監訳者まえがき

Railsの進化は止まることなく続き、さまざまな機能が追加・改良されてきました。Railsでコードを書いた後にはサーバのセットアップとデプロイ（作成したアプリケーションをサーバで実際に使えるようにすること）が待っていますが、Railsが登場した当初、アプリケーションが動作する環境のセットアップは難しいものでした。Railsを解説する書籍は何冊も出版されましたが、たいていの場合、セットアップの説明に割いてあるのは数ページ程度です。これまで、他にWebアプリケーションの開発を行ってきたバックグラウンドがあればアプリケーションを動作させるために何をすればいいのかある程度想像もつき、知りたい情報はその数ページで十分なこともあるでしょう。しかし、RailsからWebアプリケーションの開発を始めた人はデプロイについてももっといろいろ知りたいと思ったことでしょう。

そんな読者のため、本書ではRailsアプリケーションのデプロイやサーバのセットアップを中心に説明しています。また、バージョン管理システムを利用したアプリケーションの管理方法やアクセス数が増えてきた場合の対応方法、パフォーマンスについてもひととおり説明されています。Linuxを対象とした説明だけでなく、WindowsでのRailsアプリケーションの セットアップについてもひととおり説明されています。Windowsで作業を行いたい場合の参考になるでしょう。

Railsアプリケーションを初めて作った読者の次のステップとして本書をお勧めします。きっとあなたのアプリケーションをよりよいものにするための手助けをしてくれることでしょう。うまくいったら、そのRailsアプリケーションを世界中に公開することをお勧めします。きっとさまざまな反応をもらえるはずです。

それでは始めましょう。

ネットワーク応用通信研究所
前田 修吾、橋本 将、小倉 正充

賞賛の声†

“*Deploying Rails Applications*”はすばらしく、そしてきわめて重要な 1 冊です。本当にありがとう！

—— Stan Kaufman
（The Epimetrics Group LLC 社，プリンシパル）

私は VPS（Virtual Private Server、仮想的な専用サーバ）のセットアップに関する章を読み、1 インスタンス当たりわずか 30 分足らずで 3 つの VPS インスタンスを立ち上げることに成功しました。本書のおかげでサーバの準備にかかる手間を何日分も軽減でき、コードの作成に集中できました。また、本書は私が今までに読んだ中で最も優れた Capistrano のチュートリアルでもあります。私にとって Capistrano はもはや謎の呪文ではなく、デプロイのためのカスタムタスクも自作できるようになりました。最終版が待ち遠しい！

—— Barry Ezell
（Balance Engines LLC 社，CTO）

本書を購入するまでは、私はこの種の情報を求めて何時間も Web を探し回っていました。本書には求めていた情報がすべて（しかも正確に）収められており、Rails のプロジェクトを成功裏に稼働させる上で非常に役立ってくれました。どうもありがとう！

—— Eric Kramer
（Nationwide Children's Hospital，プログラマ）

† 訳注：原書の刊行に先立って公開された、本書のベータ版に対する識者の反響です。

まえがき

Railsアプリケーションの開発は楽しいのものであるはずです。しかし、ここで筆者は読者に告白しなければなりません。昔、筆者はRailsが大嫌いだった時期がありました。

当時の筆者はまず、Railsアプリケーションを作るための簡単なチュートリアルを読みました。ヘルパやプラグインそしてジェネレータを使うと、自分が書かなければならないコードの量を大幅に減らせることを学びました。ファイルは一定の論理的関係や階層構造に従っており、必要なものをすぐに見つけることができました。そしてActive Recordのドメイン固有言語は筆者のアイデアをシンプルかつ強力な方法で表現させてくれました。このようにRailsフレームワークは筆者に対してとても従順であり、ささいなミスを犯すこともありましたが簡単にアプリケーションを完成させることができました。それは喜びに満ちあふれた経験でした。

しかし、完成させたアプリケーションはデプロイしなければなりません。デプロイ（配備と訳されることもあります）とは、アプリケーションを開発環境から実運用環境に移し、ユーザが実際にアクセスできるようにすることを意味します。Webアプリケーションをデプロイするには、デプロイ先のホストを選択し、Webサーバやデータベースをセットアップし、作成したすべてのファイルを適切な場所に置き、適切なアクセス権を設定する必要があります。

