

Guia Completo: Deploy em Produção

Data Analyzer - Sistema de Análise de Formulários Google

Pré-requisitos

- Conta no GitHub
 - Conta no Google Cloud Console
 - Conta no Brevo (envio de emails)
 - Conta no Gemini AI (Google)
 - Código do projeto funcionando localmente
-

Escolha da Plataforma de Deploy

Recomendamos **Render.com** pela facilidade de uso e tier gratuito generoso.

Alternativas:

- **Railway.app** - Ótimo para iniciantes, PostgreSQL integrado
 - **Heroku** - Clássico, mas tier gratuito limitado
 - **Google Cloud Run** - Mais avançado, escalável
 - **PythonAnywhere** - Simples, mas com limitações
-

PARTE 1: Preparação do Código

1.1. Criar arquivo `Procfile` (para Render/Heroku)

Crie na raiz do projeto:

```
web: gunicorn run:app --bind 0.0.0.0:$PORT
```

1.2. Atualizar `requirements.txt`

Adicione o Gunicorn:

```
Flask==2.3.2
Flask-SQLAlchemy==3.0.5
Flask-Bcrypt==1.0.1
Flask-Login==0.6.2
psycpg2-binary==2.9.6
pandas==2.0.3
scikit-learn==1.3.0
google-auth==2.22.0
google-auth-oauthlib==1.0.0
google-auth-http2==0.1.0
google-api-python-client==2.95.0
google-generativeai==0.3.0
sib-api-v3-sdk==7.6.0
python-dotenv==1.0.0
itsdangerous==2.1.2
gunicorn==21.2.0
```

1.3. Criar arquivo `runtime.txt` (opcional)

```
python-3.11.7
```

1.4. Atualizar `.gitignore`

Certifique-se de que contém:

```
# Ambiente Virtual
venv/
env/

# Arquivos de ambiente com segredos
.env
*.env

# Arquivos compilados do Python
__pycache__/
*.pyc
*.pyo

# Credenciais do Google
client_secret.json
credencial_google.json

# Banco de dados local
*.db
*.sqlite

# IDEs
.vscode/
.idea/
*.swp
*.swo

# Logs
*.log
```

1.5. Subir código para o GitHub

```
bash
```

Inicialize o repositório (se ainda não fez)

`git init`

Adicione todos os arquivos

`git add .`

Faça o primeiro commit

`git commit -m "Preparação para deploy em produção"`

Crie repositório no GitHub e conecte

`git remote add origin https://github.com/seu-usuario/data-analyzer.git`

Envie o código

`git push -u origin main`



PARTE 2: Configurar Banco de Dados PostgreSQL

Opção A: Neon.tech (Recomendado - Gratuito)

1. Acesse <https://neon.tech/>
2. Crie uma conta (pode usar GitHub)
3. Clique em "Create Project"
4. Escolha:
 - **Project name:** data-analyzer-db
 - **Region:** US East (ou mais próxima de você)
5. Copie a **Connection String** (formato: `postgresql://user:pass@host/dbname`)
6. Guarde para usar nas variáveis de ambiente

Opção B: Supabase (Gratuito com features extras)

1. Acesse <https://supabase.com/>
2. Crie um projeto
3. Vá em "Settings" > "Database"
4. Copie a "Connection string" (modo URI)
5. Guarde para as variáveis de ambiente

Opção C: Railway (Integrado com deploy)

1. Será configurado junto com o deploy (veja Parte 4B)

PARTE 3: Configurar Credenciais e APIs

3.1. Google Cloud Console (OAuth + APIs)

Passo 1: Criar Projeto

1. Acesse <https://console.cloud.google.com/>
2. Clique em "Select a project" > "New Project"
3. Nome:
4. Clique em "Create"

Passo 2: Ativar APIs

1. No menu lateral, vá em "APIs & Services" > "Library"
2. Busque e ative (clique "Enable"):
 - **Google Sheets API**
 - **Google Drive API**
 - **Google Forms API**

