Como estamos melhorando a gestão de demandas com Python: Um case de ETL com Python e Asana

#PythonNordeste
Trilha Totó : 28/08/2022

Essa história começa num Domingo…

Quando resolvi tentar acessar uma API …

```
#### ASANA PARAMETERS ####
personal_access_token = [redacted] #Access Token generated on January 23th 2022.
client = asana.Client.access_token(personal_access_token)
client.LOG_ASANA_CHANGE_WARNINGS = False # Disable Asana API Warnings
headers = {
    'Accept': 'application/json',
    'Authorization': [redacted]
}
##########################
```

Eu tinha:

um problema

um software de gestão de demandas

uma API

O Problema:

Agregar trabalho lançado no software de gestão de demandas (Gestão de Projetos – Programas – Portfólio)

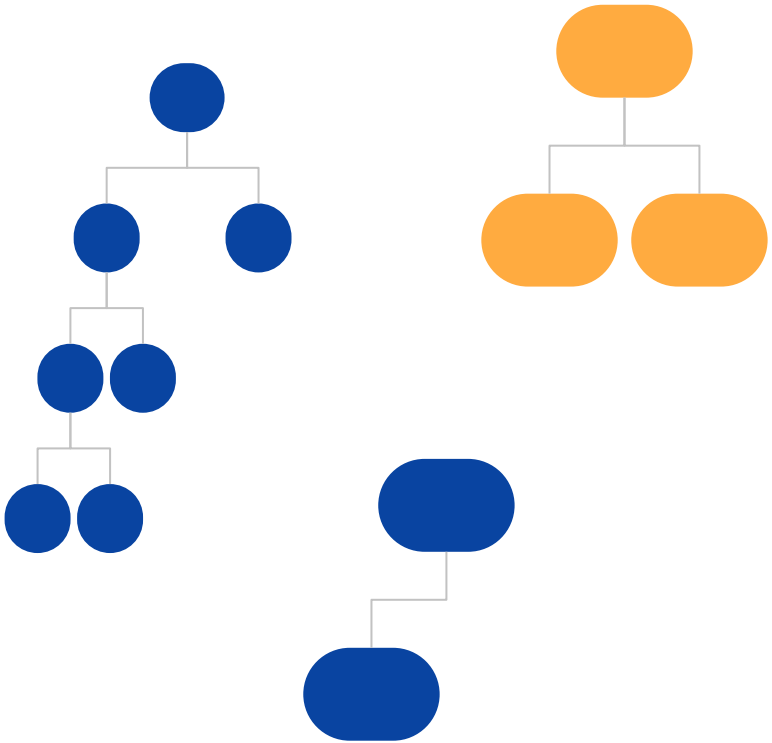
Oportunidade para automatizar processos relacionados

SOFTWARE DE GESTÃO DE DEMANDAS:

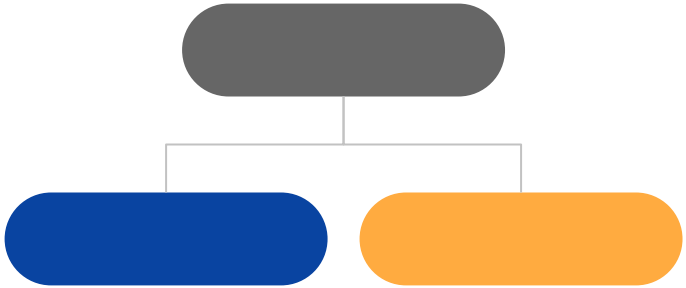
Alguns DENTRE OUTROS conhecidos no mercado

AGREGAR TRABALHO LANÇADO NO SOFTWARE DE GESTÃO DE DEMANDAS:

Esse “caos”



Na verdade é:



Isso me “custava” cerca de:

⅕ do tempo:

- a cada semana, pelo menos, 1 dia para focar em agregar o trabalho

Depois de 4 meses de muito trabalho…


WHAT
??

