さあ Heroku をはじめよう

Node.js 編

HEROKU

1. はじめに

このチュートリアルでは、Node.js 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、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)と、Node.js および npm がローカルにインストールされていることを前提としています。

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: zeke@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。
先へ進む前に、前提条件の項目がすべて適切にインストールされていることを確認します。次の各コマンドを入力し、インストールされているバージョンが表示されることを確認します (お持ちのバージョンがこの例とは異なる場合があります)。バージョンが表示されない場合、このチュートリアルの『はじめに』に戻り、前提条件の項目をインストールします。
次のローカル設定はすべて、『アプリケーションの依存関係の宣言』以降のステップを完了するために必要です。
このチュートリアルは、バージョン 4 以降の Node がインストールされていることを前提としています。

```
$ node -v
v5.9.1
```

`npm` は Node と一緒にインストールされます。インストールされていることを確認し、インストールされていない場合は、より新しいバージョンの Node をインストールします。

```
$ npm -v
3.7.3
```

`git` がインストールされていることを確認します。インストールされていない場合はインストールし、再度テストします。

```
$ git --version
git version 2.2.1
```

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/node-js-getting-started.git
$ cd node-js-getting-started
```

これで、シンプルなアプリケーションおよび Node のパッケージマネージャーで使用される `package.json` 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 sharp-rain-871... done, stack is cedar-14
http://sharp-rain-871.herokuapp.com/ | https://git.heroku.com/sharp-rain-871.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`sharp-rain-871`）。
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
Counting objects: 343, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (224/224), done.
Writing objects: 100% (250/250), 238.01 KiB, done.
Total 250 (delta 63), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Node.js app detected
remote:
remote: -----> Creating runtime environment
remote:
remote:       NPM_CONFIG_LOGLEVEL=error
remote:       NPM_CONFIG_PRODUCTION=true
remote:       NODE_MODULES_CACHE=true
remote:
remote: -----> Installing binaries
remote:       engines.node (package.json):  5.9.1
remote:       engines.npm (package.json):   unspecified (use default)
remote:
remote:       Downloading and installing node 5.9.1...
remote:       Using default npm version: 2.7.4
      ....
remote: -----> Build succeeded!
```

```
remote:       ├─── ejs@2.4.1
remote:       └─── express@4.13.3
remote:
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing... done, 9.4MB
remote: -----> Launching... done, v8
remote:        http://sharp-rain-871.herokuapp.com deployed to Heroku
To https://git.heroku.com/nameless-savannah-4829.git
 * [new branch]      master -> master
```

これで、アプリケーションがリリースされました。
アプリケーションのインスタンスを少なくとも 1 つ実行します。

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

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
2011-03-10T10:22:30-08:00 heroku[web.1]: State changed from created to starting
2011-03-10T10:22:32-08:00 heroku[web.1]: Running process with command: `node index.js`
2011-03-10T10:22:33-08:00 heroku[web.1]: Listening on 18320
2011-03-10T10:22:34-08:00 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: node index.js
```

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。
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): `node index.js`
web.1: up 2014/04/25 16:26:38 (~ 1s 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`package.json`ファイルがあると、アプリケーションが Node.js であると認識されます。
独自のアプリケーションの場合、`npm init --yes`を実行することでこのファイルを作成できます。
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`package.json`が含まれています。

```
{
  "name": "node-js-getting-started",
  "version": "0.2.5",
  ...
  "engines": {
    "node": "5.9.1"
  },
  "dependencies": {
    "ejs": "2.4.1",
    "express": "4.13.3"
  },
  ...
}
```

この`package.json`ファイルにより、Heroku でのアプリケーションの実行に使用される Node.js のバージョンと、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が決まります。
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て適切なバージョンの Node.js をインストールし、`npm install`コマンドを用いて依存関係をインストールします。
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、このコマンドをローカルディレクトリで実行し、依存関係をインストールします。

