さあ Heroku をはじめよう

Ruby 編

HEROKU

1. はじめに

このチュートリアルでは、Ruby アプリケーションを短時間でリリースするためのステップを説明します。
全体的なステップをご理解いただき、Heroku の活用にお役立てください。
このチュートリアルは、次を前提としています。

- 無料の Heroku アカウントをお持ちであること。
  (Herokuアカウントの登録方法は http://bit.ly/Heroku_SignUp_JP をご覧ください)
- Ruby 2.2.5 がローカルにインストールされていること (OS X、Windows、Linux 用の Ruby および Rails のインストールガイドを参照)
- Bundler がローカルにインストールされていること (`gem install bundler` を実行)

```
$ dir c:\openssl
bin   include   lib   ssl
```

次を実行します。

```
$ gem install puma -- --with-opt-dir=c:\openssl
```

完了したら、次を実行します。

```
$ bundle update puma
```

2. 設定

このステップでは、Heroku Command Line Interface（CLI）（旧 Heroku Toolbelt）をインストールします。この CLI を使用することで、ローカルでのアプリケーションの実行に加え、Heroku でのアプリケーションの管理および拡張、アドオンのプロビジョニング、アプリケーションの動作ログの表示が可能になります。

インストールが完了したら、コマンドシェルから`heroku`コマンドを使用できます。
Windows でコマンドシェルにアクセスするには、コマンドプロンプト（cmd.exe）または Powershell を起動します。
Heroku アカウントの作成時に使用したメールアドレスとパスワードを使用してログインします。

```
$ heroku login
Enter your Heroku credentials.
Email: ruby@example.com
Password:
...
```

heroku コマンドと git コマンドの両方を実行するためには、認証が必要です。
ローカルの開発環境がファイアウォールの内側にあり、外部の HTTP/HTTPS サービスに接続するためにプロキシを使用する必要がある場合は、`heroku`コマンドを実行する前に、ローカルの開発環境で環境変数`HTTP_PROXY`または`HTTPS_PROXY`を設定します。

3. アプリケーションの準備

このステップでは、リリース用のシンプルなアプリケーションを準備します。
サンプルアプリケーションのクローンを作成して Heroku にリリースできるコードのローカル版を準備するには、ローカルのコマンドシェルまたはターミナルで次のコマンドを実行します。

```
$ git clone https://github.com/heroku/ruby-getting-started.git
$ cd ruby-getting-started
```

これで、シンプルなアプリケーションおよび Ruby のパッケージマネージャー、Bundler で使用される`Gemfile`ファイルが含まれるアクティブな git リポジトリが準備できました。

4. アプリケーションのリリース

このステップでは、アプリケーションを Heroku にリリースします。
Heroku にアプリケーションを作成し、Heroku がソースコードを取得できるように準備します。

```
$ heroku create
Creating polar-inlet-4930... done, stack is cedar-14
http://polar-inlet-4930.herokuapp.com/ | https://git.heroku.com/polar-inlet-4930.git
Git remote heroku added
```

アプリケーションを作成すると、同時に git リモート（`heroku`）も作成され、この git リモートはローカルの git リポジトリに関連付けられます。
作成したアプリケーションに対し、Heroku がランダムな名前を生成します（今回は`polar-inlet-4930`）。
あるいは、パラメーターを渡してアプリケーション名を指定することもできます。
ここで、コードをリリースします。

```
$ git push heroku master
Fetching repository, done.
Counting objects: 10, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 876 bytes | 0 bytes/s, done.
Total 6 (delta 4), reused 0 (delta 0)

remote: -----> Ruby app detected
remote: -----> Compiling Ruby/Rails
remote: -----> Using Ruby version: ruby-2.2.5
remote: -----> Installing dependencies using 1.7.12
remote:        Running: bundle install --without development:test --path vendor/bundle --
binstubs vendor/bundle/bin -j4 --deployment
remote:        Fetching gem metadata from https://rubygems.org/.........
remote:        Fetching additional metadata from https://rubygems.org/..
remote:        Using rake 10.4.2
       ....
remote:        Bundle completed (38.43s)
```

```
remote:       Cleaning up the bundler cache.
remote: -----> Preparing app for Rails asset pipeline
remote:       Running: rake assets:precompile
remote: -----> Discovering process types
remote:       Procfile declares types -> web
remote:       Default types for Ruby  -> console, rake, worker
remote:
remote: -----> Compressing... done, 30.7MB
remote: -----> Launching... done, v16
remote:       https://pacific-river-8854.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To git@heroku.com:polar-inlet-4930.git
   2e435c5..035d5f5  master -> master
```

