Deploy de uma aplicação

Plataforma utilizada

Temos algumas opções conhecidas e que podemos utilizar para fazer o deploy com plano gratuito para uma aplicação com Flask e Python. O Render, PythonAnywhere, Railway e algumas outras.

Render

Render é uma plataforma de hospedagem em nuvem que simplifica o processo de deployment e gerenciamento de aplicações web, APIs e serviços. Com uma interface fácil de usar, Render permite que desenvolvedores implementem suas aplicações rapidamente, sem a necessidade de gerenciar a infraestrutura subjacente.

Render é uma ótima opção para desenvolvedores que desejam uma solução de hospedagem ágil e eficiente, com menos complexidade na configuração e manutenção.

PythonAnywhere

PythonAnywhere é um ambiente de desenvolvimento integrado e serviço de hospedagem web baseado na linguagem de programação Python. Ele fornece acesso, por meio de navegador web, ao Python baseado em servidor e interfaces de linha de comando Bash, juntamente com um editor de código com realce de sintaxe.

Ele funciona como um SaaS. O modelo SaaS permite às empresas usar softwares hospedados na nuvem na modalidade de assinatura, evitando custos com hardware e atualizações.

Railway

Railway é uma plataforma moderna de desenvolvimento e implantação que torna o processo de criar, gerenciar e escalar aplicações mais acessível. Com uma interface intuitiva, permite que desenvolvedores configurem seus projetos rapidamente, sem complicações. A implantação é facilitada por meio de comandos simples e integração com repositórios Git, o que otimiza o fluxo de trabalho.

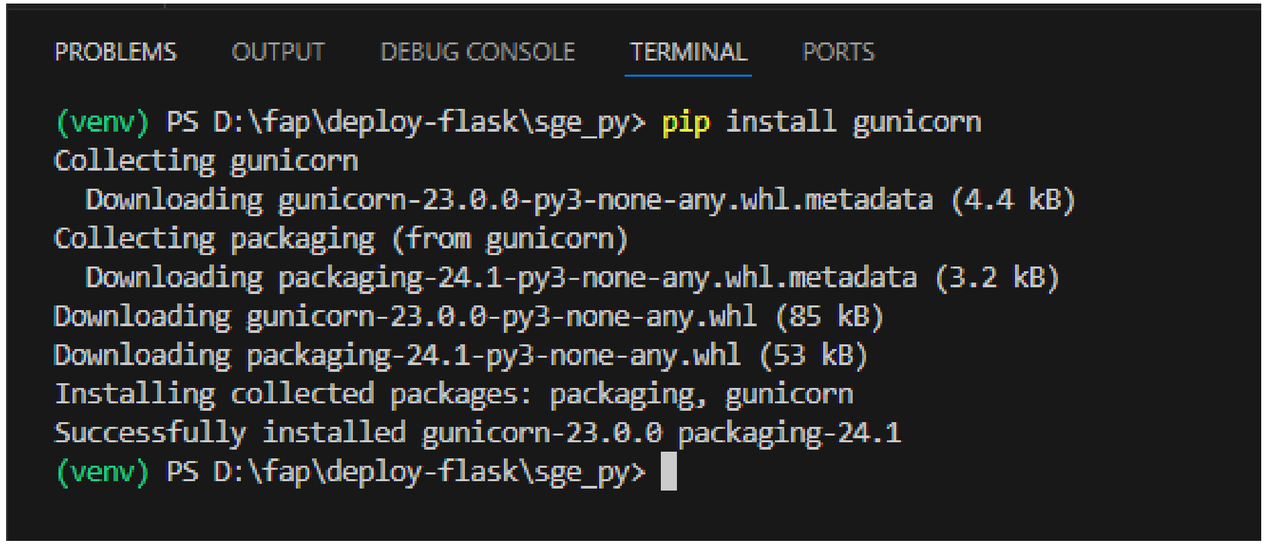
Embora seja popular para aplicações Node.js, Railway também suporta Python, Ruby, Go e outras linguagens, tornando-se uma opção versátil. A plataforma oferece suporte integrado para bancos de dados como PostgreSQL e MySQL, permitindo a configuração e gestão direta dos bancos.

Código e configurações necessárias para o deploy com Render

Clone o repositório no link https://github.com/andrelbribeiro/sge_py.git.
Após isso apague o ambiente virtual que está no repositório (/venv), e crie um novo ambiente virtual: python -m venv venv
Ative o ambiente virtual : .\venv\Scripts\activate
Ou faça usando alguma outra biblioteca

E instale o gunicorn: pip install gunicorn
O gunicorn será responsável por gerenciar seu aplicativo em produção.

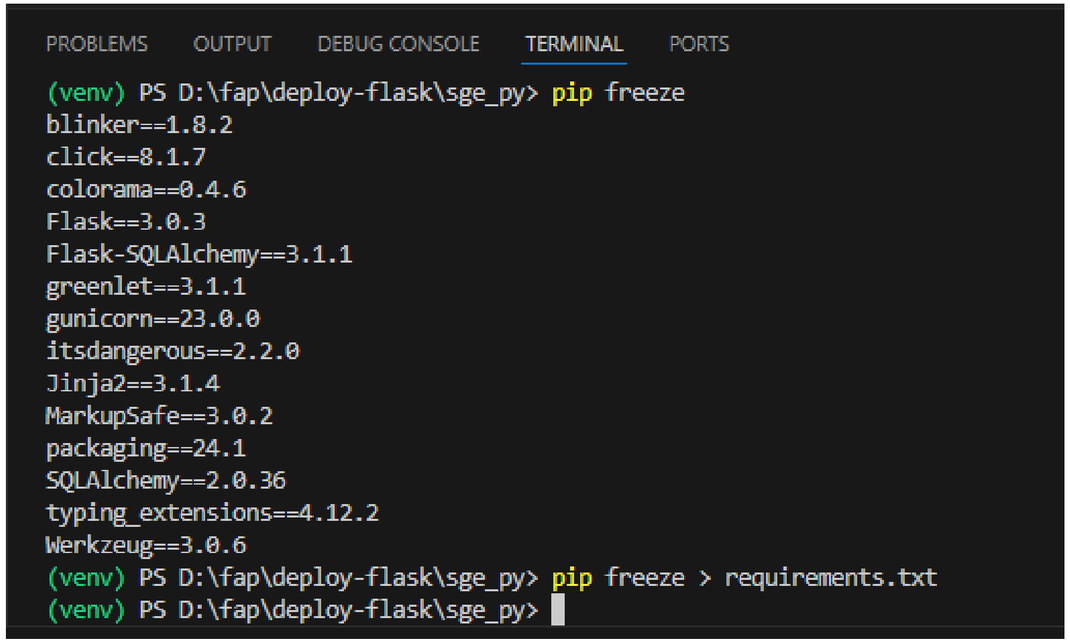
Instale também o Flask, SQLAlchemy e Flask-SQLAlchemy nesse ambiente caso tenha feito o clone: pip install Flask SQLAlchemy Flask-SQLAlchemy

