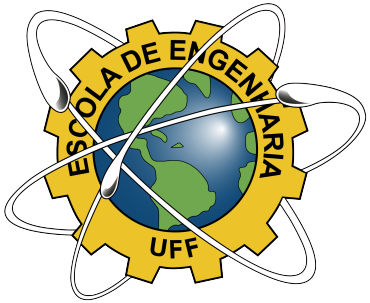

ESCOLA DE ENGENHARIA
Universidade Federal Fluminense

PPGEET
Programa de Pós-Graduação em Engenharia Elétrica e de Telecomunicações

Prefect: Uma Revisão

Tópicos Especiais em Sistemas de Energia Elétrica II: Desenvolvimento Ágil com Python e Containers

Prof. Angelo Colombini

Rene Cruz Freire

Universidade Federal Fluminense
Campus Niterói

renefreire@id.uff.br

Sumário

Introdução

- Nos últimos anos da década de 2010, o mundo presenciou uma explosão no volume de dados gerados por aplicações, dispositivos IoT, redes sociais, sistemas de monitoramento e muito mais.

- Esse aumento trouxe a necessidade de:
  - processar grandes volumes de dados com frequência;
  - integrar múltiplas fontes e destinos de dados (ETL/ELT);
  - automatizar análises, treinamento de modelos de *Machine Learning*, atualizações de dashboards, entre outros;
  - garantir confiabilidade em pipelines complexos e interdependentes.

- Com isso, a **orquestração de dados** se tornou um componente essencial na infraestrutura de dados moderna.

Introdução

- Ferramentas como o Apache Airflow (lançada em 2015) dominavam o mercado, mas se deparavam com desafios como:
  - configuração complexa e difícil de escalar;
  - pouca flexibilidade com programação dinâmica em Python;
  - dependência de DAGs (*Directed Acyclic Graphs* - Grafos Acíclicos Dirigidos) estáticos, que limitam a lógica condicional;
  - pouco suporte nativo à execução assíncrona;
  - baixa observabilidade nativa (precisava de muitos plugins ou customizações).

- Desenvolvedores e engenheiros de dados começaram a buscar alternativas mais modernas, leves e fáceis de usar.

Introdução

- O **Prefect** foi criado oficialmente em 2018 por Jeremiah Lowin, um ex-engenheiro de dados e executivo de tecnologia.

- Ele tinha vivenciado em primeira mão os desafios de gerenciar workflows complexos com ferramentas tradicionais como Airflow e Luigi, e decidiu desenvolver uma solução mais adequada à realidade dos times modernos de dados.

- A missão do Prefect desde o início foi ”eliminar o trabalho oculto (invisible work) da orquestração de dados”— tarefas como tratamento de falhas, monitoramento, reexecução, logs e controle de dependências, que normalmente exigiam muitos ajustes manuais.

Introdução

Documentação Oficial do Prefect

"O Prefect é uma plataforma moderna de orquestração de workflows (fluxos de trabalho) que permite o gerenciamento, agendamento, monitoramento e execução confiável de pipelines de dados e automações complexas."

- Ele foi projetado para facilitar a vida de engenheiros de dados, cientistas de dados e desenvolvedores que lidam com tarefas repetitivas, dependências entre processos e execução em ambientes diversos.
- Enquanto outras ferramentas de orquestração, como Apache Airflow, também lidam com esses problemas, o Prefect se destaca pela sua abordagem "*engineered for simplicity*", com foco em facilidade de uso, observabilidade e flexibilidade.

Principais Objetivos

1. Orquestração confiável de workflows complexos.
2. Gerenciamento de dependências e controle de fluxo.
3. Monitoramento e observabilidade integrados.
4. Resiliência a falhas e reexecução automática.
5. Execução local, em nuvem, ou em ambientes híbridos.

Componentes Fundamentais

1. Flow
   - É a definição do pipeline como um todo.
   - Um flow é uma coleção de tarefas interconectadas, com uma lógica de dependência entre elas.
   - É o que representa o workflow no Prefect.
2. Task (Tarefas)
   - Cada etapa de um fluxo é representada por uma tarefa.
   - Elas podem ser funções Python comuns que realizam operações específicas, como consultar uma API, processar um arquivo, carregar dados em um banco, etc.
3. Deployment (Implantação)
   - No Prefect 2.0+, os deployments são configurações que ligam um fluxo a um agendador e a um agente.
   - Um mesmo fluxo pode ter vários deployments com parâmetros diferentes.

