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

宮下 剛輔[†1,†2,a)]　栗林 健太郎[†1]　松本 亮介[†3]

概要：システムの大規模・複雑化に伴い，サーバの構築・運用を効率化するために，サーバの状態をコードで記述する手法が数多く提供されている．それらの手法を効率良く扱うプロセスとして，テスト駆動開発の手法をサーバ構築に応用した Test-Driven Infrastructure が提案されている．このプロセスを支援するテストフレームワークもいくつか登場しているが，あるものは特定の構成管理ツールに依存，またあるものは OS 毎の違いを自ら吸収しなければならないなど，汎用性に難がある．そこで，本論文では，特定の構成管理ツールや OS に依存することなく，サーバの状態を汎用的かつ可読性の高いコードでテスト可能なテストフレームワークを提案する．提案手法では，汎用性を高めるために，これまでの OS や構成管理ツール固有の振る舞いを整理して一般化し，運用業務で発生するコマンド群，特に確認作業に関するコマンド群を体系化・抽象化した汎用コマンド実行フレームワークを定義する．続いて，テストコード記述の抽象度を高め可読性を上げるために宣言的かつ自然言語に近い記法で汎用コマンド実行フレームワークを操作できる制御テストフレームワークを定義する．これにより，管理者が OS や構成管理ツールの違いを気にすることなくサーバの状態を容易にテストできるようになり，サーバの運用・管理コストを低減できる．また，フレームワークを用途別に分離して定義することにより，制御テストフレームワークを独自の記述に変更する事も容易である．提案するテストフレームワークを serverspec と名付けた．

serverspec: A Versatile Test Framework for Testing States of Servers by Delarative Description

Gosuke Miyashita[†1,†2,a)]　Kentaro Kuribayashi[†1]　Ryosuke Matsumoto[†3]

Abstract: As the increase of large and complex systems, many ways to describe states of servers as code are supplied.As the effective process to handle these ways, Test-Driven Infrastructure is proposed.It is the application of Test-Driven Development to the server configuration.Several test frameworks that support this process are appearing, but they have difficulty in vesatility because some frameworks depend on specific configuration management tools and others need the consideration of differences between OSes to write test code.In this paper, we propose a test framework that doesn't depend on specific configuration management tools or OSes for testing states of servers by versatile and readable code.In our proposal, we arrange and generalize the behavior of OSes and configuration management tools and define the versatile command execution framework for increasing the versatility. Next, we define the test framework that control the command execution framework by declarative and natural language like description to increase the level of abstraction and readability.By this test framework, system administrators can test states of servers easily without considering the diffrences between OSes or configuration management tools, and can decrease the costs of server operations.Also by defining the frameworks separated into purposes, we can replace the control test framework with another one that have original notation easily.We named the proposed test framework serverspec.

[†1] 現在，株式会社 paperboy&co.
Presently with paperboy&co., Inc.
[†2] 現在，帝京大学 理工学部 情報科学科 通信教育課程
Presently with Department of Information Science Correspondence Course, Faculty of SCIENCE and ENGINEERING, Teikyo University
[†3] 現在，京都大学 情報科学研究科，京都市
Presently with Graduates School of Infromatics, Kyoto Uni-

1. はじめに

科学やビジネス領域における問題の複雑化への要求に応えるため，システムが大規模化・複雑化する [1] のに伴い，UNIX シェルにより書かれたプログラムに代わり，サーバの設定を宣言的なコードで扱う構成管理手法が広く提供されている．その実装として 1993 年に CFEngine[2] が登場し，その後様々な構成管理ツールが生み出されているが [3]，2005 年の Puppet の登場 [4] と 2006 年の Amazon EC2 の登場 [5] をきっかけに “Infrastructure as Code” という概念が台頭した．この概念における “Infrastructure” はアプリケーションを載せるためのインフラを意味し，OS やミドルウェアといったソフトウェアレイヤーを含む．そして，インフラをコードで扱うことから，アジャイルソフトウェア開発 [6] と同様の手法がサーバ構築・運用にも適用できるのでは，という発想が生まれ “Agile infrastructure and operations”[7] という流れが生じている．