開発はとても楽しかったのですが、デプロイは退屈でつまらないということはすぐに明らかになりました。開発中に味わった幸福感は、延々と続くサーバのクラッシュログ、Railsのエラーページ、つまらないインストールスクリプトなどによってすっかり台なしにされてしまいました。解決策を求めてRailsに関するWikiやブログ、書籍などを何時間も探し回ったのですが、いずれも筆者が必要としていることのほんの一部分しか提供してはくれませんでした。しかも多くの情報は矛盾しており、まったくの誤りだったこともありました。そしてアプリケーションを提供する「家」を定めたら、今度はユーザに最善の環境を提供しなければなりません。このための努力もデプロイのプロセスに含まれます。そしてもちろんここでも筆者はつまづきました。サイトが正しく機能するところまでこぎ着けたものの、実際にアクセスすると反応が遅すぎて使い物になりませんでした。苦労してページキャッシュを導入しましたが、大した改善は見られず、ユーザはいらいらしながらブラウザ上の回るアイコンを見続けることになりました。メモリリークやマイグレーションの失敗、つまらないサーバ設定のミスなどと格闘し、ようやく筆者のサイトはまともに使えるようになりました。そしてこのサイトはある程度の成功を収め、ユーザも増えてきたのですが、するとまた新た

な種類のエラーが現れてきました。筆者は大声で叫びたい衝動に駆られました。当時の筆者は本当にRailsが嫌いでした。

筆者はRailsを美化するつもりはありません。自分が何をしようとしているのか分かっていないと、Railsでのデプロイは根気や忍耐力の訓練にしかならないでしょう。しかも、Railsでの開発が容易な分野に限って、デプロイは難しいことが多いのです。いくつか例を挙げてみましょう。

- Railsを使った開発に関するドキュメントは多数存在し、簡単に見つけることができます。しかしデプロイに関しては、必要な情報のうちほんの一部分しか見つけられないことが多いでしょう。人はデプロイよりも開発に関するドキュメントを書くことを好むようです。
- 開発環境は自分で選ぶことができますが、デプロイ先の環境は自分で選べるとは限りません。Railsに対応しているホストの多くではLinux系のシステムが使われていますが、FreeBSDやSolarisが使われていることもあります。そしてそれぞれのホストに用意されているソフトウェアスタックは千差万別であり、アプリケーションに対する制約条件もさまざまです。
- 開発環境で動作しているアプリケーションがエラーを起こした場合、ブレークポイントや開発者向けログ、コンソールなどを通じてエラーに関する情報をいくらでも集めることができます。一方、実運用環境ではそうはいきません。得られる情報は限られており、ユーザは多数存在し、不確定要素も増えます。オペレーティングシステムやアプリケーションサーバ、システムリソース、プラグイン、データベースなど、エラーはあらゆるところから発生する可能性があります。そしてキャッシュの働き方は開発環境と実運用環境では異なります。
- Railsを使った開発では、ユーザが選択しなければならないことはさほど多くありません。ほとんどの場合、データの永続化にはActive Recordを使い、コントローラやビューにはAction Packが使われます。そしてAjaxにはscript.aculo.usやPrototypeが使われることでしょう。しかし、デプロイの際にはRailsで規定されていないさまざまなことを自分で決めなければなりません。最も基本となるWebサーバの選択さえ、自分で行う必要があります。

しかしこのような面倒の数々も、適切な学習によって克服できると筆者は断言します。時が経つにつれて、以前の筆者が取ってきたアプローチは性急すぎ、計画性にやや欠けるものだったということが分かってきました。デプロイには開発時と同じ心構えが必要です。それぞれのステップについて適切に計画を立て、それに基づいて作業を行うとともに、可能な限り処理を自動化して不確定要素を減らす必要があります。また、起こり得る問題を想定しておけば、その発生を事前に予測でき、問題が発生しても早期に状況を把握できるようになります。筆者の勤務するEngine Yard社では世界で最も大規模で有名なRailsサイトを運営しており、ここで得られたノウハウを読者と共有したいと思います。

Railsは新しいプラットフォームであり、高性能でスケーラブルかつ安定性の高いアプリケーションなど作れるのかと懐疑的に考える人もいます。ここで筆者のEngine Yardでの経験に基づき、いくつかの誤解を解いてみたいと思います。