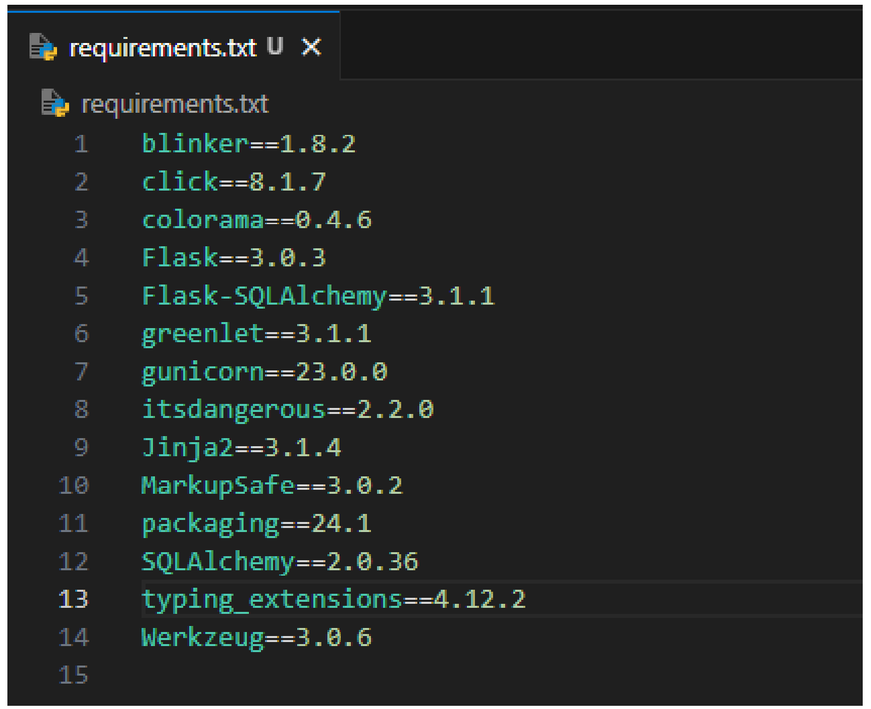


Agora você precisará criar um arquivo com todas as bibliotecas instaladas em seu ambiente de desenvolvimento para que o Render possa utilizá-las no ambiente de produção.

1. Crie um arquivo chamado “requirements.txt” na raiz do projeto.
2. Em seu ambiente rode o comando: pip freeze

Você pode copiar e colar todos as dependências que foram mostradas com o comando anterior ou rodar o comando: pip freeze > requirements.txt





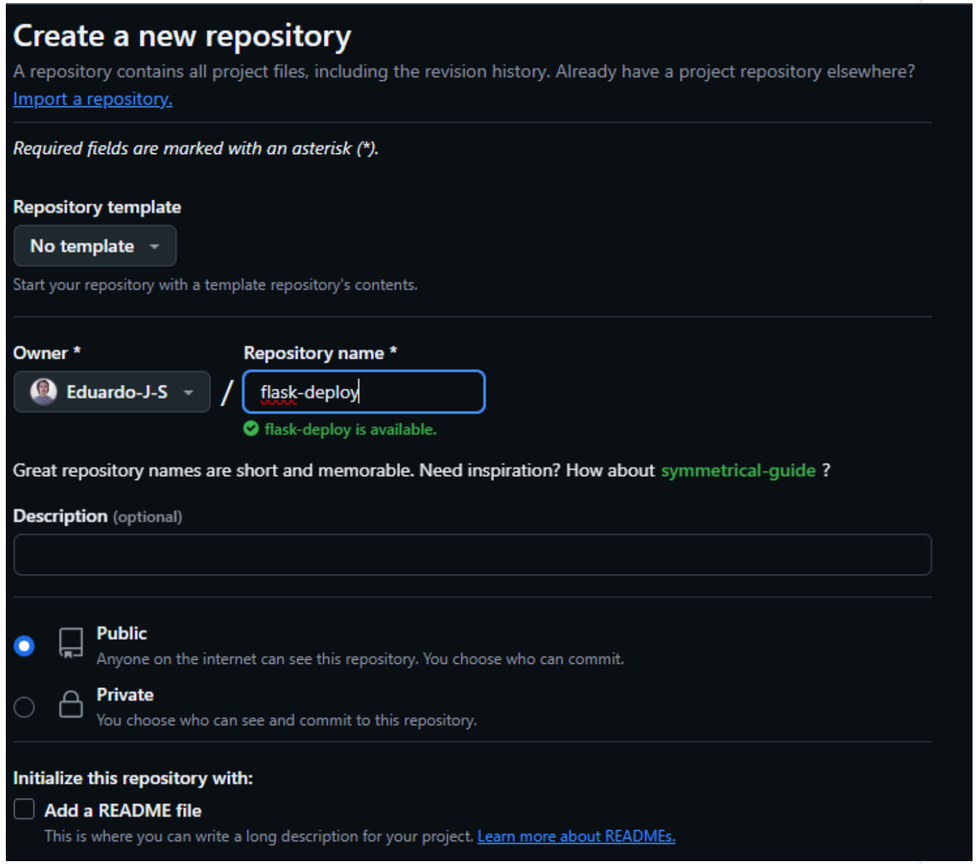
Modifique o arquivo app.py para que ele crie a aplicação com a função criar_app() e exponha a instância da aplicação. Faça isso removendo a linha app.run(debug=True) de dentro da função criar_app e criando uma instância app fora do if __name__ == '__main__'

app.py > ...

```python
from flask import Flask    Import "flask" could not be resolved
from models import db
from config import Config
from controllers.usuario_controller import usuario_bp
from controllers.produto_controller import produto_bp
from controllers.cliente_controller import cliente_bp
from controllers.pedido_controller import pedido_bp
from controllers.detalhepedido_controller import detalhePedido_bp

# Função para criar a aplicação
def criar_app():
    # Instância do Flask
    app = Flask(__name__)
    app.config.from_object(Config)
    db.init_app(app)

    with app.app_context():
        db.create_all()

    app.register_blueprint(usuario_bp)
    app.register_blueprint(produto_bp)
    app.register_blueprint(cliente_bp)
    app.register_blueprint(pedido_bp)
    app.register_blueprint(detalhePedido_bp)

    return app

# Instância da aplicação
app = criar_app()

if __name__ == '__main__':
    app.run(debug=True)

```

Adicionamos o arquivo que criamos para commit com: git add requirements.txt app.py
E fazemos o commit: git commit -m “add requirements.txt and changed app.py”
Se nosso projeto ainda não estiver versionado podemos criar um novo repositório no Github antes de subir