A solução que me permitiu tirar férias:

Pipeline de dados (ETL - Extract, Load, Transform)

[‘AWS Lambda’, ‘Scripts Python + lib BOTO3’, ‘AWS EC2’, ‘Python 3.7’, ‘Asana API’]

Extrai dados > Agrega > Gera informação/valor

A solução que me permitiu tirar férias:

1. **Função Lambda inicia automaticamente 1x por semana (que chama um script Python para ligar uma instância de EC2)**

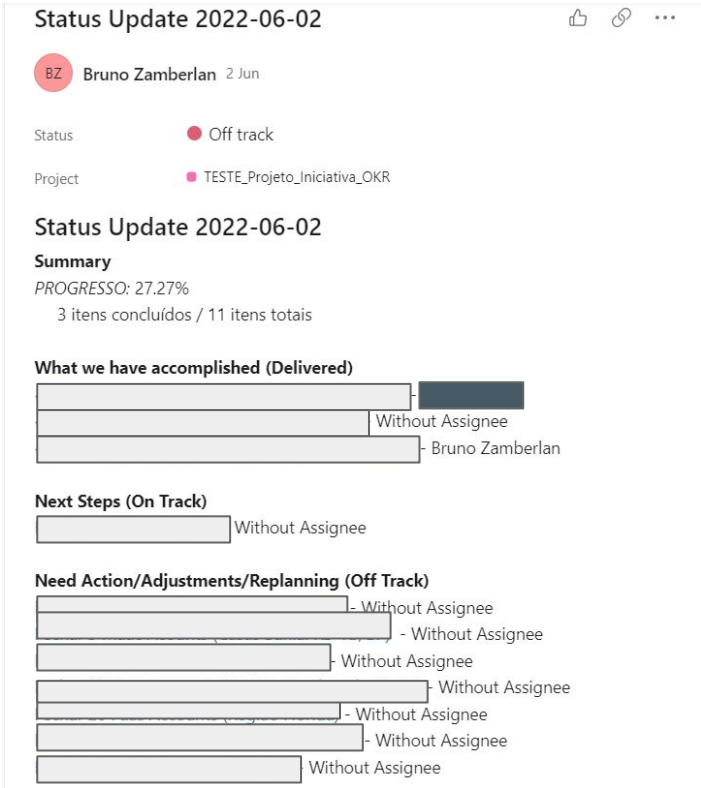
2. **Solução (Python) está na crontab da máquina, o que faz com que ela seja executada assim que a máquina ligar**

3. **Solução executa (em cerca de 1 hora) fazendo todo o trabalho de agregação e gera informação/valor:**

    relatórios automatizados com a visão agregada

4. **Após 1 hora, outra função Lambda inicia um script Python para interromper a instância EC2 economizando custos**

Informação/valor:



Status Update 2022-06-02

BZ Bruno Zamberlan 2 Jun

Status Off track

Project TESTE_Projeto_Iniciativa_OKR

Status Update 2022-06-02

Summary

PROGRESSO: 27.27%

3 itens concluídos / 11 itens totais

What we have accomplished (Delivered)

- Without Assignee
- Bruno Zamberlan

Next Steps (On Track)

- Without Assignee

Need Action/Adjustments/Replanning (Off Track)

- Without Assignee
- Without Assignee
- Without Assignee
- Without Assignee
- Without Assignee
- Without Assignee
- Without Assignee