Ruby on Rails を使った開発のためのフレームワークは、デプロイのためのフレームワークよりもはるかに進歩的である

これは誤解です。Rails のデプロイツールが注目を集めることはあまりありませんが、さまざまなツールが存在し、機能も強化され続けています。きちんと探しさえすれば、高い評価を得ている効率的なデプロイツールが無料で利用できることに気付くでしょう。これらのツールは、最も進歩的と言われる Java や C# 向けのデプロイツールと比べて一歩も引けを取らない機能を備えています。このようなツールを使えば管理者はコマンド 1 つで Rails アプリケーションをデプロイでき、問題が発生した場合は同様にコマンド 1 つで元のバージョンに戻せます。デプロイ対象のサーバが 1 台でも複数台でも、必要な変更はほとんどありません。今まで手作業でファイルをコピーしていたり rsync コマンドを使って複数台のサーバにファイルを転送していたユーザはきっと驚き、とても簡単になったデプロイに喜ぶことでしょう（もちろん、従来の方法を使い続けてもかまいませんが）。デプロイツールについては「4 章　Capistrano」で詳しく紹介します。

Rails はまだ歴史が浅く、大規模で高性能なアプリケーションの稼働実績はまだない

これも誤解です。Ruby on Rails は複数台のサーバで構成されるような大規模なサイトでも使われています。このようなサイトでは専用のサーバが多数用意され、ユーザに強力な機能を提供しています。大規模な Rails サイトは増加を続けており、例えば Twitter、Basecamp、43 Things はいずれも Rails を使って構築されています。これら以外にも多くのサイトが続々とサービスを開始しています。

Ruby という言語は型付けの機構を持たないため、Web アプリケーションにはまったく適さない

これもほとんど誤解です。Ruby は動的に型付けされるインタプリタ型言語であり、大量のアクセスが集中する実運用環境での利用には適さないと考えられていますが、その問題点とそれに伴うリスクの多くは Rails フレームワークによって軽減されています。また、Rails のキャッシュ機構やベンチマーキングツールは高性能なサイトの構築を助けてくれます。Rails のテストフレームワーク（Ruby 自体のテストフレームワークと組み合わされることもあります）を使えば、静的に型付けされる言語を使った場合にコンパイラが検出するようなエラーをテスト時に発見できます。そして Rails アーキテクチャの shared-nothing（何も共有しない）という方針は他の多くの大規模 Web サイトと共通であり、ハードウェアの追加によって性能を向上できます。このようなクラスタリングについては「6 章　スケールアウト」で解説します。

Rails では、何をしようとしているのか理解せずに作業すると問題が起こりやすい

これは本当です。深刻なトラブルを避けたいなら、自分で選んだツールの使いこなし方を学ぶようにしましょう。また、どんなプログラミング言語を使った開発でも、アプリケーションの設計には誤りが付き物です。そして誤って設計されたアプリケーションにはさまざまな問題が発生します。理論武装し、自らを守りましょう。本書がその助けとなることを願っています。

脅しめいたことを言ってしまいましたが、Rails を使ったデプロイは悪いものではありません。きっと、アプリケーションを信頼性の高い方法でデプロイでき、その結果をあらかじめ予期できるようになることでしょう。そして反復的なアプローチに基づけば、システムがどの程度の性能を発

揮するか理解できるでしょう。また、知識と時間そして忍耐力があれば、システムの安定性とスケーラビリティを向上できるはずです。さっそく始めましょう。まず、どのようなWebアプリケーションのデプロイでも共通のロードマップ（地図）について俯瞰してみましょう。

背景

新たなWebアプリケーションの分類を表すWeb 2.0という言葉が流行していますが、これはインターネットに対する考え方を変革するような新技術の集合体であるとも言えます。しかしデプロイという観点では、Web 2.0によって変わったことはほとんどありません。