4. Agent
   - Um agente é um processo que escuta por execuções de fluxos e os executa quando acionado.
   - Ele pode rodar localmente, em contêineres Docker, em Kubernetes, em nuvem, entre outros ambientes.

5. Orion (Prefect 2.0)
   - O Orion é o nome do engine subjacente que impulsiona o Prefect 2.0.
   - Ele é construído para ser leve, rápido, e com uma arquitetura mais modular e assíncrona em comparação com o Prefect 1.0.

Vantagens e Desvantagens

Vantagens

- **Pythonic**: usa Python puro para construir workflows, sem DSLs complicadas.
- **Flexibilidade**: pode ser usado em diferentes ambientes, sem necessidade de modificar o código para rodar localmente ou na nuvem.
- **Observabilidade**: fornece *dashboards* ricos para monitoramento e controle de execuções.
- **Resiliência**: tolerância a falhas com *retries* automáticos.
- **Integração Facilitada**: Conecta-se facilmente a ferramentas como Airflow, Dask, Kubernetes e Snowflake.

Desvantagens

- Comunidade menor em comparação com o Airflow.
- Algumas funcionalidades avançadas estão disponíveis apenas no Prefect Cloud.

Funcionamento

- O Prefect trabalha em um modelo híbrido: a orquestração de tarefas é feita em um servidor, os códigos das tarefas são armazenados em outro e a execução é feita em um terceiro.

- Essa arquitetura permite a descentralização completa das execuções em diferentes centros de custo, além de permitir uma maior privacidade e segurança de todos os servidores envolvidos.

- O site oficial da plataforma lista os passos do desenvolvimento com Prefect.

1. **Construir seu Flow**

- Uma pipeline, no Prefect, recebe o nome de Flow.
- Você deve utilizar o pacote Python prefect (em nosso caso, na versão 0.15.9) para desenvolver seu fluxo de trabalho.
- Depois de desenvolver, você pode executá-lo e testá-lo em sua máquina.

2. **Registrar seu Flow**

- Esse passo consiste em armazenar seu código em um repositório de objetos, de forma que ele possa ser acessado posteriormente.
- Depois de armazená-lo, sabendo sua localização, você envia esse metadado para o servidor do Prefect (registra o Flow).
- Isso permite que você possa localizá-lo na interface web do Prefect e inspecioná-lo.

3. Executar um Agente

- Agentes do Prefect são processos que se comunicam com o servidor do Prefect, procurando por cargas de trabalho agendada.
- Quando ele encontra, lança sua execução.
- Agentes podem ser executados em qualquer lugar, o que permite que as cargas de trabalho sejam lançadas em diferentes centros de custo.

4. Agendar Cargas de Trabalho

- É possível agendar cargas de trabalho para serem executadas em um determinado horário, ou com certa recorrência.
- Você pode fazer isso através do próprio código, em Python (recomendado para manter consistência), ou através da interface web do Prefect.

5. Executar o Flow

- O agente do Prefect encontrará a carga de trabalho agendada e a executará.
- Dessa forma, a carga de trabalho será executada onde o agente estiver rodando.
- Ao mesmo tempo, todo o processo de execução e logs poderão ser acompanhados através da interface web do Prefect, já que o agente os comunica ao servidor.

6. Monitorar e Gerenciar

- É possível monitorar e gerenciar as cargas de trabalho através da interface web do Prefect.
- Você pode ver o status de cada carga de trabalho, visualizar os logs, inspecionar o código, etc. de quaisquer cargas de trabalho, seja em execução ou já finalizadas.

Desenvolvendo o Primeiro Flow

- Aqui, vamos tratar do desenvolvimento de uma pipeline de exemplo, que consiste em:
  1. baixar dados de uma API;
  2. converter esses dados em um formato tabular;
  3. armazenar esses dados em um arquivo CSV.

- Esse exemplo tratará exclusivamente de uma execução local, em sua máquina, sem interações com o servidor do Prefect ou mecanismos de armazenamento.