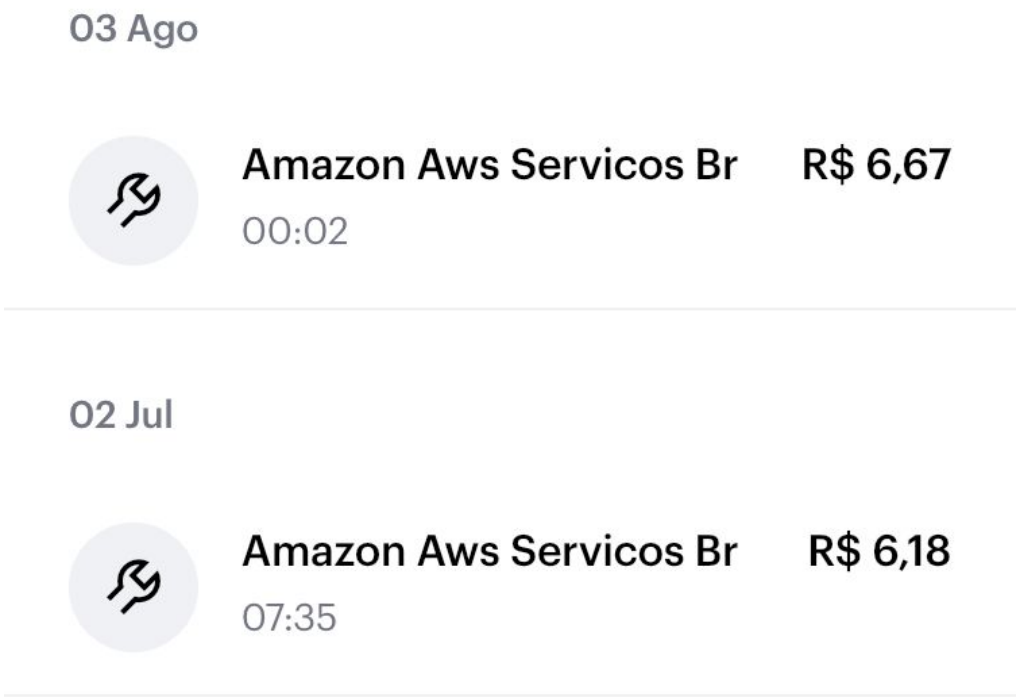
A solução que me permitiu tirar férias:

- ⅕ DO TEMPO

+ TRANQUILIDADE

A solução que me permitiu tirar férias:

A solução que me permitiu tirar férias:



R$ 6,42 / MÊS

Boas práticas/estruturas que utilizei na solução:

- **Stateless application**
- **Data Classes**
- **Composition over inheritance**
- **Choose the right data structure**
- **Asyncio**

STATELESS APPLICATION:

- Nenhuma referência ou informação sobre transações antigas são armazenadas

- Todo histórico dos projetos está na API:
  - Posso ‘refazer’ determinado momento do projeto apenas utilizando a consulta correta na API

