Guia Rápido — Criar, Executar e Publicar um Aplicativo Streamlit

Paulo César Crepaldi

Versão: 14 de dezembro de 2025

Sumário

Objetivo

Este guia apresenta um procedimento **genérico, direto e reprodutível** para:

- criar a estrutura mínima de um app Streamlit;
- executar localmente; e
- publicar no *Streamlit Community Cloud*.

O foco é reduzir falhas típicas de *deploy* relacionadas a dependências e credenciais.

1 Pré-requisitos

Para seguir este tutorial, você precisa de:

- Python instalado e disponível no terminal (`python -version`);
- Git instalado (`git -version`);
- Uma conta no GitHub;
- Acesso ao Streamlit Community Cloud.

Dica prática

Se o comando `python` não for reconhecido no Windows, verifique se o Python foi adicionado ao *PATH* durante a instalação.

2 Estrutura mínima do projeto

A estrutura abaixo é suficiente para executar o aplicativo localmente e realizar *deploy* no Streamlit Community Cloud. Recomenda-se que essa estrutura seja mantida em um repositório *GitHub*, com versionamento do código-fonte e dos arquivos de configuração necessários ao *deploy*. No Streamlit Community Cloud, a publicação é realizada a partir da integração com o repositório (seleção do *branch* e do arquivo principal, por exemplo `app.py`).

```
nome-do-repositorio/
|
|-- .gitignore # [CRITICO] Seguranca: define o que nao vai para o GitHub
|-- README.md # Documentacao: resumo, instalacao e como usar
|-- requirements.txt # Dependencias: lista de bibliotecas (pandas, etc.)
|
|-- app.py # Arquivo principal que o Streamlit executa
|
|-- src/ # Codigo-fonte: funcoes e modulos
| |-- __init__.py # Indica que esta pasta e um pacote Python
| |-- processamento.py # Limpeza e tratamento de dados
| |-- graficos.py # Geracao de mapas ou graficos
|
|-- assets/ # Arquivos estaticos (visual)
| |-- logo_universidade.png
| |-- diagrama_fluxo.png
| |-- style.css # (Opcional) CSS customizado
|
|-- data/ # Dados (apenas amostras/arquivos pequenos)
| |-- dataset_exemplo.csv
|
|-- tests/ # (Opcional) Testes unitarios
| |-- test_funcoes.py
|
|-- .streamlit/ # Configuracoes do Streamlit (tema, cores)
    |-- config.toml
```

Dica prática

Use nomes simples e que representem a finalidade do arquivo. No *deploy*, você selecionará o *Main file path* (apontando para `app.py` ou equivalente) (Streamlit Docs, 2025c).

3 Execução local (passo a passo)

Nesta etapa, o código Python do aplicativo é escrito em um **arquivo .py** (por exemplo, `app.py`) usando um editor (recomendado: VS Code). Os comandos para instalar dependências e iniciar o servidor local do Streamlit são executados no **terminal** (por exemplo, PowerShell no Windows).

3.1 Criar o arquivo principal (`app.py`)

1. Abra a pasta do projeto no editor (ex.: VS Code).
2. Crie o arquivo `app.py` na raiz do repositório (mesmo nível de `requirements.txt`).
3. Cole o exemplo mínimo abaixo e salve o arquivo.

```python
import streamlit as st

st.set_page_config(page_title="Meu App", layout="wide")
st.title("Meu aplicativo Streamlit")
st.write("Ambiente local OK.")
```

Listing 1: Exemplo mínimo de aplicativo Streamlit (arquivo `app.py`).

Dica prática

Compatibilidade por sistema operacional. Os comandos de ativação do ambiente virtual variam conforme o sistema:

Sistema	Ativação do ambiente virtual
Windows (PowerShell)	`.\.venv\Scripts\activate`
Mac/Linux (bash/zsh)	`source .venv/bin/activate`

Após ativar, a instalação de dependências via `pip install -r requirements.txt` é a mesma.

3.2 Abrir o terminal (PowerShell ou terminal integrado do VS Code)

- **Windows (PowerShell):** abra o PowerShell e navegue até a pasta do projeto.
- **VS Code:** use *Terminal* → *New Terminal*.