```
$ npm install
npm http GET https://registry.npmjs.org/express
npm http 304 https://registry.npmjs.org/express
npm http GET https://registry.npmjs.org/stream-spigot
npm http GET https://registry.npmjs.org/connect/2.12.0
...
express@4.13.3 node_modules/express
+---methods@0.1.0
+---merge-descriptors@0.0.1
+---range-parser@0.0.4
+---cookie-signature@1.0.1
+---debug@2.0.0
+---fresh@0.2.0
+---buffer-crc32@0.2.1
```

```
+---cookie@0.1.0
+---mkdirp@0.3.5
+---send@0.9.2 (destroy@1.0.3, ms@0.6.2, mime@1.2.11)
...
```

npm のバグが原因で、Windows では
[Error: ENOENT, stat `C:\Users\YOURUSERNAME\AppData\Roaming\npm`]といった
エラーメッセージが表示される場合があります。
この場合、`mkdir C:\Users\YOURUSERNAME\AppData\Roaming\npm`コマンドでこの
ディレクトリを手動で作成してから`npm install`を再度実行します。
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

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
14:39:04 web.1    | started with pid 24384
14:39:04 web.1    | Listening on 5000
```

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。
まずは、`package.json` に `cool-ascii-faces` の依存関係を追加します。次のコマンドを実行します。

```
$ npm install --save --save-exact cool-ascii-faces
cool-ascii-faces@1.3.3 node_modules/cool-ascii-faces
└─── stream-spigot@3.0.5 (xtend@4.0.1, readable-stream@1.1.13)
```

このモジュールが最初に `require` されるように、`index.js` を変更します。さらに、このモジュールを使用する新しいルート（`/cool`）を追加します。完成したコードは次のようになります。

```
var cool = require('cool-ascii-faces');
var express = require('express');
var app = express();

app.set('port', (process.env.PORT || 5000));

app.use(express.static(__dirname + '/public'));

// views is directory for all template files
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');

app.get('/', function(request, response) {
 response.render('pages/index')
});

app.get('/cool', function(request, response) {
 response.send(cool());
});

app.listen(app.get('port'), function() {
 console.log('Node app is running on port', app.get('port'));
});
```

ローカルでテストします。

```
$ npm install
$ heroku local
```

http://localhost:5000/cool でアプリケーションにアクセスすると、更新するたびにかわいらしい顔文字 (⊙ _ ⊙) が表示されます。
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 cool
```

もう 1 つ顔文字が表示されます。

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。
データ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を利用できるサービスもあります。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 sharp-rain-871... done, v4 (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。

```
May 13 14:43:03 jonwashere heroku/web.1: State changed from down to starting
May 13 14:43:05 jonwashere heroku/web.1: Starting process with command `node web.js`
May 13 14:43:07 jonwashere app/web.1: Listening on 26766
May 13 14:43:08 jonwashere heroku/web.1: State changed from starting to up
May 13 14:43:09 jonwashere heroku/router: at=info method=GET path=/ host=jonwashere.herokuapp.com
  request_id=f6ac74f1-68bf-4cb3-b363-3aa54e5b420f fwd="94.174.204.242" dyno=web.1 connect=2ms service=12ms
  status=200 bytes=191
May 13 14:43:09 jonwashere app/web.1: 10.236.149.233 - - [Tue, 13 May 2014 21:43:08 GMT] "GET / HTTP/1.1" 200
  13 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/34.0.1847.131 Safari/537.36"
May 13 14:43:29 jonwashere heroku/router: at=info method=GET path=/favicon.ico host=jonwashere.herokuapp.com
  request_id=51f36ddf-9b81-4f54-ae5f-f17573d30e4a fwd="94.174.204.242" dyno=web.1 connect=0ms service=6ms
  status=404 bytes=193
```

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 node
Running `node` attached to terminal... up, run.2132
Detected 512 MB available memory, 512 MB limit per process (WEB_MEMORY)
Recommending WEB_CONCURRENCY=1
>
```

