



Heroku

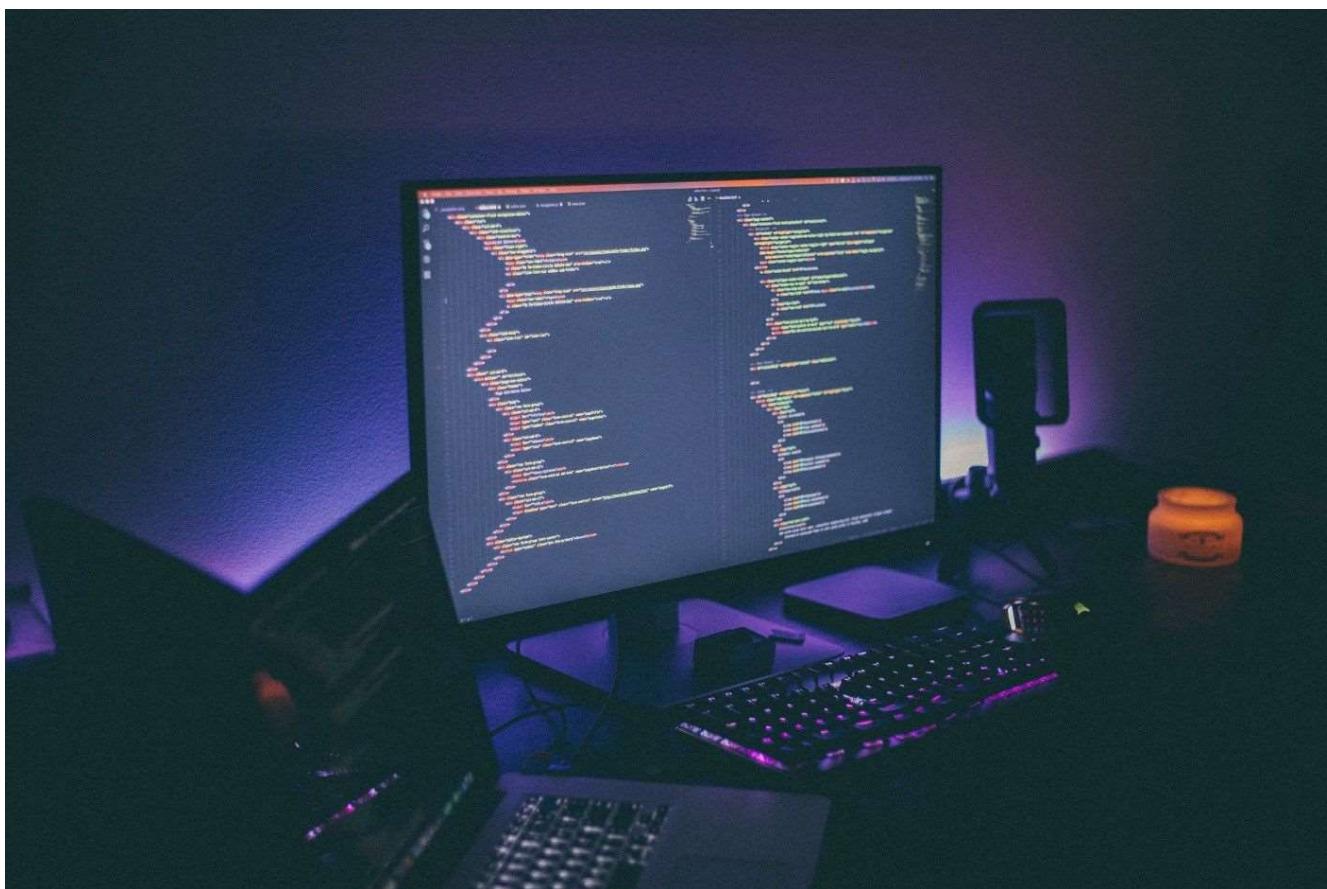
Como implantar um monorepo em vários aplicativos Heroku usando o GitHub Actions

Este artigo mostra como implantar um monorepo Node.js e React.js em vários aplicativos Heroku usando o GitHub Actions.