Passo 3: Configurar OAuth Consent Screen

1. Vá em "APIs & Services" > "OAuth consent screen"
2. Escolha **External** (para qualquer usuário Gmail)
3. Preencha:
 - **App name:** Data Analyzer
 - **User support email:** seu-email@gmail.com
 - **Developer contact:** seu-email@gmail.com
4. Clique em "Save and Continue"
5. Em "Scopes", clique "Add or Remove Scopes" e adicione:
 -
 -
 -
 -
 -
 -
6. Clique "Save and Continue"
7. Em "Test users", adicione seu email (para testes iniciais)
8. Clique "Save and Continue"

Passo 4: Criar Credenciais OAuth 2.0

1. Vá em "APIs & Services" > "Credentials"
2. Clique "Create Credentials" > "OAuth 2.0 Client ID"
3. Escolha "Web application"
4. Configure:
 - **Name:** Data Analyzer Web Client
 - **Authorized JavaScript origins:**

`https://seu-app.onrender.com`

- **Authorized redirect URIs:**

`https://seu-app.onrender.com/google_callback`
`http://localhost:5000/google_callback`

5. Clique "Create"
6. **COPIE e GUARDE:**
 - `Client ID`
 - `Client Secret`
7. Baixe o JSON e salve como `client_secret.json`

3.2. Google Gemini AI (IA para mapeamento de colunas)

1. Acesse <https://makersuite.google.com/app/apikey>.
2. Clique "Create API Key"
3. Escolha o projeto criado anteriormente
4. Copie a API Key gerada
5. Guarde para variáveis de ambiente

3.3. Brevo (Envio de Emails)

1. Acesse <https://www.brevo.com/>
2. Crie conta gratuita (300 emails/dia)
3. Confirme seu email
4. Vá em "Settings" > "SMTP & API"
5. Clique "Generate a new API key"
6. Nome: `data-analyzer-prod`

7. Copie a API Key

8. **IMPORTANTE:** Vá em "Senders" e verifique seu email remetente

9. Guarde a API Key

3.4. Gerar SECRET_KEY

No terminal Python:

```
python

import secrets
print(secrets.token_urlsafe(32))
# Copie o resultado
```

3.5. Gerar WEBHOOK_SECRET_KEY

```
python

import secrets
print(secrets.token_urlsafe(24))
# Copie o resultado
```

PARTE 4: Deploy na Plataforma

Opção A: Render.com (Recomendado)

Passo 1: Preparar client_secret.json

Como o Render não permite upload direto de arquivos, precisamos passar o conteúdo via variável de ambiente.

1. Abra seu arquivo `client_secret.json`
2. Copie TODO o conteúdo (é um JSON)
3. Minimize-o em uma linha usando: <https://jsonformatter.org/json-minify>

Passo 2: Criar Web Service no Render

1. Acesse <https://dashboard.render.com/>
2. Crie conta (pode usar GitHub)
3. Clique "New +" > "Web Service"
4. Conecte seu repositório GitHub
5. Configure:
 - **Name:** `data-analyzer`
 - **Region:** Oregon (ou mais próxima)

- **Branch:** `main`
- **Runtime:** `Python 3`
- **Build Command:** `pip install -r requirements.txt`
- **Start Command:** `gunicorn run:app --bind 0.0.0.0:$PORT`
- **Instance Type:** `Free`

Passo 3: Configurar Variáveis de Ambiente

Na seção "Environment Variables", clique "Add Environment Variable" e adicione:

```
SECRET_KEY=sua-chave-secreta-gerada-anteriormente

DATABASE_URL=postgresql://user:pass@host/dbname
(cole a URL do Neon/Supabase)

GOOGLE_CLIENT_ID=seu-client-id.apps.googleusercontent.com

GOOGLE_CLIENT_SECRET=sua-client-secret

GEMINI_API_KEY=sua-api-key-gemini

BREVO_API_KEY=sua-api-key-brevo

WEBHOOK_SECRET_KEY=sua-webhook-secret-key

GOOGLE_CLIENT_SECRET_JSON={"web":{"client_id":"...", "project_id":"...", ...}}
(cole o JSON minificado do client_secret.json)
```