STATELESS APPLICATION:

```python
@staticmethod
def snapshot_of_a_given_time(tag_gid: str, reference_date:str) -> dict:

    iterator_of_tasks_gid = (task['gid'] for task in client.tasks.get_tasks_for_tag(tag_gid))
    datetime_object_of_reference_date = datetime.strptime(reference_date, "%Y-%m-%d")
    datetime_object_for_without_date = datetime.strptime('2019-01-01', "%Y-%m-%d")

    def extract_task_date_added_to_project(task_id):

        task_history = list(client.stories.get_stories_for_task(task_id))
        task_added_to_project = task_history[0]['created_at'][:-14]
        return task_added_to_project

    def extract_task_marked_complete_to_project(task_id):

        task_marked_complete_to_project = []
        task_history = list(client.stories.get_stories_for_task(task_id))

        # To deal when the task has not already completed
        if 'marked_complete' not in (task['resource_subtype'] for task in task_history):
            return '2019-01-01'
        else:
            for element in task_history:
                if element['resource_subtype'] == 'marked_complete':
                    task_marked_complete_to_project.append(element['created_at'][:-14])

        # To deal with when the tasks has more than 1 completed date
        # (in the case of, the user first mark completed and, then, marked incompleted)

        if len(task_marked_complete_to_project) > 0:
            task_marked_complete_to_project = task_marked_complete_to_project[-1]

        return task_marked_complete_to_project
```

```python
    def identify_opened_tasks(iterator_with_task_list):
        dict_open_tasks_with_date_added_to_project_at_a_given_moment = {}

        for task in iterator_with_task_list:
            date_added_to_project = datetime.strptime(extract_task_date_added_to_project(task), "%Y-%m-%d")

            if date_added_to_project <= datetime_object_of_reference_date:
                dict_open_tasks_with_date_added_to_project_at_a_given_moment[task] = date_added_to_project

        return dict_open_tasks_with_date_added_to_project_at_a_given_moment


    def add_marked_complete_date(dict_with_open_tasks):
        dict_open_tasks_with_date_added_and_marked_complete_at_a_given_moment = {}

        task_list = list(dict_with_open_tasks.keys())
        opened_date_list = list(dict_with_open_tasks.values())
        marked_complete_date_list = [datetime.strptime(extract_task_marked_complete_to_project(taskid), "%Y-%m-%d") for taskid in list(dict_with_open_tasks.keys())]

        for task, opened_date, marked_complete_date in zip(task_list, opened_date_list, marked_complete_date_list):
            dict_open_tasks_with_date_added_and_marked_complete_at_a_given_moment[task] = [opened_date, marked_complete_date]

        return dict_open_tasks_with_date_added_and_marked_complete_at_a_given_moment

    def classify_tasks_by_stats(dict_tasks_opened_marked_complete):
        dict_tasks_with_status_open_date_and_marked_complete = {}
        opened_list = []
        marked_complete_list = []
        dict_tasks_classification = {}


        for task, value in dict_tasks_opened_marked_complete.items():
            if (value[1] <= datetime_object_of_reference_date ) and (value[1] != datetime_object_for_without_date):
                dict_tasks_with_status_open_date_and_marked_complete[task] = ['complete', value[0], value[1]]
                marked_complete_list.append(task)
            else:
                dict_tasks_with_status_open_date_and_marked_complete[task] = ['opened', value[0], value[1]]
                opened_list.append(task)
                    #

        dict_tasks_classification['opened'] = [len(opened_list), opened_list]
        dict_tasks_classification['marked_complete'] = [len(marked_complete_list), marked_complete_list]
        dict_tasks_classification['total'] = [len(opened_list) + len(marked_complete_list)]


        return dict_tasks_classification


    return classify_tasks_by_stats(add_marked_complete_date(identify_opened_tasks(iterator_of_tasks_gid)))
```

DATA CLASSES:

- Usadas para armazenar estrutura de dados
- Não requerem método construtor
- Mais eficientes
- Ajudam a melhorar a compreensão do código

DATA CLASSES:

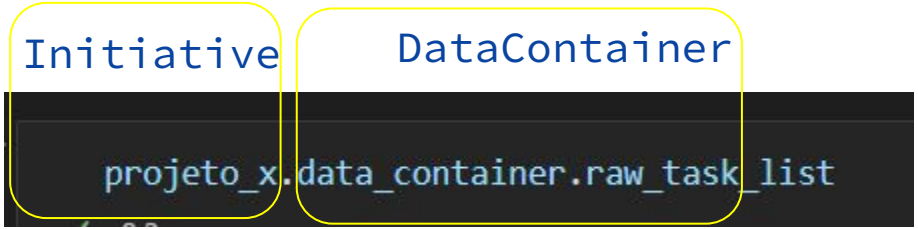
```python
@dataclass
class Task:
    '''Object for storing all the tasks infos for a given task.'''

    # Attributes based on Asana task structure
    gid: str #
    name: str
    created_at: str
    assignee_name: str
    modified_at: str
    completed_at: str
    due_on: str
    permalink_url: str

    # My own attributes created in order to aggregate info and do the calculations
    Project: str = "Not classified Yet"
    Program_Status: str = "Not classified Yet"
    Achieved_Percentage: str = "Not generated Yet"
    Type: str = 'Not defined yet'
    KR_associated_with:str = "Not associated Yet"
    KR_contribution:float = '0.0'
    KR_result: int = 0
    Milestone: str = 'Not associated Yet'
    Task_lead_time: int = 0
```