Gerald Haxhi

9 de setembro de 2020 • 5 minutos de leitura



A Heroku é um dos melhores provedores de serviços em nuvem, o que nos ajuda a criar e implantar aplicativos com facilidade. Entre outros motivos, é muito

preferido pelos desenvolvedores, pois oferece um [plano gratuito](#) em que você nem precisa registrar um cartão de crédito.

Como dito acima, o Heroku torna as implementações fáceis e diretas, desde que você tenha criado um arquivo especial onde você escreve algumas instruções que são necessárias para construir e servir o aplicativo (ou seja, o Procfile). O problema é que a configuração padrão do Heroku requer um repositório por aplicativo. E se quisermos manter toda a pilha em um único repositório e implantar cada aplicativo como um aplicativo Heroku separado?

Neste artigo, forneceremos uma configuração que nos permite implantar um monorepo em vários aplicativos Heroku, usando algumas ferramentas existentes. A pilha que implantaremos é composta por um aplicativo cliente React e uma API Node.js Express. Para automatizar a compilação e a implantação, usaremos o GitHub Actions.

Configuração da pilha de aplicativos

(Todos os comandos são executados em um sistema operacional macOS 10.14)

Etapa 1: configuração do GitHub

Antes de criar nossos aplicativos, precisamos configurar um repositório GitHub e clonar o projeto em nosso próprio dispositivo:

```
$ cd ~/Desktop$ git clone git@github.com:zeroabsolute/MonorepoHerokuDeployment.git$
```

Etapa 2: configuração da API

Nesta etapa, criaremos uma API Express muito básica que fornecerá um endpoint de integridade simples.

Primeiramente, vamos criar o diretório da API e adicionar um package.json nele:

```
$ mkdir src  
$ cd src  
$ mkdir api  
$ npm init
```

Em seguida, vamos instalar os pacotes necessários:

```
$ npm install express cors
```

Agora vamos criar um arquivo de índice e colocar algum código nele:

```
const Express = require('express');
const Cors = require('cors');

const PORT = process.env.PORT || 5000; // https://help.heroku.com/P1AVPANS/why-is-my-node-js-app-crashing-with-an-r10-error
const app = Express();
const server = require('http').Server(app);

app.use(Cors());
app.use('/health', (req, res) => {
  res.status(200).json({
    appName: 'API',
    version: process.env.npm_package_version,
    status: 'OK',
  });
});

server.listen(PORT, (error) => {
  if (error) {
    console.log(`\n\n${error}\n\n`);
  } else {
    console.log(`\n\n${error}\n\n`);
  }
  API:
  Status: Error
  Log: ${error}

  \n\n`);

});
```

Por fim, vamos executar a API e testá-la chamando o endpoint de integridade.

```
$ npm start
```

```
> api@1.0.0 start MonorepoHerokuDeployment/src/api
> node index.js
-----
-----
API:
Name: Express API
Port: 5000
Status: OK
-----
```

Devemos obter o seguinte resultado:

```
$ curl localhost:5000/health
{"appName":"API","version":"1.0.0","status":"OK"}
```

Etapa 3: configuração do cliente

Nesta etapa, configuraremos um aplicativo React minimalista com [create-react-app](#), que usará o endpoint de integridade da API e renderizará a carga útil na tela.

Em primeiro lugar, vamos configurar os diretórios e o clichê do CRA.

```
$ cd src
$ npx create-react-app web-client
$ cd web-client
$ npm install axios
```

