

# Fazendo deploy de uma API para o Heroku

Apr 15, 2018 • 🕒 Reading time ~8 minutes

Quer aprender a fazer deploy @? Vem que eu te ensino!

## O que você vai encontrar nesse tutorial

- Como criar uma aplicação Flask nos moldes do Heroku
- Como fazer deploy
- Um pouquinho de Git
- Um pouquinho de consumo de API

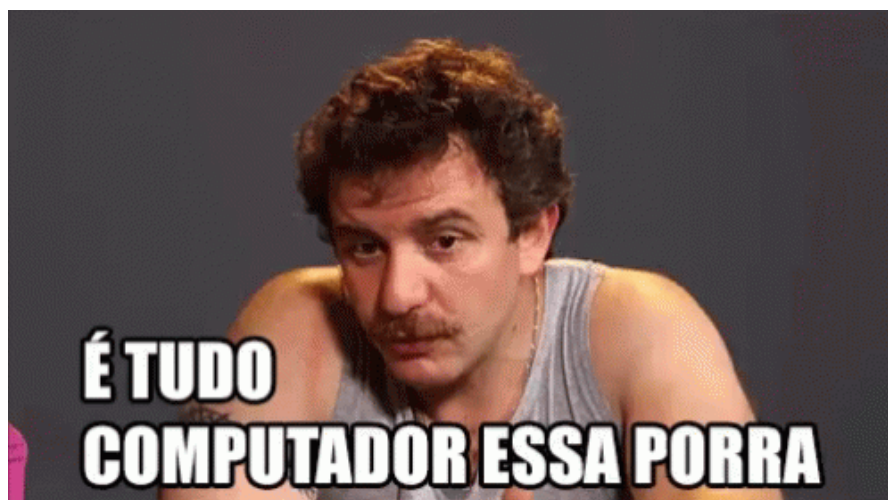
**Avisos:** Dos dois posts que eu usei de base para escrever esse aqui, um deles era muuuito antigo e o outro, apesar de recente, não tinha instruções que correspondem a versão mais atual do Heroku, eles estão na sessão de links para quem quiser lê-los.

## Heroku quem?

Se você chegou até aqui, é provável que você já saiba o que o Heroku é, mas pra quem ainda não sabe, Heroku é uma plataforma na nuvem que permite que pessoas “transformem suas ideias o mais rápido possível em uma URL” e sem ter dores de cabeça com a parte de infraestrutura (<https://www.heroku.com/what>).

Essa plataforma provê um serviço que, a partir de uma estrutura pré-definida de uma aplicação, consegue empacotar essa aplicação e colocar ela pra rodar num servidor em um dos data centers deles. O Heroku aceita aplicações escritas em várias linguagens (<https://www.heroku.com/languages>), mas hoje vamos usar Python (!

Resumo:



## Partiu código!

Pra esse tutorial eu fiz uma API em um microframework Python chamado [Flask](http://flask.pocoo.org/) (<http://flask.pocoo.org/>). Não irei explicar em detalhes como o Flask funciona, mas se você quiser aprender sobre isso, o Bruno Rocha tem um conjunto de posts chamado "What the Flask" que explicam super bem e estão todos lá nos links.

## Nossa API

Ela terá apenas um *endpoint* definido pelo `/`. Esse *endpoint* poderá dar duas respostas dependendo da requisição que você faça. São eles:

---

Requisição	Resultado
GET sem cabeçalhos (headers) extras	Não entre em pânico!
GET com Authorization 42	a resposta para a vida, o universo e tudo mais

---

O código pra fazer isso fica pequenininho, dá uma olhada:

```
import os
from flask import Flask, jsonify, request

app = Flask(__name__)

@app.route('/')
def nao_entre_em_panico():
    if request.headers.get('Authorization') == '42':
        return jsonify({"42": "a resposta para a vida, o universo e tudo mais"})
    return jsonify({"message": "Não entre em pânico!"})

if __name__ == "__main__":
    port = int(os.environ.get("PORT", 5000))
    app.run(host='0.0.0.0', port=port)
```

Esse código fica dentro de um arquivo chamado aqui de `nao_entre_em_panico.py` mas você pode nomear da forma que você achar melhor. Além disso, esse código tem uma peculiaridade: as três linhas finais, elas servem para configurar o servidor Flask quando ele estiver rodando no Heroku.

