## Laboratório Calculadora Simples

**Projeto:** Uma calculadora modular e orientada a objetos com funcionalidades crescentes.

**Objetivo didático:** Mostrar como o uso do Prefect traz:

* Clareza no versionamento e evolução de código.
* Automação do processo de CI/CD (push → deploy).
* Observabilidade (logs, histórico, status, falhas).
* Controle granular com retries, agendamento, dependências.
* Profissionalismo no ciclo de entrega contínua.

**🚀 Estrutura do novo laboratório**

| **Fase** | **Funcionalidade** | **Justificativa técnica** |
| --- | --- | --- |
| V0 | Adição e subtração | MVP básico, execução local |
| V1 | + Multiplicação | Atualização simples + novo deploy |
| V2 | + Divisão | Introduz tratamento de exceções |
| V3 | + Porcentagem | Função que simula cálculo financeiro simples |
| V4 | CLI simples ou Web API (opcional) | Interface de uso real do sistema |
| VFinal | Observabilidade via Prefect | Histórico, versionamento, controle de execução |

**📁 Estrutura sugerida do projeto**

bash

CopiarEditar

calculator\_prefect/

├── calculator/

│ ├── \_\_init\_\_.py

│ ├── base.py # classe abstrata / interface

│ ├── operations.py # implementação OO das operações

├── flows/

│ └── run\_calculator.py # fluxo Prefect que orquestra chamadas

├── tests/

│ └── test\_operations.py

├── .github/workflows/

│ └── deploy.yml

├── prefect.yaml

├── requirements.txt

└── README.md

**🧪 Diferenciais didáticos**

* Mostra evolução real de um projeto de software.
* Permite aplicar CI/CD a um sistema **que cresce com as entregas**.
* Pode ser integrado com testes, cobertura de código e até Sentry.
* Dá espaço para introduzir conceitos como: modularização, boas práticas, automação e versionamento.

**📝 E no relatório?**

Posso incluir:

* ✅ Nova seção com objetivo e plano do novo laboratório.
* ✅ Descrição detalhada de cada etapa (do V0 ao VFinal).
* ✅ Explicação técnica sobre o valor do Prefect em cada evolução.
* ✅ Código comentado das versões incrementais.

Se estiver de acordo, posso:

1. Criar o esqueleto da estrutura do projeto.
2. Escrever o prefect.yaml inicial.
3. Gerar o deploy.yml do GitHub.
4. Adicionar a nova seção no relatório .docx como parte do conteúdo didático.

**✅ Confirmação da Estrutura Didática do Novo Laboratório**

**🎯 Dois eixos centrais:**

**1. Orquestração com Prefect — O "Por quê"**

Mostrar aos alunos que **orquestrar não é apenas automatizar**, mas sim:

* **Controlar, versionar, auditar e escalar** workflows.
* Garantir **resiliência com retries**, **condições de execução**, agendamentos e visualização clara do pipeline.
* Evitar que falhas silenciosas prejudiquem a produção (monitoramento e logs).
* Preparar o projeto para integração com ferramentas como S3, bancos de dados, serviços externos, ML pipelines, etc.

**2. Engenharia de Software Moderna — O "Como"**

Construir um projeto:

* Modular.
* Escalável.
* Orientado a objetos.
* Com documentação, testes e CI/CD.

A proposta da **calculadora evolutiva** (com adição → subtração → multiplicação → divisão → porcentagem) **é perfeita** para ilustrar isso em pequenas doses funcionais, com cada versão gerando um novo deployment automatizado.

**✍️ Ação Imediata**

Proponho criar um novo documento com o seguinte conteúdo:

**🧩 Nome: Calculadora Orquestrada - Projeto Didático com Prefect**

E nele vamos estruturar:

1. **Objetivo pedagógico**: detalhar esses dois eixos.
2. **Etapas do projeto**:
   * V0 (adição e subtração)
   * V1 (+ multiplicação)
   * V2 (+ divisão com tratamento de erro)
   * V3 (+ porcentagem, simulação financeira)
   * V4 (CLI simples)
3. **Como o Prefect entra em cada etapa**.
4. **Pipeline de automação GitHub Actions** e gatilho por push.
5. **Estrutura modular do código (orientada a objetos)**.
6. **Visão final do ganho de orquestrar e versionar projetos reais.**

# Laboratório Didático: Orquestração com Prefect aplicada ao Desenvolvimento de uma Calculadora Modular

**🎯 Objetivo**

Desenvolver um laboratório didático que evidencie a importância da orquestração de tarefas com Prefect, aliado às boas práticas de Engenharia de Software. O laboratório deverá mostrar o ciclo de vida completo de um projeto de software: desde a fase de projeto (fundamental e anterior à codificação), passando pelo desenvolvimento modular orientado a objetos, até a automação de deploys e execuções usando GitHub Actions e Prefect Cloud. A proposta também busca destacar a importância da reprodutibilidade, escalabilidade, organização de sistemas reais e a necessidade de um bom projeto da solução antes de iniciar a codificação.

**🧩 Justificativa**

Alunos e profissionais iniciantes geralmente subestimam a importância de ferramentas de orquestração, acreditando que apenas "rodar um script" seja suficiente. Este laboratório irá demonstrar, de maneira prática e visual, os reais ganhos de controle, rastreabilidade, automação e confiabilidade trazidos por uma ferramenta como o Prefect. Simultaneamente, a proposta reforça a importância do projeto da solução e da aplicação das melhores práticas de Engenharia de Software desde o início do desenvolvimento, evidenciando o papel das etapas de análise, modelagem e planejamento no sucesso da implementação.

**🏗️ Estrutura Didática (Etapas)**

**✅ Etapa 0 — Preparação do Ambiente**

* Instalação e configuração do ambiente virtual, dependências e Prefect.
* Criação do repositório no GitHub com estrutura inicial.
* Testes de conectividade com Prefect Cloud.

**🔸 Etapa 1 — Projeto da Solução e Estrutura Base**

* Definição do escopo inicial: Calculadora com operações básicas (adição e subtração).
* Uso de ferramentas de modelagem (Canvas, Lucidchart, draw.io).
* Estrutura modular e orientada a objetos.
* Criação da estrutura de diretórios e arquivos principais.
* Documentação do projeto com UML simples (classes e fluxo).

