情報処理学会論文誌 Vol.61 No.3 677–686 (Mar. 2020)

Serverspec：宣言的記述でサーバの設定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[3] 松本 亮介[1]

受付日 2019年6月18日, 採録日 2019年11月29日

概要：システムの大規模・複雑化にともない，サーバの構築・運用を効率化するために，サーバの設定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 Provisioning Testing という手法が提案されている．この手法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 ごと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．シェルコマンドを直接記述する必要があり，可読性に難があるものも存在する．そこで本論文では，生産性や保守性を向上するために，サーバの設定状態を汎用的かつ可読性の高い宣言的な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，OS や構成管理ツール固有の振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な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，可読性の高いコードでサーバの設定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．提案するテストフレームワークを Serverspec と名付けた．提案手法の評価は，生産性，保守性に加え，拡張性，実装公開後の影響の 4 つの観点で行った．

キーワード：test-driven infrastructure，provisioning testing，serverspec，specinfra

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

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

Received: June 18, 2019, Accepted: November 29, 2019

Abstract: As the increase of large and complex systems, many ways to describe states of servers as code are supplied. As an effective process to handle these ways, Provisioning Testing 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 versatility because some frameworks depend on specific configuration management tools, and others need the consideration of differences between OSes to write test code. Also, others need to write shell commands directly, which is difficult to read.

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 for productivity and maintainability. In our proposal, we arrange and generalize the behavior of OSes and configuration management tools and define the versatile command execution framework to increase versatility. Next, we define the test framework that controls the command execution framework by declarative and natural language to increase the level of abstraction and readability. By this test framework, system administrators can test states of servers easily without considering the differences 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 has original notation easily. We named the proposed test framework Serverspec. We evaluated this proposed method on four perspectives, productivity, maintainability, extendability, and impact after the publication of the implementation.

Keywords: test-driven infrastructure, provisioning testing, serverspec, specinfra

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] のにともない，UNIX シェルにより書かれた手続き的な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．宣言的なコードで扱う手法では，サーバの最終的な設定状態を記述するだけで，どのように設定するかは記述しない．そのため，記述が簡潔になり，覚えるルールが少なくて済むという点で，生産性や保守性が向上する．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” [6] 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．Infrastructure as Code によるサーバ設定を宣言的なコードで扱う構成管理手法は，元々自動化に焦点が当てられていた．コードで扱うことから，バージョン管理，テスト，継続的インテグレーションなどのアジャイルソフトウェア開発 [7] 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” [8] 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 [9] の手法をサーバ構築・運用に応用した “Test-Driven Infrastructure” [10] が提案され，現在はサーバ構築時のテスト全般を指す手法として，Provisioning Testing [11] と呼ばれている．このプロセスを支援するテストフレームワークがいくつも登場している [12], [13], [14], [15], [16], [17]．これらのうち，ChefSpec [12]，rspec-puppet [13] 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．ここでいう単体テストとは，実際にコードをサーバに適用せずに行う静的テストであり，結合テストとは，コードをサーバに適用して行う動的テストである．

Cucumber-chef [14]，minitest-chef-handler [15] は Chef [18] 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 [16] や rspec-system [17] 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，特定の構成管理ツールに依存している．また，標準で持つテスト実行機能は，OS やディストリビューションに応じたシェルコマンドを直接記述する必要がある．そのため，どのようにテストを行うのかを手続き的に記述する必要があり，何をテストするのかを宣言的に記述するのと比較し，記述が冗長で理解しにくいものになる．また，OS やディストリビューション固有のシェルコマンドを覚える必要がある．このように，記述が冗長で理解しにくいものになる，覚えるべきルールが増える，という点で生産性や保守性が低下するという問題がある．

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舞い，たとえば，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ごとに分離，そのうえで 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，宣言的な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．たとえば，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 [19] 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 [20] と名付けた．

本論文では，Infrastructure as Code や Test-Driven Infrastructure という潮流のなかでの Serverspec の立ち位置を明らかにし，他の類似フレームワークと比較することで，Serverspec を有用たらしめる手法を体系的に分析することを試みる．そして，分析して得られた手法を応用することで，さらに有用なサーバインフラ管理手法の研究開発に役立てることを目指している．

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さらに詳しく述べる．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，実装公開後の実績や影響について論じ，5 章でまとめとする．