## Pipfile

Lembra que eu falei lá no começo que os posts base eram antigos? Pois bem, hoje o Heroku exige um `Pipfile` para preparar o ambiente em que a nossa aplicação vai rodar. O `Pipfile` vai servir para instalar as bibliotecas Python necessárias para rodar a nossa aplicação. Se você, assim como eu, não gosta de usar o `Pipenv` (<https://docs.pipenv.org/>) - desculpa Cássio - dá pra criar o arquivo na mão, como criei o meu abaixo:

```
[[source]]

url = "https://pypi.python.org/simple"
verify_ssl = true

[packages]

Flask = "*"

[requires]

python_version = "3.6"
```

Para desenvolver (preparar o ambiente local) eu gosto do bom e velho requirements.txt (<http://jtemporal.com/requirements-txt/>) que fica assim:

```
click==6.7
Flask==0.12.2
itsdangerous==0.24
Jinja2==2.10
MarkupSafe==1.0
Werkzeug==0.14.1
```

## Procfile

No nosso computador, para rodar (colocar de pé) a nossa API o comando é o seguinte:

```
FLASK_APP=nao_entre_em_panico.py flask run
```

Como não será você quem irá executar um comando pra colocar a API de pé, você vai precisar de um arquivo que vai dizer “*Coloca a API de pé aí*” para o Heroku, esse arquivo é o Procfile.

Uma atençãozinha especial para um detalhe: esse arquivo não possui extensão, dependendo do seu sistema operacional e seu editor de texto, na hora de salvar esse arquivo pode aparecer uma extensão `.txt` nesse arquivo e isso vai causar uma falha no nosso deploy então, lembre-se de remover a extensão do arquivo caso ela apareça ;)

Para esta aplicação, o Procfile vai ter uma linha só que é a seguinte:

```
web: python nao_entre_em_panico.py
```

Tá beleza, e o que mais?

## Versionamento

Como o Heroku funciona com o sistema de versionamento Git, independentemente de você hospedar ou não o código no GitHub, nós precisamos criar nosso histórico de versão para a aplicação.

Então, depois de criar todos esses arquivos aqui de cima, você precisa adicionar tudo isso na sua árvore Git. Se você não é lá muito fã de Git pode fazer o seguinte:

```
$ git init
$ git add .
$ git commit -m "criando a aplicação"
```

Essa sequência de comandos acima vai colocar todos os arquivos num mesmo commit, isso não é uma boa prática e caso esteja interessada(o) em boas práticas lá no fim desse tutorial vai ter links sobre isso.

## 3, 2, 1... Deploy \o/

Agora você tem uma escolha: existem algumas formas de enviar o seu código para o Heroku. Você pode conectar com o GitHub ou com Dropbox ou ainda usar o Heroku CLI. Nesse post nós vamos fazer da última forma.

Então, pra começar você precisa [instalar o Heroku CLI \(https://devcenter.heroku.com/articles/heroku-cli#download-and-install\)](https://devcenter.heroku.com/articles/heroku-cli#download-and-install).

## Fazendo login

Começando pelo login:

```
$ heroku login
```

Esse comando vai te pedir para digitar e-mail e senha de acesso da sua conta no Heroku:

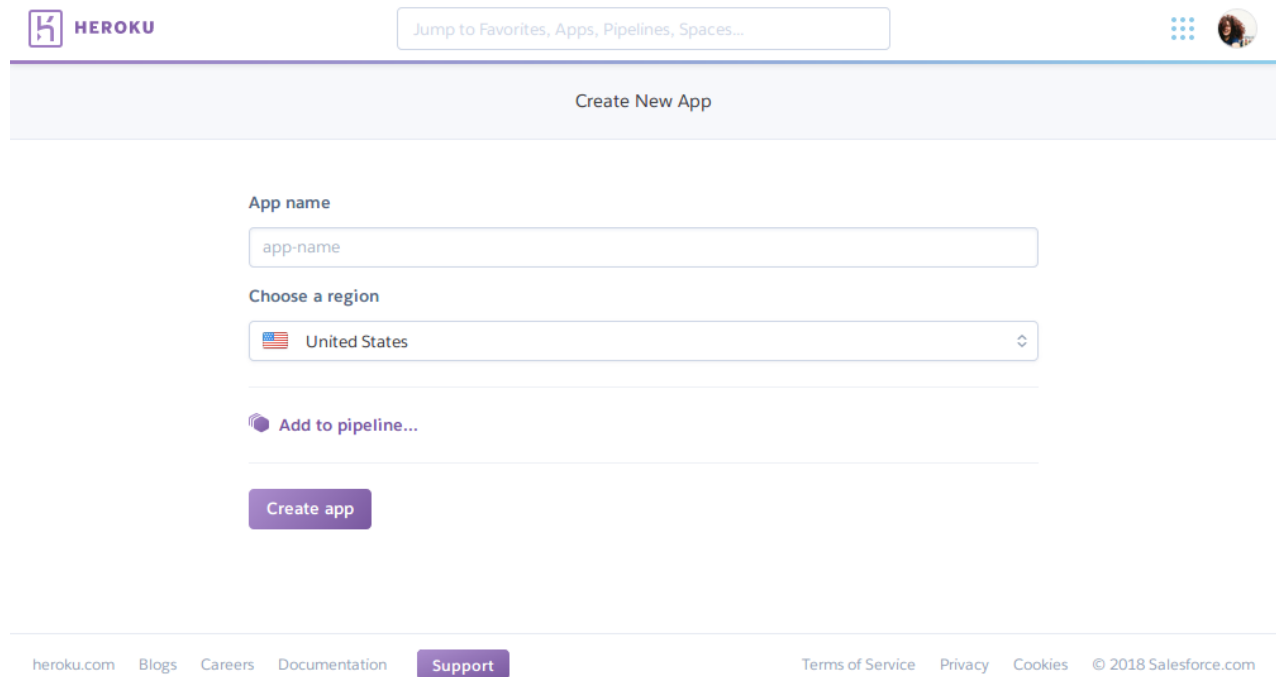
```
at api-example-flask-heroku $ heroku login
Enter your Heroku credentials:
Email:
Password:
Logged in as jessicatemporal
at api-example-flask-heroku $
```

*Tela de login da CLI do Heroku*

# Criando uma aplicação

## Usando a interface

É possível criar uma aplicação pela interface do site (<https://dashboard.heroku.com/new-app>), onde você vai ver a seguinte tela:

The screenshot shows the Heroku dashboard's 'Create New App' page. At the top, there's a navigation bar with the Heroku logo, a search bar with the text 'Jump to Favorites, Apps, Pipelines, Spaces...', and a user profile icon. Below the navigation bar is a large light blue button labeled 'Create New App'. Underneath this button, there's a form with two main sections. The first section is 'App name' with a text input field containing 'app-name'. The second section is 'Choose a region' with a dropdown menu showing 'United States' and a small flag icon. Below the dropdown is a link that says 'Add to pipeline...'. At the bottom of the form is a purple button labeled 'Create app'. The footer of the page contains links for 'heroku.com', 'Blogs', 'Careers', 'Documentation', and a 'Support' button, followed by 'Terms of Service', 'Privacy', 'Cookies', and '© 2018 Salesforce.com'.

Depois disso você ainda precisa adicionar o `remote` do Heroku no nosso repositório:

```
heroku git:remote -a guiaapi
```

E olha como é a resposta desse comando:

```
at api-example-flask-heroku (master)$ heroku git:remote -a guiaapi
set git remote heroku to https://git.heroku.com/guiaapi.git
at api-example-flask-heroku (master)$
```

## Usando o terminal

Mas também podemos criar a aplicação usando a CLI do Heroku. O lado bom de usar a CLI é que ao criar a aplicação o `remote` é criado automaticamente:

```
heroku apps:create guiaapi
```

O que deve mostrar algo assim ó:

```
at api-example-flask-heroku (master)$ heroku apps:create guiaapi
Creating ● guiaapi... done
https://guiaapi.herokuapp.com/ | https://git.heroku.com/guiaapi.git
at api-example-flask-heroku (master)$
```

Pausa para mais um momento de atenção: desse jeito que eu mostrei aí em cima, a gente consegue nomear nossa aplicação dentro do Heroku, aí que entra uma "pegadinha", o nome da minha aplicação tem que ser única dentro do Heroku, pois é o nome da aplicação que é usado para criar a URL dela, se você por acaso tentar criar uma aplicação com um nome já utilizado, a resposta para o comando acima vai ser esta:

```
at api-example-flask-heroku (master)$ heroku apps:create guia
Creating ● guia... !
> Name is already taken
```

