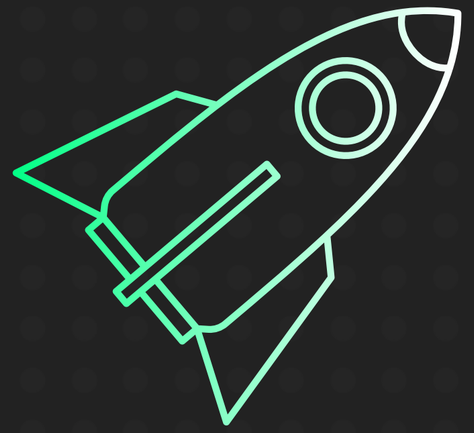
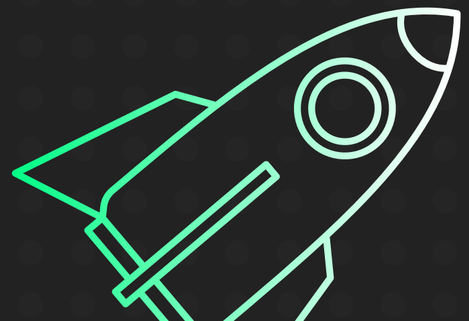
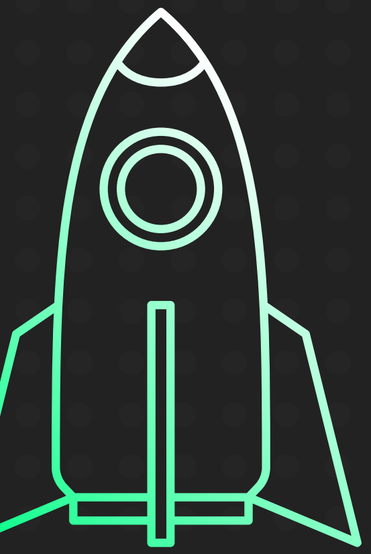
SPRINT 4
COMPUTAÇÃO NEUROMÓRFICA