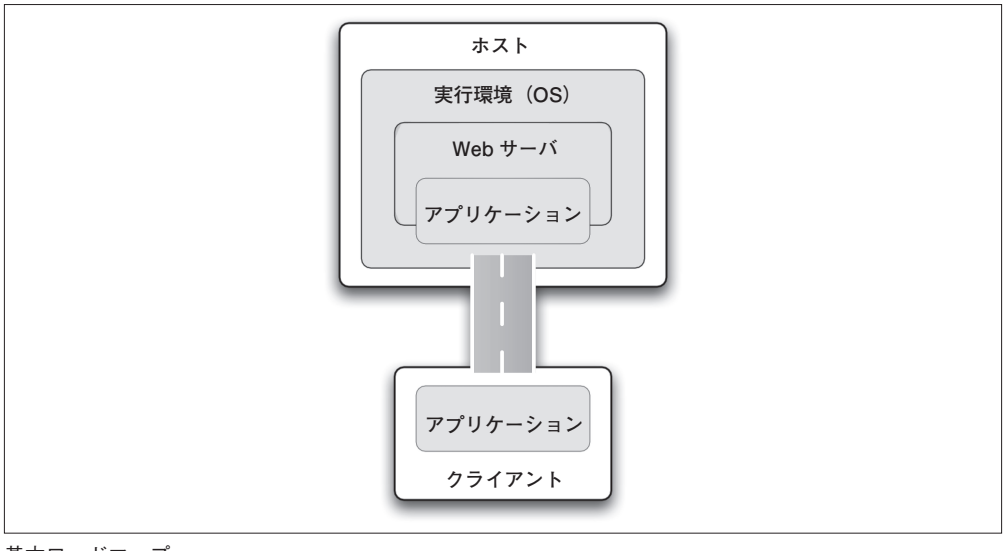
- インターネット上で使われるプロトコル（通信規約）やサーバは以前と変わりません。
- 以前と同様に、クラスタリングによってスケーラビリティを向上できます。
- 今まで使ってきたサーバを引き続き利用でき、新しく導入したサーバも同様に機能するでしょう。
- ソースコードは同じリポジトリに置いておけます。
- オペレーティングシステムについても変更はありません。主にUnixベースのものが使われ続けるでしょう。

Web 2.0によってどれほどアプリケーションが変わったとしても、デプロイの手順は以前とまったく変わりません。インターネットをロードマップにたとえながら、説明を進めていくことにします。

ここではサーバやクライアント側のブラウザ、ルータ、ファイアウォール、ロードバランサを建物に見立て、これらをつなぐネットワークやプロトコルを道路にたとえます。結局のところ、インターネットはデータを別の場所に移動させるための仕組みであり、地図のたとえが適しているのではと筆者は考えています。そしてデプロイの際には、インターネットを使ってアプリケーションを適切な場所に移動させる必要があります。本書で紹介するすべてのデプロイのシナリオは、この地図のたとえを使って表現できます。最も一般的で単純化したものが次ページの図です。

ここに描かれているそれぞれの要素について見てみましょう。まず、ホスト上に実行環境が用意されています。実行環境はアプリケーションを提供するという重要な役割を果たしており、本書のコンテンツの大部分は実行環境の構築に割かれています。ここでの実行環境には、Webアプリケーションが必要とするさまざまなコンポーネントがすべて含まれます。これらのコンポーネントは自分で用意することも、他のベンダに用意してもらうこともでき、本書ではそれぞれの方法を紹介します。例えばオペレーティングシステム、Ruby言語のランタイム、Railsフレームワーク、そしてこれらをつなぐさまざまなコンポーネントが実行環境に含まれます。ホストとは、ユーザがWebアプリケーションによるサービスを求めて訪れる場所です。本書を読み進めてゆくにつれて、ホスト上にさまざまな要素が追加されてゆくことになります。

この図には開発のためのクライアントも描かれています。筆者はMacBook Proを信頼して日常的に使っていますが、Windowsプラットフォーム上で開発を行ったこともあります。以降の本文でも、本書はクライアント上のアプリケーションからサーバに通じるロードマップについて解説して



基本ロードマップ

ゆきます。図示した基本的なロードマップでは、アプリケーションをサーバに転送するための手段としては単純な FTP が使われます。

ただし、デプロイが図のように単純なものであることはほとんどありません。共有ホストを利用している場合、我々が制御できる要素はこれよりも限られていることに読者はもうすぐ気付くことでしょう。古くから使われてきた FTP はシンプルではあるけれど、サイトを効率的に管理したいという要求には応えてくれません。デプロイにはもっと優れたツールが必要です。ソースコード管理のリポジトリも導入したくなることでしょう。また、アクセス数が増えてくれば Web サーバが 1 つでは足りないということがあるかもしれません。最終的には、「6.2　クラスタリングによるスケールアウト」で示すように高度なロードマップが必要になるでしょう。

このように複雑なロードマップでは、デプロイのシナリオも大きく異なります。ユーザにとっては単に 1 つのサイトとしか見えないものも、実際にはそれぞれ異なる環境下で動作しています。まず、Web サイトが異なります。1 つのサイトが 1 つのホストで構成されるとは限りません。複数のホストがすべて 1 つのマシン上の仮想環境で動作することもあり、すべてのホストがそれぞれ異なるハードウェア上で動作することもあります。デプロイのシナリオによっては、アプリケーションをすべてのホスト上の Rails 環境にインストールしなければならず、そしてそれぞれの環境が協力して動作するようにしなければなりません。これが本書の主題です。