3.3 Criar ambiente virtual e instalar dependências

A recomendação é criar um ambiente virtual (`venv`) para isolar bibliotecas do projeto.

Windows (PowerShell):

```powershell
cd caminho\para\nome-do-repositorio

python -m venv .venv
.\.venv\Scripts\activate

pip install -r requirements.txt
```

Listing 2: Comandos no PowerShell para preparar o ambiente local.

Observação: O arquivo `requirements.txt` deve conter, no mínimo, `streamlit` (além das demais bibliotecas do projeto).

Exemplo de `requirements.txt` (mínimo + comum). O arquivo deve listar as bibliotecas necessárias para reproduzir o ambiente local e do *deploy* (uma por linha) (Streamlit Docs, 2025a).

```
streamlit>=1.30
pandas>=2.0
numpy>=1.24
plotly>=5.15
requests>=2.31
```

Listing 3: Exemplo de requirements.txt.

Nota sobre versões mínimas. As expressões no formato >= indicam **versões mínimas** aceitas. Para **reprodutibilidade**, considere:

- testar localmente no mesmo ambiente do *deploy* e validar o *build* pelos logs;
- quando necessário, **fixar** versões com == ou usar faixas controladas;
- atualizar dependências de forma planejada e com verificação após cada atualização.

3.4 Executar o aplicativo localmente

Com o ambiente virtual ativado, execute:

```
streamlit run app.py
```

Listing 4: Execucao local do aplicativo Streamlit.

Ao executar o comando, o Streamlit iniciará um servidor local e abrirá o navegador (tipicamente em `http://localhost:8501`). Para encerrar, utilize `Ctrl+C` no terminal.

4 Senhas e credenciais (quando o app usa APIs)

4.1 O que é uma API e por que ela exige credenciais

Uma **API** (*Application Programming Interface*) é uma interface formal que permite que um software utilize funcionalidades e dados de um serviço externo. Em geral, ocorre via requisições autenticadas, pois o provedor precisa controlar *quem* acessa, *quanto* acessa e *o que* pode acessar.

Na prática, isso significa que o aplicativo pode exigir **credenciais**, como:

- **API key** (chave de API): identificador usado para autorizar chamadas; deve ser protegida.
- **OAuth token** (credencial de usuário): usado quando o serviço autentica um usuário final.
- **Service account** (conta de serviço): identidade usada para execução automática em servidores.

Aviso crítico

Credenciais (chaves/tokens/arquivos JSON de conta de serviço) devem ficar **fora do repositório** e não devem ser versionadas. Use mecanismos de *secrets* no ambiente de execução (Google Cloud, 2025; Google, 2025).

4.2 Passo a passo: credencial do Google Earth Engine (GEE)

Este exemplo detalha como configurar credenciais quando o aplicativo utiliza uma API externa, como o Google Earth Engine (GEE), que requer autenticação para acesso e processamento.

1) Onde declarar e instalar bibliotecas

As bibliotecas necessárias ao aplicativo devem ser:

1. **Declaradas no projeto** (arquivo `requirements.txt`), para garantir que o mesmo conjunto de dependências seja instalado tanto no ambiente local quanto no *deploy* (Streamlit Docs, 2025a);
2. **Instaladas no ambiente local** via terminal, com o ambiente virtual (`.venv`) ativado.

A) Declarar no `requirements.txt` (na raiz do repositório). Inclua as bibliotecas do GEE utilizadas pelo app. Exemplo:

```
streamlit>=1.30
earthengine-api>=0.1
geemap>=0.30
```

Listing 5: requirements.txt com dependencias para Streamlit + GEE (exemplo).

B) Instalar localmente (PowerShell ou terminal do VS Code). No Windows, dentro da pasta do projeto e com o ambiente virtual ativado:

```
cd caminho\para\nome-do-repositorio
python -m venv .venv
.\.venv\Scripts\activate

pip install -r requirements.txt
```

Listing 6: Instalacao das dependencias no ambiente local (Windows/PowerShell).

