動化、自動化、自動化・・・
  - クラウドの普及とともに、アプリエンジニアもインフラの作業を担当になります

これからのインフラ

- インフラCI（継続的インテグレーション）
  - アプリケーション開発と同じように、テスト駆動したら次は CI です。
  - serverspecとchefspecが出てきました。
  - これまでの構築結果の確認作業を従来通り、目視の手動チェックから解放できます。

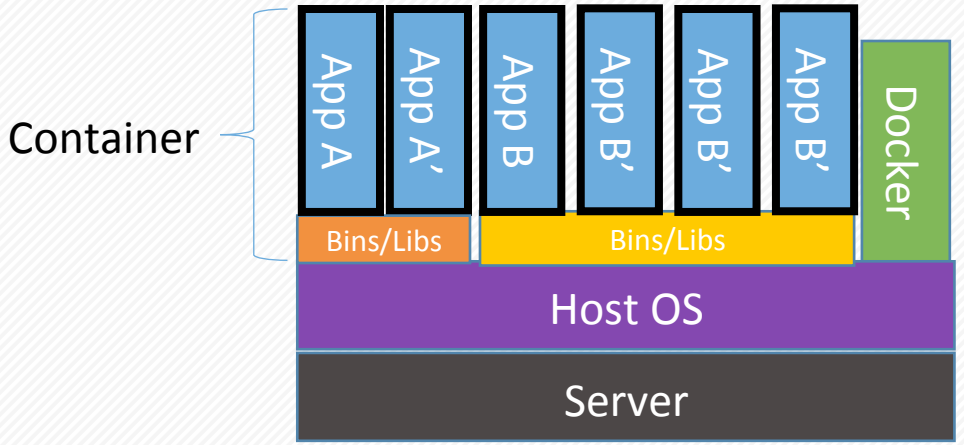
- Container Base Deployment
  - Immutable Infrastructure は EC2 のような、ハイパーバイザー型仮想マシンでの話が主流だと思われるが、今後はコンテナ型仮想マシンで同じようなことをやる、という話が増えてくると思います。
  - DockerとVagrantによって、ローカルでリモートVMを抽象化して同じように扱えるような仕組みを提供できます。

Containers vs. VMs



Containers are isolated, but share OS and, where appropriate, bins/libraries

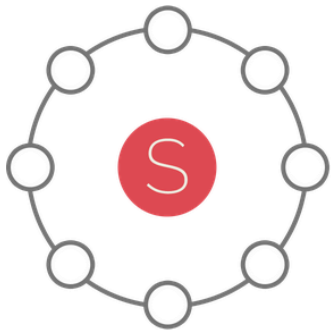
…result is significantly faster deployment, much less overhead, easier migration, faster restart



Dockerオフィシャルサイトの紹介資料から引用

Serf（自律的なインフラ運用）

サーバ毎に微妙に異なる設定（ホスト名やIPアドレスにひもづいた設定）や、サーバの増減によって動的に変わるような設定（ロードバランサや Nagios の監視対象など）なんかも、Chef でうまく対応できなさそう



GOSSIP-BASED MEMBERSHIP

Serf relies on an efficient and lightweight gossip protocol to communicate with nodes. The Serf agents periodically exchange messages with each other in much the same way that a zombie apocalypse would occur: it starts with one zombie but soon infects everyone. In practice, the gossip is very fast and extremely efficient.

Chef

予想できるトレンド

- Stateless Server
  - アプリケーションサーバーなどの状態をもたないホストは使い捨て可能なImmutableなものになり、開発プロセスにも組込まれていきます。
  - アプリ開発エンジニアとインフラエンジニアの作業を融合します。
- Stateful Server
  - データベースサーバーなど状態を持つホストは、まだまだ従来からの連続的な進化が続きます。
- クラウドでリソースを柔軟に使えるようになったが、リソースを効率的に使うための取り組みが注目されます。

Chef

THANKS!

For more information, visit
http://sjitech.github.io/

Follow me

Twitter : @allegewhd