最初の方の章では、例が単純すぎると感じられるかもしれませんが、心配する必要はありません。巻末まで読み進めるにつれ、FTP や共有ホストは使われなくなってゆきます。順次新しいロードマップが紹介され、そのたびに少しずつ内容が追加されてゆきます。ロードマップの拡張は巻末まで続きます。それぞれの章で行われる拡張について、次の節で簡単に紹介することにします。

デプロイと家探し

本書で紹介するロードマップは一貫したテーマに基づいています。それぞれのロードマップは、読者が開発したコードを最善の方法で最終的な「家」へと移すことを目指しています。アプリケーションをどこに置きたいかによって、ロードマップの種類は異なります。これを把握できない限りデプロイについて適切な理解を得ることはできませんが、適切なデプロイ先のプラットフォームを探すのは容易ではありません。あたかも、不動産業者やインターネットあるいは雑誌などの助けを一切借りずに家を探すようなものです。本書は読者の助けとなることを目指し、Rails アプリケーションのデプロイという未知の宇宙を案内してゆきたいと考えています。具体的には、本書では以下のような点について学ぶことができます。

- スムーズなデプロイを念頭に置いた Rails アプリケーションの開発。
- 共有ホスト、VPS（Virtual Private Server、仮想的な専用サーバ）、専用サーバの中から適切なものを選択。
- 効率的なデプロイのためのソフトウェアに対する理解。
- Web サーバやその他のサービスのビルドと設定。
- 実際のユーザがアプリケーションに負荷をかける前に、人工的に負荷を再現する方法。
- キャッシュなどの高度な手法を用いて、実運用環境での性能やスケーラビリティを改善する方法。

本書を通じて、筆者はデプロイを家の購入にたとえてゆきたいと考えています。それぞれの章で家を選択あるいは準備し、毎日の生活のために環境を整え、時にはよりリッチな近所の家に引っ越すこともあるでしょう。具体的には以下のような点について解説していきます。

荷造り（Rails の世界ではアプリケーションの準備）

引っ越しのためには荷造りが必要です。そして引っ越しをスムーズに済ませるには、荷物を適切にまとめておかなければなりません。これと同様に、Rails アプリケーションではアプリケーションを適切に手入れしておく必要があります。例えばソースコード管理を導入したり、実運用のアプリケーションに大きな影響を与える決定（マイグレーションの構成やセキュリティへの考慮など）を 1 章で行います。ロードマップにはソースコード管理が追加されるでしょう。

初めての部屋探し（共有ホスト）

誰もが一軒家を購入できるわけではありません。家を出て独立する場合、多くの人々はまずアパートや寮などに引っ越すのではないでしょうか。Rails でも、ブログやフォトログといったシンプルなアプリケーションのデプロイ先としてはまず共有ホストが利用されるでしょう。アプリケーションのホスティング形式にはいくつか選択肢がありますが、まずここで紹介する共有ホストが最も安価です。共有ホスト向けのアプリケーションのセットアップは初めてのアパート探しに似ており、まず条件に合った物件を探し、転居届を提出し、そして部屋をカスタマイズしていくことになります。また、共有ホストにはアパート暮らしと同様の長所と短所もあります。共有ホストは安上がりですが、居住者のルールは守らなければならず、一方でルールを守らない隣人ともうまくやっていかなければなりません。2 章では、初めての「家」を見つけ

てそれを活用するための方法を学びます。ここでのデプロイは単純であり、共有ホストと簡単なアプリケーション、そしてファイルを共有ホスト上に置くための簡単な仕組み（FTP など）があれば十分です。

グレードアップ（仮想ホストと専用ホスト）

アパート暮らしに慣れてくると、自分だけの一軒家やコンドミニアムが欲しくなってきます。Rails アプリケーションの世界でも事情は似ており、共有ホストでは不十分だという場合には仮想ホストや専用ホストに移転できます。自分の家を持つと、今までなかった利点を享受でき、同時に新たな責任も負うことになります。家具などは自由に追加できますが、トイレの修理は自分でしなければならず、庭の芝刈りもしなければなりません。仮想ホストや専用ホストも一軒家のようなもので、共有ホストより安定していますが同時に多くの知識が求められ、責任も重大です。自分だけのためにホストを用意するなら、地主としての心構えが求められます。Web サーバから Rails 環境に至るまでのソフトウェアスタックを、自分でビルドし設定しなければなりません。3 章ではホスト環境の構築について順を追って見てゆきます。環境構築を行わなければならないためロードマップは少し複雑になりますが、他の部分については変化はありません。