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

(venv) PS D:\fap\deploy-flask\sge_py> pip freeze > requirements.txt
(venv) PS D:\fap\deploy-flask\sge_py> git add requirements.txt app.py
(venv) PS D:\fap\deploy-flask\sge_py> git commit -m "add requirements.txt and changed app.py"
(venv) PS D:\fap\deploy-flask\sge_py>
```

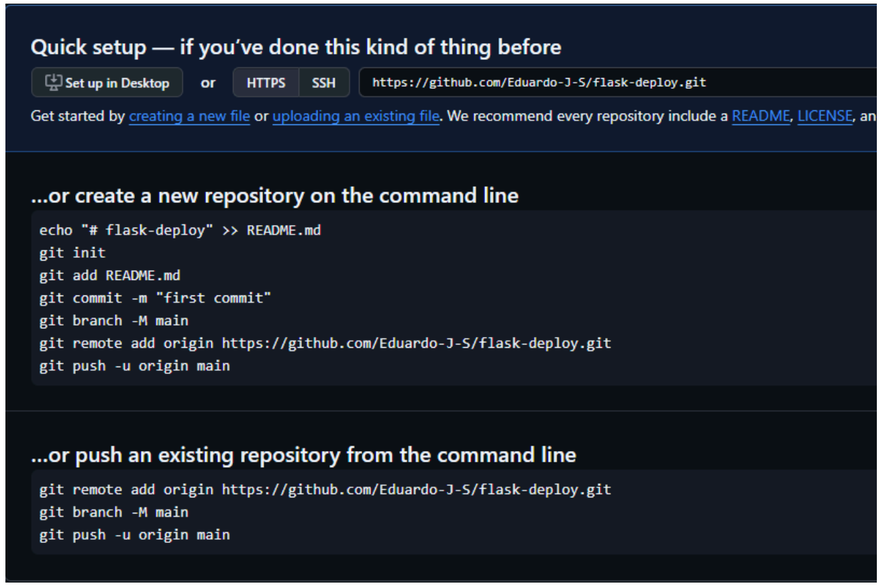


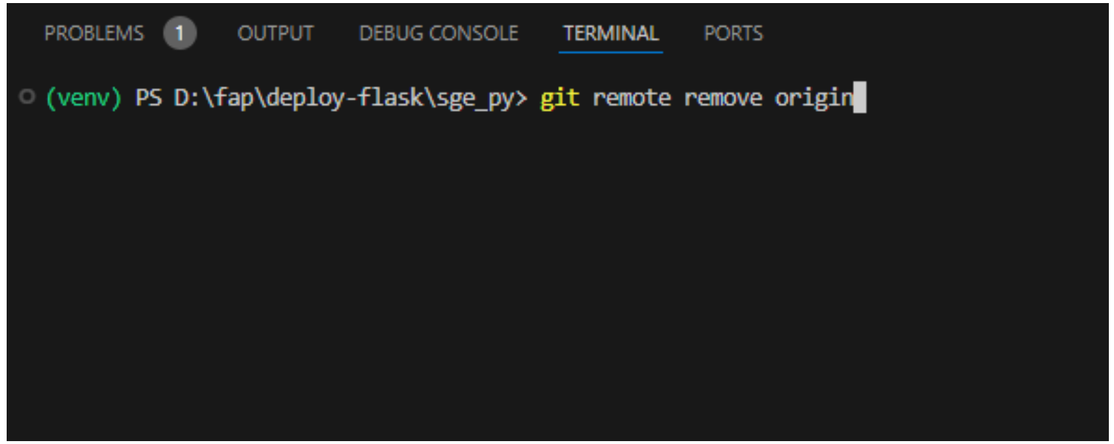
Remova o link para o repositório remoto original (para evitar acidentalmente fazer commits no repositório original): git remote remove origin