**🔸 Etapa 2 — Primeiro Deploy (Adição e Subtração)**

* Implementação inicial das operações básicas.
* Testes locais com Prefect Flow.
* Deploy da primeira versão com GitHub Actions.

**🔸 Etapa 3 — Ampliação Funcional (Multiplicação e Divisão)**

* Expansão incremental do projeto.
* Refatorações, modularização e tratamento de erros.
* Novo commit + push → novo deploy automatizado.

**🔸 Etapa 4 — Finalização de Funcionalidades (Porcentagem)**

* Adição da última funcionalidade planejada.
* Validação final do comportamento.
* Push final e execução orquestrada via Prefect Cloud.

**🔸 Etapa 5 — Integração com Prefect e Visualização**

* Conversão dos scripts em flows.
* Observação da execução e logs no painel Prefect.
* Avaliação dos ganhos e comparação com execução local/manual.

**🔍 Metodologia**

* **Projeto da solução antes do código**, com apoio de ferramentas de modelagem (Canvas, Lucidchart, draw.io).
* Aplicação de princípios de modularidade, encapsulamento e orientação a objetos.
* Uso de diagramas UML e documentação progressiva.
* Git e GitHub como base de versionamento e integração.
* Prefect para controle de execução e rastreabilidade.
* GitHub Actions para automação do deploy e execução.
* Avaliação visual e técnica dos resultados com e sem orquestração.
* Reflexão crítica ao fim de cada etapa sobre os ganhos obtidos.

**🧭 Resultado Esperado**

* Um projeto de software simples, mas tecnicamente bem estruturado, escalável e modular.
* Orquestração eficaz com Prefect, evidenciando benefícios de automação e rastreabilidade.
* CI/CD funcional com deploy automático a partir do GitHub.
* Relatório técnico final detalhando todo o processo, decisões e aprendizados.

**✅ Pontos-Chave**

* Prefect é mais do que um agendador: é uma ferramenta de orquestração e confiabilidade.
* Engenharia de Software começa com **projeto da solução**, não com código.
* Boas práticas geram código reusável, manutenível e escalável.
* CI/CD não é um luxo — é um diferencial competitivo real.

**🧠 Observação Didática**

Este laboratório poderá ser adaptado para diferentes níveis (graduação, pós) ajustando o grau de complexidade da calculadora e os requisitos de automação. Também poderá ser expandido com integração a banco de dados, autenticação ou visualização via Dash/Streamlit em etapas futuras.

## Estrutura Didática do Repositório (com explicações)

### ✅ Etapa 0 — Preparação do Ambiente e Documentação Inicial

🧭 Contexto

Estamos iniciando o projeto calculadora\_simples, cujo objetivo final é demonstrar a importância da orquestração com Prefect, aliada às boas práticas de Engenharia de Software.

O projeto será construído incrementalmente, com deploys progressivos após cada nova funcionalidade, simulando um ciclo real de evolução de sistemas.

### 🔹 Ambiente de Desenvolvimento

* 📁 Repositório: calculadora\_simples
* 💻 Ambiente virtual: DEV
* 🐍 Python: 3.12.5
* 📂 Diretório atual: /Users/accol/Library/Mobile Documents/com~apple~CloudDocs/UNIVERSIDADES/UFF/CARREIRA\_POS/PPGEET/ORQUESTRACAO/LABORATORIO/calculadora\_simples

### 📁 Estrutura Inicial do Projeto

Texto

O conteúdo gerado por IA pode estar incorreto.

**✅** Ações obrigatórias nesta etapa

#### 1. Criar o arquivo docs/readme.txt

Este arquivo conterá a **especificação da versão inicial da calculadora**, com foco apenas em **adição e subtração**. Deve conter:

* Nome do sistema
* Objetivo
* Funcionalidades mínimas
* Público-alvo
* Forma de uso esperada

#### 2. Gerar requirements.txt

Com os pacotes usados no projeto e versões fixas. Incluir:

prefect==2.16.4

griffe==0.35.2

Outros pacotes serão adicionados conforme a evolução.

#### 3. Configurar infra/prefect.yaml

Este arquivo será responsável por indicar como o Prefect deve executar os flows. Terá a configuração git\_clone para repositório GitHub.

#### 4. Criar .gitignore e .prefectignore

Para evitar que arquivos locais/ocultos sejam versionados:

Texto

O conteúdo gerado por IA pode estar incorreto.

#### 5. Visibilidade no VSCode (macOS)

Para visualizar arquivos ocultos como .gitignore e .prefectignore:

1. Pressione ⌘ + , para abrir Preferências;
2. Busque por files.exclude e certifique-se que arquivos como .git\* não estão ocultos;
3. Alternativamente, abra a paleta de comandos (⌘ + Shift + P) e busque por Preferences: Open Settings (JSON) e ajuste:

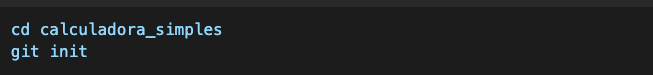
Uma imagem contendo Forma

O conteúdo gerado por IA pode estar incorreto.

#### 6. Criar repositório remoto no GitHub

* Nome: CALCULADORA\_SIMPLES
* Inicializar sem README (para evitar conflito)

#### 7. Inicializar versionamento com Git localmente



#### 8. Conectar repositório remoto



#### 9. Primeiro commit e push

Texto

O conteúdo gerado por IA pode estar incorreto.

## 📘 ETAPA 1 — Modelagem da Solução e Projeto da Calculadora OO

### 🎯 Objetivo

Desenhar e documentar a arquitetura inicial da calculadora com base nos princípios da Engenharia de Software, aplicando boas práticas de modularização, orientação a objetos e escalabilidade. Antes de codificar, será realizado o **projeto da solução**, que norteará todas as etapas seguintes.

### 🧠 Justificativa

É fundamental que o projeto de um sistema, por mais simples que pareça, seja realizado com base em boas práticas. Isso inclui:

* Clareza nos requisitos e escopo
* Organização modular e reutilizável
* Documentação desde o início
* Adoção de estratégias que favoreçam o crescimento do projeto

Além disso, será evidenciada a **importância da orquestração** em ciclos de evolução, por meio de deploys programados com o Prefect a cada nova funcionalidade adicionada.