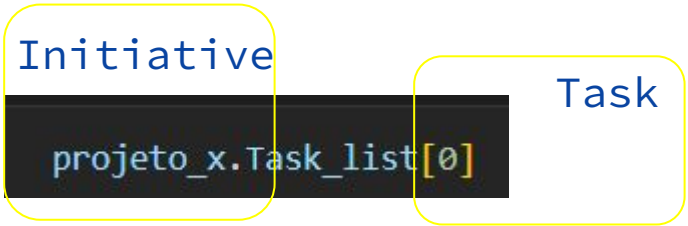
```python
def __post_init__(self):
    """
    First, we need to use DataContainer raw_list attribute to create create a list with all Tasks objects

    transform_raw_task_list_into_Task_list(). It has all the business rules needed to adjust the raw_tasks
    for a Pandas Dataframe

    Finally, we need to set the status of this Initiative with load_status_initiative_kr_okr() method,

    To update the Asana Progress Report with load_PROJECT_progress_report_Asana method
    """


    def transform_raw_task_list_into_Task_list(raw_task_list: list) -> list[Task]:

        Tasks_list = []

        #Iterave over the task list stored into DataContainer object
        for task in raw_task_list:
```

COMPOSITION OVER INHERITANCE:

- **Evita criar uma classe SUPER para herdar os atributos**
- **Deixa a lógica mais simples (Clean Code)**
- **Estrutura criada:**
  - ***Task < Data Container < Milestones < Initiative***

COMPOSITION OVER INHERITANCE:





CHOOSE THE RIGHT DATA STRUCTURE:

- Lazy Evaluation x Eager Evaluation
- Iterators -> Lazy Evaluation ()
- Listas -> Eager Evaluation []

CHOOSE THE RIGHT DATA STRUCTURE:

Lazy Evaluation

```
iterator_of_task_lists_by_given_tag = (task['gid'] for task in client.tasks.get_tasks_for_tag(self.gid))
        # Using iterators instead of using list can give us 4% of efficiency.
```

Eager Evaluation

```
def check_if_test_task_is_completed(task):
            if task == 0.0:
                return "not a valid task"
            else:
                result = str()
                if list(client.stories.find_by_task(task))[-1]['resource_subtype'] == 'marked_complete':
                    result = 'completed'
                else:
                    result = 'not_completed'
                return result
```

ASYNCIO:

- Enviar requisições assíncronas
- Difícil de entender, documentação confusa
- Vale a pena para quem precisa enviar muitas requisições http

ASYNCIO:

```
async def extract_task_information(task_gid):

    list_of_task_attributes = ()

    def load_custom_field_display_value(list_of_custom_fields, custom_field_name):
        dict_name_display_value = {element['name']: element['display_value'] for element in list_of_custom_fields}

        try:
            return dict_name_display_value[custom_field_name]
        except:
            return f"{custom_field_name} not found"

    async with aiohttp.ClientSession(headers=headers) as session:
        async with session.get(f'https://app.asana.com/api/1.0/tasks/{task_gid}') as resp:
            try:
```

ASYNCIO:

```python
async def extract_task_information_from_task_list(task_list: list) -> list:
    tasks = [asyncio.create_task(Task.extract_task_information(id)) for id in task_list]
    objects = await asyncio.gather(*tasks)
    return objects
```

```python
[await Data_Container.extract_task_information_from_task_list([id['gid']
```


Let's keep a little optimism here.

ASYNCIO:

Síncrono

110 MIN

- (1HR E 50 MIN)

Assíncrono

7 MIN

https://www.linkedin.com/in/bruno-zamberlan-62418145/