“Agile infrastructures and operations” を実践するためのプロセスとして，テスト駆動開発 [8] の手法をサーバ構築・運用に応用した “Test-Driven Infrastructure”[9] が提案されており，このプロセスを支援するテストフレームワークがいくつも登場している [10][11][12][13][14][15]．これらのうち，ChefSpec[10]，rspec-puppet[11] は，構成管理ツール固有の言語で書かれたコードの内容をテストするのみで，実際にコードをサーバに適用し，設定が正しく行われたかどうかまではテストしない．そのため，単体テストしては利用できるが結合テスト用途には利用できない．Cucumber-chef[12]，minitest-chef-handler[13] は Chef という特定の構成管理ツールに依存している．そのため，Chef 以外の構成管理ツールでは利用することができない．Test Kitchen[14] や rspec-system[15] は，テスト用 Virtual Machine（以降 VM とする）の作成，テスト用 VM への構成管理ツールの適用，テストの実行をトータルで行う統合テストスイートであるが，標準で持つテスト実行機能は汎用性に乏しく，特定の構成管理ツールに依存しており，OS やディストリビューション毎の違いを意識したテストコードを書く必要がある．

我々は，テストフレームワークの汎用性を高めるために，構成管理ツール特有の振る舞い，例えば，パッケージのインストールやシステムユーザの作成などを抽出，一般化し，それらをテストするためのコマンドを OS やディストリビューション毎に分離，その上で OS や実行形式の違いを吸収するレイヤーを設けることにより，汎用コマンド実行フレームワークを定義した．続いて，テストコードの記述の抽象度を高め可読性を上げるために，宣言的かつ自然言語に近い記法で汎用コマンド実行フレームワークを操作できる制御テストフレームワークを定義した．これにより，テスト対象の環境の違いを気にすることなく，直観的にテストが書ける．また，フレームワークを用途別に分離して定義することにより，制御テストフレームワークを独自の記法に変更することも容易である．例えば，本論文で提案するテストフレームワークでは，テスト記法として RSpec[16] を採用しているが，他のテストフレームワークに差し替えたり，あるいはまったく独自の記法に差し替えたりすることも可能である．このテストフレームワークを serverspec[17] と名付けた．serverspec を採用している企業も既に存在する [18][19]．

本論文の構成について述べる．2 章ではサーバの構成管理とテスト手法について更に詳しく述べる．3 章では提案するサーバテスト手法と実装について述べ，4 章では提案手法の評価について論じ，5 章で結びとする．

2. サーバの構成管理とテスト手法

本論文で提案するテストフレームワークは，構成管理ツールによりサーバ構成を記述したコードのテストをいかに効率よく行うか，という観点から出発している．そこで代表的な構成管理ツールを例に，ツールと言語の特徴，テスト手法ならびに従来手法の問題点について言及する．

2.1 CFEngine から Chef へ

第 1 章で触れた CFEngine は 1993 年に登場した．2005 年には CFEngine に影響を受け，より抽象度と拡張性を高める形で発展させた Puppet[4] が登場，2009 年には Puppet から影響を受け，コードのプログラマブルな側面を強めた Chef[20] が登場している．

2.2 Chef から Test-Driven Infrastructure へ

CFEngine，Puppet は独自のドメイン固有言語でサーバの構成を記述するが，Chef は実装言語と同じ言語を利用したドメイン固有言語，すなわち Ruby[21] という汎用プログラミング言語で記述する．その特性ゆえ，Chef はシステム管理者よりも開発者に強く訴求し，Amazon EC2[5] の様な開発者自身がサーバ構築・運用を行える環境の普及とともに広まりを見せている．しかし，汎用プログラミング言語であるが故に，システムが複雑になるにともない，それを記述するコードも複雑になる．そこでサーバ構成を記述したコードに対し，アジャイル開発におけるテスト駆動開発のプロセスを適用する事で，効率よくコードを記述することができる，といった考えが生まれる．Test-Driven Infrastucture という概念は Chef コミュニティ周辺で発生したものであるが，それは偶然ではなく，このような Chef が持つ言語特性ゆえである．