Definimos a branch: git branch -M main

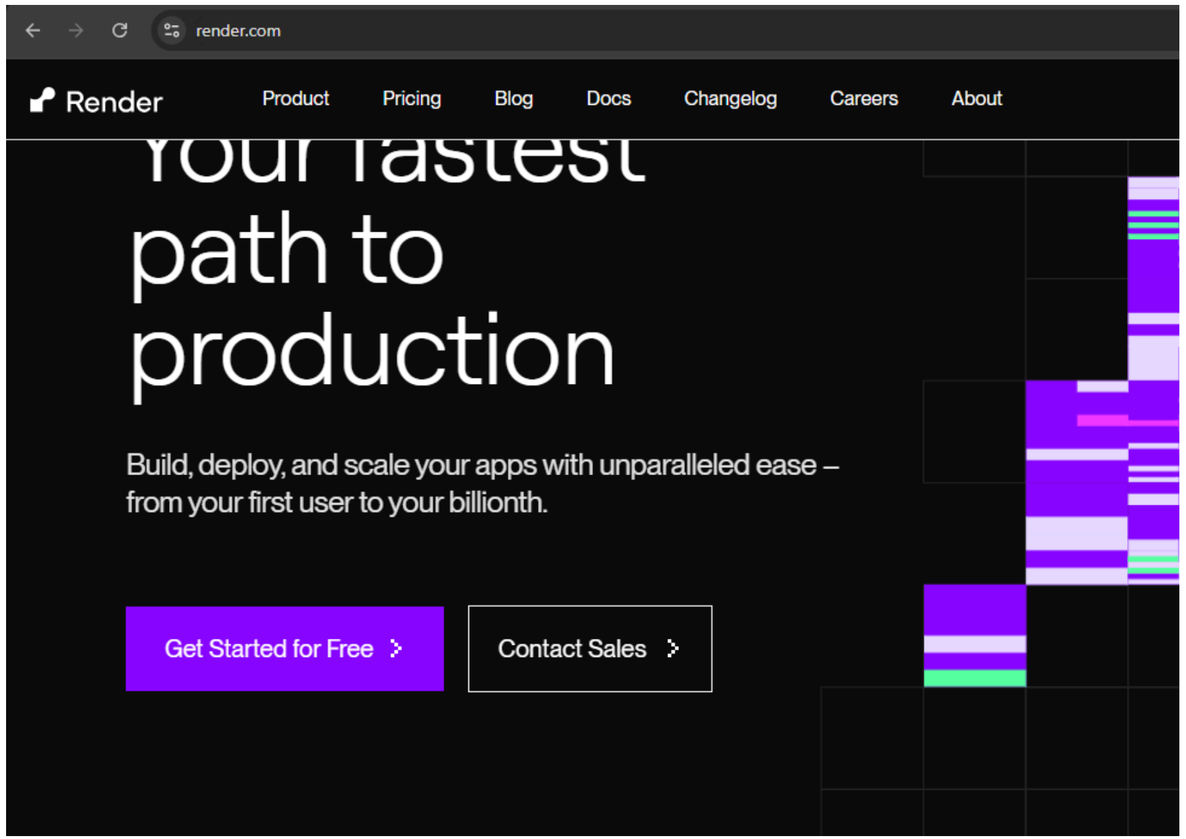
Adicione o link do seu novo repositório como origin: git remote add origin <url-do-seu-repositorio>

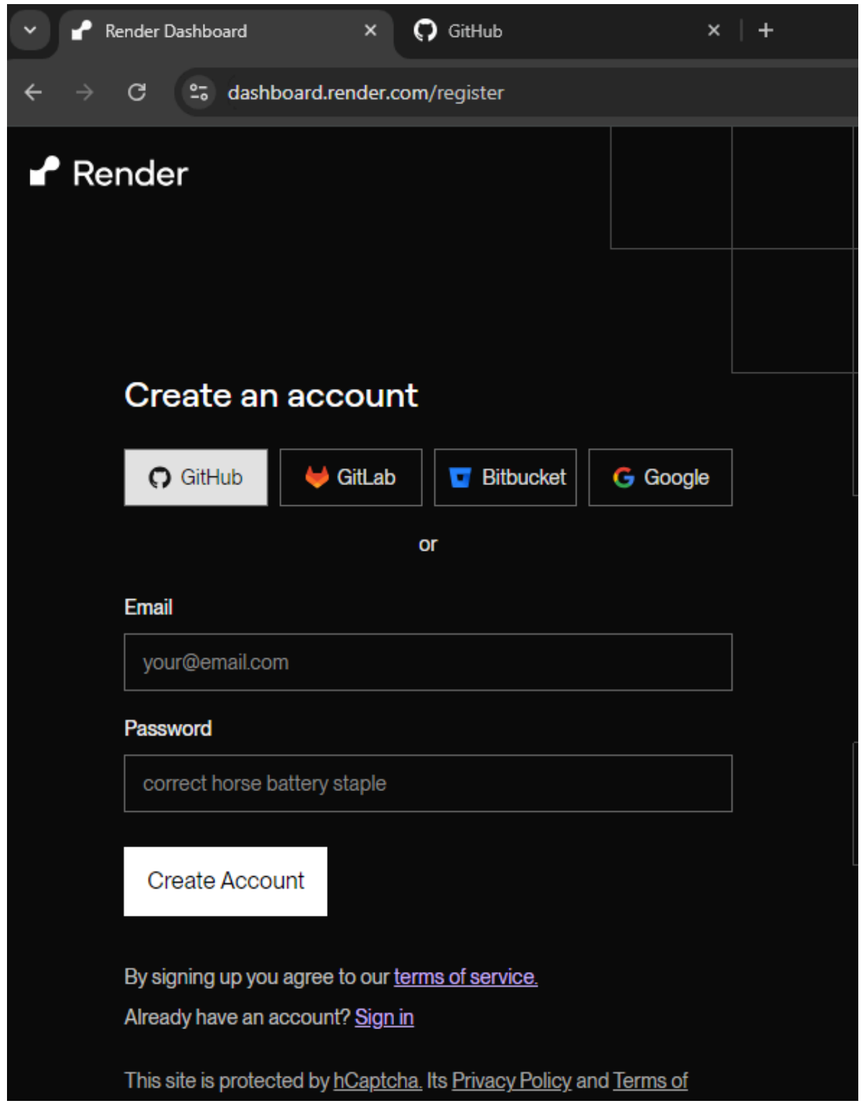
Suba o código para o seu repositório: git push -u origin main



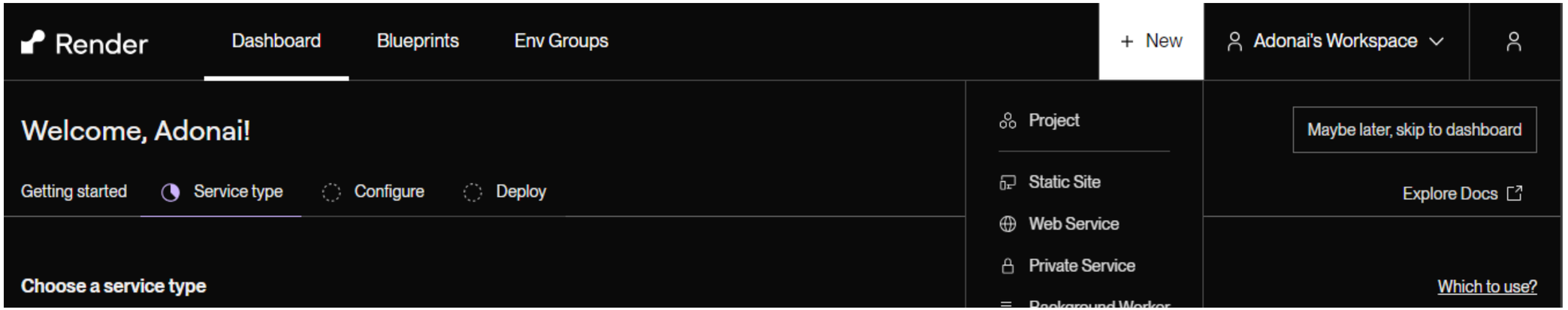


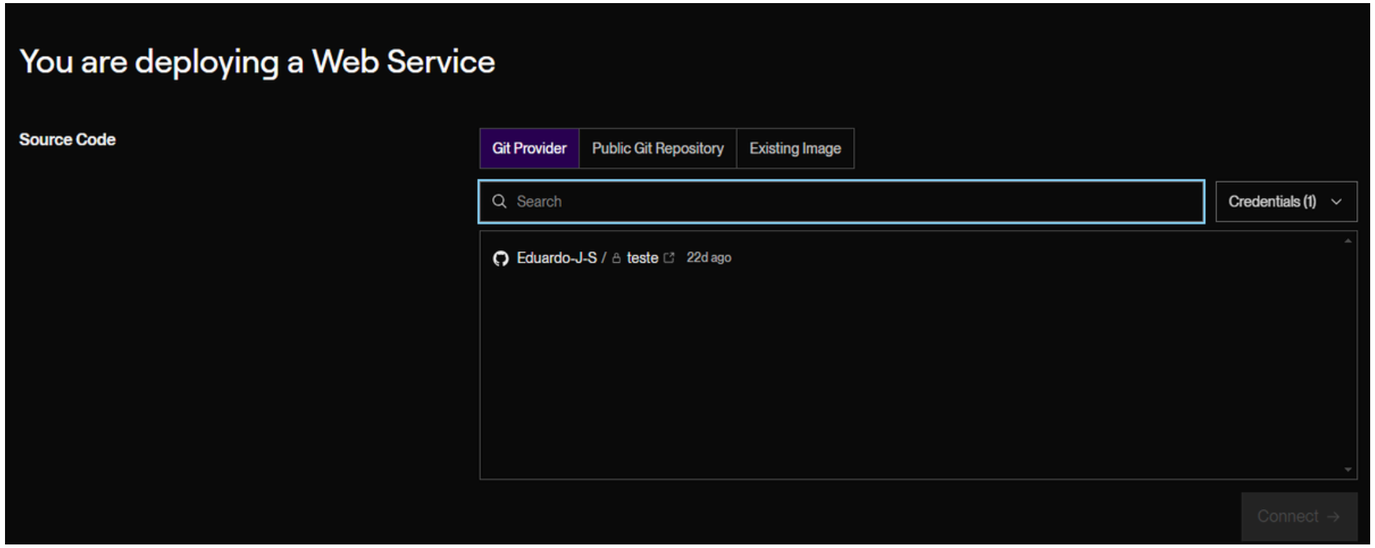