[1] さくらインターネット株式会社さくらインターネット研究所
SAKURA internet Research Center, SAKURA internet Inc., Shinjuku, Tokyo 160–0023, Japan
[2] 合同会社 Serverspec Operations
Serverspec Operations, LLC, Sagamihara, Kanagawa 252–0242, Japan
[3] GMO ペパボ株式会社ペパボ研究所
Pepabo Research and Development Institute, GMO Pepabo, Inc., Shibuya, Tokyo 150–8512, Japan
[a)] miyashita@serverspec-operations.com

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良く行うか，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フレームワークについて言及する．

2.1 CFEngine から Chef, Chef から Test-Driven Infrastructure へ

CFEngine [2] 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 [18] が登場している．

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] のよう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，独自のドメイン固有言語と比較して自由度が高く制約が少ないため，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．コードが複雑になると，そのコードが正しいコードなのか，見ただけで判別するのが難しくなる．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ことで，複雑なコードでも正しさを保証することができる，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．

2.2 Test-Driven Infrastructure から Provisioning Testing へ

Test-Driven Infrastructure という言葉からは，テストコードを書いた後，サーバ構成を記述したコードを書く，というプロセスが想起される．しかし実際には，サーバ構成を記述したコードを書いた後，テストコードを書く，というプロセスをふむことも多い．そのため現在では，Test-Driven Infrastructure よりも，サーバ構築時のテスト全般を指す Provioning Testing [11] という言葉が広く利用されている．

2.3 Provisioning Testing におけるテスト手法の分類

Provisioning Testing に関連するテストの種類は，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．ここでいうモジュールとは，単一の目的を持ったコードの集まりのことを指す．

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，コードが期待どお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．

(1) に分類される単体テストフレームワークとしては，Chef には ChefSpec [12]，Puppet には rspec-puppet [13] というフレームワークが存在する．その名が示すとおり，それぞれ Chef と Puppet 専用のフレームワークであり，サーバに対してコードを適用して設定を行うことなく，コードの整合性などのテストを行う．

(2) に分類される結合テストフレームワークとしては，minitest-chef-handler [15]，Test Kitchen [16]，rspec-system [17] が存在する．

(3) に分類される受け入れテストフレームワークとしては cucumber-chef [14]，leibniz [22] が存在する．

サーバ構成を記述したコードのテストは，コードによってもたらされたサーバ内部の設定状態をテストすべきものである．(2) の結合テストは，サーバの内部状態を網羅的にテスト可能であり，サーバ構成を記述したコードのテストとしてふさわしいものである．よって以降は (2) の結合テストに対象を絞って論を進める．

2.4 結合テストにおける従来手法の考察

minitest-chef-handler は Chef 専用のテストフレームワークであり，Chef 以外の構成管理ツールと組み合わせて利用することができない．一方，OS やディストリビューションが異なっていても，テストコードが汎用的に記述でき，覚えることが少ない．また，宣言的な記述により，どうテストするのかではなく，何をテストするのかを簡潔に記述することができる．そのため，テストコードが読みやすくテストの意図が分かりやすい．

Test Kitchen はテスト用仮想マシンの作成，仮想マシンへの構成管理ツールの適用，テストの実行，仮想マシンの破棄といった，テストに必要なライフサイクルをトータルでサポートする統合テストスイートである．構成管理ツールには Chef を利用することが前提となっている．標準の

表 1 従来手法のメリットとデメリット
Table 1 Advantages and disadvantages of conventional methods.

従来手法	単体利用可	OS汎用性	コード可読性	構成管理ツール非依存	様々な実行形式	ライフサイクルサポート
minitest-chef-handler	×	○	○	×	×	×
Test Kitchen	×	×	×	×	○	○
rspec-system	×	×	×	×	×	○

テストフレームワークはシェルコマンドを直接記述する必要があり，OS やディストリビューションごとに記述が異なる．また，記述が宣言的ではなく手続き的なため，何をテストするのかではなく，どうテストするのかを記述する必要があり，可読性が低くテストの意図が分かりにくい．

統合テストスイートなので，サポートするライフサイクルをそのまますべて利用できれば，非常に便利で強力なフレームワークであり，実行形式やテストフレームワークをプラグインで柔軟に差し替えることができる．ここでいう実行形式とは，ローカルホストでの直接実行，SSH や WinRM を介してのリモートホスト上での実行，Docker [23] API 経由での Docker コンテナ上での実行などを指す．

rspec-system も Test Kitchen 同様，テストに必要なライフサイクルをトータルでサポートする統合テストスイートである．こちらはテスト用仮想マシンのセットアップにシェルコマンドを利用するか，構成管理ツール Puppet を利用することが前提となっている．また，標準のテストフレームワークはシェルコマンドを直接記述する必要があり，OS やディストリビューションごとに記述が異なり，可読性も低い，という点は Test Kitchen と同様である．