versity
a) gosukenator@gmail.com

2.3 Test-Driven Infrastructure におけるテスト手法の分類

Test-Driven Infrastructure におけるテストの種類は，テスト駆動開発における以下の 3 つに分類できる．

(1) 単体テスト
(2) 結合テスト
(3) 受け入れテスト

単体テストは構成管理ツールにおける “モジュール” に対するテストで，実際にコードをサーバに適用する前の段階で行うテストである．結合テストはコードをサーバに適用した後に行うテストで，コードが期待通りにサーバの設定を行ったかどうかをテストする．受け入れテストもコードをサーバに適用した後に行うテストだが，結合テストがサーバ内部の状態をテストするホワイトボックステストであるのに対し，受け入れテストはサーバの外から見た振る舞いをテストするブラックボックステストであるという違いがある．

2.4 従来テスト手法の問題点

(1) に分類される単体テストツールとしては Chef には ChefSpec，Puppet には rspec-puppet というツールが存在する．その名が示すとおり，それぞれ Chef と Puppet 専用のツールであり，単体テストツールとしては十分である．しかし，実際にコードをサーバに適用した結果をテストすることはできないため，単体テストツールだけではサーバのテスト手法としては不十分である．

(2) に分類される結合テストツールとしては，minitest-chef-handler，Test Kitchen，rspec-system が存在する．この内 minitest-chef-handler と Test Kitchen は Chef に依存したツールであり，他の構成管理ツールとともに利用することができない．Test Kitchen や rspec-system は，テスト用 VM の作成，テスト用 VM への構成管理ツールの適用，テストの実行をトータルで行う統合テストスイートであるが，ツール標準のテスト機構は汎用性に乏しく，特定の構成管理ツールに依存しており，OS やディストリビューション毎の違いを意識したテストコードを書く必要がある．

(3) に分類される受け入れテストツールとしては cucumber-chef，leibniz が存在するが，これらは Chef に依存したツールであり，他の構成管理ツールとともに利用することができない．

表 1 に各テストツールの比較を示す．これにより，従来テスト手法では，どのテスト種別においても，構成管理ツールからの独立性と OS・ディストリビューションの汎用性双方を満たすものが存在しないことがわかる．

表 1 テストツール比較表

ツール名	テスト種別	ツール独立性	OS 汎用性
ChefSpec	単体	×	○
rspec-puppet	単体	×	○
minitest-chef-handler	結合	×	○
Test Kitchen	結合	×	×
rspec-system	結合	○	×
Cucumber-Chef	受入	×	○
leibniz	受入	×	○

3. 提案するサーバテスト手法

3.1 テスト種別の選択

2 章にて，従来のテスト手法では構成管理ツール独立性と OS・ディストリビューション汎用性の双方を満たすものが，どのテスト種別においても存在しない，ということを述べた．そこでまず，本論文で提案する手法を，どのテスト種別に適用するのか選択を行う．

種別（1）の単体テストは既に述べたように，サーバのテスト手法としては不十分である．また，性質上構成管理ツール固有の言語と密に関連しているため，構成管理ツール独立性を満たすことは不可能である．よって単体テストは提案手法の対象から除外する．種別（3）の受け入れテストは，サーバの外からの振る舞いをテストするもので，内部状態をテストするものではない．したがって，サーバ構成を記述したコードによってもたらされたサーバの内部状態を網羅的にテストする用途には向かない．よって，内部状態を網羅的にテスト可能な（2）の結合テストを提案手法の適用対象とする．

3.2 要件の考察

次に，結合テストにおいて構成管理ツール独立性と OS・ディストリビューション汎用性の双方を満たすための要件について考察する．

特定の構成管理ツールからの独立性を満たせない理由は 2 つある．ひとつは，テストスイートはテストに必要なプロセスをすべて実行するため，テスト用 VM を構築する機能も併せ持つが，この機能が特定の構成管理ツールに依存しているためである．もうひとつは，テストコードに OS・ディストリビューション汎用性を持たせるために，テストの実装が構成管理ツールが元々持っている機能に依存しているためである．

OS・ディストリビューション汎用性が満たせないのは，テストがシェルコマンドを直接記述する実装になっており，OS・ディストリビューションの違いをテストコードを書く者自らが意識しないといけないからである．この考察から，提案するテスト手法に必要な要件は以下の通りとなる．