Agora vamos criar uma conta no Render. Primeiro entramos no site do Render e então podemos criar nossa conta com o próprio GitHub que ele já integra com os nossos repositórios



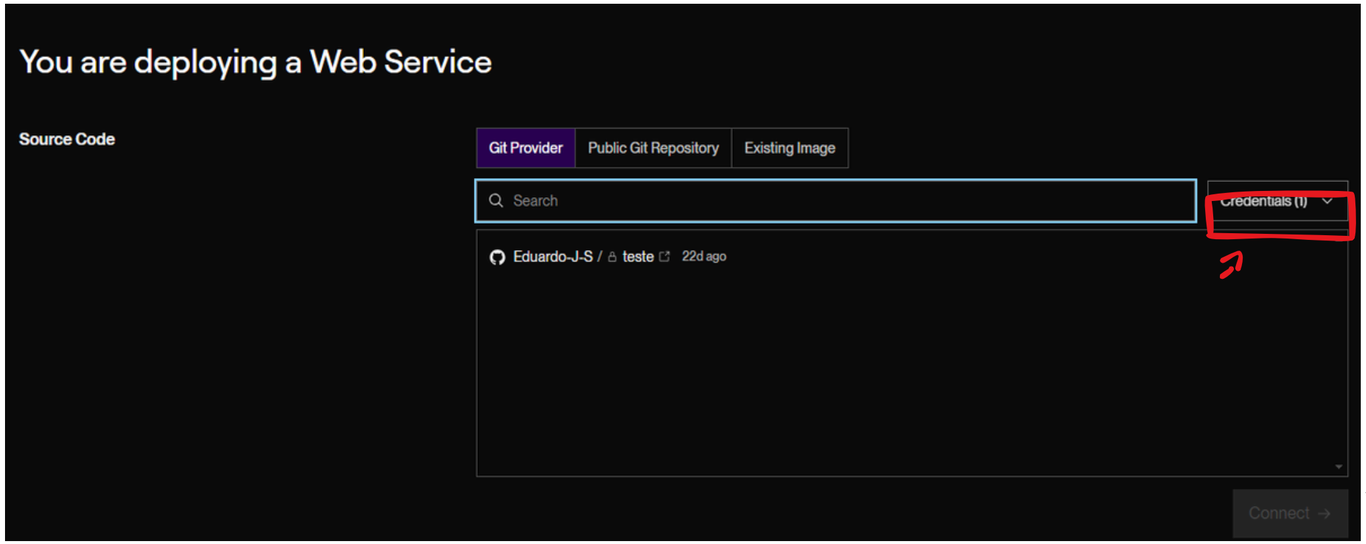


Próximo passo é clicar em “New” no topo da página, em seguida em “web service“. Se o GitHub estiver conectado ele vai reconhecer os repositórios. Então é só selecionar o que acabamos de subir.

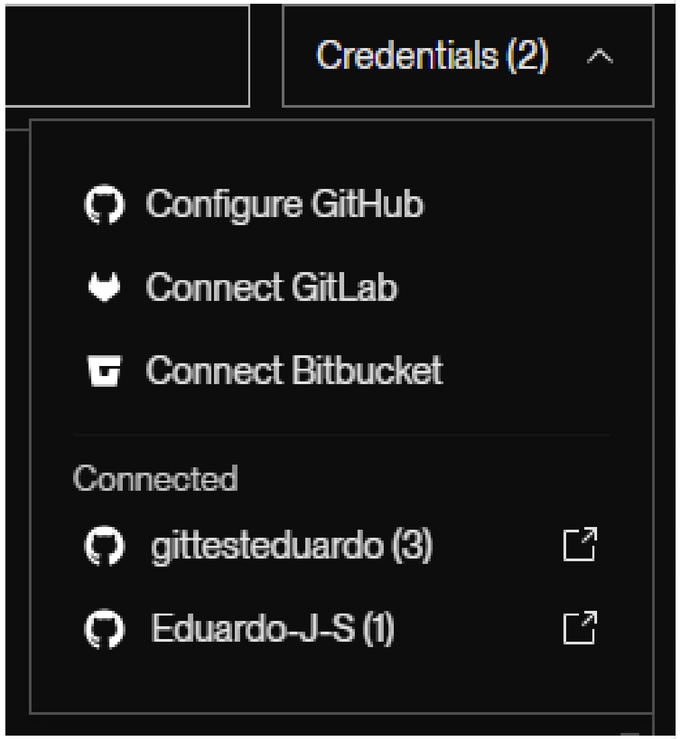




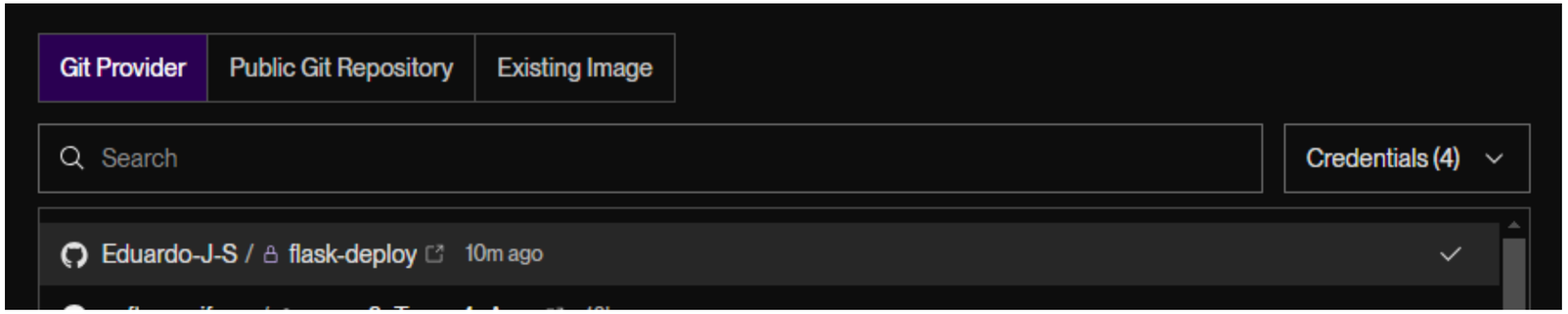