これで、アプリケーションがリリースされました。
アプリケーション名をもとに生成された URL でアプリケーションにアクセスします。
次の便利なショートカットを利用してアクセスすることもできます。

```
$ heroku open
```

5. ログの表示

Heroku では、すべてのアプリケーションおよび Heroku コンポーネントの出力ストリームからイベントを集約し、時系列に並べたイベントストリームとしてログを処理するため、すべてのイベントを単一のチャネルで確認できます。
ログ記録コマンドの 1 つ、`heroku logs --tail` を使用して、実行中のアプリケーションに関する情報を確認します。

```
$ heroku logs --tail
2014-07-07T11:42:26.829065+00:00 heroku[web.1]: Starting process with command
`bundle exec puma -C config/puma.rb`
2014-07-07T11:42:35.334415+00:00 app[web.1]: I, [2014-07-07T11:42:35.334301 #2]
 INFO -- : listening on addr=0.0.0.0:19146 fd=9
2014-07-07T11:42:35.707657+00:00 app[web.1]: I, [2014-07-07T11:42:35.707293 #5]
INFO -- : worker=0 ready
2014-07-07T11:42:35.772074+00:00 app[web.1]: I, [2014-07-07T11:42:35.771727 #11]
INFO -- : worker=2 ready
2014-07-07T11:42:35.767750+00:00 app[web.1]: I, [2014-07-07T11:42:35.764688 #2]
INFO -- : master process ready
2014-07-07T11:42:35.777268+00:00 app[web.1]: I, [2014-07-07T11:42:35.777006 #8]
INFO -- : worker=1 ready
2014-07-07T11:42:35.618291+00:00 heroku[web.1]: State changed from starting to up
```

ブラウザーでアプリケーションに再度アクセスすると、別のログメッセージが生成されて表示されます。
`Control+C` を押して、ログのストリーミングを終了します。

6. Procfile の定義

アプリケーションのルートディレクトリにあるテキストファイル、Procfile を使用して、アプリケーションを起動するときに実行するコマンドを明示的に宣言します。
今回の例でリリースしたアプリケーションの `Procfile` は、次のとおりです。

```
web: bundle exec puma -C config/puma.rb
```

これは 1 つのプロセスタイプ `web` と、その実行に必要なコマンドを宣言しています。ここで重要なのが `web` という名前です。このプロセスタイプは Heroku の HTTP ルーティングスタックにアタッチされ、リリース時に Web トラフィックを受信することを宣言しています。
ここで使用しているコマンドは、設定ファイルを指定して Puma（Web サーバー）を起動するためのものです。Procfile に追加のプロセスタイプを含めることもできます。たとえば、キューにあるアイテムをすべて処理するバックグラウンドワーカープロセスのプロセスタイプを宣言することもできます。

7. アプリケーションの拡張

ここまでの操作により、アプリケーションが 1 つの web dyno 上で動作しています。
dyno とは、Procfile に指定されたコマンドを実行する軽量のコンテナのようなものです。
動作している dyno の個数は、ps コマンドを使用して確認できます。

```
$ heroku ps
=== web (Free): `bundle exec puma -C config/puma.rb`
web.1: up 2015/05/12 11:28:21 (~ 4m ago)
```

デフォルトでは、アプリケーションは無料の dyno にリリースされます。無料の dyno は、30 分間操作がない場合（トラフィックの受信がまったくない場合）、スリープします。この状態になると、次のリクエスト時にスリープが解除されるまでに数秒の遅延が生じます。それに続くリクエストでは通常どおり動作します。
また、無料の dyno は、毎月アカウントレベルで割り当てられる無料 dyno 時間を消費します。
割り当てられた時間を使い切らない限り、すべての無料アプリケーションの動作が継続します。
dyno がスリープするのを防ぐため、『Dyno Types（dyno タイプ）』の記事に記載されている hobby または プロフェッショナル用途の dyno タイプにアップグレードすることができます。
たとえば、アプリケーションをプロ用途の dyno に移行すると、コマンドによって指定した数の dyno を起動し、それぞれで web プロセスタイプを実行するよう Heroku に命令することで、簡単に拡張できるようになります。Heroku でアプリケーションを拡張または縮小するには、動作している dyno の個数を変更します。web dyno の個数を 0 にするには次のようにします。

```
$ heroku ps:scale web=0
```