Dica prática

No *deploy*, o Streamlit Community Cloud instala automaticamente as dependências a partir do `requirements.txt` versionado no GitHub. Portanto, incluir `earthengine-api` e `geemap` no arquivo e fazer *commit* é essencial (Streamlit Docs, 2025a).

2) Gerar a credencial (Conta de serviço) e baixar o JSON

No Google Cloud Console, crie uma credencial do tipo **Conta de serviço** e gere uma chave no formato **JSON**. Após a criação, o arquivo JSON é baixado e contém as informações necessárias para autenticação do GEE.

3) Configurar a senha no ambiente local (Windows)

No Windows, crie (ou use) a pasta `.streamlit` dentro do perfil do usuário e crie o arquivo:

```
C:\Users\<SEU_USUARIO>\.streamlit\secrets.toml
```

Em seguida, copie as informações do JSON para esse arquivo `secrets.toml`, mas **no formato TOML** (com uma seção, por exemplo `[earthengine]`). Após formatar, salve como `secrets.toml`.

Aviso crítico

Importante. Ao transcrever o arquivo JSON (conta de serviço) para o `secrets.toml`, o conteúdo deve ser convertido para o **formato TOML exatamente** conforme o modelo apresentado na Figura 1, incluindo a seção `[earthengine]`. Divergências no formato podem impedir a leitura por `st.secrets` e causar falha de autenticação.

Segurança ao publicar/documentar. Se você for inserir a credencial em relatório, slides, repositório ou anexos:

- aplique **tarja preta** em todo o conteúdo de `private_key` (e, preferencialmente, também em `private_key_id`);
- nunca publique o arquivo real `secrets.toml` nem o JSON original da conta de serviço;
- garanta que `.streamlit/secrets.toml` esteja no `.gitignore`.

Atenção

Observe com muita atenção o exemplo .toml da Figura 1. Compare com o JSON gerado. Por exemplo, veja que o .toml não tem o comando `\n`