Se os seus repositórios não estiverem aparecendo automaticamente, você irá precisar adicionar as credencias. Clique onde está marcado, depois selecione a opção “Configure GitHub”, selecione a credencial onde está seu repositório e depois clique em “install”



1

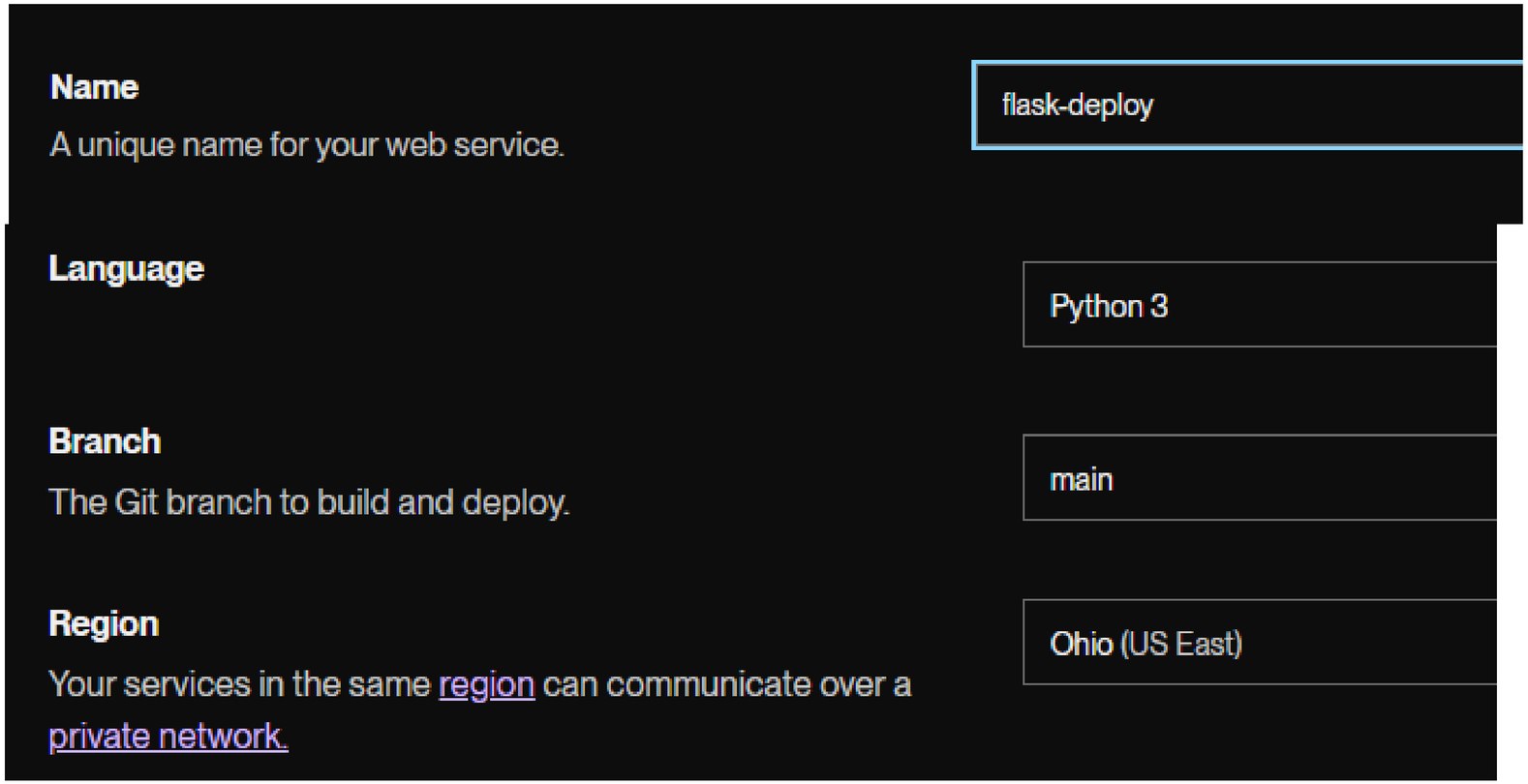


2



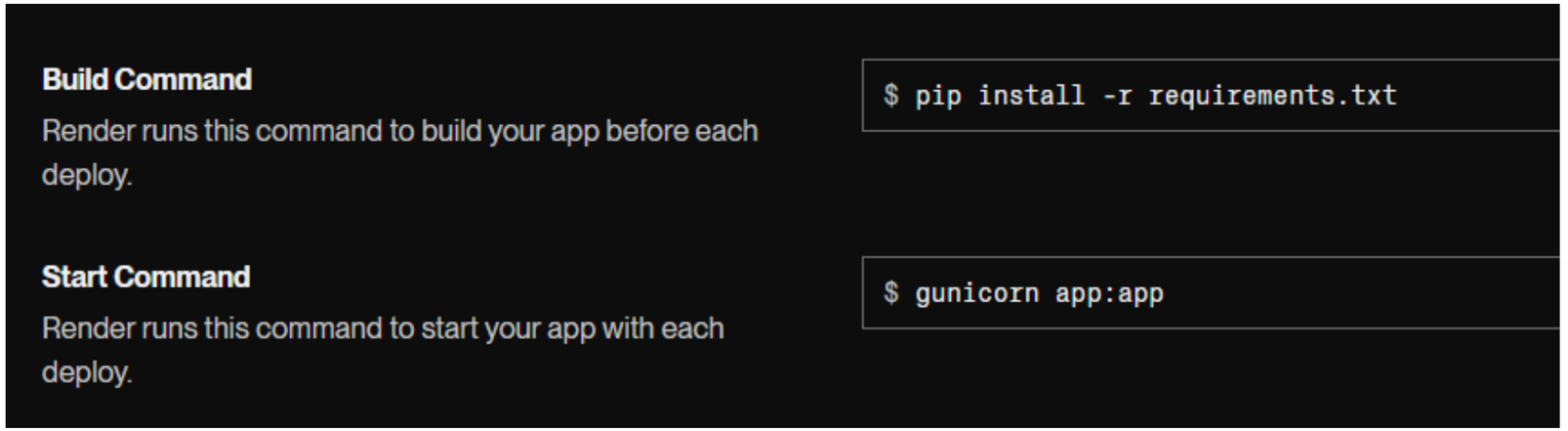
3

Configurações básicas para o deploy. Defina nome, linguagem, branch e região.

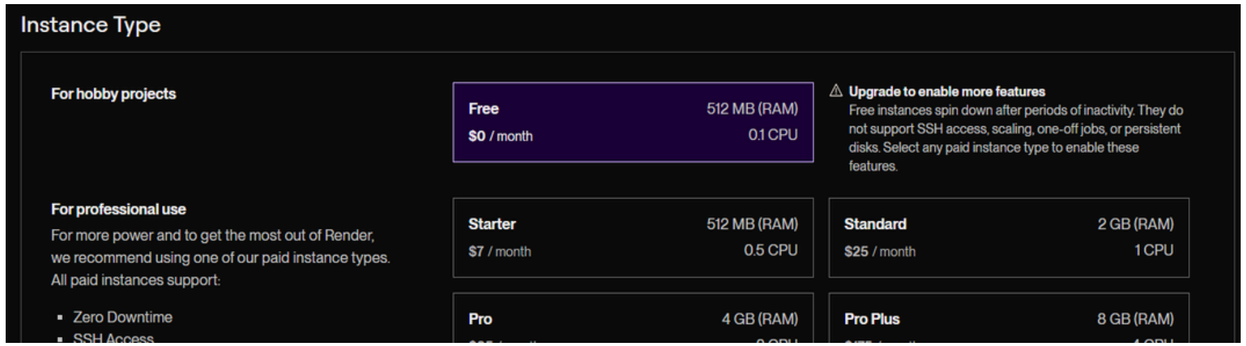