搬入（Capistrano）

引っ越すべき場所を決めて準備を済ませたら、いよいよ荷物を運び込みます。ただし家具や身の回りの道具と異なり、Rails では搬入作業が何度も発生するのが普通です。そこで、可能なことはすべて自動化し、面倒な作業ができるだけ発生しないようにしましょう。このために広く使われているデプロイツールが Capistrano です。4 章では、あらかじめ指定された手順に基づいたデプロイをコマンド 1 つで実行する方法についてまず紹介します。そして問題が発生した際に元のバージョンに戻す方法や、Capistrano の挙動をカスタマイズするための手法などについても学びます。ロードマップ上では、デプロイ先と開発環境との間により便利な道路が引かれることになります。

増築（プロキシと負荷分散）

家が手狭になってきたら、増築するかより広い家に引っ越す必要があります。後者については 3 章で解説しています。6 章では前者すなわちクラスタリングについて解説します。まず、大金をかけずにデプロイ形式を改善する方法として、静的なコンテンツとアプリケーションによる動的なコンテンツとでサービスを分割するというものがあります。ここでは実運用環境の設定を変更し、より多くの負荷に耐えられるようにします。Apache あるいは nginx という Web サーバが静的なコンテンツを提供し、Mongrel が動的なコンテンツを提供するようにするための手順を紹介します。また、負荷分散のための簡単なクラスタを構成し、それぞれのサーバにアプリケーションを配布する方法も学びます。必要になるであろうデータベースについてのデプロイについても解説します。サーバ側が単一のホストではなくクラスタになるため、ロードマップは非常に複雑なものになります。しかし Capistrano があれば、クライアント側についての変更はまったく必要ありません。

将来の計画（ベンチマーキング）

年月が経つと家族の構成は変化します。この変化を考慮しないと、わずか数年以内に家が家族

全員を収容できなくなってしまうかもしれません。Rails やその他のインターネット環境では、ユーザ数が急に数百倍にも増加することがあるため、変化を見越した計画が非常に重要です。正しく計画を立てるためにはベンチマーキングが必要です。ソフトウェアスタックを選択してアプリケーションをデプロイしたら、どの程度の負荷に耐えられるかテストするべきです。8章では Ruby による基本的なツールやその他のツールを使い、アプリケーションの能力を把握するための方法を解説します。ボトルネックが発見された場合にそれを解消するための方法についてもいくつか紹介します。ここではロードマップに変更はありません。

物事の管理（モニタリング）

家に住んでいると、さまざまなものを管理しなければなりません。番犬を飼って家の出入りを見張らせたり、警備会社と契約して管理を代行してもらう必要があるかもしれません。Rails での他の設定項目と同様に、このような管理の作業は自分で行うことも可能です。しかし Monit というアプリケーションを使えば、システムにエラーが発生したり発生しようとしている場合に自動的に通知してもらうようにできます。これに伴うロードマップへの変更はわずかです。

窓掃除（Windows 環境へのデプロイ）

人は窓掃除を嫌うものです。そして Rails の開発者も窓（Windows）を好みません。しかし場合によっては、選択の余地が残されていないこともあります。やむを得ず Windows 環境でデプロイを行うという場合には、7章での解説が役に立つでしょう。なるべくシンプルで、問題が起こりにくいよう記述には配慮したつもりです。ホスト側に注目し、Windows 上での環境構築という選択肢を提供します。

本書をすべて読むと、読者は自分に最も適したプラットフォームを選択できるようになるでしょう。Capistrano を使ってアプリケーションを開発マシンからターゲットの環境に移送するための技巧を学び、さまざまなデプロイのシナリオについてくまなく設定を行えるようになるでしょう。以前にデプロイの問題で Rails に悪い感情を抱いたことがあったとしても、本書があれば Rails を楽しむという本来あるべき状態へと戻ることができるでしょう。

本書の表記

本書では、以下の表記を使用しています。

太字（Bold）

重要な用語を示します。

等幅（Constant Width）

サンプルコード、コマンド、変数、属性、関数、クラス、名前空間、メソッド、モジュール、値、ファイルの内容、コマンドの出力、ファイル名、ディレクトリ名、URL などを示します。