[`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。
コンソールが起動した時点では、Node.js の標準ライブラリのみが読み込まれた状態です。
ここから`require`を使用して一部のアプリケーションファイルを読み込むことができます。
たとえば、次のコマンドを実行できます。

```
> var cool = require('cool-ascii-faces')
> cool()
( ʘ _ ʘ )
```

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連動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.3052
~ $ ls
Procfile  README.md  composer.json  composer.lock  vendor  views  web
~ $ exit
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、`index.js`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新しいルート、`/times`を追加します。

```
app.get('/times', function(request, response) {
  var result = ''
  var times = process.env.TIMES || 5
  for (i=0; i < times; i++)
    result += i + ' ';
 response.send(result);
});
```

`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=2
```

`heroku local`でアプリケーションを実行すると、毎回 2 つの数字が生成されて表示されます。
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
== sharp-rain-871 Config Vars
PAPERTRAIL_API_TOKEN: erdKhPeeeehIcdfY7ne
TIMES: 2
```

変更したアプリケーションを Heroku にリリースし、`heroku open times`を実行して開きます。

14. データベースのプロビジョニング

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、無料の Heroku Postgres Starter Tier の開発用データベースをアプリケーションに追加します。

```
$ heroku addons:create heroku-postgresql:hobby-dev
Adding heroku-postgresql:hobby-dev... done, v3 (free)
```

これにより、データベースが作成され、環境変数`DATABASE_URL`が設定されます（`heroku config`を実行することで確認可能）。
`package.json`ファイルを編集し、依存関係に pg npm モジュールを追加します。

```
"dependencies": {
  "pg": "4.x",
  "ejs": "2.3.3",
  "express": "4.13.3",
  "cool-ascii-faces": "1.3.3"
}
```

アプリケーションをローカルで実行するため、`npm install`を入力して新しいモジュールをインストールします。ここで、`index.js`ファイルを次のように編集して、このモジュールを使用して DATABASE_URL 環境変数に指定されたデータベースに接続するよう設定します。

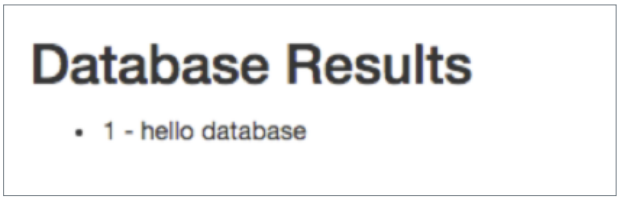
```
var pg = require('pg');

app.get('/db', function (request, response) {
 pg.connect(process.env.DATABASE_URL, function(err, client, done) {
  client.query('SELECT * FROM test_table', function(err, result) {
    done();
    if (err)
     { console.error(err); response.send("Error " + err); }
    else
     { response.render('pages/db', {results: result.rows} ); }
  });
 });
});
```

これで、`/db`ルートを使用してアプリケーションにアクセスすると、`test_table`テーブルにあるすべての行が返されるようになります。これを Heroku にリリースします。
`/db`にアクセスすると、データベースにテーブルがないためエラーが返されます。
Postgres がローカルにインストールされていることを前提として、`heroku pg:psql`コマンドを使用してリモートのデータベースに接続し、テーブルの作成および行の挿入を行います。

```
$ heroku pg:psql
psql (9.3.2, server 9.3.3)
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.
=> create table test_table (id integer, name text);
CREATE TABLE
=> insert into test_table values (1, 'hello database');
INSERT 0 1
=> \q
```

これで、アプリケーションの`/db`ルートにアクセスすると、次のように表示されるようになります。



Heroku PostgreSQL の詳細をご覧いただけます。
MongoDB および Redis アドオンのインストールにも、同様の技法を使用できます。

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。1 つ目は基礎をより一層固めるのに役立つ記事で、2 つ目はここ Dev Center のメインカテゴリ、Node.js のページです。

- 『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。
- 『Deploying Node.js Apps on Heroku（Heroku での Node.js アプリケーションのリリース）』では、既存の Node.js アプリケーションを Heroku に移植してリリースする方法を説明しています。
- Node.js カテゴリでは、Node.js 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。