Em seguida, atualizaremos ligeiramente o arquivo App.js para usar nossa API e chamar o endpoint de integridade. A resposta será renderizada na tela. Aqui está o código final após a edição:

```

import React, { useEffect, useState } from 'react';
import Axios from 'axios';

import logo from './logo.svg';
import './App.css';

const API_URL = process.env.REACT_APP_API_URL || 'http://localhost:5000';

function App() {
  const [apiStatus, setApiStatus] = useState('');

  useEffect(() => {
    Axios({
      method: 'GET',
      url: `${API_URL}/health`
    }).then((response) => {
      setApiStatus({ status: response.status, payload: response.data, error: null });
    }).catch((error) => {
      setApiStatus({ status: error.response?.status, payload: null, error: error.response?.data });
    });
  }, []);

  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <h3>API Status:</h3>
        {renderStatus(apiStatus)}
      </header>
    </div>
  );
}

function renderStatus(response) {
  if (!response) {
    return null;
  }

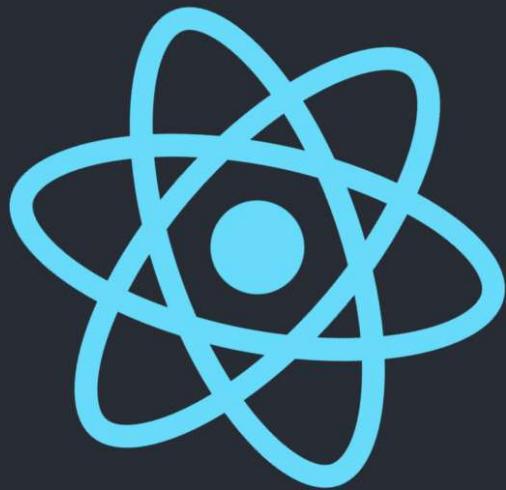
  if (response.error) {
    return (
      <div>
        Code: {response.status} <br />
        Message: {response.error?.message}
      </div>
    );
  }

  return (
    <div>
      Code: {response.status} <br />
      Version: {response.payload?.version} <br />
      Message: {response.payload?.status}
    </div>
  );
}

export default App;

```

Por fim, vamos testar o aplicativo cliente localmente. Devemos obter o seguinte resultado:



API Status:

Code: 200
Version: 1.0.0
Message: OK

Configuração e configuração do Heroku

Depois de terminar a configuração do projeto local, podemos seguir em frente e configurar nossos aplicativos Heroku.

Se você não tiver o Heroku CLI instalado em seu dispositivo, siga [este guia para instalá-lo e fazer login.](#)

Primeiramente, precisamos criar dois aplicativos Heroku separados, um para a API Express e outro para o React Web Client:

```
$ heroku create api-02092020

Creating ⚡ api-02092020... done
https://api-02092020.herokuapp.com/ | https://git.heroku.com/api-02092020.git

$ heroku create web-client-02092020

Creating ⚡ web-client-02092020... done
```

Na próxima etapa, precisamos especificar um buildpack para cada um de nossos aplicativos — o buildpack Node.js e o buildpack Create-React-App para a API e o Web Client, respectivamente.

```
$ heroku buildpacks:add -a api-02092020 heroku/nodejs
```

```
Buildpack added. Next release on api-02092020 will use heroku/nodejs.  
Run git push heroku main to create a new release using this buildpack.
```

```
$ heroku buildpacks:add -a web-client-02092020 mars/create-react-app
```

```
Buildpack added. Next release on web-client-02092020 will use mars/create-react-app  
Run git push heroku main to create a new release using this buildpack.
```

Agora precisamos adicionar outro pacote de compilação, aquele que possibilita implantar muitos aplicativos de um único repositório — Heroku-buildpack-monorepo.

Nota: Adicionamos a opção “-i 1” no final porque este buildpack precisa ser definido antes dos outros como está declarado [aqui](#) .

```
$ heroku buildpacks:add -a api-02092020 https://github.com/lstoll/heroku-buildpack-
```

```
Buildpack added. Next release on api-02092020 will use:
```

1. heroku/nodejs
2. https://github.com/lstoll/heroku-buildpack-monorepo

```
$ heroku buildpacks:add -a web-client-02092020 https://github.com/lstoll/heroku-bui
```

```
Buildpack added. Next release on web-client-02092020 will use:
```