Este primeiro campo é basicamente para instalar as dependências e preparar o ambiente da aplicação para rodar. No caso da aplicação, vamos instalar as bibliotecas especificadas no arquivo requirements.txt.
Esse comando instalará todas as dependências necessárias para que o projeto funcione corretamente.

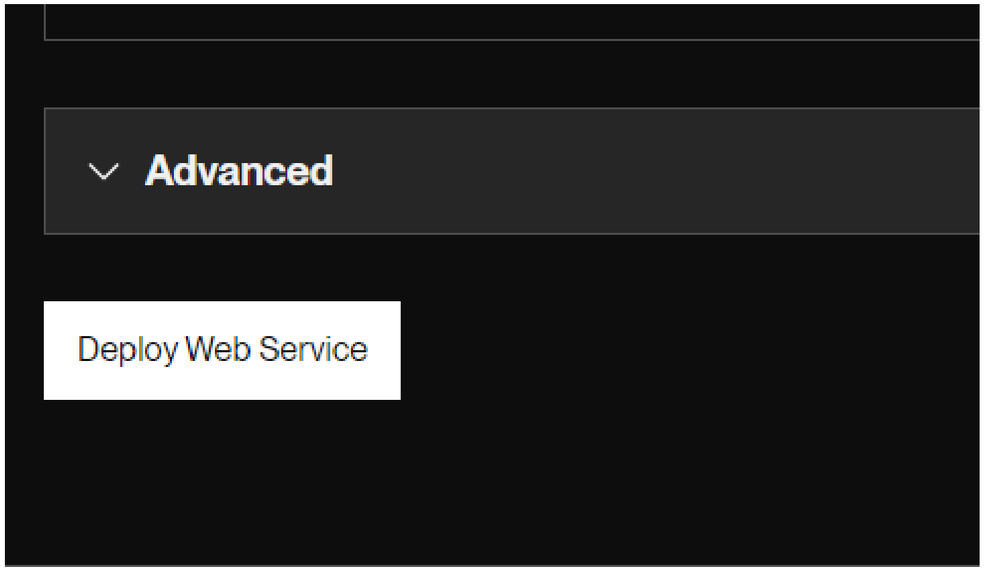
No campo "Start Command", configuramos o comando para iniciar o servidor em produção. Usamos o gunicorn como servidor WSGI para rodar a aplicação de maneira eficiente. Aqui, app:app indica o caminho do módulo (app) e a instância da aplicação Flask (app).