等幅の太字（Constant Width Bold）

コードの重要な部分と、そのとおりに打ち込まなければならないコマンドやテキストを示します。

Joeの疑問

●●●って何？

このコラムには、よくある質問とその答えが記載されています。

本書中の至るところで、コマンドラインを使った操作手順を目にすることになると思います。コマンドラインはさまざまな方式のデプロイ、セットアップ、設定などに使われます。読者も適切なコマンドを適切な個所で入力しなければなりません。不注意によって開発マシン上のコードを消去してしまったり、開発環境向けのダミーのデータを実運用環境のデータベースに上書きしてしまうようなことは避けなければなりません。安全を期すために、コマンドラインで入力するデータの表記について以下のようなルールを定めることにします。

Unix 系のほとんどのシステムでは、コマンドラインのプロンプト文字がシャープ記号（#）なら現在 root ユーザとしてログインしていることを表します。通常ユーザであれば代わりにドル記号($)が使われます。これは bash（Bourne Again Shell）での慣習に基づいています。他のシェルでは別のプロンプト文字が使われることもあるので、それぞれのシェルでのルールに従ってください。本書で取り上げる Ubuntu システムでは、bash がデフォルトのシェルとして使われています。

以下のプロンプトは、本書で使われているさまざまなシェルコマンドを実行する際に表示されます。root としてログインしたら、下のようなプロンプトが表示されます。

```
root#
```

通常ユーザのアカウントでログインした場合は、次のようなプロンプトになります。

```
ezra$
```

サーバではなくローカルのマシンからコマンドを実行する場合は、下のプロンプトに続けてコマンドが表記されます。

```
local$
```

バージョン情報とサンプルコード

本書では Rails 2.0.2 で作成したアプリケーションのデプロイについて説明しています。また、特に明記しない場合、デプロイツールは Capistrano 2.0、Web サーバは Nginx 0.5.33、Apache 2.2、Mongrel 1.0.1、データーベースサーバは MySQL 5.0 について説明しています。新しいバージョンには古いバージョンにない機能があるため異なるバージョンではサンプルが動作しないことがあります。本書のサンプルコードは以下のサイトからダウンロードできます。

```
http://www.pragprog.com/titles/fr_deploy/source_code
```

意見と質問

本書（日本語翻訳版）の内容については、最大限の努力をもって検証および確認していますが、誤りや不正確な点、誤解や混乱を招くような表現、単純な誤植に気付かれることもあるでしょう。本書を読んで気付いたことは、今後の版で改善できるように知らせていただければ幸いです。将来の改訂に関する提案なども歓迎します。

株式会社オライリー・ジャパン
〒160-0002　東京都新宿区坂町26番地27　インテリジェントプラザビル1F
電話　03-3356-5227
FAX　03-3356-5261
電子メール　japan@oreilly.co.jp

本書のWebページには、正誤表、サンプルコード、追加情報が掲載されています。以下のアドレスでアクセスできます。

http://www.oreilly.co.jp/books/9784873114002/
http://dra.shugo.net
http://oreilly.com/catalog/9780978739201/（原書）
http://www.pragprog.com/titles/fr_deploy/deploying-rails-applications（原書）

オライリーに関するその他の情報については、次のオライリーのWebサイトを参照してください。

http://www.oreilly.co.jp

謝辞

筆者一同より

Railsを私たちに与えてくれたことを、DHH（David Heinemeier Hansson）とRailsの開発コアチームに対してまず感謝します。彼らのさまざまな革新がなければ、Railsは今のようにはなっていなかったことでしょう。RailsとRubyの開発者コミュニティの皆様にも感謝します。CapistranoとMongrelの開発チームは、Railsでのデプロイのシナリオを進化させてくれました。彼らはとてもフレンドリでしかも筆者たちの助けになってくれただけでなく、筆者たちに知ることの喜びを与えてくれました。最後に、寛容なBrian HoganとGeoffrey Grosenbachによる本書への多大な貢献に感謝します。

Clinton Beginより

「約束するよ、本を書くのはこれで最後だ。」これは私が1冊目の本を書き終えた後に、妻Jenniferに告げた言葉です。まず、この約束を破ってしまった私を許してくれた妻に感謝しなければなりません。それだけでなく彼女は本書の執筆の最初から最後まで、自らを顧みることなくサポー