以上により，従来手法のメリットやデメリットは，次のように整理できる．

- minitest-chef-handler は，Chef という特定の構成管理ツールに依存している，という問題がある．
- 一方，OS やディストリビューションを問わず，汎用的かつ宣言的な，読み書きしやすいコードで記述できる，というメリットがある．
- Test Kitchen や rspec-system は，テストのライフサイクルをトータルでサポートするという点で非常に便利で強力である．
- その反面，テストフレームワークはシェルコマンドを直接記述する必要があり，OS やディストリビューションごとに記述が異なり，可読性も低い．
- また，特定の構成管理ツールの利用が前提となっている．
- Test Kitchen は様々な実行形式に柔軟に対応可能である．

これらのメリット・デメリットを整理すると**表 1** のようになる．

3. 提案するサーバテスト手法

3.1 従来手法を補うための要件の考察と要件を満たすための手法の提案

2.4 節で述べた従来手法フレームワークのメリットを活かし，デメリットを補うための要件について考察する．

まず，統合テストスイートではなく，単体で利用できるテストフレームワークとする．これにより，特定の構成管理ツールに依存することなく，任意のツールと連携しやすくなる．また，Test Kitchen や rspec-system といった既存の統合テストスイートと組み合わせて利用することもできるようになる．

次に，minitest-chef-handler のメリットを活かし，OS やディストリビューションが異なっていても，汎用的にコードが記述できるようにする．また，コードの可読性を高めるために，宣言的に記述できるようにする．宣言的な記述は minitest-chef-handler 以外にも，CFEngine，Chef，Puppet といった構成管理ツールや，ChefSpec，rspec-puppet といったテストフレームワークなどでも広く採用されている．

そして，OS やディストリビューションの違いを吸収する機能を，特定の構成管理ツールに依存しない実装とする．minitest-chef-handler は，OS やディストリビューションの違いを吸収する機能を Chef の実装に依存することで，実装の手間を省いている．しかしこの実装では，特定の構成管理ツールに依存してしまい，最初にあげた，単体で利用できるテストフレームワークとし，特定の構成管理ツールに依存しない，という要件と矛盾する．

テスト実行の対象となるマシンは，ローカルホスト，リモートホスト，仮想マシン，コンテナなど，状況により様々である．そのため，Test Kitchen のように様々な実行形式をサポートすることも要件に含める．

これらの要件をまとめると次のようになる．

(1) テストスイートではなく単体で利用できるテストフレームワークとする．
(2) OS・ディストリビューションが違っていても，汎用的にコードが記述できる．
(3) 宣言的にコードが記述でき可読性が高い．
(4) 特定の構成管理ツールに実装を依存しない．
(5) 様々な実行形式に対応できる．

これらの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ごとにコマンドを分離し，汎用的な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な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舞い（パッケージインストールなど）を抽出する．

そして要件 (1) を満たすために，振舞いのテストに特化したAPIを定義する．さらに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ごとに定義し，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．ここは要件 (4) を満たすため，構成管理ツールなど，特別なソフトウェアを必要としない方式をとる．

また，要件 (5) を満たすために，様々な実行形式を抽象化したレイヤーを設ける．このレイヤーは，OS抽象化レイヤーから実行すべきコマンドを受け取り，選択された実行形式に応じてコマンドを実行する．

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，宣言的な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．

3.2 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 [24]，制御テストフレームワークをServerspec [20] 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，宣言的に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/profile”) などがそれにあたる．これによりテスト対象が/etc/profile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．たとえば，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

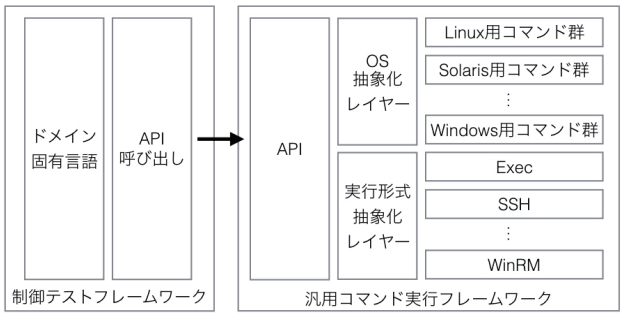


図 1 汎用コマンド実行フレームワークと制御テストフレームワークの仕組みと関係

Fig. 1 Relationship between versatile command execution framework and control test framework.

```
describe file("/etc/profile") 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

Fig. 2 Code to test files by Serverspec.

```
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

Fig. 3 Code to test a user and a group by Serverspec.