- O exemplo a seguir foi extraído do site do Escritório de Dados da Prefeitura do Rio de Janeiro

Preparando o Ambiente

- Requisitos:
  - Python 3.9, pois a versão deve coincidir com a utilizada pelo escritório.
  - pip, para instalar dependências.
- A primeira boa prática é criar um arquivo requirements.txt para armazenar as dependências do projeto.
- Nesse arquivo, deve-se colocar todas as bibliotecas que serão utilizadas no desenvolvimento do seu Flow.
- Neste exemplo será usado o seguinte:

requirements.txt

```
prefect==0.15.9
requests
pandas
```

Preparando o Ambiente

- Note que o único pacote que tem sua versão especificada é o Prefect, pois a versão dele deve corresponder à utlizada nos servidores.
- Em seguida, para instalar as dependências declaradas nesse arquivo, deve-se executar o seguinte comando:

cmd.exe

```
pip install -r requirements.txt
```

- E pronto, tudo que é necessário para desenvolver o Flow está disponível.
- Caso, no futuro, haja necessidade de adicionar novas dependências, basta adicionar no arquivo requirements.txt e executar o comando acima novamente.

Preparando o Ambiente

- Para manter um padrão de desenvolvimento, deve-se criar os seguintes arquivos:
  - tasks.py: arquivo que conterá as funções que serão utilizadas no Flow;
  - flows.py: arquivo que conterá o Flow propriamente dito;
  - utils.py: arquivo que conterá funções auxiliares, como por exemplo, funções para imprimir logs.

Criando Funções Auxiliares

- Começando então no arquivo utils.py, vamos criar uma função que imprime logs no console.
- Essa função será útil para acompanhar o andamento do Flow.
- Como é uma função que envolve conhecer o Prefect um pouco mais a fundo, vamos explicar o que ela faz.

utils.py

```python
import prefect

def log(message) -> None:
    """Logs a message"""
    prefect.context.logger.info(f"\n{message}")
```

Criando Funções Auxiliares

- A função “log” recebe uma mensagem e a imprime no console.
- Ela faz isso através do objeto “prefect.context.logger”, que é um objeto que contém o logger do Prefect.
- Esse logger é um objeto que está dentro do “prefect.context”, esse que contém informações sobre o Flow e a execução atual, como por exemplo, o nome do Flow e o ID da execução atual.
- Por isso, é importante que essa função seja chamada dentro de um Flow, pois ela depende do objeto “prefect.context” para funcionar.

Criando *Tasks*

- Aqui é onde serão criadas as funções que implementam os passos da pipeline.
- Essas funções serão chamadas de tasks.
- Então, abrindo o arquivo tasks.py, cria-se as seguintes funções:

Criando *Tasks*

tasks.py (parte I)

```python
from io import StringIO

import pandas as pd
from prefect import task
import requests

from utils import log

@task
def download_data(n_users: int) -> str:
    """
    Baixa dados da API https://randomuser.me e retorna um texto em formato CSV.

    Args:
        n_users (int): número de usuários a serem baixados.

    Returns:
        str: texto em formato CSV.
    """
    response = requests.get(
        "https://randomuser.me/api/?results={}&format=csv".format(n_users)
    )
    log("Dados baixados com sucesso!")
    return response.text
```

Criando *Tasks*

tasks.py (parte II)

```python
@task
def parse_data(data: str) -> pd.DataFrame:
    """
    Transforma os dados em formato CSV em um DataFrame do Pandas, para facilitar sua manipulação.

    Args:
        data (str): texto em formato CSV.

    Returns:
        pd.DataFrame: DataFrame do Pandas.
    """
    df = pd.read_csv(StringIO(data))
    log("Dados convertidos em DataFrame com sucesso!")
    return df
```

Criando *Tasks*

tasks.py (parte III)

```python
@task
def save_report(dataframe: pd.DataFrame) -> None:
    """
    Salva o DataFrame em um arquivo CSV.

    Args:
        dataframe (pd.DataFrame): DataFrame do Pandas.
    """
    dataframe.to_csv("report.csv", index=False)
    log("Dados salvos em report.csv com sucesso!")
```

- Vale ressaltar que as funções são decoradas com o "@task", que é um decorator do Prefect.
- Esse decorator é necessário para que o Prefect reconheça as funções como tasks.