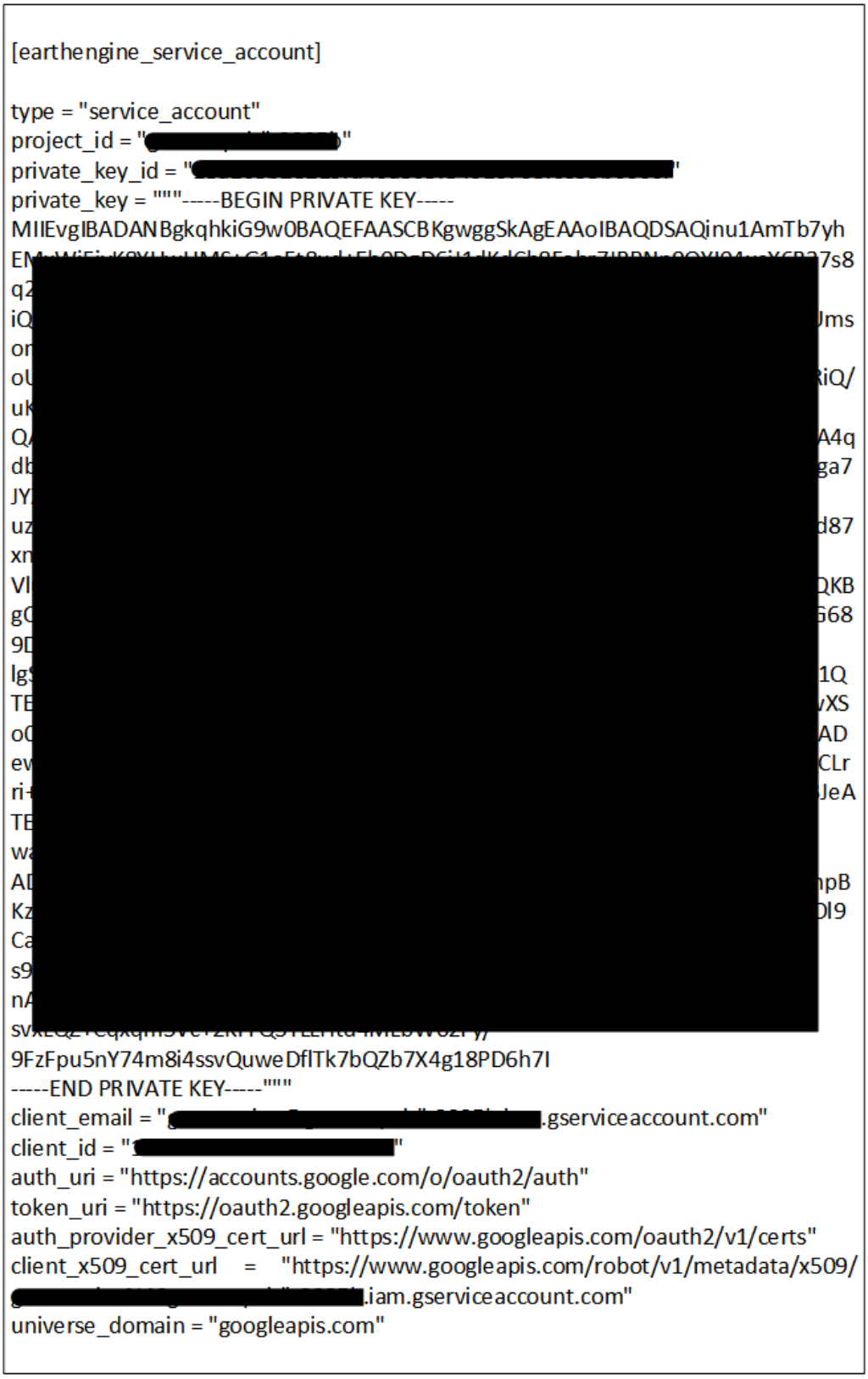

Figura 1: Exemplo de secrets.toml.

5) Configurar Secrets no Streamlit Community Cloud

Ao publicar no Streamlit Community Cloud, acesse *Settings* → *Secrets* e **cole o conteúdo TOML** do seu `secrets.toml` no painel de Secrets. Salve as configurações.

5 Publicação no Streamlit Community Cloud

O *deploy* no Community Cloud é realizado conectando o GitHub e selecionando repositório, *branch* e arquivo principal (Streamlit Docs, 2025c).

5.1 Preparar o repositório (GitHub)

Checklist mínimo:

- `app.py` (ou equivalente) está no repositório;
- `requirements.txt` existe e está coerente com os `imports` (Streamlit Docs, 2025a);
- senhas não foram versionadas (ex.: `.streamlit/secrets.toml`).

```
# Secrets do Streamlit
.streamlit/secrets.toml

# Chaves e certificados
*.pem
*.key

# Chaves de conta de servico (exemplos de nomes comuns)
*service-account*.json
*service_account*.json
```

Listing 7: .gitignore recomendado (evitar vazamento de credenciais sem bloquear JSON genéricos).

5.1.1 Enviar os arquivos para o GitHub (Git básico)

Se o projeto ainda está apenas na sua máquina, siga o procedimento abaixo para inicializar o repositório local e enviar os arquivos ao GitHub.

```
cd caminho/para/nome-do-repositorio

git init
git add .
git commit -m "Primeiro commit"

git branch -M main
git remote add origin https://github.com/SEU_USUARIO/NOME_REPO.git
git push -u origin main
```

Listing 8: Comandos mínimos para subir o projeto ao GitHub (exemplo).

Dica prática

Após o `push`, o repositório passa a existir no GitHub e então aparecerá na lista **Repository** do Streamlit Community Cloud (quando sua conta estiver conectada ao GitHub).

5.2 Versão do Python no Community Cloud (ponto crítico)

Atenção

No Streamlit Community Cloud, a versão do Python é escolhida no momento do *deploy* (em *Advanced settings*). Para mudar a versão do Python depois, em geral é necessário **recriar/republicar** o app.

5.3 Criar o app no Community Cloud

Fluxo padrão (Streamlit Docs, 2025c):

1. **Acessar o Community Cloud e iniciar a criação do app.** Entre em `share.streamlit.io` e clique em **Create app**.
2. **Selecionar o repositório do GitHub (Repository).** No campo **Repository**, escolha o repositório que contém o código do aplicativo.
3. **Selecionar o ramo de publicação (Branch).** No campo **Branch**, selecione o ramo que será publicado (tipicamente `main`).