### 🧱 Ferramentas e Estratégias Utilizadas

* **Modelagem visual:** ferramenta https://app.diagrams.net para os diagramas
* **Canvas de Projeto:** descrevendo contexto, escopo, usuários e funcionalidades
* **UML simplificada:** para representação das classes e modularização
* **Documentação contínua:** no diretório docs/

### 🛠️ Tarefas desta Etapa

**1. Criar o Canvas do Projeto (arquivo .md ou .pdf em docs/)**

* Nome do sistema
* Problema que resolve
* Usuários
* Funcionalidades esperadas
* Restrições e premissas

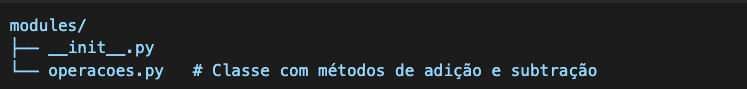
**2. Criar o Diagrama de Classes Inicial**

* Representar as classes Calculadora (base), e Operacoes
* Métodos públicos: somar(), subtrair()
* Salvar como imagem em docs/diagrama\_classes\_v1.png

**3. Documentar Arquitetura da Solução**

* Arquivo docs/arquitetura.md
* Descrever:
  + Modularização pretendida (modules/, flows/, tests/)
  + Justificativa das separações
  + Como será feita a expansão futura (multiplicação, divisão, porcentagem)

### 4. Criar estrutura de classes base em modules/

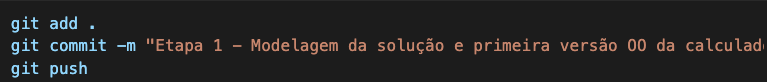


## 5. Planejar primeiro Flow em flows/

* Nome: flow\_versao1.py
* Responsável por orquestrar a execução da versão inicial da calculadora

## 6. Atualizar requirements.txt se necessário

## Commit e Push da Etapa 1



## Ações da ETAPA 1

## 📌 Canvas do Projeto – Calculadora Simples com Orquestração Prefect

|  |  |
| --- | --- |
| Bloco | Descrição |
| Nome do Sistema | Calculadora Simples Orquestrada |
| Problema que Resolve | Automatiza operações básicas de forma estruturada, modular e escalável. |
| Público-Alvo / Usuários | Estudantes e profissionais aprendendo boas práticas de Engenharia de Software e Orquestração de Projetos |
| Funcionalidades Esperadas | - Soma - Subtração - Multiplicação - Divisão - Porcentagem |
| Módulos do Sistema | - modules/operacoes.py (lógica OO) - flows/ (orquestração) - infra/ (deploy futuro) |
| Expansões Previstas | Novas operações, interface web (futura), relatórios de uso |
| Premissas | - Código em Python 3.12.5 - Projeto modular - Deploy via Prefect Cloud com GitHub Actions |
| Restrições | - Código deve ser funcional a cada commit - Testes obrigatórios - Compatível com GitHub Actions |

## 📌 **UML – Diagrama de Classes (Versão 1)**

### Objetivo:

Representar graficamente as classes e seus relacionamentos iniciais no projeto da calculadora orientada a objetos. Nesta versão inicial, focaremos nas operações de **soma** e **subtração**, com possibilidade de extensão modular.

### 🧩 Estrutura proposta:

Diagrama

O conteúdo gerado por IA pode estar incorreto.

### 📁 Local para salvar:

O arquivo será salvo como:

📄 docs/diagrama\_classes\_v1.png

### Criando chaves – importante:

Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

Agora que você está com a **API Key recém-criada** do Prefect Cloud visível (conforme imagem), o próximo passo é usá-la **IMEDIATAMENTE**, antes de fechar esse modal — pois ela **não poderá mais ser visualizada depois**.

### ✅ Passos Imediatos – Agora no **Terminal (VSCode)**

Execute o seguinte comando abaixo diretamente no terminal (com o ambiente virtual DEV ativado):

pnu\_IbAMNqegMi3HB3yNGxPvAqiklD1uWQ2kjdQA

prefect cloud login -k pnu\_lbAMNqegMi3HB3yNGxPvAqikID1uWQ2kjdQA

**Nota:** Isso salvará a chave API no seu perfil local do Prefect, permitindo autenticar seu agente local e o CLI com o **Prefect Cloud**.

## Próximo Passo:

**👉 Definir os Secrets no GitHub**

1. Acesse o repositório no GitHub
2. Vá até: Settings > Secrets and variables > Actions
3. Clique em **New repository secret**
4. Adicione:
   * Name: PREFECT\_API\_KEY
   * Value: pnu\_lbAMNqegMi3HB3yNGxPvAqikID1uWQ2kjdQA
5. Clique em **New repository secret** novamente:
   * Name: PREFECT\_API\_URL
   * Value:  
     https://api.prefect.cloud/api/accounts/284c9335-75c6-4022-ac98-4aa4d49d7f95/workspaces/fbc659c6-2423-42f4-b053-8224ac12d982

Depois disso, validamos juntos o prefect-deploy.yml para garantir que o workflow do GitHub esteja correto e preparado para **executar deploys automáticos** com segurança e eficiência.

Quando estiver pronto, só me dar o sinal que seguimos com os secrets e depois revisamos o arquivo prefect-deploy.yml. Vamos fazer desse laboratório o padrão de excelência!

### ✅ Notas Importantes

* **Gatilho:** somente em push no main e com alterações relevantes.
* **Secrets esperados:**
  + PREFECT\_API\_KEY: sua chave do Prefect Cloud
  + PREFECT\_API\_URL: URL do seu workspace Prefect
* **Deploy com retry:** evita falhas momentâneas por conectividade ou instabilidade de rede.

### 📌 Explicações Didáticas (para seus alunos)

|  |  |
| --- | --- |
| Seção | Explicação |
| on.push.paths | Garante que o deploy só será feito se algo relevante mudar |
| prefect deploy com retry | Evita falhas momentâneas de rede ou API |
| secrets | O segredo nunca fica visível e é seguro via GitHub |
| curl -I | Confirma conectividade com a Prefect Cloud antes de tentar o login |

### ✅ Objetivo agora

Verificar e, se necessário, redefinir a variável de ambiente PREFECT\_API\_KEY no seu terminal, para evitar conflitos com chaves antigas.

