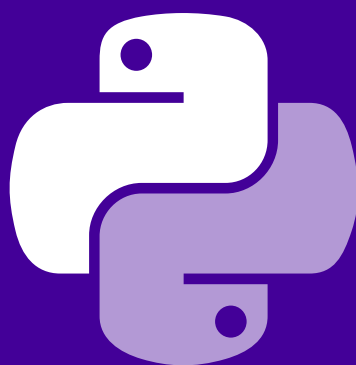




# さあ Heroku をはじめよう

## Python 編



|                        |       |
|------------------------|-------|
| ■ 1. はじめに              | 3     |
| ■ 2. 設定                | 3     |
| ■ 3. アプリケーションの準備       | 4     |
| ■ 4. アプリケーションのデプロイ     | 4~6   |
| ■ 5. ログの表示             | 6     |
| ■ 6. Procfile の定義      | 7     |
| ■ 7. アプリケーションの拡張       | 7~8   |
| ■ 8. アプリケーションの依存関係の宣言  | 8~9   |
| ■ 9. ローカルでのアプリケーションの実行 | 9     |
| ■ 10. ローカルで行った変更のプッシュ  | 10~11 |
| ■ 11. アドオンのプロビジョニング    | 12    |
| ■ 12. コンソールの起動         | 13~14 |
| ■ 13. 設定変数の定義          | 14    |
| ■ 14. データベースの使用        | 15~17 |
| ■ 15. 次のステップ           | 18    |

## 1. はじめに

このチュートリアルでは、Python アプリケーション (シンプルな Django アプリケーション) を短時間でデプロイ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](http://bit.ly/Heroku_SignUp_JP) をご覧ください)
- Python 3.6 がローカルにインストールされていること ([OS X](#)、[Windows](#)、[Linux](#) 用のインストールガイドを参照)
- [virtualenv](#) がローカルにインストールされていること (`pip install virtualenv` を実行してインストール)
- [Postgres](#) がローカルにインストールされていること (アプリケーションをローカルで実行する場合に必要)

## 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。



Heroku CLI をダウンロード (環境別)



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: python@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、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/python-getting-started.git  
$ cd python-getting-started
```

これで、シンプルなアプリケーションおよび Python のパッケージマネージャー、Pip で使用される `requirements.txt` が含まれる git リポジトリが準備できました。

### 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。  
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 lit-bastion-5032 in organization heroku... done, stack is cedar-14  
http://lit-bastion-5032.herokuapp.com/ | https://git.heroku.com/lit-bastion-5032.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 `lit-bastion-5032`)。

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、コードをデプロイします。

```
$ git push heroku master  
Counting objects: 232, done.  
Delta compression using up to 4 threads.  
Compressing objects: 100% (217/217), done.  
Writing objects: 100% (232/232), 29.64 KiB | 0 bytes/s, done.  
Total 232 (delta 118), reused 0 (delta 0)  
remote: Compressing source files... done.  
remote: Building source:  
remote:  
remote: -----> Python app detected  
remote: -----> Installing python-3.6.0
```