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．

4. 提案手法の評価と実装公開後の影響

提案手法を，生産性，保守性，拡張性において評価を行う．また，提案手法実装の公開後の影響についても述べる．

4.1 提案手法実装の生産性評価

提案手法要件 3.1 節 (2) を満たすことによる生産性の評価を示す．

比較対象となる従来手法のテストフレームワークとしては，minitest-chef-handler と Test Kitchen 標準の bats [25] が存在する．minitest-chef-handler は構成管理ツール Chef に依存しているという制約がある．一方, bats は特定の構成管理ツールに依存していない．また，minitest-chef-handler は，汎用的なコードで記述ができる．一方 bats はシェルコマンドを手続き的に記述する必要があり，OS やディストリビューションごとに記述が異なり，可読性も低い．したがって，構成管理ツールに依存していない，という同じ条件の下で，テストコードの記法の違いによる生産性の比較を行うため，bats を比較対象として評価を行う．

Serverspec のバージョンは v2.41.4 [*1]，bats のバージョンは v0.4.0 [*2]で比較を行う．

bats による Ubuntu [26] 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 CentOS 向けに書く場合の例を**図 5** に，Solaris [27] 向けに書く場合の例を**図 6** に示す．Serverspec によるテストコードは，OS が何であっても**図 7** で示すようなコードになる．このように，提案手法では OS やディストリビューションの違いを意識することなく，効率的にテストコードを記述することができるという点で，生産性が高いといえる．

4.2 提案手法実装の保守性評価

提案手法要件 3.1 節 (3) を満たすことによる保守性の評価を示す．

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 8** に示すコードとなり，Serverspec では**図 9** に示すコードとなる．このように，bats は手続き的なコードで記述するため，テストコードだけでは何をテストしているのか理解することが難しい．一方 Serverspec は，file(“/etc/sudoers”) という形でテスト対象が明示的に示されており，実行コマンドではなくテストしたい意図を宣言的に記述しているため，何をテストしているのか理解しやすい．したがって，Serverspec によるテストコードは，コードが簡潔で意図が理解しやすいという点で，保守性が高いといえる．

[*1] https://github.com/mizzy/serverspec/releases/tag/v2.41.4
[*2] https://github.com/sstephenson/bats/releases/tag/v0.4.0