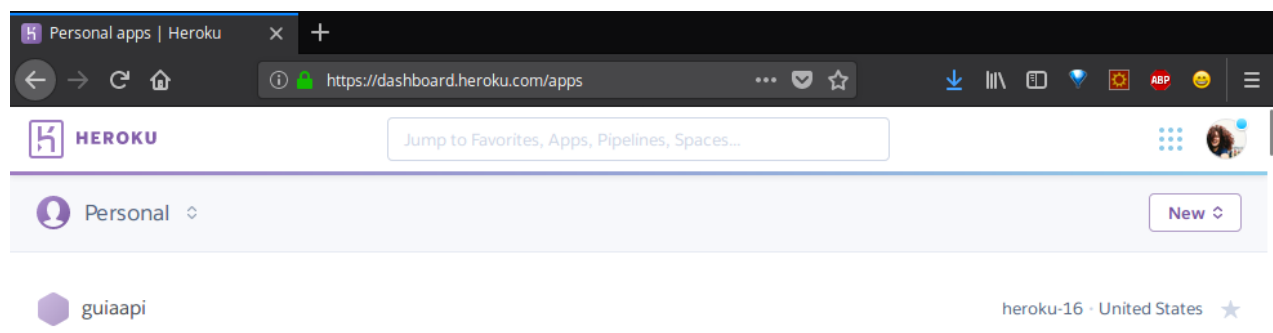
Se o nome da sua aplicação não for algo importante, você pode executar o seguinte comando no lugar do comando anterior:

```
heroku create
```

Esse comando vai criar uma aplicação com um nome randomizado, por exemplo:

```
at api-example-flask-heroku (master)$ heroku create
Creating app... done, ● warm-spire-86593
https://warm-spire-86593.herokuapp.com/ | https://git.heroku.com/warm-spire-86593.git
at api-example-flask-heroku (master)$
```

Depois de criada a aplicação, você pode dar uma espiada lá na dashboard do Heroku na área de aplicações (<https://dashboard.heroku.com/apps>) para ver a lista das suas aplicações:



## Finalmente fazendo o bendito deploy

Depois de commitar tudo, fazer login e criar a sua aplicação no Heroku chegou a hora do

*deploy.*

A ação de fazer o deploy, aqui nesse contexto, significa enviar o código da aplicação para o servidor e rodar o processo para colocar nossa API de pé no servidor.

Outros contextos podem trazer variações e passos intermediários desse que estou apresentando, mas aqui como nossa aplicação é simples e o Heroku é feito para nos ajudar a colocar aplicações de pé, o maior trabalho se torna a criação dos arquivos de configuração (que já criamos) ;)

Então, o comando que vai executar o envio do código para o servidor é:

```
git push heroku master
```

Em caso de sucesso você deve ver algo parecido com isso:

```
at api-example-flask-heroku (master)$ git push heroku master
Counting objects: 15, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (13/13), done.
Writing objects: 100% (15/15), 1.79 KiB | 0 bytes/s, done.
Total 15 (delta 2), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Python app detected
remote: !      No 'Pipfile.lock' found! We recommend you commit this into your
remote: repository.
remote: -----> Installing python-3.6.4
remote: -----> Installing pip
remote: -----> Installing dependencies with Pipenv 11.8.2...
remote: -----> Installing dependencies from Pipfile...
remote: -----> Discovering process types
remote: Procfile declares types -> web
remote:
remote: -----> Compressing...
remote: Done: 53.4M
remote: -----> Launching...
remote: Released v3
remote: https://guiaapi.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/guiaapi.git
* [new branch]      master -> master
at api-example-flask-heroku (master)$
```

Uma última configuração que pode ser necessária fazer é garantir que a nossa aplicação tem pelo menos um *dyno* rodando com o seguinte comando:

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



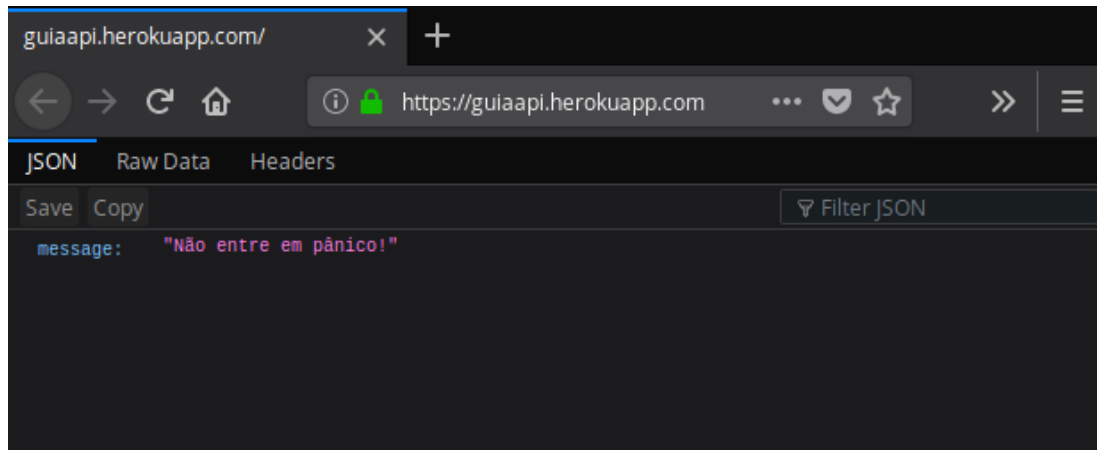
## Dynos

E o que é um *dyno*? Em resumo um *dyno* é um Linux Container, o Heroku utiliza um modelo de container que isola a sua aplicação e possibilita a fácil escalabilidade do sistema. Em outras palavras menos técnicas: Pega todos os arquivos de código e configuração que escrevemos até agora e enfia num caixa e bota pra rodar, isso é um *dyno*.

Agora pra ver o resultado da sua API podemos (nesse caso) abrir um navegador e acessar <https://guiaapi.herokuapp.com/> ou usar a linha de comando pra abrir a URL direto:

```
heroku open
```

Se deu tudo certo (e você usar o Firefox como navegador) você verá algo assim:



O que aconteceu foi o seguinte: ao acessar a URL da nossa aplicação no Heroku nós fizemos uma requisição GET pra API com o cabeçalho (header) padrão, ou seja, sem o campo `Authorization`. Que seria a mesma coisa que rodar no terminal, isso aqui em baixo:

```
curl -X GET -k -i 'https://guiaapi.herokuapp.com/'
```

Que me dá este resultado:

```
at api-example-flask-heroku (master)$ curl -X GET -k -i 'https://guiaapi.herokuapp.com/'
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Content-Length: 50
Server: Werkzeug/0.14.1 Python/3.6.4
Date: Sun, 15 Apr 2018 02:48:30 GMT
Via: 1.1 vegur

{
  "message": "N\u00e3o entre em p\u00e2nico!"
}
at api-example-flask-heroku (master)$
```

Como sabemos, essa nossa API vai ter uma resposta diferente se você passar o cabeçalho `Authorization` com o valor de `42`, veja:

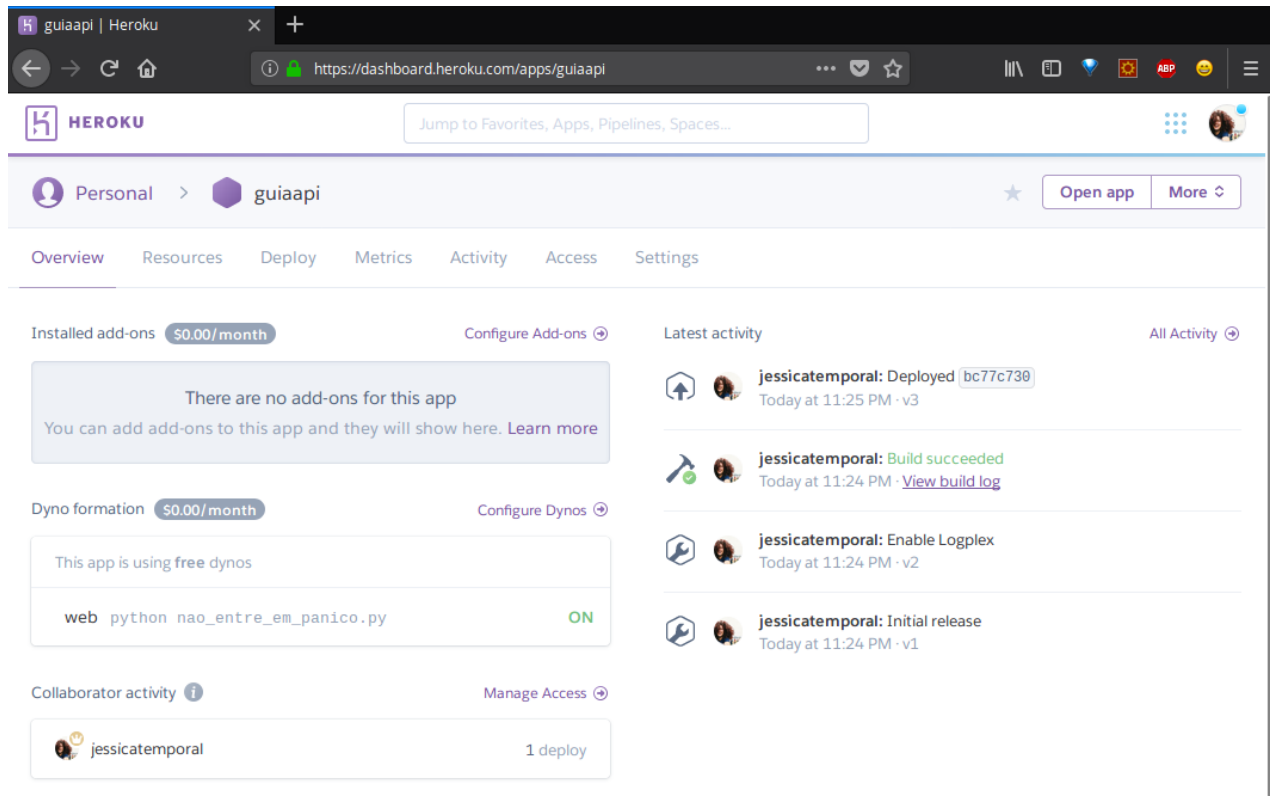
```
curl -X GET -k -H 'Authorization: 42' \
-i 'https://guiaapi.herokuapp.com/'
```