(1) テストスイートではなくテストのみに特化する
(2) テストの実装を特定の構成管理ツールに依存しない
(3) OS・ディストリビューションの違いをテストコードを書く者に意識させない

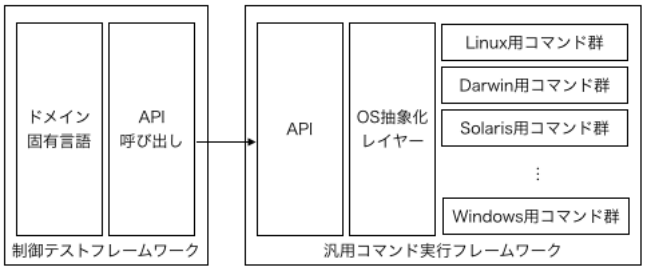

図 1 汎用コマンド実行フレームワークと制御テストフレームワークの仕組みと関係

3.3 要件を満たすための手法の提案

考察した要件を満たすために，運用業務で発生するコマンド群，特に確認作業に必要なコマンド群の体系化・抽象化を行う．そのためにまずは OS・ディストリビューション毎にコマンドを分離し，統一的な API でコマンドを呼び出すことができる汎用コマンド実行フレームワークを定義する．次に，汎用コマンド実行フレームワークを宣言的かつ自然言語に近い記法で操作できる制御テストフレームワークを定義する．汎用コマンド実行フレームワークと制御テストフレームワークの仕組みおよびその関係を**図 1** に示す．汎用コマンド実行フレームワークではまず，構成管理ツール固有の振る舞い（パッケージインストール等）を抽出する．そして要件（1）を満たすために，振る舞いのテストに特化した API を定義する．更に API から呼び出されるコマンドを OS・ディストリビューション毎に定義し，要件（2）を満たすために，API とコマンド群の間には OS・ディストリビューションを判別して自動で適切なコマンドを返すレイヤーを設ける．ここは構成管理ツール等，特別なソフトウェアを必要としない方式をとる．制御テストフレームワークではまず，要件（3）を満たすため，テストコード記述の抽象度を高め可読性を上げるために，宣言的かつ自然言語に近い記法で汎用コマンド実行フレームワークを操作するための記法の定義を行う．次に記法内の各命令と実際に呼び出す汎用コマンド実行フレームワークの API メソッドをひもづける．

3.4 提案手法の実装

提案手法に基づき実装した汎用コマンド実行フレームワークを specinfra[22], 制御テストフレームワークを serverspec[17] と名付けた．specinfra，serverspec ともに実装には Ruby を採用している．Ruby を採用した理由は RSpec を利用するためである．RSpec はテストフレームワークとして実績があり，自然言語に近い形でテストコードを記述することができるという要件を満たし，記法の拡張が可能であることから採用した．

RSpec を採用することによりテストコードがどのように書けるのかを例で示す．**図 2** に serverspec によりファイルに対してテストを行うためのコードを示す．describe で

```
describe file("/etc/password") do
  it { should be_file }
end

describe file("/tmp") do
  it { should be_directory }
end

describe file("/var/run/unicorn.sock") do
  it { should be_socket }
end

describe file("/etc/httpd/conf/httpd.conf") do
  its(:content) do
    should match /ServerName www.example.jp/
  end
end
```

図 2 serverspec によりファイルをテストするためのコード

```
describe user("root") do
  it { should exist }
  it { should have_uid 0 }
  it { should belong_to_group "root" }
  it { should have_home_directory "/root" }
  it { should have_login_shell "/bin/bash" }
end

describe group("root") do
  it { should exist }
  it { should have_gid 0 }
end
```

図 3 serverspec によりシステムユーザ/グループをテストするためのコード

はテストの対象となるサーバ上のリソースを指定する．この図では file(”/etc/passwd”) などがそれにあたる．これによりテスト対象が/etc/passwd というファイルであることを指定する．テストしたい内容は it { should ... } といった形で記述する．例えば，it { should be_file } は，対象リソースがファイルとして存在する，ということをテストするためのコードである．また，its(:content) といった形で指定することで，対象リソースそのものだけではなく，リソースに付随するもの，この例ではファイルの内容についてもテストすることができる．