### 🧪 **1. Verificar valor atual**

Execute o comando abaixo para verificar a variável que está carregada atualmente:

prefect config view

### 🧹 **2. Limpar a variável antiga do ambiente**

Se a PREFECT\_API\_KEY exibida ainda estiver apontando para uma chave antiga, rode o seguinte comando para **desvinculá-la temporariamente da sessão atual**:

unset PREFECT\_API\_KEY

Caso também tenha configurado no seu shell de forma permanente (.zshrc, .bash\_profile, etc), comente ou remova a linha correspondente.

### 🔐 **3. Reconfigurar com a nova chave (caso necessário)**

Depois disso, configure manualmente com a nova chave (copiada da Prefect Cloud, já validada):

export PREFECT\_API\_KEY="sua-nova-chave-aqui"

Você pode também salvar isso em seu shell profile se desejar manter permanente, mas isso é opcional neste momento.

### 🌐 **4. Verificar novamente**

Agora rode:

prefect cloud login --key $PREFECT\_API\_KEY --workspace-url $PREFECT\_API\_URL

E depois:

prefect config view

Confirme se está tudo certo com os dois valores:

* PREFECT\_API\_KEY: sua nova chave
* PREFECT\_API\_URL: o link do seu workspace, como:

<https://api.prefect.cloud/api/accounts/284c9335-75c6-4022-ac98-4aa4d49d7f95/workspaces/fbc659c6-2423-42f4-b053-8224ac12d982>

### 🔄 Próximo passo:

Se quiser garantir que não há nenhuma outra variável de ambiente sobrescrevendo isso, você pode rodar:

env | grep PREFECT

Isso vai listar todas as variáveis do sistema que começam com PREFECT\_. Deve aparecer apenas o que foi configurado no profile — sem variáveis de ambiente conflitantes.

...**não retornou nada**, o que significa que:

Isso **significa que nenhuma variável de ambiente com prefixo PREFECT\_ está atualmente definida** na sua sessão do terminal.

**Isso é esperado?**

➡️ **Sim, é perfeitamente normal**, **a menos que** você já tenha feito login no Prefect Cloud ou configurado as variáveis manualmente nesta sessão atual.

### ✅ Como checar se o Prefect está autenticado?

Execute:

#### prefect config view

#### Possíveis respostas: Esse comando mostra a configuração ativa do Prefect. Procure por:

### PREFECT\_API\_KEY = "..."

### PREFECT\_API\_URL = <https://api.prefect.cloud/api/accounts/...>

### Se essas variáveis estiverem definidas, você está autenticado no Prefect Cloud.

### ✅ **2. Verificar o perfil ativo e workspace**

prefect profile inspect

### Exibe o conteúdo do perfil atual (default por padrão), incluindo o workspace ativo.

No name provided, defaulting to 'default'

PREFECT\_API\_URL='https://api.prefect.cloud/api/accounts/284c9335-75c6-4022-ac98-4aa4d49d7f95/workspaces/fbc659c6-2423-42f4-b053-8224ac12d982'

PREFECT\_API\_KEY='pnu\_LCasaq34elD7d4nlHTGim1SZlW8IZa2PkymB'

### ✅ **3. Listar os deployments ativos (confirma integração com o Cloud)**

prefect deployment ls

#### Se retornar os deployments (ex: hello-deploy), significa que o Prefect está conectado corretamente e operando com o Cloud.

Tabela

O conteúdo gerado por IA pode estar incorreto.

### ✅ Como se autenticar no Prefect Cloud (caso não esteja)?

Se necessário, use:

prefect cloud login

Você precisará:

Do **API Key**

Do **Workspace** (ex: uff-eletrica/default)

Ou, se quiser já configurar as variáveis de ambiente (como simuladas pelo GitHub Actions), você pode definir manualmente com:

export PREFECT\_API\_KEY="sua-chave"

export PREFECT\_API\_URL="https://api.prefect.cloud/api/accounts/<account-id>/workspaces/<workspace-id>"

### ✅ **Diagnóstico Perfeito para Seguirmos**

* **Não há variáveis de ambiente PREFECT\_API\_KEY ou PREFECT\_API\_URL ativas no seu terminal.**
* O Prefect está **usando exclusivamente os valores salvos no perfil padrão**, conforme retornado por prefect config view.

Isso é **exatamente o comportamento desejado** neste momento:

|  |  |  |
| --- | --- | --- |
| Variável | Fonte | Status |
| PREFECT\_API\_KEY | profile | ✅ OK |
| PREFECT\_API\_URL | profile | ✅ OK |
| `env | grep PREFECT` | terminal env |

### 📌 Agora podemos seguir com segurança para:

1. **Configuração dos secrets no GitHub**
   * PREFECT\_API\_KEY → cole a chave nova (você gerou e viu hoje).
   * PREFECT\_API\_URL → https://api.prefect.cloud/api/accounts/284c9335-75c6-4022-ac98-4aa4d49d7f95/workspaces/fbc659c6-2423-42f4-b053-8224ac12d982
2. **Rodar um prefect deploy localmente para validar.**
3. **Depois disso**, rodar o GitHub Actions com o .yml atualizado.

### ✅ Verificação antes do deploy automatizado:

|  |  |
| --- | --- |
| Item | Status |
| Repositório GitHub CALCULADORA\_SIMPLES | ✅ Criado |
| prefect.yaml | ✅ Refatorado (gatilho por push, sem agendamento) |
| prefect-deploy.yml | ✅ Preparado |
| Variáveis secretas no GitHub | ✅ Presentes (PREFECT\_API\_KEY e PREFECT\_API\_URL) |
| Flow flow\_versao1.py | ✅ Pronto |
| Script de operações base (operacoes.py) | ✅ Implementado |
| Requisitos (requirements.txt) | ✅ Atualizado |
| Documentação (Canvas, UML, arquitetura) | ✅ Concluída na docs/ |

### 📦 **Próximo passo agora**:

#### 👉 Rodar o primeiro deploy manualmente para validar se tudo está funcionando:

No terminal com o ambiente virtual DEV ativado, execute:

prefect deploy -n hello-deploy

Nota: se **o deploy funcionar com sucesso**, **fazemos um push no GitHub** e **validamos a execução automática** via GitHub Actions (deploy automatizado).

⚠️ Caso algo falhe no deploy, o erro nos mostrará **se é de conectividade, autenticação, work pool ou estrutura do flow** — e resolveremos juntos.

