さあ Heroku をはじめよう

PHP 編

HEROKU

1. はじめに

このチュートリアルでは、PHP アプリケーションを短時間でデプロイ(インターネットに公開)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)
- PHP がローカルにインストールされていること
- Composer がローカルにインストールされていること

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。Mac の場合はターミナルを起動します。
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: dz@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、PHP がインストールされていることを前提としています。インストールされていることを確認します。

```
$ php -v
PHP 7.0.5 (cli) (built: Apr 26 2016 04:39:48) ( NTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
```

`composer`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。

```
$ composer -V
Composer version 1.4.1 2017-03-10 09:29:45
```

`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 2.12.2
```

3. アプリケーションの準備

このステップでは、デプロイ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 にデプロイ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/php-getting-started.git
$ cd php-getting-started
```

これで、シンプルなアプリケーションおよび`composer.json`ファイルが含まれる git リポジトリが準備できました。必ず Composer をインストールしておきます。
Heroku では、PHP プロジェクトの依存関係の管理に Composer を使用します。
`composer.json`ファイルがあることで、アプリケーションが PHP で書かれていることを Heroku に示します。

4. アプリケーションのデプロイ

このステップでは、アプリケーションを Heroku にデプロイします。
まず、Heroku 上に開発アプリをデプロイする枠（この枠をアプリケーションと呼びます）を作成します。

```
$ 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割り当てます（今回は`sharp-rain-871`）。
オプションで名前を指定してアプリケーションを作成することも可能です。
ここで、ソースコードをデプロイします。

```
$ git push heroku master
remote: Building source:
remote:
remote: -----> PHP app detected
remote: -----> Bootstrapping...
remote: -----> Installing platform packages...
remote:       NOTICE: No runtime required in composer.json; requirements
remote:       from dependencies in composer.lock will be used for selection
remote:       - php (7.1.3)
remote:       - apache (2.4.20)
remote:       - nginx (1.8.1)
remote: -----> Installing dependencies...
remote:       Composer version 1.4.1 2017-03-10 09:29:45
remote:       Loading composer repositories with package information
remote:       Installing dependencies from lock file
remote:       Package operations: 12 installs, 0 updates, 0 removals
remote:         - Installing psr/log (1.0.2): Loading from cache
remote:         - Installing monolog/monolog (1.22.1): Loading from cache
...
```

```
remote:        - Installing symfony/twig-bridge (v3.2.7): Loading from cache
remote:       Generating optimized autoload files
remote: -----> Preparing runtime environment...
remote: -----> Checking for additional extensions to install...
remote: -----> Discovering process types
remote:       Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:       Done: 14.8M
remote: -----> Launching...
remote:       Released v17
remote:       https://gsphpjon.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/gsphpjon.git
 + 264e577...4f2369c master -> master (forced update)
```

これで、アプリケーションがデプロイされました。
次のコマンドで、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。
次のコマンドを使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ログで確認できます。
ログ表示コマンドの 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
014-05-27T11:03:21.033331+00:00 app[web.1]: Booting on port 28661...
2014-05-27T11:03:21.127050+00:00 app[web.1]: Using Apache2 configuration file
'vendor/heroku/heroku-buildpack-php/conf/apache2/heroku.conf'
2014-05-27T11:03:21.068192+00:00 app[web.1]: Using PHP configuration (php.ini) file
'vendor/heroku/heroku-buildpack-php/conf/php/php.ini'
...
2014-05-27T11:03:23.883157+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。

```
2014-05-30T13:18:13.581545+00:00 app[web.1]: [30-May-2014 13:18:13] WARNING:
[pool www] child 59 said into stderr: "[2014-05-30 13:18:13] myapp.DEBUG: logging output. [] []"
```

つまり、ログ記録は出力結果を`stdout`または`stderr`にリダイレクトしているだけです。
Heroku では、すべてのアプリケーションおよびシステムコンポーネントの出力結果が収集されます。
`web/index.php`ファイルを参照すると、出力結果を`stderr`に書き込むように`Monolog`サービスを設定する方法を確認できます。
`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。
今回の例でデプロイしたアプリケーションの`Procfile`は、次のとおりです。