トし続けてくれました。本職の合間に執筆するということは、父親から家族と過ごす貴重な時間が奪われ、母親からただでさえ少ない休息時間がさらに奪われるということを意味しました。執筆作業を通じて得られた経験から、専業主婦というのは私が今までに得たどんな資格よりも過酷だということを痛感しました。また、私の息子 Cameron と Myles にも感謝します。彼らはいつも、私がどうあるべきかという点について私に教えてくれました。

家族以外では、私を本書の執筆に誘ってくれた Bruce と Ezra にも感謝したいと思います。デプロイはとても重要であるにもかかわらず、Rails に関する書籍のほとんどがうわべだけの説明に終始し、他のことで紙幅を尽きさせています。これに正面から取り組んだのが本書です。また、私のキャリアをスタートさせ、実運用環境に Rails アプリケーションをデプロイするというプロジェクトに取り組ませてくれた ThoughtWorks にも感謝します。ThoughtWorks には Alexey Verkhovsky と Jay Fields をはじめとして、Ruby と Rails に関するきわめて優秀な頭脳が集まっています。最後に、Dave Thomas にも感謝します。3 年前、私は生意気にも彼に「Ruby はほかの子供だましの言語と何が違うのですか」と尋ねたものでした。彼は返事する代わりに 1 冊の本を送ってくれました。彼の人柄を端的に表す出来事でした。

Bruce Tateより

以前は、Ruby on Rails でのデプロイに関する書籍を私が執筆しているとは想像もできませんでした。私はプログラミングを得意としており、大規模なインターネット関連のプロジェクトではプログラマあるいはマネージャとしてデプロイの作業は他人に任せてしまっていました。この私の不得意分野を埋めるべく、本書の執筆作業に私を招いてくれた Dave と Andy そして Ezra に感謝します。そして私の上司 Robert Tolmach にも感謝します。彼は私の親友の 1 人で、しかも私が執筆作業を行うことを認めるという大胆な賭けに打って出てくれました。彼は本書のために時間を割き、WellGood LLC や ChangingThePresent で私たちが学んだことを読者にフィードバックしてくれています。また、作業の最終段階で執筆者に加わったにもかかわらず、本書の中で最も重要な章を執筆してくれた Clinton Begin にも特別の感謝を送りたいと思います。彼は他の執筆者に新鮮な刺激を与え、生産性の向上に貢献してくれました。

ここまでの謝辞が不完全に思えるのは、書くことや新しいテクノロジに没頭してしまった私のために、家族があまりにも多くのものを犠牲にしてまで尽くしてくれたからです。書き物ばかりしていた私に耐えてくれた Maggie、Kayla そして Julia に感謝します。私が本を書いたのは今回が初めてではなく、きっと今回が最後でもないと思われます。彼女たちへの感謝はこれからも尽きることはないでしょう。

Ezra Zygmuntowiczより

本書の執筆作業を理解し協力してくれた妻 Regan に感謝します。週末も夕暮れも、彼女と過ごすべき時間を執筆に費やさなければなりませんでした。執筆の過程の中で、原稿を査読あるいは批評してくださったすべての皆様にも感謝します。また、すばらしい執筆者たちによる寄稿にも感謝します。彼らの助力がなければ本書は完成しませんでした。Francois Beausoleil は原稿をまとめる際に、Subversion に関して私に協力してくれました。執筆の初期段階できわめて重要な貢献を果たし

てくれた Geoffrey Grosenbach にも感謝します。

Brian Hoganより

まず、執筆に参加する機会を与えてくれた Ezra と、私がプログラミングに嫌気がさしていたころに Ruby を紹介してくれた Bruce に感謝します。彼らの助けと導きがなければ今日の私はありませんでした。Zed Shaw の名前もここに記すに値します。彼は私に、Windows 環境でのデプロイというやりがいに満ちた課題を与えてくれました。Mongrel を Windows サービス化し、私の作業の手間を大幅に軽減してくれた Luis Laverna は私の中のヒーローです。

本書の執筆中もそうでないときも、妻 Carissa の私への絶えることない忍耐はすばらしいものでした。Ana と Lisa も娘としてがんばってくれました。また、父と母はどれほど困難だと思えることでも一生懸命取り組み、決してあきらめてはならないということを私に教えてくれました。素敵な家族に恵まれた私は果報者です。

最後に、私を助けてくれた最高の仲間たち Erich Tesky、Adam Ludwig、Mike Weber、Chris Warren、Chris Johnson そして Josh Swan に感謝します。