## Comando para executar um teste local com Python:

python flows/flow\_versao1.py

Nota: a **Etapa 1 só estará completa** quando incluirmos os **testes unitários automatizados**, mesmo que simples, pois eles são parte essencial da estruturação profissional e da validação contínua da solução.

### ✅ Atualizações pendentes para concluirmos a **ETAPA 1**:

#### 📌 Inclusão dos Testes Automatizados

**1. Estrutura esperada no diretório tests/:**

Texto

O conteúdo gerado por IA pode estar incorreto.

2. Conteúdo inicial sugerido para test\_operacoes.py:

Texto

O conteúdo gerado por IA pode estar incorreto.

### **Os testes automatizados devem ser incorporados ao fluxo de orquestração com Prefect**, para garantir que:

1. **Cada nova versão da aplicação passe por testes antes do deploy**
2. **Falhas nos testes bloqueiem a propagação de versões defeituosas**
3. **A orquestração agregue valor ao pipeline de CI/CD**

### ✅ Ações que tomaremos agora:

#### 1. **Refatorar o** flow\_versao1.py

Adicionaremos uma tarefa que executa os testes automatizados antes da tarefa principal da calculadora. Isso pode ser feito de forma direta com subprocess.run(["python", "-m", "unittest", "discover"]).

#### 2. **Refatorar** prefect.yaml

Precisaremos refletir a nova dependência no flow, garantindo que o entrypoint corresponda ao script que executa **primeiro os testes e depois a lógica principal**.

#### 3. **Refatorar o** .github/workflows/prefect-deploy.yml

Este workflow será atualizado para garantir:

* Execução dos testes via prefect (ou em um step separado do GitHub Actions, conforme decidirmos)
* Falha explícita se os testes não passarem

### 📌 O que muda com essa refatoração?

Texto

O conteúdo gerado por IA pode estar incorreto.

* O fluxo agora **prioriza os testes**.
* Qualquer **erro nos testes interrompe imediatamente** o processo.
* Toda execução do Prefect, seja via prefect deploy ou CLI, passará por validação da lógica de negócio.

### ✅ Refatorado prefect.yaml

Texto

O conteúdo gerado por IA pode estar incorreto.

### ✅ O que foi ajustado?

|  |  |
| --- | --- |
| Item | Ajuste |
| 🔁 schedules removido | Para evitar execuções automáticas desnecessárias. O fluxo só roda via CLI ou GitHub Action |
| 🧪 Testes no flow | Agora os testes são executados dentro do fluxo, com validação antes do deploy |
| 🎯 Entrada principal | Aponta diretamente para flow\_versao1.py com o nome da função correta |
| 🏷️ Tags | Permitem classificação no Prefect Cloud (versao1, testes, validado) |
| 💬 Descrição clara | Explica o propósito e comportamento do fluxo |

## ✅ Novo .github/workflows/prefect-deploy.yml

### Linha do tempo O conteúdo gerado por IA pode estar incorreto.

### 📌 O que está diferente (e melhor):

|  |  |
| --- | --- |
| Etapa | Explicação |
| python -m unittest | Executa os testes antes de qualquer deploy |
| curl | Testa a conectividade com a API do Prefect |
| prefect deploy -n | Usa o nome definido no prefect.yaml |
| for i in {1..5} | Reforça estabilidade com tentativas múltiplas em caso de falhas momentâneas |
| Secrets | Usa as **mesmas variáveis já criadas** no GitHub: PREFECT\_API\_KEY e PREFECT\_API\_URL |

Teste local e manual: python flows/flow\_versao1.py

Nota: passando neste teste estamos prontos para fazer um deploy via GitHub Actions, simulando uma entrega profissional.

## 📌 **Próximo Passo: Executar o Deploy com GitHub Actions**

### 🔁 O que devemos fazer agora:

1. **Verificar se o arquivo .github/workflows/prefect-deploy.yml está correto**  
   (com nome do flow, chave correta e sem agendamento a cada 60s)
2. **Fazer um commit no branch principal (main) para disparar o deploy**
3. **Ir até o GitHub > Actions** e verificar se:
   * O workflow foi iniciado
   * O deploy foi executado com sucesso
4. **Validar no Prefect Cloud** se o flow aparece e está ativo após o deploy.

### ⚙️ Se o deploy falhar:

* Vamos revisar logs do GitHub Actions
* Verificar os secrets PREFECT\_API\_KEY e PREFECT\_API\_URL no GitHub
* Corrigir qualquer falha no prefect-deploy.yml ou prefect.yaml

### 💡 Só após isso:

Podemos passar com tranquilidade para a **Etapa 3: novas operações e novos fluxos.**

Lembre-se que para o GitHub Action executar seu fluxo, tudo deve estar exatamente nos locais especificados.

### No caso desse laboratório:

CALCULADORA\_SIMPLES/.github/workflows/prefect-deploy.yml

Se tudo estiver correto, basta acompanhar no GitHub Action, e esperar para que seu resultado seja como este:

Texto

O conteúdo gerado por IA pode estar incorreto.

## 🧭 Próximos Passos Sugeridos

### 🔁 **ETAPA 3 – Expansão Funcional e Operacional**

#### 1. 📈 **Adicionar novas operações matemáticas**

* Multiplicação, Divisão, Porcentagem
* Refatorar classe Calculadora
* Atualizar flow\_versao1.py e test\_operacoes.py

#### 2. 🧪 **Ampliar cobertura de testes**

* Garantir testes para as novas operações
* Testes de exceção (divisão por zero, tipos inválidos, etc.)