1. mars/create-react-app
2. https://github.com/lstoll/heroku-buildpack-monorepo

Para cada aplicação, devemos também definir a variável de ambiente *APP_BASE*, que representará o novo diretório raiz do projeto.

```
-a api-02092020 APP_BASE=src/api
```

```
restarting ⚡ api-02092020... done, v4

-a web-client-02092020 APP_BASE=src/web-client REACT_APP_API_URL=https://api-02092020.

ACT_APP_API_URL and restarting ⚡ web-client-02092020... done, v5
rc/web-client
https://api-02092020.herokuapp.com/
```

Vamos tentar enviar nossas alterações manualmente para o Heroku, antes de automatizar o processo com GitHub Actions:

```
$ git remote add heroku-api https://git.heroku.com/api-02092020.git
$ git remote add heroku-web-client https://git.heroku.com/web-client-02092020.git
$ git push heroku-api master
$ git push heroku-web-client master
```

A implantação para ambos os aplicativos deve ser bem-sucedida. Se tentarmos acessar cada um deles nas URLs fornecidas pelo Heroku, devemos ver alguma saída na tela.

Automatizando a implantação com o GitHub Actions

Para evitar digitar os comandos acima manualmente, podemos declarar um script GitHub Actions que lidará com a configuração e implantação do ambiente Heroku automaticamente, assim que enviarmos nosso código para o GitHub. Usaremos o seguinte plugin para implantar no Heroku:

<https://github.com/AkhileshNS/heroku-deploy>

Etapa 1: armazene a chave da API Heroku no GitHub Secrets

Navegue até <https://dashboard.heroku.com/account> e role até o final da página para obter a chave de API para sua conta. Depois de pressionar “Reveal”, copie a chave e navegue até a guia Secrets no seu projeto GitHub usando o seguinte URL: <https://github.com/{username}/{repository}/settings/secrets>. Pressione “Novo segredo” e cole o valor da chave de API. Você pode colocar o nome que quiser para a chave (no nosso caso, o nome da variável será HEROKU_API_KEY).

Etapa 2: adicionar o script GitHub Actions

No diretório raiz do projeto, adicionaremos o seguinte arquivo: “`.github/workflows/.deploy.yaml`”. A implementação será a seguinte:

```
name: Push stack to heroku

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v1
      - name: Release API
        uses: akhileshns/heroku-deploy@v3.0.4
        with:
          heroku_api_key: ${{secrets.HEROKU_API_KEY}}
          heroku_app_name: "api-02092020"
          heroku_email: ${{secrets.HEROKU_API_KEY}}
        env:
          HD_APP_BASE: "src/api"
      - name: Release Web Client
        uses: akhileshns/heroku-deploy@v3.0.4
        with:
          heroku_api_key: ${{secrets.HEROKU_API_KEY}}
          heroku_app_name: "web-client-02092020"
          heroku_email: ${{secrets.HEROKU_API_KEY}}
        env:
          HD_APP_BASE: "src/web-client"
          HD_REACT_APP_API_URL: "https://api-02092020.herokuapp.com"
```