アプリケーションにアクセスし直すため、Web タブを更新するか、Web タブにアプリケーションを開く heroku open コマンドを実行すると、エラーメッセージが表示されます。
リクエストに応答できる web dyno が 1 つもなくなったためです。
再度拡張します。

```
$ heroku ps:scale web=1
```

不正使用を防止するため、アプリケーション内で有料の dyno を 2 つ以上 に拡張するためにはアカウントの認証が必要です。

8. アプリケーションの依存関係の宣言

Heroku では、ルートディレクトリに`Gemfile`ファイルがあると、アプリケーションが Ruby であると認識されます。
今回リリースしたデモアプリケーションには、すでに次のような`Gemfile`が含まれています。

```
source 'https://rubygems.org'
ruby '2.2.5'

# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.2.5'
# Use postgresql as the database for Active Record
gem 'pg'
gem 'rails_12factor', group: :production
# Use SCSS for stylesheets
gem 'sass-rails', '~> 4.0.3'
# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'
...
```

この`Gemfile`ファイルが、アプリケーションとともにインストールする必要がある依存関係を指定します。
また、Heroku でのアプリケーションの実行に使用される Ruby のバージョンを指定するためにも使用します。
アプリケーションのリリース時、Heroku はこのファイルを参照し、`bundle install`コマンドを用いて適切なバージョンの Ruby を依存関係とともにインストールします。
どのアプリケーションの場合でも、ローカルで実行するための前提条件は、依存関係も合わせてローカルにインストールされていることです。この特定の Gemfile には依存関係`pg`があり、これは Postgres がローカルにインストールされている場合にのみ解決します。先へ進む前に Postgres をインストールしてください。
アプリケーションをローカルで実行できるようシステムを準備するため、`bundle install`をローカルディレクトリで実行し、依存関係にあるファイルもインストールします。

```
$ bundle install
Using rake 10.4.2
Using i18n 0.7.0
Using json 1.8.2
....
Using puma 2.9.1
Your bundle is complete!
```

依存関係をインストールしたら、アプリケーションをローカルで実行する準備は完了です。

9. ローカルでのアプリケーションの実行

ローカルにある独自の開発環境でアプリケーションを実行するためには、いくらか作業が必要です。
Rails では通常、データベースが必要になります。このサンプルアプリケーションでは Postgres を使用します。
ローカルへのインストールについては、こちらの手順を参照してください。
この Rails アプリケーションもデータベースを使用するため、Rake タスクを使用して適切なデータベースおよびテーブルを作成する必要があります。

```
$ bundle exec rake db:create db:migrate
== 20140707111715 CreateWidgets: migrating ====================================
-- create_table(:widgets)
   -> 0.0076s
== 20140707111715 CreateWidgets: migrated (0.0077s) ===========================
```

Heroku CLI の一部としてインストールされた`heroku local`コマンドを使用してアプリケーションをローカルで起動します。

```
$ heroku local web
13:15:47 web.1  | started with pid 67489
13:15:47 web.1  | I, [2014-07-07T13:15:47.655153 #67489]  INFO -- : Refreshing Gem list
13:15:48 web.1  | I, [2014-07-07T13:15:48.495226 #67489]  INFO -- : listening on addr=
0.0.0.0:5000 fd=10
13:15:48 web.1  | I, [2014-07-07T13:15:48.621967 #67489]  INFO -- : master process ready
13:15:48 web.1  | I, [2014-07-07T13:15:48.624523 #67491]  INFO -- : worker=0 ready
13:15:48 web.1  | I, [2014-07-07T13:15:48.626285 #67492]  INFO -- : worker=1 ready
13:15:48 web.1  | I, [2014-07-07T13:15:48.627737 #67493]  INFO -- : worker=2 ready
```

Heroku での場合と同様、`heroku local`が`Procfile`を検証し、何を実行すべきかを特定します。
Web ブラウザーで http://localhost:5000 を開きます。アプリケーションがローカルで実行されていることを確認します。
ローカルでのアプリケーションの実行を停止するには、`CLI`で`Ctrl`+`C`を押して終了します。