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

@test "The apache service is running" {
  service nginx 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
Fig. 4 Test code for Ubuntu by bats.

```
@test "The package nginx is installed" {
  rpm -q nginx
}

@test "The apache service is running" {
  service nginx status
}

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

図 5 bats による CentOS 上でのテストコード
Fig. 5 Test code for CentOS by bats.

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

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

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

図 6 bats による Solaris 上でのテストコード
Fig. 6 Test code for Solaris by bats.

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

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

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

図 7 Serverspec によるテストコード
Fig. 7 Test code by Serverspec.

4.3 提案手法実装の拡張性評価

提案手法要件 3.1 節 (2)，(5) を満たすことによる拡張性の評価を示す．

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

図 8 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

Fig. 8 Test code by bats to confirm that /etc/sudoers is not readable by others.

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

図 9 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

Fig. 9 Test code by Serverspec to confirm that /etc/sudoers is not readable by others.

OS・ディストリビューションと実行形式の抽象化により，Serverspec は様々な環境上でのテスト実行に対応している．Serverspec は公開当初，OS は Red Hat 系 Linux，実行形式は SSH でのリモート実行にしか対応していなかった．しかしその拡張性の高さにより，様々な OS・ディストリビューションや実行形式が追加され，現在では以下の OS・ディストリビューションと実行形式に対応している．

- OS・ディストリビューション
  - Linux
    - Alpine Linux，CoreOS，Red Hat Linux，Amazon Linux，Cumulus Linux，Debian，Devuan，elementary OS，Arista EOS，Fedora，Gentoo，Linux Mint，NixOS，openSUSE，Plamo Linux，Poky，Raspbian，SUSE，Ubuntu
  - Darwin
  - FreeBSD
  - OpenBSD
  - SmartOS
  - Solaris
  - Windows
- 実行形式
  - ローカルでの直接実行
  - SSH/Telnet/WinRM でのリモート実行
  - Docker API による Docker コンテナ上での実行
  - liblxc による LXC コンテナ上での実行
  - jexec による FreeBSD jail 環境上での実行

4.4 Serverspec の特徴と公開後の影響

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 1 つとして，提案手法要件 3.1 節 (4) を満たすことにより，特定の構成管理ツールに依存していないことがあげられる．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．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フレームワークと比べて利用の間口が広いといえる．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 は様々な実行形式に対応しており，SSH でのリモート実行を選択した場合，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，すでに動いているサーバに対してテストを行いたい場合も，影響を最小限に抑えることができる．ただし，サーバに Ruby を入れる必要がないといっても，Serverspec を実行するクライアント側には Ruby が必要である．

2 つ目の特徴は，提案手法要件 3.1 節 (1) を満たすことにより，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フレームワークとも組み合わせやすく，同種フレームワークとしてとりあげた Test Kitchen や rspec-system には，フレームワーク標準のテスト機構を Serverspec で置き換えるためのプラグインが存在する．

また，Vagrant [28] と連携して VM のテストを行う vagrant-serverspec [29]，Sensu [30] と連携してサーバ監視に Serverspec を利用する sensu-plugins-serverspec [31]，Packer [32] によるプロビジョニング後に Serverspec でテストを行う packer-provisioner-serverspec [33]，Ansible [34] によるプロビジョニング後に Serverspec でテストを行う ansible_spec [35] など，Serverspec と連携するための様々なツールが公開されている．これは裏を返すと，Serverspec 単体では不十分で他のツールと連携して利用する必要があり，連携のためのツールをつくる必要がある，ということの現れでもある．そのため，必要なことをすべて行ってくれるオールインワンなフレームワークを求めるケースには Serverspec は向かない．逆に，様々なツールを組み合わせて要件に合うシステムを構築したい，というケースに向いている．

3 つ目の特徴は，提案手法要件 3.1 節 (2)，(3) を満たすことによる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．逆に，汎用性を高めるための実装を，他の構成管理ツールなどに依存することなく自前で実装しているため，実装が複雑になっている．汎用性や抽象度にも限界がある．Serverspec はテスト実行対象マシン上でコマンドを実行してテストを行っている．したがって，コマンドで実行可能なテストであれば，どのようなテストでも対応できる．しかしそのためには，テスト対象マシン上に必要なコマンドがあらかじめ存在する必要がある．また，GUI で