Temos o seguinte resultado:

```
at api-example-flask-heroku (master)$ curl -X GET -k -H 'Authorization: 42' \
> -i 'https://guiaapi.herokuapp.com/'
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Content-Length: 61
Server: Werkzeug/0.14.1 Python/3.6.4
Date: Sun, 15 Apr 2018 02:52:20 GMT
Via: 1.1 vegur

{
  "42": "a resposta para a vida, o universo e tudo mais"
}
at api-example-flask-heroku (master)$
```

Dá pra passar o cabeçalho de autorização no navegador? Dá! mas vou deixar isso pra outro post/tutorial que eu prometo que linko aqui quando escrever.



## Pro tip: Arquivos de log

Pode ser que você tenha cometido alguma falha no processo de configuração da sua aplicação e, na hora do deploy, o Heroku avise que ele foi mal-sucedido. Nesse caso o primeiro lugar para procurar informações do que deu errado são os logs.

Um bom sistema faz logs de todas as ações executadas. Nesse nosso caso, o Heroku se encarrega de logar todos os detalhes pra gente, então basta pedir esse log usando a CLI. Aqui eu ainda escrevo os logs num arquivo para facilitar a inspeção, veja:

```
heroku logs > out.log
```

Exemplos de linhas que você irá encontrar num desses logs:

```
...
2018-04-15T02:24:39.000000+00:00 app[api]: Build succeeded
2018-04-15T02:25:11.165813+00:00 heroku[web.1]: Starting process with command `python nao_entre_em_panico.py`
...
2018-04-15T02:52:20.075995+00:00 app[web.1]: 10.5.207.142 - - [15/Apr/2018 02:52:20] "GET / HTTP/1.1" 200 -
```

Agora se alguém falar “Faz um deploy ae!” você já sabe como ;)

*Ps.: Se tiver dúvidas ou comenta ali em baixo ou me manda mensagem que eu tento responder, sem crise as DMs tão sempre abertas ;)*