10. ローカルで行った変更のプッシュ

このステップでは、アプリケーションに対してローカルで行った変更を Heroku に伝搬する方法を説明します。
この例では、依存関係とそれを使用するためのコードを追加するという変更をアプリケーションに加えます。
`Gemfile`を変更して`gem 'cowsay'`の行を追加することで、`cowsay`gem の依存関係を追加します。

```
ource 'https://rubygems.org'
ruby '2.2.5'

gem 'cowsay'

# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.2.5'
...
```

app/views/welcome/index.erb に対し、ファイルの最初の数行が次になるよう変更することで、
この gem が使用されるようにします。

```
<pre>
<%= Cowsay.say("Hello", "Cow") %>
</pre>
...
```

ローカルでテストします。

```
$ bundle install
$ heroku local
```

http://localhost:5000 でアプリケーションにアクセスすると、かわいらしい ASCII アートが表示されます。
これから、ローカルで行った変更を Heroku にリリースします。
Heroku へのリリースは、ほぼ毎回同じパターンで行います。
まず、変更したファイルをローカルの git リポジトリに追加します。

```
$ git add .
```

次に、このリポジトリに変更をコミットします。

```
$ git commit -m "Demo"
```

そして、先ほどと同様の方法でリリースします。

```
$ git push heroku master
```

最後に、すべてが正常に動作していることを確認します。

```
$ heroku open
```

11. アドオンのプロビジョニング

アドオンは、アプリケーションに標準の追加サービスを提供する、サードパーティのクラウドサービスです。情報の保持からログ記録、監視まで、さまざまな機能を提供します。
デフォルトでは、Heroku に保管できるアプリケーションログは 1,500 行です。ただし、サービスとして完全なログストリームも利用できます。また、アドオンプロバイダー数社からログ記録サービスが提供されており、ログの保持、検索、メールおよび SMS でのアラートといった機能を利用できます。
このステップでは、こうしたログ記録アドオンの 1 つ、Papertrail をプロビジョニングします。
Papertrail ログ記録アドオンをプロビジョニングします。

```
$ heroku addons:create papertrail
Adding papertrail on polar-inlet-4930... done, v11 (free)
Welcome to Papertrail. Questions and ideas are welcome (support@papertrailapp.com).
Happy logging!
Use `heroku addons:docs papertrail` to view documentation.
```

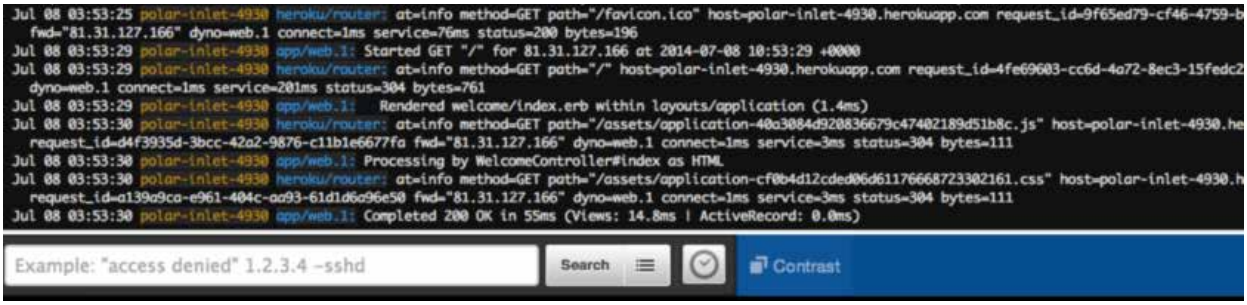
不正使用を防止するため、アドオンをプロビジョニングする際にはアカウントの認証が必要です。
アカウントの認証を行っていない場合は、認証サイトに転送されます。
これで、アドオンがアプリケーション用にリリースおよび設定されました。
次の方法で、アプリケーションのアドオンの一覧を確認できます。

```
$ heroku addons
```

この特定のアドオンの動作を確認するには、アプリケーションの Heroku URL に数回アクセスします。
アクセスするたびに新しいログメッセージが生成され、Papertrail アドオンに転送されるようになっています。ログメッセージを確認するには、Papertrail コンソールにアクセスします。

```
$ heroku addons:open papertrail
```