   Dica prática

   O app publicado no Cloud refletirá o conteúdo do *branch* selecionado. Ao fazer *push* de novos commits nesse ramo, o Community Cloud pode reconstruir e atualizar o app.

4. **Informar o arquivo principal (Main file path).** Em **Main file path**, informe o caminho do arquivo de entrada (por exemplo `app.py`).
   - Se o arquivo estiver na raiz do repositório: use apenas `app.py`.
   - Se estiver em subpastas: informe o caminho relativo (ex.: `src/app.py`).

   Atenção

   Erro típico. Informar um caminho que não existe no *branch* selecionado resulta em falha imediata no *deploy*.

5. **Definir o endereço do app (App URL) — opcional.** Escolha um identificador curto (por exemplo `meuapp2025`) para compor a URL final `<nome>.streamlit.app`.
6. **Abrir configurações avançadas (Advanced settings) — quando necessário.** Use **Advanced settings** caso precise ajustar opções específicas do *deploy*.

7. **Executar o deploy.** Clique em **Deploy**. O Streamlit irá clonar o repositório, instalar dependências e iniciar o app.
8. **Acompanhar logs e corrigir falhas.** Se houver erro, verifique a aba **Logs**. Problemas frequentes:
    - dependências ausentes (*ModuleNotFoundError*);
    - caminho incorreto do arquivo principal;
    - secrets ausentes quando o app consome APIs (Streamlit Docs, 2025d).

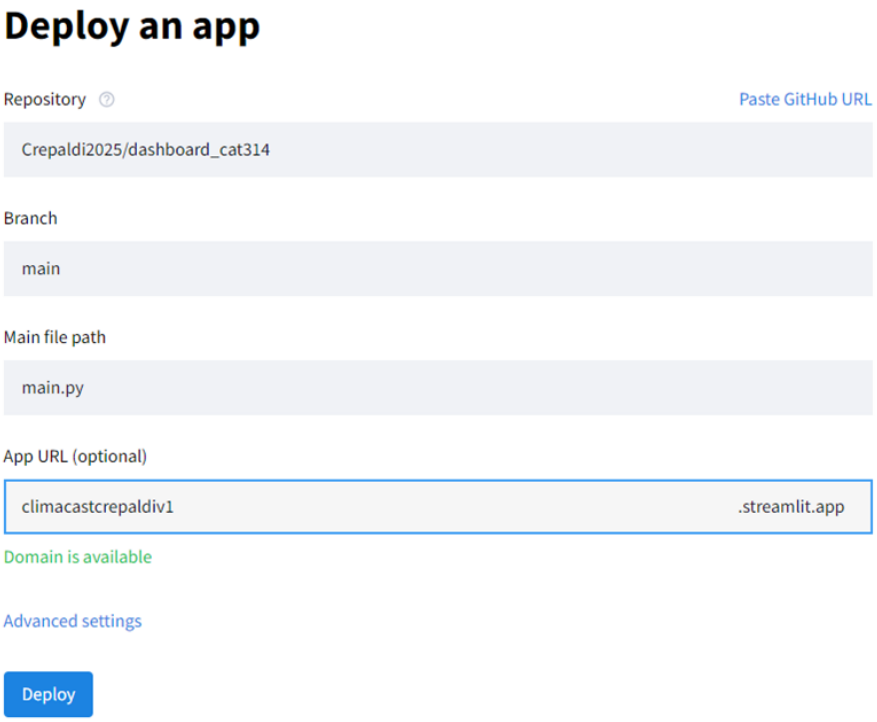

Figura 2: Início do *deploy* no Streamlit Community Cloud (Create app).

5.4 Inserir Secrets no Cloud (se necessário)

Se o app requer credenciais, utilize o painel de *secrets* do Community Cloud (Streamlit Docs, 2025d).