しか行えないようなテストには対応することができない．さらに，OS やディストリビューションごとの差違をすべて吸収できているわけではない．そのため，同じ目的のテストでも，OS が異なると同じ記述では動作しない場合がある．また，特定の OS でしか実行できないテストも存在する．

4 つ目の特徴は，提案手法要件 3.1 節 (5) を満たすことによる，様々な実行形式への対応である．ローカルでの直接実行，SSH や Telnet や WinRM 経由でのリモート実行，仮想マシン，Docker や LXC [36] コンテナ，FreeBSD jail [37] といった様々な仮想環境上での実行に対応している．また，新たな実行形式にも容易に対応することができる．

公開後の影響について述べる．

Serverspec の公開が Provioning Testing というプロセスが世界中で認知されるきっかけの 1 つとなった．広く認知されることになったのは，RSpec というテスト駆動開発において実績があり広く知られているライブラリを実装として採用したことが大きい．これにより，テスト駆動開発の手法をサーバ構築と結び付ける，という発想を，既存ライブラリの力を借りることで，分かりやすく示すことができた．

ThoughtWorks Technology Radar の Provisioning Testing の項 [11] には，Serverspec の名が記載されている．また，Black Duck Open Source Rookies of the Year 2013 [38] や第 10 回日本 OSS 奨励賞 [39] を受賞している．このように，Serverspec は日本だけでなく海外でも認知されている．

Serverspec に影響を受けて開発されたテストフレームワークに，Testinfra [40], Goss [41], InSpec [42], awspec [43] といったものがある．1 章で「Test Kitchen や rspec-system は標準で持つテスト実行機能は汎用性に乏しい」と述べたが，Serverspec をプラグインとして利用する，という形で改善されている [44], [45]．また，Dockerspec [46] という，Serverspec を利用して Docker コンテナのテストを行うフレームワークや，Container Structure Test [47] というコンテナ用テストフレームワークが開発されるなど，コンテナの世界にも Priviosioning Testing が適用されている．このように Serverspec は，煩雑化していたサーバ構築の確認作業にテスト駆動開発の手法を取り入れ，サーバ構築のコード化をさらにソフトウェアのテスト駆動開発に近づけたという有用性により，他の Provisioning Testing 関連フレームワークに大きな影響を与えている．

Serverspec は 2019 年 6 月 17 日現在，rubygems.org 上で 1,900 万回以上ダウンロードされている [48]．これは rubygems.org 全体の上位 0.3%以内に位置する．このことから，Serverspec は多くのユーザから利用されていることが分かる．

また，日経 SYSTEMS の特集記事で取り上げられたり [49]，日経電子版のサーバ環境構築に利用されていたり [50]，任天堂社内でも利用されていたり [51] と，企業での導入も進んでいる．その他数多くの企業でも採用されており，世界中で広く利用されている．

このように，Serverspec は日本だけでなく海外でも認知され，多くのユーザや企業に利用されており，他の Provisioning Testing 関連フレームワークに多大なる影響を与えている．

定量的な評価としては，Serverspec 導入前と導入後で，以下のような数値を比較する，といった方法が考えられる．

- サーバ構築や設定変更時のテストにかかる時間
- サーバ構築や設定変更後に発生する障害の数

このような評価を行うためには，Serverspec 導入前の数値計測，Serverspec の導入，Serverspec 導入後の数値計測が必要となる．数値での比較は今後の課題とする．

5. まとめ

本論文では，Provisioning Testing を支援するための従来テストフレームワーク手法のメリットとデメリットについて整理し，メリットを活かしデメリットを補うための要件について整理した．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 2 つの実装例として Specinfra と Serverspec を紹介した．

Serverspec の公開が Provisioning Testing というプロセスが世界中で認知されるきっかけの 1 つとなり，影響を受けたテストフレームワークがほかにも生まれたり，コンテナの世界にも Provisioning Testing が適用されたりするようになった．オライリー・ジャパンから Serverspec に関する書籍 [52] も出ている．サーバの設定状態をテストするテストフレームワークのデファクトスタンダードとしてのポジションを確立したといえるだろう．

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や構成管理以外への Specinfra の応用も期待していたが，これらはまだ実現していない．これについては新たな研究テーマとして解決を試みたい．

Serverspec と同時期にリリースされた Docker によって，コンテナが広く使われるようになり，サーバの構成管理とテストのあり方が変わってきている．Docker によるコンテナイメージ作成は，従来のサーバプロビジョニングと比べ簡素なため，サーバ構成管理ツールの重要性が低下しており，それにともない，Provisioning Testing の重要性も低下している．

コンテナのプロビジョニングは簡素になったが，システム全体としてはマイクロサービス化により依存関係が複雑になっている．その複雑な依存関係において，問題が発生

した際に原因を特定するための，分散トレーシングという技術が注目されている．しかし，問題が発生しないよう事前に行うテストについては，これといった手法はまだ存在していない．このような課題を，本論文での提案手法や別の手法により解決できないか，という試みも興味深い研究テーマとなるであろう．

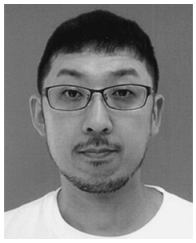
謝辞 Serverspec 実装にあたり，提案手法の原型となる実装を示してくれた GMO ペパボ株式会社伊藤洋也氏に感謝する．また，複数 OS 対応の実装やリソースタイプという重要な実装上の概念を提案してくれた Tatsuya Takamura 氏，実行形式を切り替える機構を実装してくれた Raphaël Pinson 氏，その他多くのコントリビュータの方々[*3]にも感謝する．

参考文献

[1] Prodan, R. and Ostermann, S.: 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] Burgess, M.: CFEngine: A system configuration engine, available from ⟨http://cfengine.com/markburgess/papers/cfengine_history.pdf⟩ (accessed 2019-06-17).
[3] Wikipedia: Comparison of open-source configuration management software, available from ⟨http://en.wikipedia.org/wiki/Comparison_of_open_source_configuration_management_software⟩ (accessed 2019-06-17).