ブラウザーで Papertrail の Web コンソールが開き、最近のログイベントが表示されます。
このインターフェースから検索やアラートの設定を行えます。



12. コンソールの起動

アプリケーションの一部であるスクリプトやアプリケーションのコマンドは、`heroku run`コマンドを使用して one-off dyno で実行できます。このコマンドは、アプリケーション環境でのテストを目的として、ローカルのターミナルにアタッチされた REPL プロセスを起動する際にも使用できます。

```
$ heroku run rails console
Running `rails console` attached to terminal... up, run.1594
Loading production environment (Rails 4.2.5)
irb(main):001:0>
```

[`Error connecting to process`]というエラーが表示された場合、ファイアウォールの設定が必要である可能性があります。
コンソールが起動すると、アプリケーション全体が読み込まれます。
たとえば`puts Cowsay.say("hi","Cow")`と入力すると、動物が表示されて「hi」と言います。
`exit`を入力してコンソールを終了します。

```
irb(main):001:0> puts Cowsay.say("hi", "Cow")
 ____
| hi |
 ----
  \
   \
        .--.
       |o_o |
       |:_/ |
      //   \ \
     (|     | )
    /'\_   _/`\
    \___)=(___/
=> nil
irb(main):002:0> exit
```

dyno の動作をより詳しく理解するために、one-off dyno をもう 1 つ作成して、その dyno 上でシェルを開く`bash`コマンドを実行できます。その後、このシェルからコマンドを実行できます。
各 dyno には、アプリケーションおよび依存関係が取り込まれた一時的なファイルスペースがあります。
コマンドが完了すると (今回は`bash`)、この dyno は削除されます。

```
$ heroku run bash
Running `bash` attached to terminal... up, run.1421
~ $ ls
app  config db   Gemfile.lock  log   public   README.rdoc  tmp
bin  config.ru  Gemfile  lib       Procfile  Rakefile  test   vendor
~ $ exit
```

必ず exit を入力してシェルを閉じ、dyno を終了します。

13. 設定変数の定義

Heroku では、暗号化鍵や外部リソースのアドレスといったデータを設定変数に格納することで、設定を外部化することができます。
アプリケーションの実行時、設定変数は環境変数としてアプリケーションに公開されます。
たとえば、`app/views/welcome/index.erb`を変更して、環境変数`TIMES`の値の回数だけアクションを繰り返すよう変更します。最初の数行が次になるようファイルを変更します。

```
<% for i in 0..(ENV['TIMES'] ? ENV['TIMES'].to_i : 2) do %>
  <p>Hello World #<%= i %>!</p>
<% end %>
```

`heroku local`を実行すると、ローカルディレクトリにある`.env`ファイルの内容に応じて環境が自動的に設定されます。プロジェクトの最上位ディレクトリには、次の内容の`.env`ファイルがすでにあります。

```
TIMES=10
```

`heroku local`でアプリケーションを実行すると、「Hello World」が 10 回表示されます。
この設定変数を Heroku で設定するには、次を実行します。

```
$ heroku config:set TIMES=2
```

`heroku config`を使用して、設定された設定変数を確認します。

```
$ heroku config
== polar-inlet-4930 Config Vars
PAPERTRAIL_API_TOKEN: erdKhPeeeehIcdfY7ne
TIMES: 10
```

変更したアプリケーションを Heroku にリリースし、動作を確認します。

14. データベースの使用

アドオンのマーケットプレイスには、Redis や MongoDB から Postgres や MySQL まで、プロバイダーによって数多くのデータストアが用意されています。
このステップでは、Rails アプリケーションのリリースで自動的にプロビジョニングされる、無料の Heroku Postgres アドオンについて説明します。
データベースはアドオンであるため、CLI の `addons` コマンドを使用すれば、アプリケーション用にプロビジョニングされたデータベースについて追加の情報を確認できます。

```
$ heroku addons
=== polar-inlet-4930 Configured Add-ons
heroku-postgresql:hobby-dev  HEROKU_POSTGRESQL_BROWN
papertrail:choklad
```

アプリケーションの設定変数の一覧を表示すると、アプリケーションがデータベースへの接続に使用している URL、`DATABASE_URL` を確認できます。

```
$ heroku config
=== polar-inlet-4930 Config Vars
DATABASE_URL:            postgres://xx:yyy@host:5432/d8slm9t7b5mjnd
HEROKU_POSTGRESQL_BROWN_URL: postgres://xx:yyy@host:5432/d8slm9t7b5mjnd
```