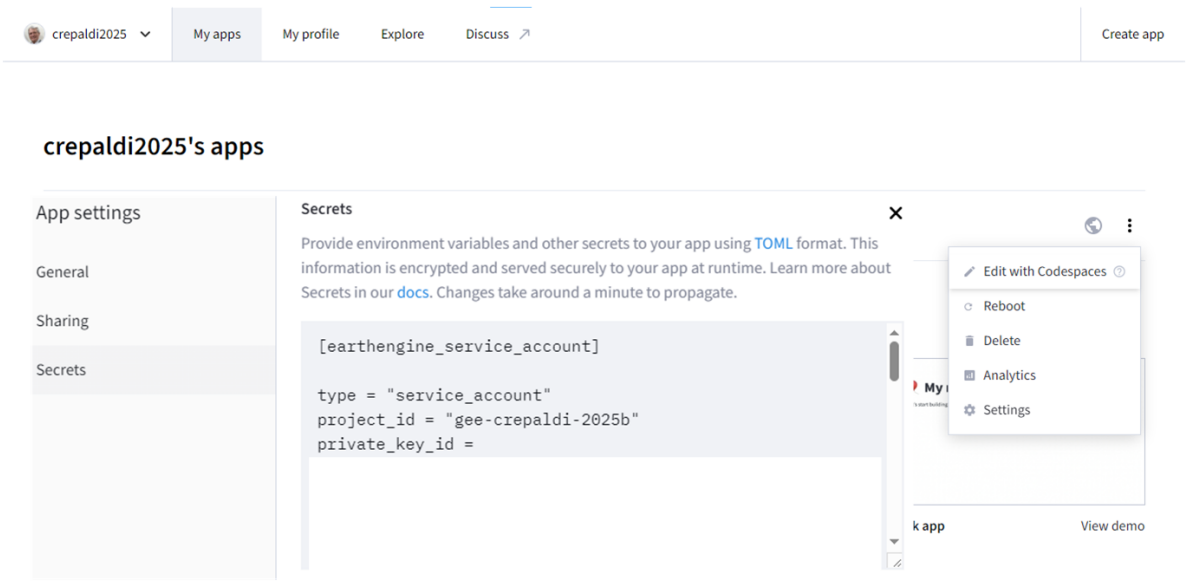

Figura 3: Editor de *Secrets* no Streamlit Community Cloud.

5.5 Dependências e logs

Se ocorrer falha de *build*, o primeiro diagnóstico deve ser feito pelos **logs do app no Community Cloud** e pela verificação do arquivo de dependências (Streamlit Docs, 2025a).

Onde acessar os logs (Community Cloud):

1. Acesse `share.streamlit.io` e abra **My apps**;
2. Clique no nome do aplicativo que apresentou falha;
3. Abra a área **Logs**;
4. Leia as mensagens desde o início do *build*.

6 Erros comuns e correções rápidas

Tabela 1: Diagnóstico rápido de falhas comuns no *deploy*.

Sintoma (log)	**Causa provável**	**Correção**
`ModuleNotFoundError`	Pacote não está no `requirements.txt`	Adicione o pacote e faça *commit/push*.
`No such file or directory` (main file)	Caminho do *Main file path* incorreto	Ajuste para o arquivo real no *branch* publicado.
Erro ao ler `st.secrets`	Secrets ausentes ou TOML inválido	Corrija o TOML e configure no painel de Secrets.
Falha em dependência "externa"	Falta biblioteca do sistema	Adicione dependência externa conforme suporte do Cloud (ex.: `packages.txt`).

