ChefScript: 運用支援のための時系列記述が可能な構成管理手法

青山 真也† 廣津 登志夫†

概要 サービス基盤の大規模化と複雑化に伴い，サービス基盤の状態をコードで記述しておき自動的に構築および運用する構成管理ツールの利用が広まっている．構成管理ツール Chef では冪等性により記述されている通りにシステムが構成されるため，構成を順次変更していくような時間軸に沿った記述をすることが出来ず，運用業務の実現が困難である．そこで本研究では，運用業務を分析・モデル化し，運用業務のワークフローを実現する宣言的な記述性を導入したシステムを実装することで，Infrastructure as Code の概念及び Chef を拡張した．また，提案するシステムを ChefScript と名付けた．

1. はじめに

現在，データセンタなどの大規模なサービス基盤ではハイパーバイザ上に展開した複数の仮想マシンやネットワーク機器などで複雑に構成されている．こうした環境では，サービス基盤を構成するノードの手動構築が困難となっており，サービス基盤の状態をコードで記述しておき，自動的に構築する枠組みとして Infrastructure as Code の概念が広まってきている[1]．現状では，Infrastructure as Code の概念の元となった構成管理ツールでシステム構成を記述して運用している．しかし，Chef[2]・Puppet[3]・Ansible に代表される構成管理ツールでは，サーバの状態をコード化することは可能であるが，サービス基盤管理のすべてをコード化しきれておらず，完全には Infrastructure as Code を実現できていないという問題点が存在する．本研究では，特に運用業務に関するコード化管理に着目する．

2. 構成管理ツールの概要と問題点

Chef などに代表される構成管理ツールでは，サーバの状態をコードとして記述する．そのコードをツールで適用することにより，自動的にサーバの状態がコードで記述した状態に構成される．また，構成管理ツールには冪等性が保障されていることが多く，コードの適用が一度行われても複数回行われても実行結果となるサーバの状態が変わらないことが主な特徴となっている．コードの記述には，多くの場合は JSON 形式やプログラミング言語のブロック形式を用いて行われている．

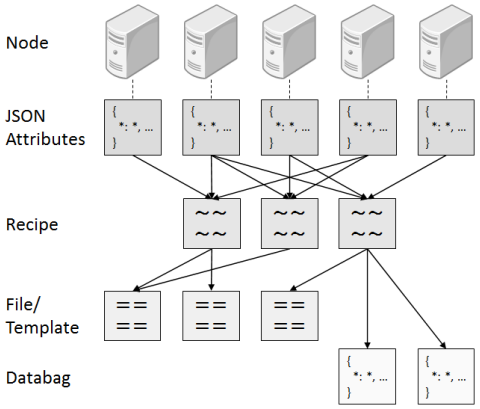

図 1 Chef を構成するコード

構成管理ツールには，運用業務へ対応が得意ではないという問題点が存在する．運用業務では，ローリングアップデートやサーバの切り替えなど，複数のサーバを対象とした順次操作や時系列操作が必要である．しかし，現状の構成管理ツールでは，単体のサーバまたは複数のサーバの状態をコードで記述するため，一連の操作を記述することができず，こうした操作を上手く扱うことができない．

3. Chef のアーキテクチャ

本研究では，構成管理ツールの中でも Chef を対象とする．Chef では図 1 のように主に 4 種類のコードからサーバの状態を決定している．(1) JSON Attributes は，各 Node 用に定義された属性を集めた JSON ファイル，(2) Recipe は，図 2 のように多種多様な Chef 用の Ruby 内部 DSL で書かれたサ

† 法政大学情報科学部
Faculty of Computer and Information Sciences,
HOSEI University

```
package “memcached” do
   action :install
   version “1.4.10”
end

template “/etc/sysconfig/memcached” do
   source “/etc/sysconfig/memcached.erb”
end
```