[4] Kanies, L.: Puppet: Next-Generation Configuration Management, *USNIX; login*, Vol.31, No.1 (2006).
[5] Amazon Web Services, Inc: AWS Release Notes, available from ⟨https://aws.amazon.com/jp/releasenotes/release-amazon-ec2-on-2006-08-23/⟩ (accessed 2019-06-17).
[6] Hummer, W., Rosenberg, F., Oliveira, F. and Eilam, T.: Testing Idempotence for Infrastructure as Code, *Middleware 2013: ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp.368–388 (2013).
[7] Cunningham, W.: アジャイルソフトウェア開発宣言，入手先 ⟨http://agilemanifesto.org/iso/ja/manifesto.html⟩（参照 2019-06-17）.
[8] Debois, P.: Agile infrastructure and operations: How infra-gile are you?, *Agile 2008 Conference* (*AGILE '08*), pp.202–207 (2009).
[9] Beck, K.: *Test Driven Development: By Example*, Addison-Wesley Professional (2002).
[10] Nelson-Smith, S.: *Test-Driven Infrastructure with Chef, 2nd Edition*, O'Reilly Media (2013).
[11] ThouhghtWorks: Provisioning testing — ThoughtWorks, available from ⟨https://www.thoughtworks.com/radar/techniques/provisioning-testing⟩ (accessed 2019-06-17).
[12] Chef: ChefSpec, available from ⟨https://docs.chef.io/chefspec.html⟩ (accessed 2016-06-17).
[13] Sharpe, T.: rspec-puppet, available from ⟨http://rspec-puppet.com/⟩ (accessed 2019-06-17).
[14] Nelson-Smith, S.: Atalanta/cucumber-chef, available from ⟨https://github.com/Atalanta/cucumber-chef⟩ (accessed 2019-06-17).
[15] Calavera, D.: chef-boneyard/minitest-chef-handler, available from ⟨https://github.com/chef-boneyard/minitest-chef-handler⟩ (accessed 2019-06-17).
[16] Chef Software, Inc.: Test Kitchen, available from ⟨http://kitchen.ci/⟩ (accessed 2019-06-17).
[17] Barber, K.: puppetlabs/rspec-system, available from ⟨https://github.com/puppetlabs/rspec-system⟩ (accessed 2019-06-17).
[18] Chef Software, Inc.: Chef: Deploy new code faster and more frequently, Automate infrastructure and applications, available from ⟨http://www.chef.io/⟩ (accessed 2019-06-17).
[19] RSpec: RSpec, available from ⟨http://rspec.info/⟩ (accessed 2019-06-17).
[20] Miyashita, G.: mizzy/serverspec, available from ⟨https://github.com/mizzy/serverspec⟩ (accessed 2019-06-17).
[21] Matsumoto, Y.: オブジェクト指向スクリプト言語 Ruby，入手先 ⟨https://www.ruby-lang.org/ja/⟩（参照 2016-06-17）.
[22] Nelson-Smith, S.: Atalanta/leibniz, available from ⟨https://github.com/Atalanta/leibniz⟩ (accessed 2019-06-17).
[23] Docker Inc.: Enterprise Container Platform — Docker, available from ⟨https://www.docker.com/⟩ (accessed 2019-06-17).
[24] Miyashita, G.: mizzy/specinfra, available from ⟨https://github.com/mizzy/specinfra⟩ (accessed 2016-06-17).
[25] Stephenson, S.: sstephenson/bats, available from ⟨https://github.com/sstephenson/bats⟩ (accessed 2019-06-17).
[26] Ubuntu: The world's most popular free OS — Ubuntu, available from ⟨http://www.ubuntu.com/⟩ (accessed 2019-06-17).
[27] Oracle: Oracle Solaris 11, available from ⟨https://www.oracle.com/solaris/solaris11/⟩ (accessed 2019-06-17).
[28] Hashimoto, M.: Vagrant, Hashicorp (online), available from ⟨http://www.vagrantup.com/⟩ (accessed 2019-06-17).
[29] Voorhis, J.: jvoorhis/vagrant-serverspec, available from ⟨https://github.com/jvoorhis/vagrant-serverspec⟩ (accessed 2019-06-17).
[30] Workflow automation for monitoring — Sensu, Sensu (online), available from ⟨https://sensu.io/⟩ (accessed 2019-06-17).
[31] Smith, T.: sensu-plugins/sensu-plugins-serverspec, available from ⟨https://github.com/sensu-plugins/sensu-plugins-serverspec⟩ (accessed 2019-06-17).