#### 3. 🧱 **Atualizar estrutura modular (se necessário)**

* Dividir operacoes.py em múltiplos arquivos? (ex: aritmetica.py, validador.py)
* Avaliar se o \_\_init\_\_.py do módulo precisa de ajustes

#### 4. 🧾 **Atualizar** prefect.yaml **e** prefect-deploy.yml

* Caso o flow principal mude de nome ou seja duplicado
* Caso inclua novos flows e precise registrar mais de um

#### 5. 🌐 **Deploy contínuo**

* Validar que cada novo commit (com push na main) atualiza seu deploy no Prefect Cloud
* Verificar na interface Prefect se o flow atualizado está versionado corretamente

#### 6. 📄 **Atualizar documentação**

* Diagrama de classes com novas operações
* Documentação em docs/arquitetura.md
* Atualizar README com status do CI/CD

## 🎯 Conclusão

Você completou com excelência um ciclo real de desenvolvimento moderno:

* Projeto modular e testado
* CI/CD real com GitHub Actions
* Deploy monitorado no Prefect Cloud
* Rastreabilidade e logs automáticos

## ✅ Confirmação: ETAPA 4 — Finalização de Funcionalidades (Porcentagem)

**📍Situação Atual:**

* Etapas 0 a 3 finalizadas com êxito:
  + Ambiente virtual validado
  + Prefect e GitHub Actions funcionais
  + Operações soma, subtração, multiplicação e divisão implementadas (em progresso)
* Arquitetura já documentada
* Flows e testes iniciais funcionando

## 📌 Etapa 4 — O que vamos construir agora

### 🧩 Objetivo:

### 🧩 ETAPA 4 — Multiplicação e Divisão

* Adicionar as operações:
  + multiplicar(a: float, b: float) -> float
  + dividir(a: float, b: float) -> float (com tratamento para divisão por zero)
* Atualizar:
  + modules/operacoes.py (incluir métodos e atualizar Calculadora)
  + tests/test\_operacoes.py (novos testes com assert)
  + flows/flow\_versao1.py (incluir chamadas no executar\_operacoes)
* Fazer **commit + push**
* Observar CI/CD (Prefect + GitHub Actions)

## 🛠️ 1. Atualização de modules/operacoes.py

### 🔁 Substituir o conteúdo atual por:

Texto

O conteúdo gerado por IA pode estar incorreto.

## 🧪 2. Atualização de tests/test\_operacoes.py

### ➕ Adicione abaixo dos testes existentes:

Texto

O conteúdo gerado por IA pode estar incorreto.

## ⚙️ 3. Atualização de flows/flow\_versao1.py

### 🔁 Modificar a função executar\_operacoes:

Texto

O conteúdo gerado por IA pode estar incorreto.

## ✅ 4. Testes locais

Antes do commit:

Texto

O conteúdo gerado por IA pode estar incorreto.

### ✅ **1.** python -m unittest

Este comando executa o módulo de testes do Python chamado unittest, que é a biblioteca padrão para testes unitários.

### ✅ **2.** discover

A palavra discover é um **subcomando do unittest** que serve para **descobrir automaticamente** os arquivos de teste em uma estrutura de diretórios.

Ele faz isso procurando:

* Arquivos que **comecem com test** e **terminem com .py**
* Dentro do diretório que você indicar (ou . por padrão)

### ✅ **3.** -s tests

A flag -s significa start directory, ou seja, **diretório inicial** onde o unittest começará a procurar arquivos de teste.

### 🔁 Outras flags úteis:

|  |  |  |
| --- | --- | --- |
| Flag | Significado | Exemplo |
| -s | Diretório inicial | -s tests |
| -p | Padrão de nome dos arquivos | -p "\*\_test.py" |
| -t | Diretório raiz do projeto | -t . |

### ✅ Exemplo completo alternativo:



Esse comando dirá: "procure por arquivos test\_\*.py na pasta tests/".

### 💾 5. Commit e Push

Texto

O conteúdo gerado por IA pode estar incorreto.

## 🔄 6. Verificação no GitHub Actions e Prefect Cloud

* Vá até Actions no GitHub → acompanhe o workflow
* No Prefect Cloud, verifique se o flow flow-versao1 foi atualizado corretamente e pode ser executado com os novos recursos

#### 🔥Nota importante: você pode ter excedido os limites de uso do Prefect Cloud – versão gratuita e não poderá ver seu fluxo rodando, se isso acontecer e se chegou até aqui não se preocupe, você domina o processo é apenas uma restrição financeira.

## 🐳 ALTERNATIVA **Ideal: Prefect Server + Docker (Offline)**

### ✅ Vantagens para o laboratório:

* **Sem limite de uso**
* **Execução local e auditável**
* **Compatível com workers dockerizados**
* Ideal para treinamento de:
  + Infraestrutura ágil
  + Observabilidade
  + Orquestração modular

## ⚙️ Etapas para Implantação Local com Docker

### 📁 Estrutura que vamos criar:

Forma

O conteúdo gerado por IA pode estar incorreto.

### 🔧 Funcionalidades:

* Executar Prefect UI localmente via localhost:4200
* Registrar flow\_versao1 localmente
* Executar manual ou via agendamento
* Conectar worker local com pool “hello-pool”

## ✅ agora precisamos construir:

1. docker-compose.yml completo com prefect-server, postgres e hasura
2. prefect-server.env com configurações padrão
3. Scripts de:
   * Start do servidor
   * Registro de deployment local
   * Execução do flow com CLI

E tudo isso deve ser adicionado dentro da pasta infra/ do projeto.

## 🧠 O que é o **Hasura**?

### 🔎 Definição:

**Hasura** é um **engine de GraphQL de alto desempenho**, que permite **expor automaticamente** um banco de dados (como PostgreSQL) via uma **API GraphQL em tempo real**.

## 🧩 E qual o papel do Hasura no **Prefect Server**?

O **Prefect Server (self-hosted)** é composto por vários componentes que **simulam o ambiente do Prefect Cloud**, localmente.

### Arquitetura do Prefect Server:

|  |  |
| --- | --- |
| Componente | Função |
| postgres | Banco de dados para armazenar flows, runs, logs |
| hasura | Middleware que transforma o banco PostgreSQL em API GraphQL |
| graphql | API principal usada pelo servidor Prefect (usa o Hasura por baixo) |
| orion-api (Prefect 2.x) | Serviço backend responsável pela orquestração e execução |
| UI | Interface Web (acessada via http://localhost:4200) |

## ⚙️ Por que o Prefect usa Hasura?

Porque o Prefect, desde o início, foi **construído sobre GraphQL**, e o Hasura **fornece uma camada instantânea de API** sobre o banco de dados.

### Vantagens:

* 🚀 Rapidez no desenvolvimento da API
* 📊 Querying eficiente de metadata (flows, runs, logs)
* 🔧 Integração limpa entre backend e UI

## 🎯 Preciso me preocupar com o Hasura?

**Não diretamente.** Ele será iniciado como um serviço de apoio via docker-compose, mas:

* Você **não precisa configurar o Hasura manualmente**
* Você **não interage com ele diretamente**
* Ele só **fornece a API GraphQL interna** que o Prefect UI consome

## ✅ Analogia simples

Pense no Hasura como:

“O tradutor entre o banco de dados e a interface do Prefect Server. Você não fala com ele diretamente, mas ele está lá, fazendo tudo funcionar.”