図 2　構成管理ツール Chef の Recipe 例

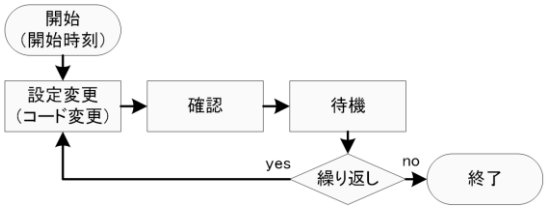

図 3　運用業務のフローチャートモデル

ーバ状態の詳細定義，(3) File/Template は，ファイルや Embedded Ruby テンプレート，(4) Databag は，暗号化やドメイン階層をサポートした JSON 形式の簡易データベースである．Chef のコード適用は JSON Attributes に定義された適用する Recipe のリストを元に行われる．また，各 Recipe は File/Template や Databag・JSON Attribute の値などを用いて各 Node を構築する．

4.　運用業務のモデル化

運用業務には，ローリングアップデートやサーバの順次切り替えといった様々な業務が存在する．そこで，様々な運用業務の分析を行い，最小単位の操作要素に分解を行った結果，運用業務は図 3 のようなワークフローで行われていると考えられる．運用業務において，特に重要になるのは待機処理である．待機処理とは，一定時間待機するだけではなく，memcached のローリングアップデート時に対象となるサーバのキャッシュが蓄積されたかを判断することなど，高度な判断を含むものである．

5.　実装

運用業務をモデル化した結果から考えられる，必要な記述性は下記のとおりである．ただし，確認処理については Chef 側で行われるため必要としない．

(i)　開始時刻
(ii)　依存関係（変更処理等の順序）
(iii)　コードの変更内容（ファイルの複製を含む）
(iv)　待機処理の内容

依存関係を記述し，コードの可読性を高めるために複数の Task が集合した Task Group を 1 つの運用業務として扱う．図 3 のフローチャートに当てはめて考えた場合，繰り返しの 1 回分が 1 つの Task となり，1 つの運用業務が 1 つの Task Group となる．また，Chef に類似した宣言的記述性で実現するために，Ruby 内部 DSL として実装する．

```
create "Create file from wget" do
   file "site-cookbooks/ex/files/default/exam.jp.crt"
   source "http://key.example.jp/keys/exam.jp.crt"
   remove false
end

interval "CHECK MEMCACHE" do
   confirm do
      system "check_memcache.sh exam.jp"
   end
   every 20
end

task "TASK_A" do
   file "nodes/exam.jp.json"
   if content["apache"]["port"] == "80"
      content["apache"]["port"] = "8080"
   end
   interval 3
end

taskgroup "TASKGROUP_1" do
   starts "2014-01-01 09:00:00"
   exec "TASK_A"
   interval "CHECK MEMCACHE"
   ...
end
```

図 4　提案手法の ChefScript のコード例

以上の実装要件を元に提案手法を ChefScript として実装した．図 4 の ChefScript のコード例にある各 DSL 記述の処理は以下のとおりである．

(a)　**create**: ファイルのコピー及び移動の内容
(b)　**interval**: 待機処理の内容及び確認間隔
(c)　**task**: コードの変更内容（Recipe，JSON）
(d)　**taskgroup**: 依存関係及び開始時間制御

6.　まとめ

現段階ではファイル単位での変更を行うコード記述性に留まっているため，複数台に関連するコードを書き換える際の同期・待機制御などを Syntax Sugar として用意する必要がある．

参　考　文　献

[1]　宮下 剛輔，栗林 健太郎，松本 亮介，“serverspec: 宣言的記述でサーバの状態をテスト可能な汎用性の高いテストフレームワーク，” 電子情報通信学会技術研究報告，pp.81-86, Feb 2014.

[2]　Chef Software, Inc.，“All about Chef ... - Chef Docs,” http://docs.getchef.com/, (2014/10/30 参照)

[3]　L. Kanies，“Puppet: Next-Generation Configuration Management，” USENIX ;login:, Vol.31, No.1, pp19-25, Feb 2006.