Passo 4: Modificar código para ler JSON da variável de ambiente

Crie arquivo `app/google_credentials.py`:

```
python
```



```
import os
import json

def get_google_client_secret():
    """Retorna o caminho ou conteúdo do client_secret"""

    # Tenta ler da variável de ambiente (produção)
    json_content = os.environ.get('GOOGLE_CLIENT_SECRET_JSON')
    if json_content:
        # Salva temporariamente
        with open('/tmp/client_secret.json', 'w') as f:
            f.write(json_content)
        return '/tmp/client_secret.json'

    # Fallback para arquivo local (desenvolvimento)
    return 'client_secret.json'
```

Em `app/routes.py`, atualize as funções que usam `client_secret.json`:

```
python

from app.google_credentials import get_google_client_secret

# Substitua 'client_secret.json' por:
flow = Flow.from_client_secrets_file(
    get_google_client_secret(),
    scopes=SCOPES,
    redirect_uri=url_for('main.google_callback', _external=True)
)
```

Passo 5: Deploy

1. Clique "Create Web Service"
2. Aguarde o build (5-10 minutos)
3. Acesse a URL gerada: `https://data-analyzer-xxxx.onrender.com`

Passo 6: Atualizar Google OAuth

1. Volte no Google Cloud Console
2. Edite as credenciais OAuth
3. Adicione a URL do Render nos redirects:

`https://data-analyzer-xxxx.onrender.com/google_callback`

Opção B: Railway.app (Mais Simples)

Passo 1: Deploy

1. Acesse <https://railway.app/>
2. Faça login com GitHub
3. Clique "New Project" > "Deploy from GitHub repo"
4. Selecione seu repositório
5. Railway detecta automaticamente que é Python/Flask

Passo 2: Adicionar PostgreSQL

1. No projeto, clique "New" > "Database" > "Add PostgreSQL"
2. Railway cria e conecta automaticamente
3. A variável `DATABASE_URL` é criada automaticamente

Passo 3: Configurar Variáveis de Ambiente

1. Clique em "Variables"
2. Adicione as mesmas variáveis da Opção A

Passo 4: Configurar Domain

1. Clique em "Settings"
2. Em "Domains", clique "Generate Domain"
3. Copie a URL gerada

Passo 5: Atualizar Google OAuth

Adicione a URL do Railway nos redirects do Google Cloud Console.

PARTE 5: Validação e Testes

5.1. Checklist de Funcionamento

- ☐ Site carrega na URL de produção
- ☐ Página de registro funciona
- ☐ Email de confirmação chega
- ☐ Login funciona com verificação por email
- ☐ Botão "Conectar com Google" funciona
- ☐ OAuth do Google redireciona corretamente

- ☐ Busca de formulários no Drive funciona
- ☐ Processamento de formulários funciona
- ☐ Gráficos aparecem no dashboard
- ☐ Análise descritiva funciona
- ☐ Análise de ML (K-Means) funciona

5.2. Teste de Fluxo Completo

1. Criar conta

- Acesse </register>
- Preencha dados
- Confirme email

2. Conectar Google

- Faça login
- Clique "Conectar com Google"
- Autorize as permissões
- Verifique se mostra "Conectado como: seu@email.com"

3. Adicionar formulário

- Clique "Gerenciar Fontes de Dados"
- Clique "Buscar Formulários no meu Google Drive"
- Selecione um formulário com respostas
- Processe os dados

4. Visualizar análises

- Vá em "Visualizar Dados"
- Vá em "Análise Descritiva"
- Vá em "Análise com ML"

5.3. Monitoramento de Erros

No Render:

1. Vá em "Logs" no dashboard
2. Filtre por "Error" ou "Exception"

No Railway:

1. Clique no serviço
2. Aba "Deployments"
3. Clique no deployment ativo > "View Logs"

🔧 PARTE 6: Otimizações Pós-Deploy

6.1. Configurar Domínio Personalizado (Opcional)

No Render:

1. Compre domínio (ex: Namecheap, Google Domains)
2. No Render, vá em "Settings" > "Custom Domain"
3. Adicione seu domínio: `www.seudominio.com`
4. Configure DNS do seu provedor:

Type: CNAME

Name: www

Value: data-analyzer-xxxx.onrender.com

6.2. Habilitar HTTPS (Automático)

Render e Railway habilitam HTTPS automaticamente com Let's Encrypt.