---

## Links

- Texto do Diego Garcia: [Publicando seu Hello World no Heroku \(http://pythonclub.com.br/publicando-seu-hello-world-no-heroku.html\)](http://pythonclub.com.br/publicando-seu-hello-world-no-heroku.html)
- Texto do John Kagga em inglês: [Deploying a Python Flask app on Heroku \(https://medium.com/@johnkagga/deploying-a-python-flask-app-to-heroku-41250bda27d0\)](https://medium.com/@johnkagga/deploying-a-python-flask-app-to-heroku-41250bda27d0)
- [What the Flask \(https://twitter.com/rochacbruno\)](https://twitter.com/rochacbruno) do Bruno Rocha
- Curso de [Git para iniciantes do Willian Justen no Udemy \(https://www.udemy.com/git-e-github-para-iniciantes/\)](https://www.udemy.com/git-e-github-para-iniciantes/)
- [Discussão sobre boas práticas de Git no Stackoverflow \(https://pt.stackoverflow.com/questions/60729/quais-seriam-as-pr%C3%A1ticas-recomendadas-para-commits-no-git\)](https://pt.stackoverflow.com/questions/60729/quais-seriam-as-pr%C3%A1ticas-recomendadas-para-commits-no-git)
- O site [Oh shit, git! \(http://ohshitgit.com/\)](http://ohshitgit.com/) em Inglês com uns poucos tópicos mais avançados em Git
- Vídeo do [Bruno Rocha \(https://twitter.com/rochacbruno\)](https://twitter.com/rochacbruno) com uma [Introdução ao HTTP \(https://www.youtube.com/watch?v=GVTEyLNyGWE\)](https://www.youtube.com/watch?v=GVTEyLNyGWE)

## TAMBÉM EM JTEMPORAL

**Como ser Cientista de Dados usando um ...**

um ano atrás • 1 comentário

A resposta simples: Nuvem! Vamos ver na prática como é viver com seus Jupyter ...

**Subindo imagens docker pro dockerhub**

2 anos atrás • 1 comentário

Já achou mágico aquelas imagens pros containers docker que qualquer um ...

**De 0 a 17 mil pageviews**

9 meses atrás • 2 comentários

Como uma troca de temas impactou na quantidade de acessos do meu blog


**Diferenciando json.loads de**

um ano atrás • 1 comentário

Aprenda a diferenciar json.loads de json.dumps em Python

9 Comentários

jtemporal

 Disqus' Privacy Policy Renato Silva ▾ Recomendar 7 Tweet Compartilhar

Ordenar por Mais votados ▾



Participe da discussão...

**Lucas Souto** • 2 anos atrás

Sobre boas práticas com o git gosto de consultar esse link às vezes:  
[www.git-tower.com/learn/git...](https://www.git-tower.com/learn/git...)

1 ^ | ▾ • Responder • Compartilhar &gt;

**Jessica Temporal** Mod ➔ Lucas Souto • 2 anos atrás

sim! muito boa essa indicação inclusive!

^ | ▾ • Responder • Compartilhar &gt;

**Diogo Jéferson** • 11 dias atrás

06/2020 e esse artigo perfeito me salvando. Obrigado!

^ | ▾ • Responder • Compartilhar &gt;

**Rodrigo Silveira** • 9 meses atrás

Fantástico, me salvou num deploy aqui!

^ | ▾ • Responder • Compartilhar &gt;

**Ronaldo** • um ano atrás

Muito legal e construtiva sua iniciativa, pizza de dados também é show e seu cabelo é marca registrada. Fico feliz pela sua disponibilidade em ser útil ao próximo. Show;) Parabéns

^ | ▾ • Responder • Compartilhar &gt;

**Groo** • 2 anos atrás

Olá, obrigado pelas dicas! Apenas com o arquivo .py o deploy foi bem sucedido, porém quando eu começo a aumentar o projeto (colocando uma página estática na página templates, por exemplo) a aplicação começa a apresentar erro no heroku:

[« Instalando Go no Windows \(https://jtemporal.com/instalando-go-windows/\)](https://jtemporal.com/instalando-go-windows/)

[Brincando com o PostgreSQL » \(https://jtemporal.com/brincando-com-postgresql/\)](https://jtemporal.com/brincando-com-postgresql/)



## Recent articles

[Trabalhando remoto \(https://jtemporal.com/trabalhando-remoto/\)](https://jtemporal.com/trabalhando-remoto/) 26 Mar 2020

[Working remotely \(https://jtemporal.com/working-remotely/\)](https://jtemporal.com/working-remotely/) 26 Mar 2020

[Fixing date error while running jekyll on macOS Catalina \(https://jtemporal.com/fixing-date-error-while-running-jekyll-on-macos-catalina/\)](https://jtemporal.com/fixing-date-error-while-running-jekyll-on-macos-catalina/) 25 Feb 2020

---

All rights reserved by [jtemporal \(https://jtemporal.com\)](https://jtemporal.com)

Built by [webjeda \(http://webjeda.com\)](http://webjeda.com)



<https://twitter.com/jesstemporal>



<http://github.com/jtemporal>



[\(mailto:jessicatemporal@gmail.com\)](mailto:jessicatemporal@gmail.com)