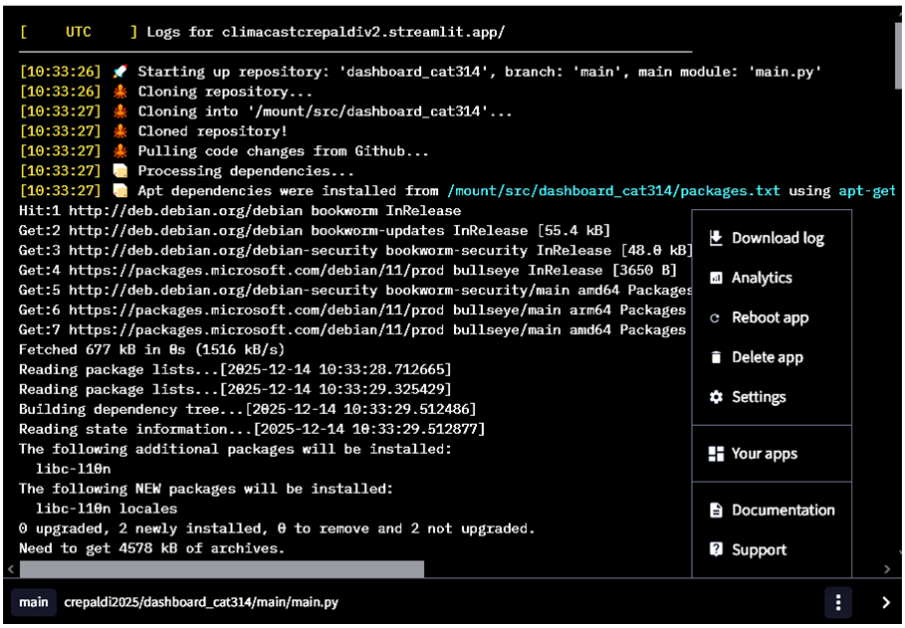

Figura 4: Tela de informações durante instalação de dependências e inicialização.

Dica prática

Nos logs, procure por palavras-chave como `ERROR`, `ModuleNotFoundError`, `No module named`, `Could not find a version`, `Main file path` e `secrets`. Elas geralmente indicam a causa raiz.