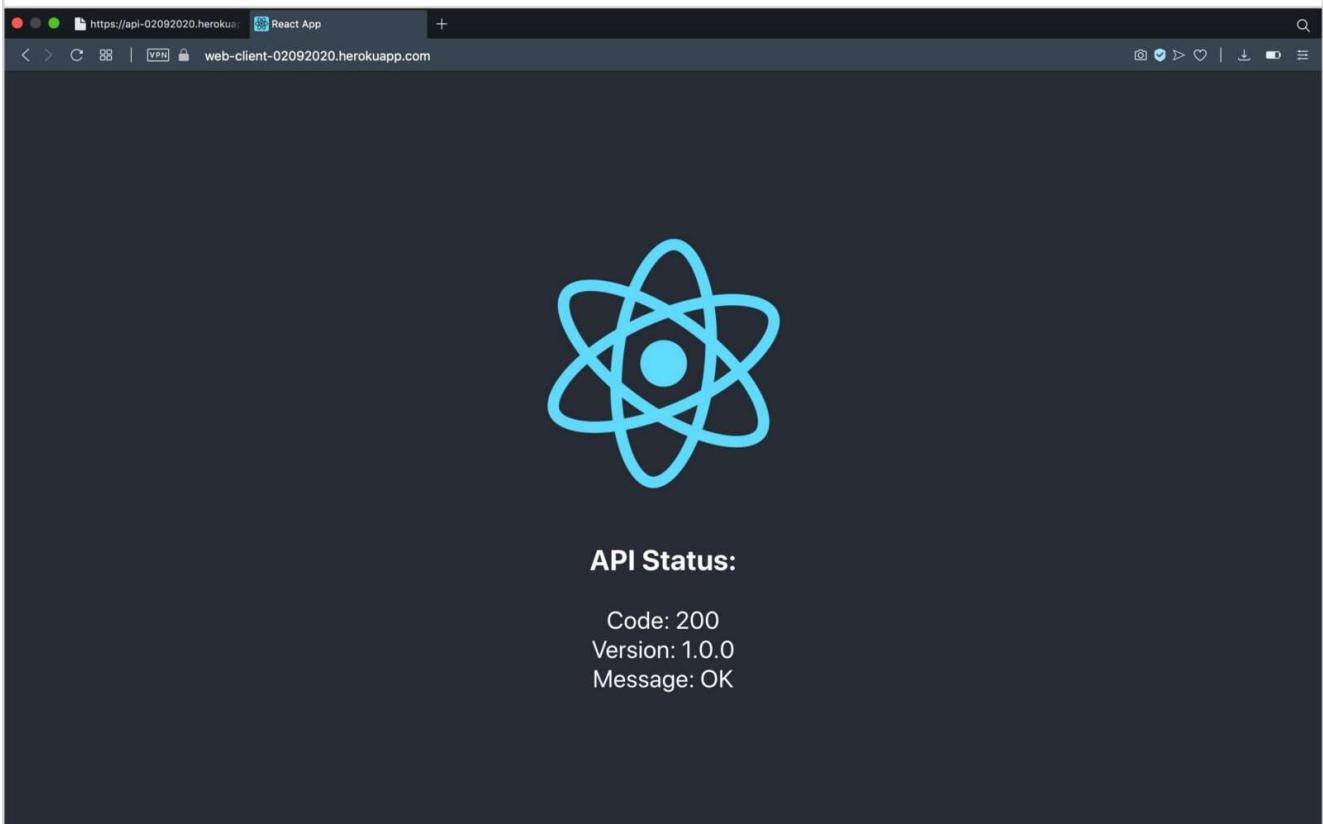
O script é composto de três etapas: Primeiro, ele faz check-out em nosso repositório. Na segunda e terceira etapas, ele usa o plug-in *heroku-deploy* para implantar a API e os aplicativos Web-Client no Heroku e configurar seus ambientes.

Etapa 3: enviar para o GitHub

Assim que terminarmos a implementação do script GitHub Actions, podemos enviar nossas alterações para o GitHub. O fluxo de trabalho deve iniciar automaticamente e nossos aplicativos serão implantados.

Para ver se as implantações foram bem-sucedidas, podemos usar as URLs fornecidas pelo Heroku.

```
< > C | VPN | api-02092020.herokuapp.com/health
{"appName": "API", "version": "1.0.0", "status": "OK"}
```



O código-fonte completo pode ser encontrado aqui:

<https://github.com/zeroabsolute/MonorepoHerokuDeployment>

Inscreva-se para mais como este.

Digite seu e-mail

Se inscrever



Como estruturamos nosso código Terraform

A parte difícil de estruturar um projeto Terraform é que existem várias soluções e não existe um guia universal. A estrutura do código dependeria muito do tamanho do projeto,...

28 de abril de 2022 — 5 minutos de leitura

Tecnologias Softup - Blog © 2022

Serviços. tecnologias. perícia. portfólio. carreira.

Alimentado por Fantasma