```
remote: $ pip install -r requirements.txt
remote: Collecting dj-database-url==0.4.0 (from -r requirements.txt (line 1))
remote:   Downloading dj-database-url-0.4.0.tar.gz
remote: Collecting Django==1.9.2 (from -r requirements.txt (line 2))
remote:   Downloading Django-1.9.2-py2.py3-none-any.whl (6.6MB)
remote: Collecting gunicorn==19.4.5 (from -r requirements.txt (line 3))
remote:   Downloading gunicorn-19.4.5-py2.py3-none-any.whl (112kB)
remote: Collecting psycopg2==2.6.1 (from -r requirements.txt (line 4))
remote:   Downloading psycopg2-2.6.1.tar.gz (371kB)
remote: Collecting whitenoise==2.0.6 (from -r requirements.txt (line 5))
remote:   Downloading whitenoise-2.0.6-py2.py3-none-any.whl
remote: Installing collected packages: dj-database-url, Django, gunicorn, psycopg2,
remote: whitenoise
remote: Running setup.py install for dj-database-url: started
remote: Running setup.py install for dj-database-url: finished with status 'done'
remote: Running setup.py install for psycopg2: started
remote: Running setup.py install for psycopg2: finished with status 'done'
remote: Successfully installed Django-1.9.2 dj-database-url-0.4.0 gunicorn-19.4.5
remote: psycopg2-2.6.1 whitenoise-2.0.6
remote:
remote: $ python manage.py collectstatic --noinput
remote: 58 static files copied to '/app/gettingstarted/staticfiles', 58 post-processed.
remote:
remote: -----> Discovering process types
remote:   Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:   Done: 39.3M
remote: -----> Launching...
remote:   Released v4
remote:   http://lit-bastion-5032.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
remote: To git@heroku.com:lit-bastion-5032.git
remote: * [new branch] master -> master
```

デプロイ時、Gunicorn のインストール中に `yield from self.wsgi.close()` 行の構文が無効であるという構文エラーが表示される場合があります。このエラーは無視してかまいません。これで、アプリケーションがデプロイされました。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ンネルで確認できます。

`heroku logs --tail` を利用して実行中のアプリケーションでの [ログ記録コマンド](#) からの情報を表示します。

```
$ heroku logs --tail
2014-08-15T15:17:55.780361+00:00 app[web.1]: 2014-08-15 15:17:55 [2] [INFO]
Listening at: http://0.0.0.0:19585 (2)
2014-08-15T15:17:55.780488+00:00 app[web.1]: 2014-08-15 15:17:55 [2] [INFO]
Using worker: sync
2014-08-15T15:17:55.830489+00:00 app[web.1]: 2014-08-15 15:17:55 [7] [INFO]
Booting worker with pid: 7
2014-08-15T15:17:55.779494+00:00 app[web.1]: 2014-08-15 15:17:55 [2] [INFO]
Starting gunicorn 19.0.0
2014-08-15T15:17:56.321151+00:00 heroku[web.1]: State changed from starting to up
2014-08-15T15:17:57.847806+00:00 heroku[router]: at=info method=GET
path="/" host=lit-bastion-5032.herokuapp.com request_id=
7fd99883-20ec-43d5-8b2d-5204351cdf2d fwd="94.174.204.242" dyno=web.1 connect=
1ms service=240ms status=200 bytes=679
```

ブラウザ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: gunicorn gettingstarted.wsgi --log-file -
```

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。

### Microsoft Windows

このサンプルアプリケーションには他にも、Microsoft Windows でのローカル開発用に `Procfile.windows` という `Procfile` ファイルが用意されており、これを使用するステップについては後ほどご説明します。

このファイルは、Windows と互換性がある他の Web サーバーを起動します。

```
web: python manage.py runserver 0.0.0.0:5000
```

## 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): `gunicorn gettingstarted.wsgi --log-file -`
web.1: starting 2014/12/11 11:59:02 (~ 1s 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フェッショナル用途の `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 `requirements.txt` ファイルがあると、アプリケーションが Python であると認識されます。独自のアプリケーションの場合、`pip freeze` を実行することでこのファイルを作成できます。

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 `requirements.txt` が含まれています。

```
dj-database-url==0.4.1
Django==1.9.7
gunicorn==19.6.0
psycopg2==2.6.2
whitenoise==2.0.6
```

`requirements.txt` ファイルには、アプリケーションの依存関係がバージョンと併せてリストアップされています。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、`pip install -r requirements.txt` コマンドを使用して Python の適切な依存関係をインストールします。  
これをローカルで実行するため、アプリケーションの `virtualenv` を作成します。

```
$ virtualenv venv
```

`virtualenv` を有効化します。Windows をお使いの場合は、次のコマンドを実行します。

```
> venv\Scripts\activate.bat
```