Desenvolvendo o Primeiro Flow

Definindo um Flow

- Agora, vamos criar o Flow propriamente dito.
- Aqui vamos definir como as tasks interagem entre si e os parâmetros que elas recebem.
- Para isso, abra o arquivo flows.py e crie a seguinte função:

flows.py

```python
from prefect import Flow, Parameter

from tasks import (
    download_data,
    parse_data,
    save_report,
)

with Flow("Users report") as flow:

    # Parâmetros
    n_users = Parameter("n_users", default=10)

    # Tasks
    data = download_data(n_users)
    dataframe = parse_data(data)
    save_report(dataframe)
```

Definindo um Flow

- Foi criado um Flow chamado “Users report” com o seguinte parâmetro:
  - “n_users”: número de usuários a serem baixados.
- Observe que foi configurado um valor padrão para esse parâmetro, que é 10.
- Isso significa que, caso o parâmetro não seja passado na hora de executar o Flow, ele será executado com o valor padrão.

Definindo um Flow

- Além disso, foram inseridas as três tasks criadas anteriormente.
- Observe que a task “download_data” recebe o parâmetro “n_users” como argumento.
- Isso significa que, quando a task for executada, ela receberá o valor do parâmetro “n_users” como argumento.
- Depois disso, a task “parse_data” recebe o retorno da task “download_data” como argumento, fazendo com que o dado baixado seja passado a ela, para converter em um DataFrame.
- Por fim, a task “save_report” recebe o retorno da task “parse_data” como argumento, fazendo com que o DataFrame seja salvo em um arquivo CSV.

Desenvolvendo o Primeiro Flow

Executando

- Para facilitar a execução, cria-se um arquivo run.py com o seguinte código nele:

run.py

```
from flows import flow

flow.run()
```

- Depois, pode-se executar o Flow com o seguinte comando:

cmd.exe

```
python3 run.py
```

Executando

- A saída terá o seguinte formato:

```
[2022-10-05 14:40:40-0300] INFO - prefect.FlowRunner | Beginning Flow run for 'Users report'
[2022-10-05 14:40:40-0300] INFO - prefect.TaskRunner | Task 'n_users': Starting task run...
[2022-10-05 14:40:40-0300] INFO - prefect.TaskRunner | Task 'n_users': Finished task run for task with final state: 'Success'
[2022-10-05 14:40:40-0300] INFO - prefect.TaskRunner | Task 'download_data': Starting task run...
[2022-10-05 14:40:40-0300] INFO - prefect.download_data |
Dados baixados com sucesso!
[2022-10-05 14:40:40-0300] INFO - prefect.TaskRunner | Task 'download_data': Finished task run for task with final state: 'Success'
[2022-10-05 14:40:40-0300] INFO - prefect.TaskRunner | Task 'parse_data': Starting task run...
[2022-10-05 14:40:40-0300] INFO - prefect.parse_data |
Dados convertidos em DataFrame com sucesso!
[2022-10-05 14:40:40-0300] INFO - prefect.TaskRunner | Task 'parse_data': Finished task run for task with final state: 'Success'
[2022-10-05 14:40:40-0300] INFO - prefect.TaskRunner | Task 'save_report': Starting task run...
[2022-10-05 14:40:40-0300] INFO - prefect.save_report |
Dados salvos em report.csv com sucesso!
[2022-10-05 14:40:40-0300] INFO - prefect.TaskRunner | Task 'save_report': Finished task run for task with final state: 'Success'
[2022-10-05 14:40:40-0300] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
```

Executando

- Observe que um arquivo chamado “report.csv” foi criado.
- Esse arquivo contém os dados baixados e convertidos em um DataFrame.
- Observe também que as mensagens de log foram impressas na saída.
- Isso acontece porque, como vimos anteriormente, as funções “log” são decoradas com o “@task”, que é um decorator do Prefect.
- Isso faz com que o Prefect reconheça as funções como tasks e, por isso, as mensagens de log são impressas na saída.

Referências

- https://www.prefect.io - Informações gerais sobre a missão da empresa, recursos da ferramenta, Prefect Cloud e Prefect OSS.
- https://docs.prefect.io - Informações sobre a arquitetura, componentes (Flow, Task, Agent, Deployment), diferenças entre Prefect 1.x e 2.x, entre outras.
- https://medium.com/the-prefect-blog/why-not-airflow-4cfa423299c4 - Comparação direta com Airflow e descrição dos problemas enfrentados com DAGs estáticos.
- https://docs.dados.rio/guia-desenvolvedores/pipelines - Desenvolvimento de pipelines local.
- https://discourse.prefect.io - Comunidade Prefect no Discourse