6.1 Fluxo de diagnóstico pelos logs

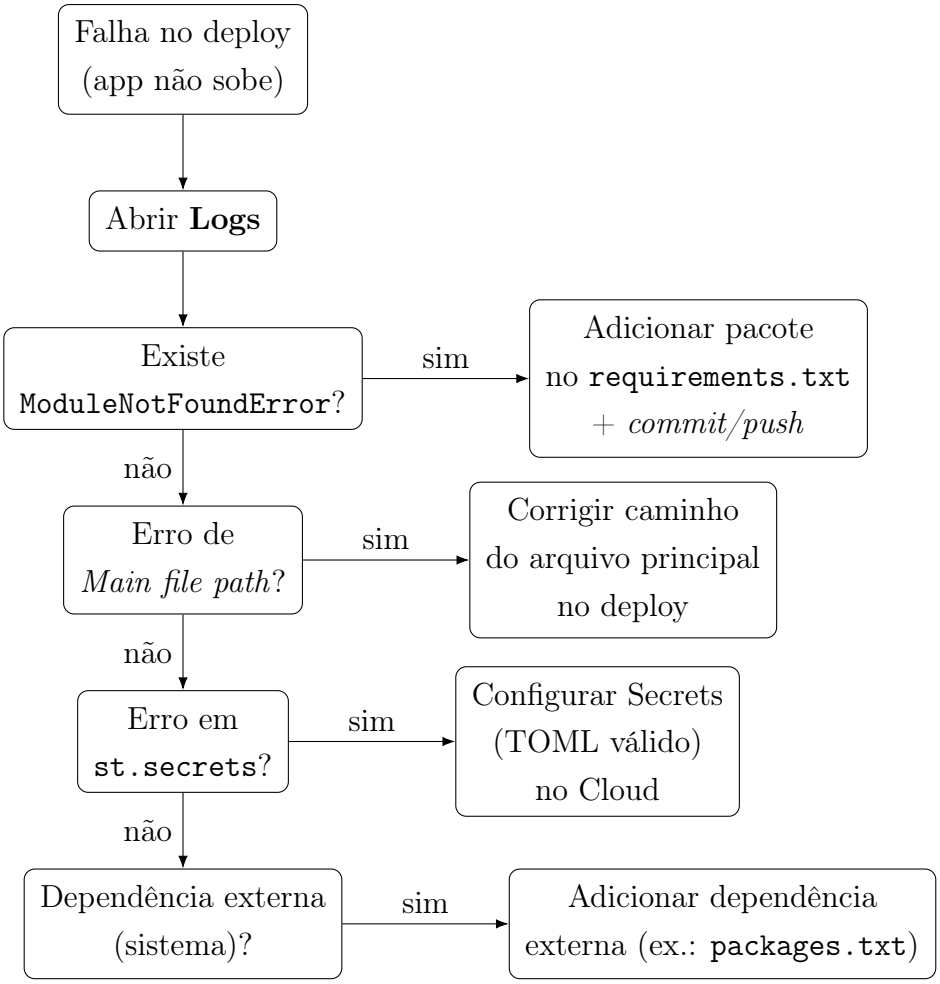

Figura 5: Roteiro de diagnóstico baseado em logs do Community Cloud.

7 Configuração opcional: `.streamlit/config.toml`

Para ajustes de comportamento (ex.: detalhes de erro, opções de servidor), use `config.toml` (Streamlit Docs, 2025b).

7.1 Exemplos práticos de `.streamlit/config.toml`

O arquivo `config.toml` (opcional) permite ajustar o comportamento do Streamlit no nível do cliente (exibição de erros e toolbar), execução (reruns) e servidor (CORS/XSRF, uploads etc.) (Streamlit Docs, 2025b).

```toml
[client]
showErrorDetails = "full" # exibe traceback completo no navegador
toolbarMode = "developer" # opcoes de desenvolvedor (menu/toolbar)

[logger]
level = "debug"

[runner]
fastReruns = true

[server]
headless = true
runOnSave = true
enableCORS = true
enableXsrfProtection = true
maxUploadSize = 200
maxMessageSize = 200
baseUrlPath = ""

[browser]
gatherUsageStats = false
```

Listing 9: Exemplo de `config.toml` para **desenvolvimento** (mais diagnóstico).

```toml
[client]
showErrorDetails = "none" # nao mostra traceback no navegador
toolbarMode = "minimal" # reduz opcoes visiveis no menu/toolbar

[logger]
level = "warning"

[runner]
fastReruns = true

[server]
headless = true
runOnSave = false
enableCORS = true
enableXsrfProtection = true
maxUploadSize = 200
maxMessageSize = 200
baseUrlPath = ""
```

Listing 10: Exemplo de `config.toml` para **produção** (menos exposição de detalhes).

Dica prática

Para listar todas as opções disponíveis no seu ambiente, use: `streamlit config show`. Em seguida, copie apenas o que for necessário para o seu caso (Streamlit Docs, 2025b).

8 Checklist final

- Tenho um *entrypoint* único (ex.:`app.py`) e ele roda localmente.
- Meu `requirements.txt` contém os pacotes que eu importo.
- Eu não versionei credenciais; usei `secrets.toml` localmente e o painel de *Secrets* no Cloud.
- No Community Cloud, selecionei repositório, branch e caminho do arquivo principal e acompanhei logs (Streamlit Docs, 2025c).

Referências

Google. *Best practices for securely using API keys*. 2025. <https://support.google.com/googleapi/answer/6310037?hl=en>. Acesso em: 12 dez. 2025.

Google Cloud. *Best practices for managing service account keys*. 2025. <https://cloud.google.com/iam/docs/best-practices-for-managing-service-account-keys>. Acesso em: 12 dez. 2025.

Streamlit Docs. *App dependencies for your Community Cloud app*. 2025. <https://docs.streamlit.io/deploy/streamlit-community-cloud/deploy-your-app/app-dependencies>. Acesso em: 12 dez. 2025.

Streamlit Docs. *config.toml*. 2025. <https://docs.streamlit.io/develop/api-reference/configuration/config.toml>. Acesso em: 12 dez. 2025.

Streamlit Docs. *Deploy your app on Community Cloud*. 2025. <https://docs.streamlit.io/deploy/streamlit-community-cloud/deploy-your-app/deploy>. Acesso em: 12 dez. 2025.

Streamlit Docs. *Secrets management for your Community Cloud app*. 2025. <https://docs.streamlit.io/deploy/streamlit-community-cloud/deploy-your-app/secrets-management>. Acesso em: 12 dez. 2025.