図 3 にシステムユーザ/グループに対してテストを行うためのコードを示す．この例では，root ユーザが存在し，uid が 0，root グループに所属，ホームディレクトリが/root，ログインシェルが/bin/bash であること，root グループが存在し，gid が 0 であることをテストしている．

3.5 serverspec の特徴

提案手法に基づき実装した serverspec の特徴について述べる．特徴の一つとして，特定の構成管理ツールに依存していないことが挙げられる．そのため，どの構成管理ツールを利用していても serverspec を利用することができる．

```
@test "The package apache2 is installed" {
  dpkg-query -f '${Status}' -W apache2 \
    | grep '^install ok installed$'
}

@test "The apache2 service is running" {
  service apache2 status
}

@test "Port 80 is listening" {
  netstat -tunl | grep ":80 "
}
```

図 4 bats による Ubuntu 上でのテストコード

それだけにとどまらず，構成管理ツールを利用していない場合でも serverspec を利用することができる．ゆえに特定の構成管理ツール依存のテストツールと比べて利用の間口が広いと言える．また，特定の構成管理ツールに依存していないということは，テスト対象のサーバに特定のソフトウェアを入れる必要がないということでもある．serverspec はテスト対象サーバで sshd が動いてさえいれば，Ruby すら入れる必要がない．そのため特定の構成管理ツール依存のテストツールと比較して利用の敷居が低い．

二つ目の特徴は Test Kitchen や rspec-system のような統合テストスイートと比較して単機能な点である．単機能であるため他のツールとも組み合わせやすく，同種ツールとしてとりあげた Test Kitchen や rspec-system には，ツール標準のテスト機構を serverspec で置き換えるためのプラグインや，Vagrant[23] と連携して VM のテストを行うプラグインが存在する．

三つ目の特徴は記法の汎用性と抽象度の高さである．汎用性を高めたため，OS・ディストリビューションの違いを気にすることなくテストを容易に書くことができる．また抽象度が高いためテストコードの可読性が高く，メンテナンス性が高い．

4. 提案手法の評価

Test Kitchen 標準の bats[24] によるテストコードと serverspec のテストコードを比較し評価する．

bats による Ubuntu[25] 上でのテストコードを**図 4** に示す．また，同じ内容のテストを Solaris[26] 向けに書く場合の例を**図 5** に示す．serverspec によるテストコードは，OS が何であっても**図 6** で示すようなコードになる．このように，提案手法では OS の違いを意識することなく，テストコードを記述することができる．

別の比較として，/etc/sudoers が他人から読めないことをテストするコードの例を示す．bats では**図 7** に示すコードとなり，serverspec では**図 8** に示すコードとなる．このように，bats はテストコードだけでは何をテストしているのか判別しにくいため，説明用のテキストが必要となる．

```
@test "The package apache2 is installed" {
  pkg list -H apache2
}

@test "The apache2 service is running" {
  svcs -l apache2 | egrep '^status *online$'
}

@test "Port 80 is listening" {
  netstat -an | grep LISTEN | grep ".80 "
}
```

図 5 bats による Solaris 上でのテストコード

```
describe package("apache2") do
  it { should be_installed }
end

describe service("apache2") do
  it { should be_running }
end

describe port(80) do
  it { should be_listning }
end
```

図 6 serverspec によるテストコード

```
@test "/etc/sudoers is not readable by others" {
  ls -l /etc/sudoers | egrep '^......-..'
}
```

図 7 bats による/etc/sudoers が他人から読めないことをテストするコード

```
describe file("/etc/sudoers") do
  it { should_not be_readable.by("others") }
end
```

図 8 serverspec による/etc/sudoers が他人から読めないことをテストするコード

一方 serverspec はテストコードだけでテスト内容が理解できるため，別途説明用のテキストを必要としない．

serverspec はオープンソースで公開されており，上述のような利用ための敷居の低さ，単機能，記法の汎用性・抽象度の高さから，利用が広がっている．また，採用している企業もいくつか見受けられる [18][19]．しかし，定量的な評価ができておらず，評価手法の検討など，今後に課題が残る．