## 🧭 Conclusão

* **Você não precisa aprender Hasura** para usar o Prefect Server
* **Ele é obrigatório** na versão self-hosted do Prefect Server 2.x (como parte da stack original)
* Ele será apenas um **serviço coadjuvante no seu docker-compose.yml**, assim como o PostgreSQL

## ✅ PLANO DE AÇÃO — ETAPA 4 com Docker

### 📁 Estrutura a ser criada em infra/

Forma

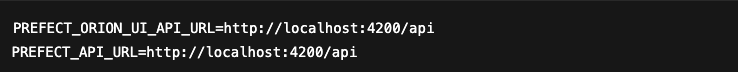
O conteúdo gerado por IA pode estar incorreto.

## 🛠️ 1. infra/docker-compose.yml

Texto

O conteúdo gerado por IA pode estar incorreto.

## 🧪 2. infra/prefect-server.env



## ▶️ 3. infra/run\_server.sh

Texto

O conteúdo gerado por IA pode estar incorreto.

## ▶️ 4. infra/run\_worker.sh

Texto

O conteúdo gerado por IA pode estar incorreto.

🎯Nota: lembre-se de executar esse comando com o ambiente virtual ativado e Prefect 2.16.4 instalado.

## ✅ Etapas de Execução

### 🔹 1. Iniciar o servidor



### 🔹 2. Verificar UI

Acesse:



### 🔹 3. Iniciar o worker local



### 🔹 4. Registrar o flow (como fizemos anteriormente)



### 🔹 5. Executar o flow pela UI ou com:



## ✅ Resultado Esperado

* Prefect UI local rodando
* Worker local conectado
* Flow flow\_versao1 disponível
* Execução local completa, com logs e testes visíveis
* Finalização formal da ETAPA 4 com Docker

## 🎯 Vantagens de usar src/

|  |  |
| --- | --- |
| Benefício | Explicação |
| ✅ Evita importações acidentais | Quando usamos src/, os testes e scripts só funcionam se a estrutura de importação estiver correta |
| ✅ Facilita empacotamento (setuptools, pip) | A maioria dos projetos Python usa src/ para preparar distribuição |
| ✅ Organiza melhor entre “código-fonte” e “infraestrutura” | Claramente separa src/, tests/, infra/, docs/, flows/ |
| ✅ Reforça boas práticas para times | Evita shortcuts como from modules import x funcionando só por acaso |
| ✅ Integra melhor com ferramentas modernas de CI/CD | Pytest, coverage, linters, etc., reconhecem melhor a estrutura com src/ |

## 📁 Estrutura sugerida com src/

Texto

O conteúdo gerado por IA pode estar incorreto.

O pacote passaria a ser calculadora, e o import seria feito como:



## ✅ Próximo passo sugerido (pós-ETAPA 4)

* Criar src/calculadora/\_\_init\_\_.py
* Mover modules/operacoes.py → src/calculadora/operacoes.py
* Atualizar imports nos flows/ e tests/
* Rodar pytest ou unittest para garantir que os imports foram atualizados corretamente
* Incluir src/ no PYTHONPATH se necessário via:



## 🔥 Dica para garantir em repositórios

Se você versionar esse script no GitHub, vale lembrar que:

* O chmod +x **não é preservado via Git para o sistema local do usuário**
* É boa prática documentar isso no README.md com:

## 🧭 Proposta de execução da nova ETAPA 4

### 🎯 Objetivo

Gerar um **roteiro à prova de falhas**, que:

* Configure o ambiente do zero
* Use Docker para orquestração local
* Estabeleça prefect.yaml, flow, deployment, work pool, worker
* Permita execução imediata e validada via UI local
* Seja 100% replicável por qualquer aluno

# Roteiro Oficial - Execução Precisa da ETAPA 4 (CALCULADORA\_SIMPLES)

## Objetivo

Executar com perfeição a ETAPA 4 do projeto CALCULADORA\_SIMPLES, utilizando Docker + Prefect Server local, garantindo que cada etapa esteja livre de erros, bem documentada e replicável.

## Etapa 1: Reset do Ambiente

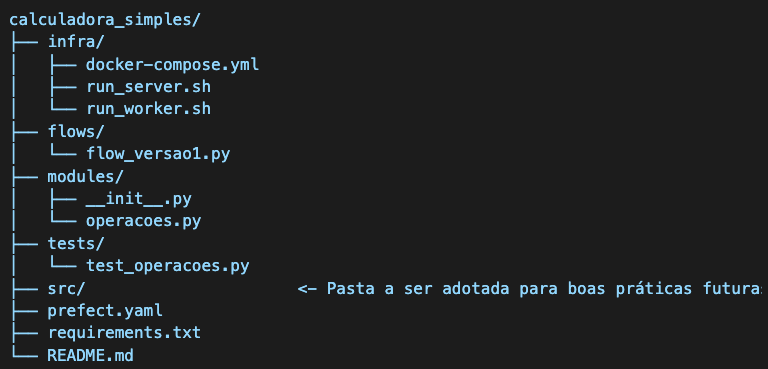
**Objetivo:** garantir que nenhum resíduo de execuções anteriores cause erros.

Texto

O conteúdo gerado por IA pode estar incorreto.

## Etapa 2: Reorganização da Estrutura do Projeto

**Objetivo:** garantir que o projeto siga boas práticas de organização.



⚠️ **Nota:** A pasta src/ será considerada na ETAPA 5 para organização modular recomendada.

## Etapa 3: Tornar Scripts Executáveis

**Objetivo:** garantir que os scripts shell possam ser executados diretamente.



## Etapa 4: Inicialização do Prefect Server local

**Objetivo:** levantar o servidor Prefect, PostgreSQL e Hasura com Docker.



**Validação:** acessar <http://localhost:4200>

Se curl http://localhost:4200/api retornar HTML, o servidor está ativo.

## Etapa 5: Definir a API local para o Prefect

**Objetivo:** garantir que os comandos CLI afetem o servidor local (e não o Prefect Cloud).



## Etapa 6: Criação do Work Pool

**Objetivo:** registrar o work pool local para execução de flows do tipo "pull" (modo process).



Validação:



Deve listar hello-pool com tipo process.