[32] HashiCorp: Packer by HashiCorp, available from ⟨https://packer.io/⟩ (accessed 2019-06-17).
[33] Carey, W.: unifio/packer-provisioner-serverspec: Packer Serverspec remote provisioner, available from ⟨https://github.com/unifio/packer-provisioner-serverspec⟩ (accessed 2019-06-17).
[34] Red Hat, Inc.: Ansible is Simple IT Automation, available from ⟨https://www.ansible.com/⟩ (accessed 2019-06-17).
[35] volanja: volanja/ansible_spec, available from ⟨https://github.com/volanja/ansible_spec⟩ (accessed 2019-06-17).
[36] Canonical Ltd.: Linux Containers, available from ⟨https://linuxcontainers.org/⟩ (accessed 2019-06-17).
[37] The FreeBSD Foundation: Chapter 14. Jails, available

[*3] https://github.com/mizzy/serverspec/graphs/contributors, https://github.com/mizzy/specinfra/graphs/contributors

from ⟨https://www.freebsd.org/doc/handbook/jails.html⟩ (accessed 2019-06-17).
[38] OSDN 株式会社：米 Black Duck が恒例の「オープンソースルーキー」を発表，Docker，OpenDaylight，Serverspec などが入選 — OSDN Magazine，入手先 ⟨https://mag.osdn.jp/14/01/29/190000⟩（参照 2019-06-17）.
[39] 日本 OSS 推進フォーラム：「第 10 回日本 OSS 貢献者賞・日本 OSS 奨励賞」受賞者を選定 — 日本 OSS 推進フォーラム，入手先 ⟨http://ossforum.jp/ossaward10th2⟩（参照 2019-06-17）.
[40] Pepiot, P.: Testinfra test your infrastructure, available from ⟨https://testinfra.readthedocs.io/⟩ (accessed 2019-06-17).
[41] Elsabbahy, A.: aelsabbahy/goss, available from ⟨https://github.com/aelsabbahy/goss⟩ (accessed 2019-06-17).
[42] Chef Software, Inc.: Chef InSpec—Audit and Automated Testing Framework, available from ⟨https://www.inspec.io/⟩ (accessed 2019-06-17).
[43] Oyama, K.: k1LoW/awspec, available from ⟨https://github.com/k1LoW/awspec⟩ (accessed 2019-06-17).
[44] ci-lab k: test-kitchen/busser-serverspec, available from ⟨https://github.com/test-kitchen/busser-serverspec⟩ (accessed 2016-06-17).
[45] Haugen, H.: puppetlabs/rspec-system-serverspec, Puppet Labs (online), available from ⟨https://github.com/puppetlabs/rspec-system-serverspec⟩ (accessed 2016-06-17).
[46] de Zuazo, X.: zuazo/dockerspec, available from ⟨https://github.com/zuazo/dockerspec⟩ (accessed 2019-06-17).
[47] Google: GoogleContainerTools/container-structure-test, available from ⟨https://github.com/GoogleContainerTools/container-structure-test⟩ (accessed 2019-06-17).
[48] xmisao: serverspec – BestGems, available from ⟨http://bestgems.org/gems/serverspec⟩ (accessed 2019-06-17).
[49] 白井 良：[PART5] “泣き所” だったサーバ検証 専用ツールつなぎ自動化する—開発・運用ツール利用実態調査 2014「黄金コンビはこれだ」：日経 xTECH Active，入手先 ⟨https://tech.nikkeibp.co.jp/it/atclact/active/15/060400056/060500005/⟩（参照 2019-06-17）.
[50] Elastic：日経電子版の記事検索およびログ解析の両方を 1 つの仕組みで実現 — Elastic Customers，入手先 ⟨https://www.elastic.co/jp/use-cases/nikkei⟩（参照 2019-06-17）.
[51] 任天堂：キャリア採用：募集要項—ソフトウェアエンジニア（京都勤務）| 採用情報 | 任天堂，入手先 ⟨https://www.nintendo.co.jp/jobs/career/kyoto_sec2.html#nwie⟩（参照 2019-06-17）.
[52] 宮下剛輔：*Serverspec*，オライリー・ジャパン (2015).



宮下 剛輔 （正会員）

1997 年北海道大学経済学部経営学科卒業．同年伊藤忠テクノサイエンス株式会社（現，伊藤忠テクノソリューションズ株式会社）入社，2004 年株式会社ネットマークス（現，ユニアデックス株式会社）入社，2006 年株式会社 paperboy&co.（現，GMO ペパボ株式会社）入社を経て，2014 年よりフリーランスのソフトウェアエンジニア．2016 年合同会社 Serverspec Operations 設立．2019 年帝京大学理工学部情報科学科通信教育課程卒業．同年よりさくらインターネット株式会社さくらインターネット研究所客員研究員．インターネットサービスの運用技術に興味を持つ．

栗林 健太郎 （正会員）

1999 年東京都立大学法学部政治学科卒業．2002 年奄美市役所入所，2008 年株式会社はてな入社を経て，2012 年株式会社 paperboy&co.（現 GMO ペパボ株式会社）入社．2016 年よりペパボ研究所所長，2017 年より GMO ペパボ株式会社取締役 CTO．インターネットサービスの開発，技術経営等に興味を持つ．人工知能学会会員．

松本 亮介 （正会員）

2015 年京都大学大学院情報学研究科博士課単位取得認定退学．同年 GMO ペパボ株式会社入社を経て．2018 年よりさくらインターネット株式会社さくらインターネット研究所上級研究員．京都大学博士（情報学）．OS やサーバソフトウェア，インターネットの運用技術やセキュリティ等に興味を持つ．IEEE，ACM 各会員．