5. まとめ

本論文では，Test-Driven Infrastructure を支援するための従来テストフレームワーク手法の問題点について指摘し，解決するために必要な要件として，構成管理ツール独立性と OS・ディストリビューション汎用性を提案した．また，これらの要件を満たすために，汎用コマンド実行フレームワークと制御テストフレームワークに分離して定義する手法についても提案した．これらふたつの実装例とし

て specinfra と serverspec を紹介した．

serverspec の登場によって Test-Driven Infrastucture というプロセスが国内でも徐々に認知され，この分野の今後の発展が期待される．また specinfra を基盤とした，serverspec よりも優れたテストフレームワーク実装の登場や，確認作業以外の運用業務に必要なコマンド群を体系化することによるテスト以外への specinfra の応用，例えば構成管理ツールへの応用なども今後期待される．

今後の課題として，提案手法による運用効率向上の定量的評価ができていないので，評価手法について検討する．

謝辞 serverspec 実装にあたり、提案手法の原型となる実装を示してくれた株式会社 paperboy&co. 伊藤洋也氏に感謝する。

参考文献

[1] R. Prodan and S. Ostermann, “A Survey and Taxonomy of Infrastructure as a Service and Web Hosting Cloud Providers”, *Grid Com puting, 2009 10th IEEE/ACM International Conference*, pp. 17–25 (2009).
[2] M. Burgess, “CFEngine: a system configuration engine”, http://cfengine.com/markburgess/papers/cfengine_history.pdf.
[3] Wikipedia,“Comparison of open-source configuration management software”, http://en.wikipedia.org/wiki/Comparison_of_open_source_configuration_management_software.
[4] L. Kanies, “Puppet: Next-Generation Configuration Management”, *USNIX; login*, Vol. 31, No. 1 (2006).
[5] Amazon WebServices,Inc., “Release: Amazon EC2 on 2006-08-23”, http://aws.amazon.com/releasenotes/Amazon-EC2/353.
[6] W. Cunningham, “アジャイルソフトウェア開発宣言”, http://agilemanifesto.org/iso/ja/.
[7] P. Debois, “Agile infrastructure and operations: how infra-gile are you?”, *Agile, 2008. AGILE ’08. Conference*, pp. 202–207 (2009).
[8] K. Beck, *Test Driven Development: By Example*, Addison-Wesley Professional (2002).
[9] S. Nelson-Smith, *Test-Driven Infrastructure with Chef, 2nd Edition*, O’Reilly Media (2013).
[10] S. Vargo: “ChefSpec”, http://code.sethvargo.com/chefspec/.
[11] T.Sharpe, “rspec-puppet”, http://rspec-puppet.com/.
[12] S. Nelson-Smith, “Cucumber-Chef”, http://www.cucumber-chef.org/.
[13] D. Calavera, “minitest-chef-handler”, https://github.com/calavera/minitest-chef-handler.
[14] Heavy Warter Opertaions, LLC., “Test Kitchen”, http://kitchen.ci/.
[15] Puppet Labs, “rspec-system”, https://github.com/puppetlabs/rspec-system.
[16] RSpec, “RSpec”, http://rspec.info/.
[17] G. Miyashita, “serverspec”, https://github.com/serverspec/serverspec.
[18] 任天堂株式会社, “採用情報：キャリア採用 募集要項 - 京都勤務の募集職種開発部門”, http://www.nintendo.co.jp/jobs/career/kyoto_sec1.html#p03.
[19] Wantedly, “serverspec の求人 - Wantedly”, https://www.wantedly.com/projects/search?q=serverspec.
[20] Chef, “Chef - Code Can — Chef”, http://www.getchef.com/.
[21] Y. Matsumoto, “オブジェクト指向スクリプト言語 Ruby”，https://www.ruby-lang.org/ja/.
[22] G. Miyashita, “specinfra”, https://github.com/serverspec/specinfra.
[23] M. Hashimoto, “Vagrant”, http://www.vagrantup.com/.
[24] S. Stephenson, “bats”, https://github.com/sstephenson/bats.
[25] Ubuntu, “The world’s most popular free OS — Ubuntu”, http://www.ubuntu.com/.
[26] Oracle, “Oracle Solaris”, http://www.oracle.com/jp/products/servers-storage/solaris/overview/index.html.