## Etapa 7: Atualizar ou Criar o prefect.yaml

Texto

O conteúdo gerado por IA pode estar incorreto.

## Etapa 8: Deploy do Flow



* Quando perguntado sobre agendamento: n
* Quando perguntado se deseja salvar: y
* Quando perguntado se deseja sobrescrever: y

## Etapa 9: Iniciar o Worker (via script ou manualmente)

**Caso deseje utilizar o script:**



Ou iniciar diretamente:



Validação: o terminal deve exibir:



## Etapa 10: Executar o Flow

Acesse a UI local: <http://localhost:4200>

1. Clique em Deployments
2. Clique em hello-deploy
3. Clique em Run

Ou via CLI:

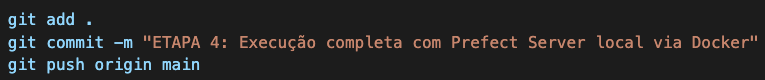


## Etapa 11: Validar Resultado

1. Acesse Flow Runs na UI
2. Clique sobre o nome do run (ex: silver-tuna)
3. Verifique os logs:
   * Testes automatizados executados
   * Resultados das operações: soma, subtração, multiplicação, divisão

## Etapa 12: Commit e Push para GitHub

**Objetivo:** consolidar a ETAPA 4 no repositório e validar o CI via GitHub Actions.



### Validação:

1. Acesse o repositório no GitHub
2. Clique em Actions
3. Verifique o workflow prefect-deploy.yml

Mesmo com limite do Prefect Cloud atingido, o deploy ainda pode ser visualizado no GitHub Actions para fins didáticos.

## O que esperar no GitHub Actions

Interface gráfica do usuário, Texto

O conteúdo gerado por IA pode estar incorreto.

# ETAPA 5 (CALCULADORA\_SIMPLES)

## Objetivo

Avançar no desenvolvimento incremental do projeto CALCULADORA\_SIMPLES, com foco em:

* Organização do projeto em conformidade com boas práticas (uso da pasta src)
* Introdução de novas funcionalidades (porcentagem, módulo, etc.)
* Integração contínua (CI) com testes automáticos
* Deploy e execução via Prefect Server local

### 🔥 Ação recomendada para antes de iniciarmos os testes da ETAPA5, visto que nosso laboratório está acontecendo em uma sequência.

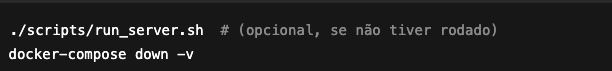
### 🥷🏼 Pode ser chamda de ETAPA 0

## 🔁 ETAPA 0 — RESET COMPLETO DO AMBIENTE

🎯 Objetivo:

Limpar completamente o ambiente anterior, **removendo containers, volumes, pools e deployments antigos**, evitando interferências com a nova fase (ETAPA 5).

## **✅ PASSO 1 — Parar e Remover Containers Docker**



📌 O -v remove os volumes associados, apagando dados do PostgreSQL e Prefect Server.

## ✅ PASSO 2 — Limpar Volumes Docker Não Utilizados



📌 Remove volumes órfãos que podem estar ocupando espaço e gerando conflitos.

## ✅ PASSO 3 — Remover Work Pool Anterior (se existir)



📌 Se o pool hello-pool já existir, deletamos para recriá-lo limpo na ETAPA5.

⚠️ se der erro de “not found”, está tudo certo — significa que já foi limpo.

Texto

O conteúdo gerado por IA pode estar incorreto.

## Por fim, devemos apontar o PYTHONPATH para src, antes precisamos saber, para onde ele está apontando:



### 🚀 Se for necessário corrigir use:



🚀 Nota Importante**:** essa “zeragem” também reforça as boas práticas didáticas para reforçar o aprendizado.

## 🎯 Configurar o VSCode para resolver os imports

Crie ou edite o arquivo .vscode/settings.json na raiz do projeto com o seguinte conteúdo:

{

"python.analysis.extraPaths": ["./src"]

}

📍Importante: Isso evita alertas do Pylance como **reportMissingImports** e melhora a experiência com autocomplete e linting.

## 🚀 (Opcional) Validar com sys.path

Dentro de um script Python, você pode validar se src está acessível:



## Etapa 1: Reorganização com pasta src

**Objetivo:** isolar o código de negócio em um módulo src conforme padrão de projetos Python.

## Etapa 1: Reorganização com pasta src

**Objetivo:** isolar o código de negócio em um módulo src conforme padrão de projetos Python.

Texto

O conteúdo gerado por IA pode estar incorreto.

✅ Atualize os **imports** nos scripts (from modules.operacoes import ...) e configure o PYTHONPATH se necessário para apontar para src.

## Etapa 2: Novas Funcionalidades

**Objetivo:** implementar e testar novas operações na calculadora.

1. Atualizaroperacoes.py

**Texto

O conteúdo gerado por IA pode estar incorreto.**

**2.** Atualizartest\_operacoes.py



## Etapa 3: Atualizar Flow Prefect

Objetivo: incluir as novas funções no flow existente no flow\_versao1.

Texto

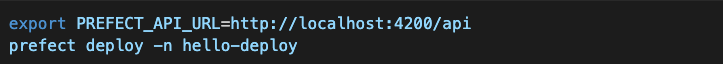
O conteúdo gerado por IA pode estar incorreto.

### Etapa 4: Atualizar prefect.yaml

Texto

O conteúdo gerado por IA pode estar incorreto.

## Etapa 5: Executar novo deploy



### Confirmar:

* [y] salvar
* [y] sobrescrever

### 🎯 Conclusão

A ETAPA 5 consolida a arquitetura modular do projeto com a introdução da pasta src, expande as funcionalidades da calculadora e mantém o ciclo de deploy contínuo funcionando via Prefect Server local e GitHub Actions.

## Etapa 6: Iniciar worker e executar

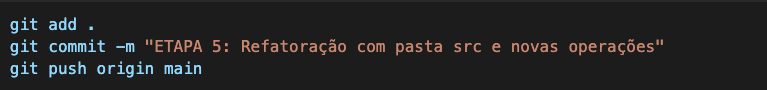


Na interface: <http://localhost:4200> → Deployments → hello-deploy → Run

Ou:



## Etapa 7: Validar CI via GitHub Actions



🔥 Acompanhar o workflow prefect-deploy.yml no GitHub Actions.