```
web: vendor/bin/heroku-php-apache2 web/
```

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、デプロイ時に 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): `vendor/bin/heroku-php-apache2`
web.1: up 2014/05/27 12:03:23 (~ 11m ago)
```

デフォルトでは、アプリケーションは無料の dyno にデプロイされます。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定義

Heroku では、ルートディレクトリに`composer.json`ファイルがあると、アプリケーションが PHP であると認識されます。
今回デプロイしたデモアプリケーションには、すでに次のような`composer.json`が含まれています。

```
{
 "require" : {
  "silex/silex": "^2.0.4",
  "monolog/monolog": "^1.22",
  "twig/twig": "^2.0",
  "symfony/twig-bridge": "^3"
 },
 "require-dev": {
  "heroku/heroku-buildpack-php": "*"
 }
}
```

この`composer.json`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。
アプリケーションのデプロイ時、Heroku はこのファイルを参照し、`vendor`ディレクトリに適切な依存関係をインストールします。
PHP アプリケーションでは、シンプルな require の呼び出し後に、依存関係を使用できるようになります。

```
require('../vendor/autoload.php');
```

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、次のコマンドを実行して依存関係をインストールします。

```
$ composer update
Loading composer repositories with package information
Updating dependencies (including require-dev)
  - Installing psr/log (1.0.0)
    Loading from cache
...
Writing lock file
Generating autoload files
```

composer.json および composer.lock を毎回 git リポジトリにコミットしてください。
vendor ディレクトリは .gitignore ファイルに含めます。

9. 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（Cowsay ライブラリ）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、composer を使用して新しい依存関係を require します。

```
$ composer require alrik11es/cowsayphp
```

これにより、`composer.json` も変更されます。自分自身で `composer.json` ファイルを変更して依存関係を追加した場合は、次を実行して依存関係を更新します。

```
$ composer update
```

このライブラリを使用するように `index.php` を変更します。既存のルートの後に `/cowsay` の新しいルートを追加します。

```
$app->get('/cowsay', function() use($app) {
 $app['monolog']->addDebug('cowsay');
 return "<pre>".\Cowsayphp\Cow::say("Cool beans")."</pre>";
});
```

このルートにアクセスすると、アスキーアートで牛の絵が表示されます。
デプロイします。Heroku へのデプロイは、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デプロイします。

```
$ 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 cowsay
```

10. 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)
...
Happy logging!
Use `heroku addons:docs papertrail` to view documentation.
```

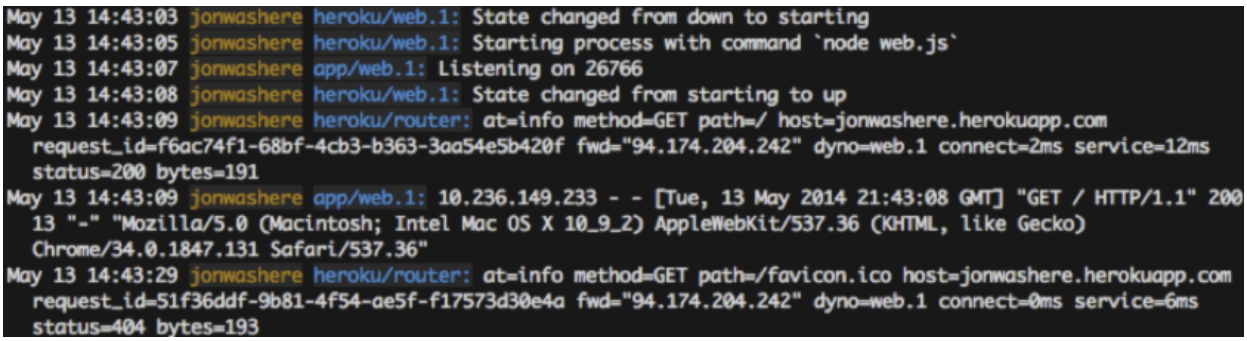
不正使用を防止するため、アドオンをプロビジョニングする際にはアカウントの認証が必要です。
アカウントの認証を行っていない場合は、認証サイトに転送されます。
これで、アドオンがアプリケーション用にデプロイおよび設定されました。次の方法で、アプリケーションのアドオンの一覧を確認できます。