Windows 以外をお使いの場合は、次のコマンドを実行します。

```
$ source venv/bin/activate
```



アプリケーションをローカルで実行できるように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。

```
$ pip install -r requirements.txt
Downloading/unpacking Django==1.9.2 (from -r requirements.txt (line 1))
  Downloading Django-1.9.2.tar.gz (6.6MB): 6.6MB downloaded
  Running setup.py egg_info for package Django
...
Successfully installed dj-database-url Django gunicorn psycopg2 whitenoise
Cleaning up...
```

注: このステップを問題なく行うために、Postgres を [適切にインストールしておく](#) 必要があります。

Linux をお使いの場合は、`libpq-dev` システムパッケージ (または、お使いのディストリビューションで同等のもの) もインストールしておく必要があります。

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

ここまでで、アプリケーションをローカルで起動する準備はほぼ整いました。

Django ではローカルのアセットが使用されるため、まずは `collectstatic` を実行する必要があります。

```
$ python manage.py collectstatic
```

「yes」を返します。

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。Microsoft Windows システムをお使いの場合は、次を実行します。

```
$ heroku local web -f Procfile.windows
```

Unix システムをお使いの場合は、次を実行してデフォルトの `Procfile` をそのまま使用します。

```
$ heroku local web
```

ローカルの Web サーバーが起動します。

```
11:48:19 web.1 | started with pid 36084
11:48:19 web.1 | 2014-07-17 11:48:19 [36084] [INFO] Starting gunicorn 19.0.0
11:48:19 web.1 | 2014-07-17 11:48:19 [36084] [INFO] Listening at: http://0.0.0.0:5000 (36084)
11:48:19 web.1 | 2014-07-17 11:48:19 [36084] [INFO] Using worker: sync
11:48:19 web.1 | 2014-07-17 11:48:19 [36087] [INFO] Booting worker with pid: 36087
```

Heroku 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、ターミナルウィンドウに戻り、`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。

requirements.txt を変更して、次に示す requests の依存関係を追加します。

```
requests==2.9.1
```

完成した `requirements.txt` は次のようになります。

```

dj-database-url==0.4.1
Django==1.9.7
gunicorn==19.6.0
psycopg2==2.6.2
whitenoise==2.0.6
requests==2.9.1

```

requests モジュールが最初にインポートされるように `hello/views.py` を変更します。

```
import requests
```

このモジュールを使用するように `index` メソッドを変更します。

現在の `index` メソッドを次のコードに置き換えます。

```
def index(request):
    r = requests.get('http://httpbin.org/status/418')
    print(r.text)
    return HttpResponse('<pre>' + r.text + '</pre>')
```

□ローカルでテストします。

```
$ pip install -r requirements.txt
$ heroku local
```

<http://localhost:5000>でアプリケーションにアクセスすると、<http://httpbin.org/status/418> を取得して得られた出力として、かわいらしいティーポットが表示されます。

```
-=[ teapot ]=-
```

.....  
 : \_ \_ :  
 | . " ^ " . \_  
 \ \_ ; " \_ \_ " | //  
 | \_ ;/  
 \ \_ \_ /  
 ^ ^ ^ ^

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
```

## 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
```

## I 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 python manage.py shell
Python 3.6.0 (default, Sep  4 2013, 10:55:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.1.63)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

[`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。

Python シェルは、アプリケーションとすべての依存関係のコンテキストで起動します。

ここからアプリケーションファイルの一部をインポートできます。たとえば、次を実行できます。

```
>>> import requests
>>> print(requests.get('http://httpbin.org/status/418').text)

-=[ teapot ]=-

      _.._
     /___\
    | .^ ^ .|
    \; " " /|//
     |  ;/
    \_ _ _/
     `----`
        \

>>> 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.3052
~ $ ls
gettingstarted hello manage.py Procfile README.md requirements.txt runtime.txt staticfiles
~ $ 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。

`hello/views.py` を編集します。最初に、`os` モジュールをインポートするための行を追加します。

```
import os
```

次に、`index` メソッドを変更して、環境変数 `TIMES` の値の回数だけアクションを繰り返すようにします。

```
def index(request):
    times = int(os.environ.get('TIMES',3))
    return HttpResponse('Hello!' * times)
```

`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、「Hello!」が 2 つ表示されます。

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: erdKhPeeeehlcdfY7ne
TIMES: 2
```