Heroku では、より詳細な情報を表示できる `pg` コマンドも提供されています。

```
$ heroku pg
=== HEROKU_POSTGRESQL_BROWN_URL (DATABASE_URL)
Plan:       Hobby-dev
Status:     Available
Connections: 0
PG Version:  9.3.3
Created:    2014-07-07 11:30 UTC
Data Size:   6.6 MB
Tables:     2
Rows:       1/10000 (In compliance)
Fork/Follow: Unsupported
Rollback:   Unsupported
```

これは、Postgres 9.3.3 を実行し、hobby データベース（無料）に 1 行のデータが格納されていることを示しています。
今回の例でリリースしたアプリケーションには、すでにデータベース機能があり、ウィジェット用のコントローラーおよびデータベースモデルが用意されています。ウィジェットには、アプリケーションの URL に「/widgets」を追加することでアクセスできます。
今の状態で URL にアクセスすると、エラーページが表示されます。`heroku logs` を使用するか Papertrail を開いてエラーメッセージを確認すると、次のようなメッセージが確認できます。

```
2014-07-08T14:52:37.884178+00:00 app[web.1]: Started GET "/widgets" for 94.174.204.242
at 2014-07-08 14:52:37 +0000
2014-07-08T14:52:38.162312+00:00 heroku[router]: at=info method=GET path="/widgets"
host=fox828228.herokuapp.com request_id=3755bb46-4de2-4434-a13a-26ec73e53694
fwd="94.174.204.242" dyno=web.1 connect=0 service=294 status=500 bytes=955
2014-07-08T14:52:38.078295+00:00 app[web.1]: Processing by WidgetsController#index
as HTML
....
2014-07-08T14:52:38.146062+00:00 app[web.1]: PG::UndefinedTable: ERROR:
relation "widgets" does not exist
```

これは、データベースに接続することはできたものの、必要なテーブルが見つからなかったことを示しています。Rails では、`rake db:migrate` を実行することでこれを修正できます。このコマンドを Heroku で実行するには、次のように one-off dyno 上で実行します。

```
$ heroku run rake db:migrate
Running `rake db:migrate` attached to terminal... up, run.3559
Migrating to CreateWidgets (20140707111715)
== 20140707111715 CreateWidgets: migrating ====================================
-- create_table(:widgets)
   -> 0.0244s
== 20140707111715 CreateWidgets: migrated (0.0247s) ===========================
```

web プロセスタイプが dyno で実行される場合と同様に、この rake コマンドが実行されました。
Heroku が新しい dyno を起動し、準備されたアプリケーションを取り込み、このコンテキストでコマンドを実行して、最終的にこの dyno を削除します。この dyno は接続されたアドオンでアクションを実行するため、期待通りの動作となります。
これで、アプリケーションの `/widgets` ページに再度アクセスして、ウィジェットのレコード一覧を確認したり、レコードを作成できるようになりました。
Postgres がローカルにインストールされている場合は、データベースを直接操作することもできます。例として、`psql` を使用してデータベースに接続し、クエリを実行する方法を次に示します。

```
$ heroku pg:psql⇤

d8slm9t7b5mjnd=> select * from widgets;
 id |   name    | description  | stock |       created_at        |       updated_at
----+-----------+--------------+-------+-------------------------+-------------------------
  1 | My Widget | It's amazing |   100 | 2014-07-08 15:05:13.330566 | 2014-07-08 15:05:13.330566
(1 row)
```

Heroku PostgreSQL の詳細をご覧いただけます。

15. 次のステップ

ここまでで、アプリケーションのリリース、設定の変更、ログの表示、拡張、アドオンの追加の方法をお伝えしました。
ここで、おすすめの補足資料をご案内します。

- 『How Heroku Works (Heroku の仕組み)』では、アプリケーションの記述、設定、リリース、実行時に直面する概念の技術的な概要を説明しています。
- Ruby カテゴリでは、Ruby アプリケーションの開発とリリースについて詳しく説明しています。
- 『Getting Started with Rails 4.x on Heroku (Rails 4.x を使用した Heroku の使用開始)』および『Getting Started with Ruby on Heroku (Ruby を使用した Heroku の使用開始)』では、特定のタイプの Ruby アプリケーションのリリースについて詳しく説明しています。