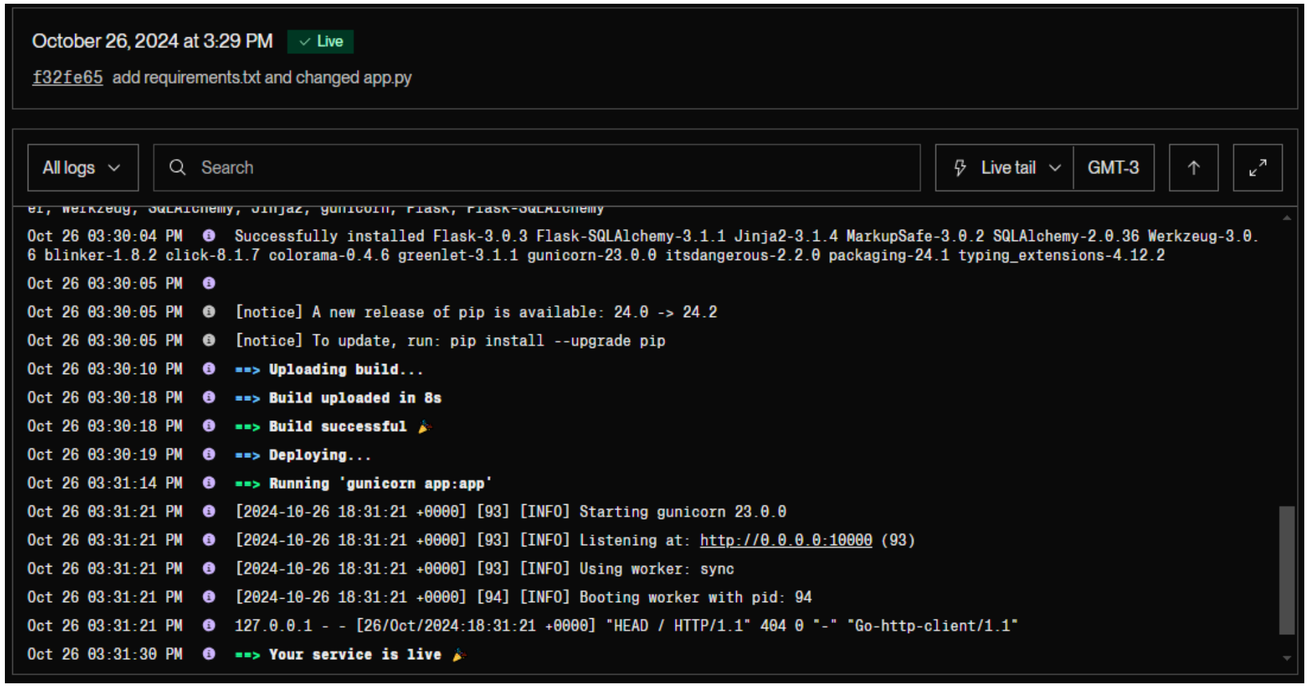
Selecionamos o plano FREE



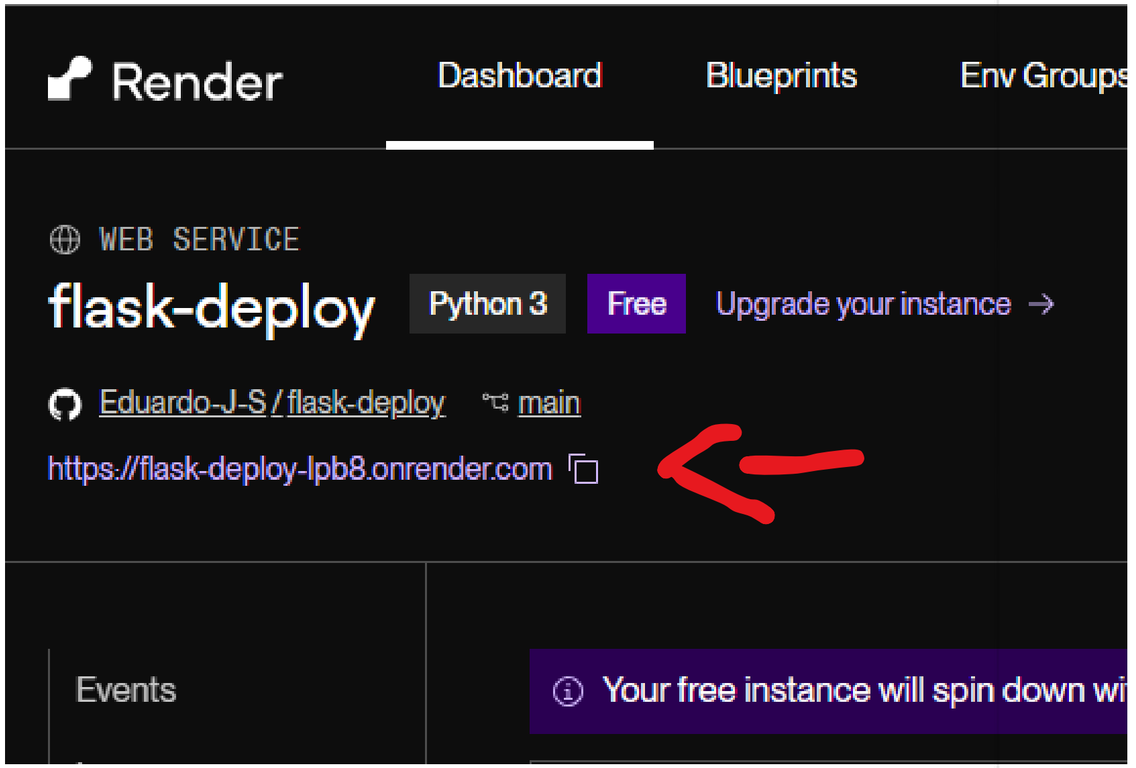
E criamos o deploy

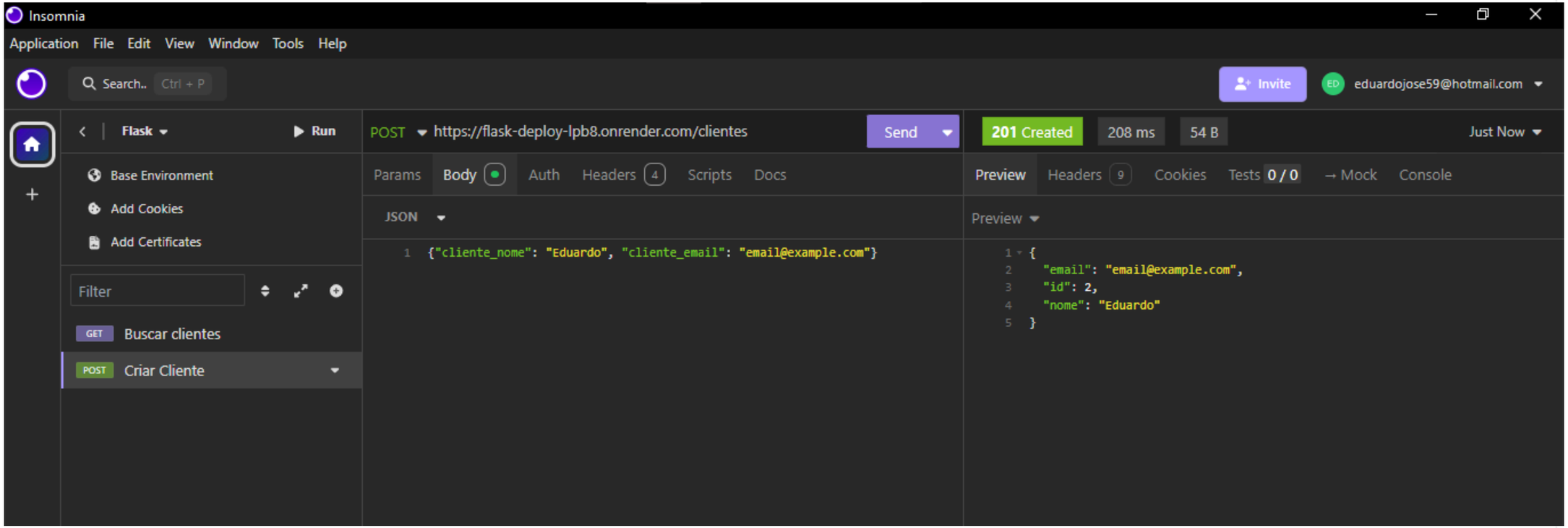


Log de quando ele subir a aplicação será assim.

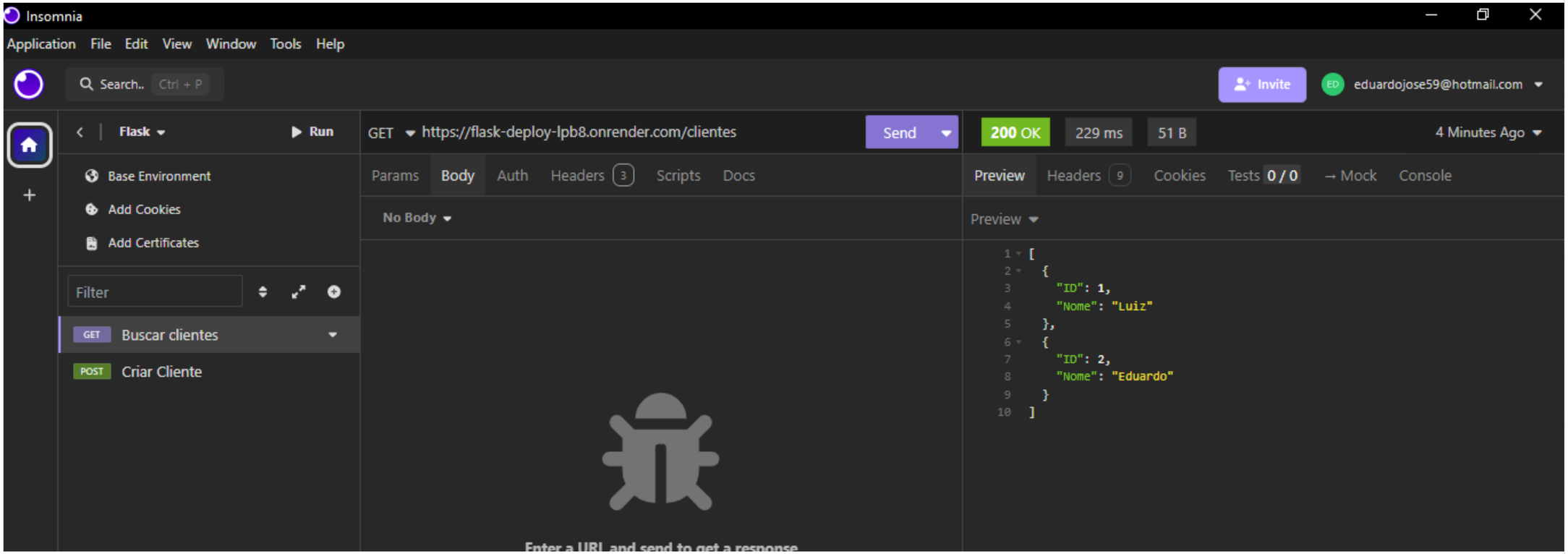


Podemos então copiar a URL e fazer as requisições





Exemplo de uma requisição POST de criação de cliente



Exemplo de uma requisição GET para buscar todos os cliente