6.3. Configurar Backup do Banco de Dados

No Neon:

1. Dashboard > Settings > Backups
2. Habilite "Point-in-time restore"

No Supabase:

1. Backups automáticos já estão ativos
2. Retenção de 7 dias no plano gratuito

6.4. Monitoramento e Alertas

Use **UptimeRobot** (gratuito):

1. Acesse <https://uptimerobot.com/>
2. Crie monitor HTTP(s)
3. URL: `https://seu-app.onrender.com/health`
4. Configure alertas por email

6.5. Melhorias de Performance

Adicione em `app/__init__.py`:

```
python

from flask_caching import Cache

cache = Cache(config={'CACHE_TYPE': 'simple'})

def create_app(config_class=Config):
    app = Flask(__name__)
    app.config.from_object(config_class)

    # ... código existente ...

    cache.init_app(app)

    return app
```

PARTE 7: Publicar para Usuários

7.1. Tirar Google OAuth do Modo Teste

1. Volte no Google Cloud Console
2. "APIs & Services" > "OAuth consent screen"
3. Clique em "Publish App"
4. Leia os termos e confirme
5. Agora qualquer usuário Gmail pode usar

7.2. Aumentar Limites de Email

No Brevo:

1. Verifique domínio próprio (aumenta credibilidade)
2. Considere upgrade se precisar mais de 300 emails/dia

7.3. Criar Página de Ajuda

Adicione rota em `app/routes.py`:

```
python

@main.route("/ajuda")
def ajuda():
    return render_template('ajuda.html', title="Central de Ajuda")
```

Crie `app/templates/ajuda.html` com tutoriais.

Troubleshooting Comum

Erro: "Application Error" no Render

Causa: Build falhou ou variáveis de ambiente incorretas

Solução:

1. Verifique logs em "Logs" > "Deploy Logs"
2. Confirme todas as variáveis de ambiente
3. Verifique se `Procfile` está correto

Erro: "invalid_grant" no OAuth

Causa: URL de redirect não configurada no Google

Solução:

1. Vá no Google Cloud Console
2. Edite credenciais OAuth
3. Adicione EXATAMENTE a URL de produção

Erro: "This app is blocked"

Causa: OAuth consent screen não publicado

Solução: Publique o app (Parte 7.1)

Erro: Banco de dados não conecta

Causa: DATABASE_URL incorreta

Solução:

1. Verifique se copiou URL completa
2. Certifique-se que começa com `postgresql://`
3. Teste conexão localmente primeiro

Gráficos não aparecem

Causa: Chart.js não carregou ou dados inválidos

Solução:

1. Abra DevTools (F12) > Console
 2. Verifique erros de JavaScript
 3. Confirme que `grafico_labels` e `grafico_values` existem
-



Próximos Passos

1. **Analytics:** Integre Google Analytics
 2. **Logs Estruturados:** Use Sentry ou Logtail
 3. **Testes Automatizados:** Adicione pytest
 4. **CI/CD:** Configure GitHub Actions
 5. **Documentação:** Crie README completo
-



Suporte

Se encontrar problemas:

1. Verifique logs da plataforma
 2. Teste localmente primeiro
 3. Confirme todas as credenciais
 4. Consulte documentação oficial:
 - [Render Docs](#)
 - [Railway Docs](#)
 - [Flask Docs](#)
-



Parabéns! Sua aplicação está em produção!

Compartilhe a URL com seus usuários e monitore o uso nos primeiros dias para identificar possíveis ajustes.