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。

## 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、アプリケーションのデプロイで自動的にプロビジョニングされている、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確認ができます。

```
$ heroku addons
Add-on          Plan    Price State
-----
heroku-postgresql (postgresql-parallel-49191) hobby-dev free  created
└─ as DATABASE
...
```

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
=== lit-bastion-5032 Config Vars
DATABASE_URL: postgres://qplhashkhqyxp:YXDPus9MrU4HglCPzjhOevee@
ec2-54-204-47-58.compute-1.amazonaws.com:5432/dc9qsdhia6v1
...
```

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_POSTGRES_BLUE_URL (DATABASE_URL)
Plan:    Hobby-dev
Status:   Available
Connections: 0/20
PG Version: 9.6.1
Created:  2017/04/19 7:52 UTC
Data Size: 6.5 MB
Tables:   0
Rows:     0/10000 (In compliance)
Fork/Follow: Unsupported
Rollback: Unsupported
Add-on:   postgresql-parallel-49191
```

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。今回の例でデプロイ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、この機能には、アプリケーションの URL に `/db` を追加することでアクセスできます。

たとえば、アプリケーションのデプロイ先が `https://wonderful-app-287.herokuapp.com/` の場合、`https://wonderful-app-287.herokuapp.com/db` でアクセスできます。

データベースは設定されていてもテーブルが作成されていないため、アクセスするとエラーが生成されます。Django 標準の `manage.py migrate` を実行してテーブルを作成します。

```
$ heroku run python manage.py migrate
Running `python manage.py migrate` attached to terminal... up, run.1059
Synchronizing apps without migrations:
  Creating tables...
    Creating table hello_greeting
    Running deferred SQL...
  Installing custom SQL...
Running migrations:
  Rendering model states... DONE
  Applying contenttypes.0001_initial... OK
...
```

[You just installed Django's auth system, which means you don't have any superusers defined. Would you like to create one now? (Django の認証システムをインストールしましたが、スーパーユーザーが定義されていません。スーパーユーザーを作成しますか?)] というメッセージが表示された場合、`no` を入力します。

これで、`/db` ルートに再度アクセスすると、アクセスするたびにわかりやすい形でページが更新されるようになります。

#### Page View Report

April 19, 2017, 8:50 a.m.

April 19, 2017, 8:52 a.m.

データベースにアクセスするためのコードは簡潔です。また、`hello/models.py` にある `Greetings` というシンプルな Django モデルを使用します。

唯一必要な設定は `gettingstarted/settings.py` で確認できます。これは、環境変数 `DATABASE_URL` にもとづきデータベースを設定するものです。



```
# Parse database configuration from $DATABASE_URL
import dj_database_url
...
db_from_env = dj_database_url.config(conn_max_age=500)
DATABASES['default'].update(db_from_env)
```

アプリケーションの `/db` ルートにアクセスするたびに、`hello/views.py` ファイルにある次のメソッドが呼び出されて新しい Greeting が作成され、既存の Greetings がすべて表示されます。

```
def db(request):

    greeting = Greeting()
    greeting.save()

    greetings = Greeting.objects.all()

    return render(request, 'db.html', {'greetings': greetings})
```

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、すべての行を表示します。

```
$ heroku pg:psql
--> Connecting to postgresql-parallel-49191
psql (9.5.2, server 9.6.1)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
compression: off)
Type "help" for help.
=> select * from hello_greeting;                                id |
when
----+-----
 1 | 2017-04-19 08:50:58.343108+00
 2 | 2017-04-19 08:52:04.626254+00
(2 rows)
=> \q
```

[Heroku PostgreSQL](#) の詳細をご覧ください。

[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。

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、Python 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、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 Python and Django Apps on Heroku \(Heroku での Python および Django アプリケーションのデプロイ\)](#)』では、既存の Python または Django 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。
- [Python カテゴリ](#)では、Python 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。