```
$ 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。



11. 対話型シェルの起動

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対話型の PHP シェルを起動する際にも使用できます。

```
$ heroku run "php -a"
Running `php -a` attached to terminal... up, run.8081
Interactive shell

php > echo PHP_VERSION;
5.5.12
```

[`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。PHP コンソールは、PHP の標準ライブラリのみが読み込まれた状態です。
PHP シェルを終了するには、`quit`を入力します。
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.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。

12. 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。
`index.php`を変更して、設定変数`TIMES`の値に応じてルートが繰り返し`Hello`という単語を返すようにします。

```
$app->get('/', function() use($app) {
  $app['monolog']->addDebug('logging output.');
  return str_repeat('Hello', getenv('TIMES'));
});
```

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=20
```

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: erdKhPeeehIcdfY7ne
TIMES: 20
```

変更したアプリケーションを Heroku にデプロイし、動作を確認します。

13. データベースの追加

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ベース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、設定変数`DATABASE_URL`が設定されます(`heroku config`を実行することで確認可能)。`composer.json`を変更して、シンプルな PDO サービスプロバイダー、csanquer/pdo-service-provider の依存関係を追加します。

```
$ composer require csanquer/pdo-service-provider=~1.1dev
```

新しい依存関係をインストールします。

```
$ composer update
```

index.php を変更して、PDO 接続を追加するようにアプリケーションを拡張します。

```
$dbopts = parse_url(getenv('DATABASE_URL'));
$app->register(new Csanquer\Silex\PdoServiceProvider\Provider\PDOServiceProvider('pdo'),
           array(
            'pdo.server' => array(
              'driver'   => 'pgsql',
              'user' => $dbopts["user"],
              'password' => $dbopts["pass"],
              'host' => $dbopts["host"],
              'port' => $dbopts["port"],
              'dbname' => ltrim($dbopts["path"],'/')
              )
           )
);
```

このコードでは、`getenv()`を使用して設定変数`DATABASE_URL`を環境から取得し、`parse_url()`を使用してこの設定変数からホスト名、データベース、ログイン情報を抽出します。
同じファイルに、データベースをクエリする新しいハンドラーを追加します。

```
$app->get('/db/', function() use($app) {
 $st = $app['pdo']->prepare('SELECT name FROM test_table');
 $st->execute();

 $names = array();
 while ($row = $st->fetch(PDO::FETCH_ASSOC)) {
  $app['monolog']->addDebug('Row ' . $row['name']);
  $names[] = $row;
 }

 return $app['twig']->render('database.twig', array(
  'names' => $names
 ));
});
```

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、`database.twig`テンプレートを使用して結果が表示されるようになりました。
このテンプレートは`web/views`ディレクトリに作成します。

```
{% extends "layout.html" %}

{% block content %}
<p>Got these rows from the database:</p>

<ul>
{% for n in names %}
 <li> {{ n.name }} </li>
{% else %}
 <li>Nameless!</li>
{% endfor %}
</ul>

{% endblock %}
```

これらの変更に関してご不明な点がある場合、サンプルアプリケーションの db ブランチをご覧ください。
アプリケーションの変更を Heroku にデプロイします。

```
$ git add .
$ git commit -m "added database access"
$ git push heroku master
```

今の状態で`/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出力結果に`Nameless`と表示されます。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、先ほどプロビジョニングした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 (9.5.2, server 9.6.2)
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。

```
Got these rows from the database:

* hello database
```

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、同様の手法を使用できます。

14. 次のステップ

ここまでで、アプリケーションのデプロイ、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、デプロイ、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 PHP Apps on Heroku（Heroku での PHP アプリケーションのデプロイ）』では、既存の PHP アプリケーションを Heroku 用に移植して Heroku にデプロイする方法を説明しています。
- PHP カテゴリでは、PHP アプリケーションの開発とデプロイについて詳しく説明しています。