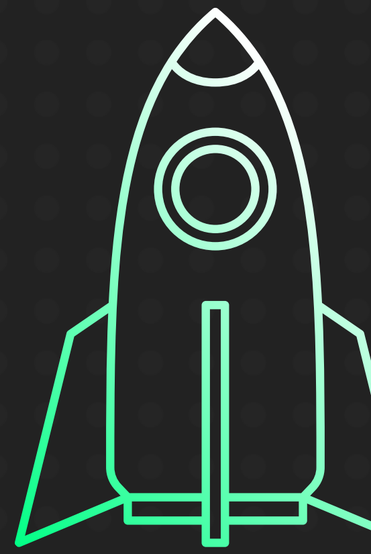
HENRICO BELA
EMILLY GABRIELLY
SARA LEAL
FELYPE NUNES
DANIEL FARIA



Sobre a performance da API

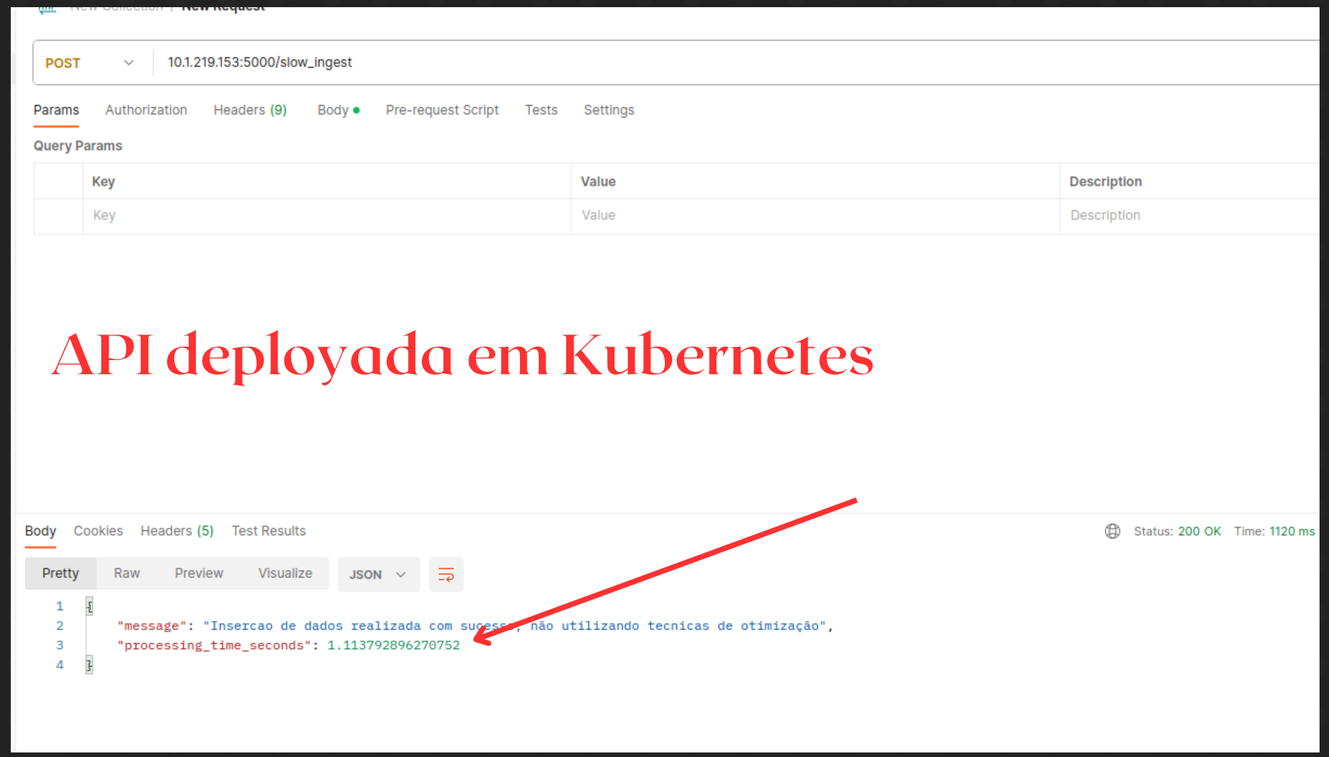
Analisando a performance da API entregue no sprint 3, foi identificado que a mesma necessitava de um upgrade em seu endpoint de “Ingest” onde foi colocada a função de gravar os dados de um endpoint publico (ThingsPeaks).

Foram executadas algumas mudanças neste endpoint de Ingest, e obtivemos os seguintes Benchmarks de Resposta temporal para a nova API.

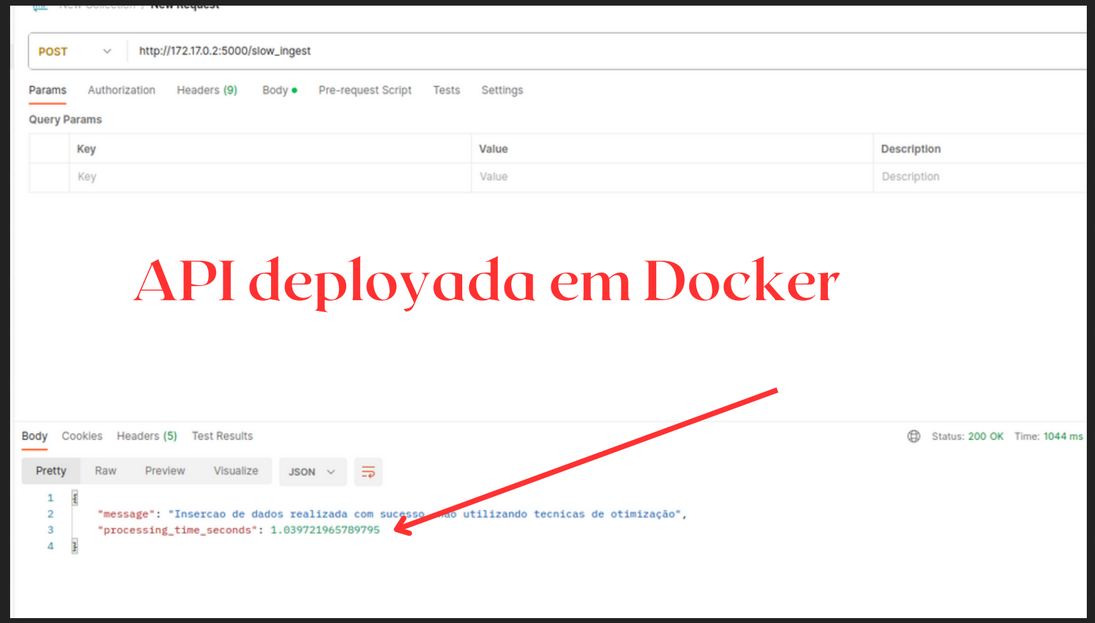


Slow Ingest

Conforme á imagem a baixo, chamamos o endpoint antigo de slow_ingest, onde o codigo necessitava de otimização. Foi calculado o tempo de requisição, e obteve-se o seguinte resultado:



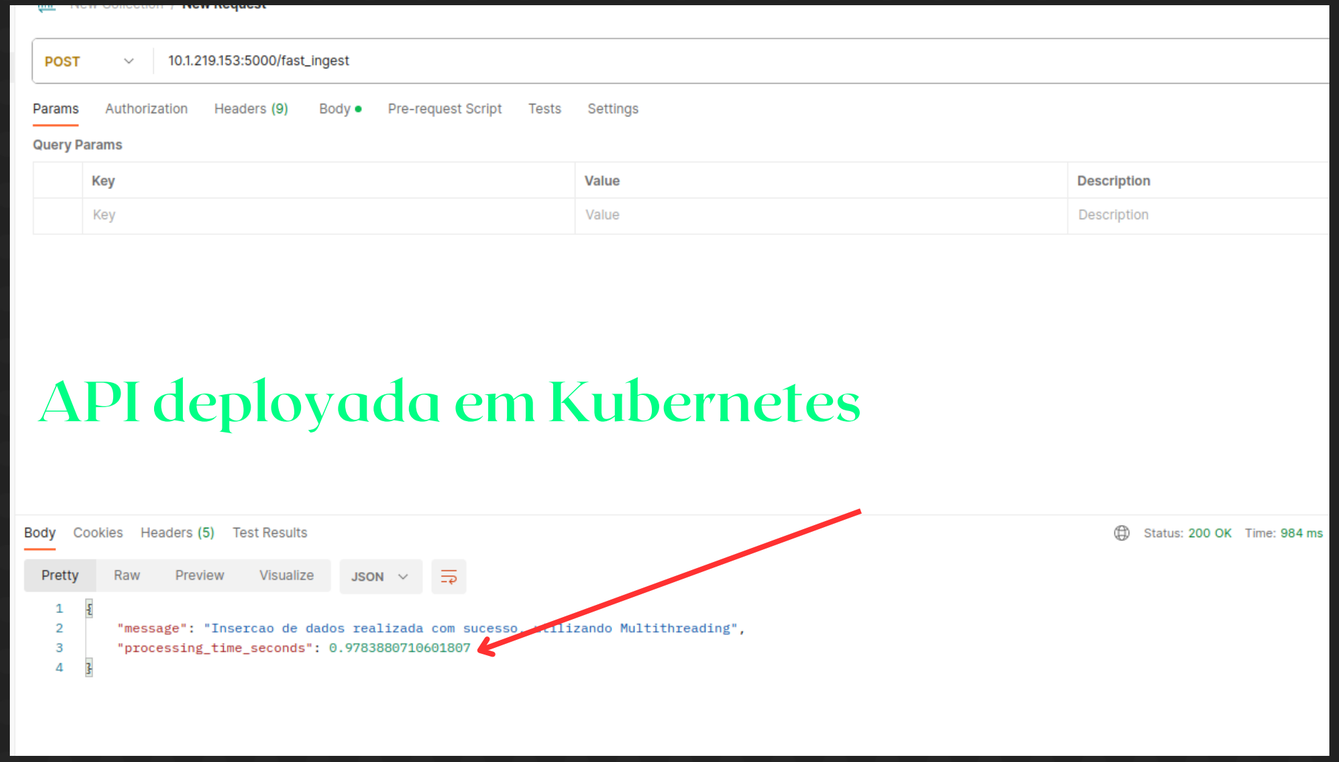
API deployada em Kubernetes



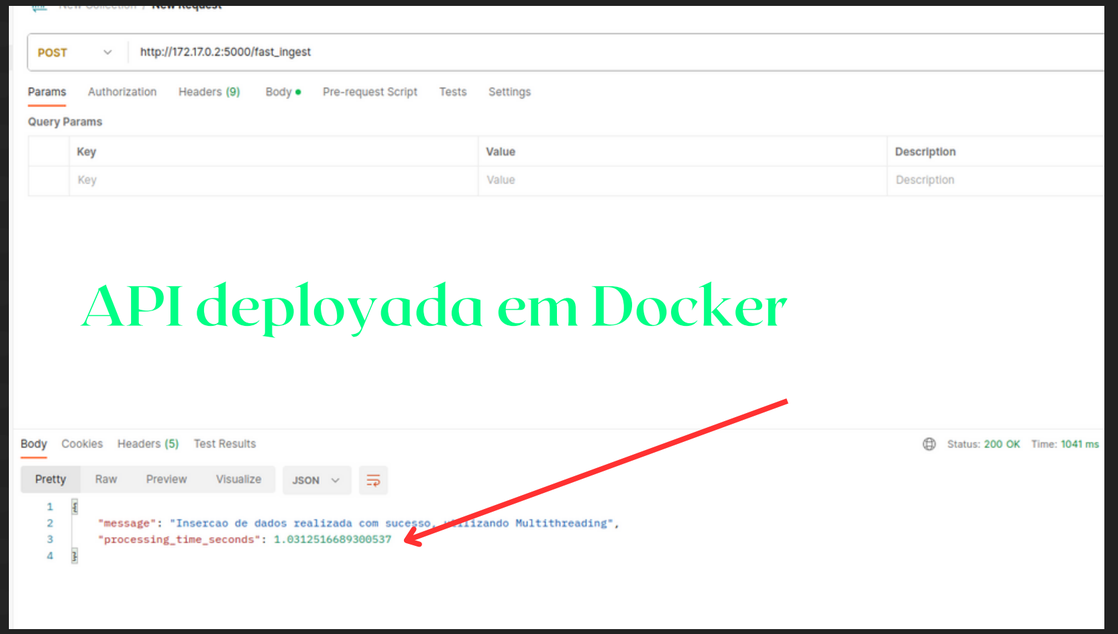
API deployada em Docker

Fast Ingest

Conforme á imagem a baixo, agora com as otimizações de multithreading, obtivemos resultados satisfatórios nas requisições!



API deployada em Kubernetes



API deployada em Docker

RESULTADOS

Com o calculo de tempo e benchmark entre as funções otimizadas e não otimizadas, ambos nos ambientes de Kubernetes, pode-se ver uma melhora de mais de 10% no tempo da requisição.

Ganho de performance

12% - tempo

Apesar de parecer pouco ganho de performance, estamos falando de uma base de dados pequena, porém, com uma base de dados maior, esta porcentagem tende a aumentar mais ainda, ou seja, o tempo ainda menor utilizando as otimizações.

Funções Alteradas e Commits

Conforme dito anteriormente, 2 funções foram alteradas para que esta otimização funcionasse.

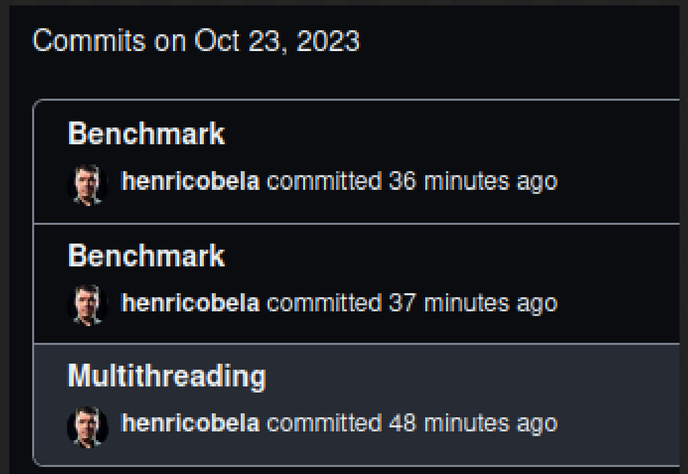
```python
# Função para inserir dados no banco
def inserir_dados():
    conn = sqlite3.connect('api/core/db/data.db')
    cursor = conn.cursor()

    get_df = TksRequest()
    df = get_df.get_data_response()
    colunas = ', '.join(df.columns)
    tabela = "configurations"
    placeholders = ', '.join(['?'] * len(df.columns))

    sql = f'CREATE TABLE IF NOT EXISTS {tabela} (id INTEGER PRIMARY KEY AUTOINCREMENT, {colunas})'
    cursor.execute(sql)
    conn.commit()

    sql = f'INSERT INTO {tabela} ({colunas}) VALUES ({placeholders})'
    values = [tuple(x) for x in df.values.tolist()]
    cursor.execute(f"DELETE FROM {tabela}")
    cursor.executemany(sql, values)
    conn.commit()

    cursor.close()
    conn.close()
```

```python
# Aplicando conceito de multithreading
@bp.route('/fast_ingest', methods=['POST'])
def fast_ingest():
    start_time = time.time()

    num_threads = 4
    threads = []

    for _ in range(num_threads):
        thread = threading.Thread(target=inserir_dados)
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

    end_time = time.time()
    processing_time = end_time - start_time

    return jsonify({'message': 'Insercao de dados realizada com sucesso, utilizando Multithreading',
                    'processing_time_seconds': processing_time})
```

Repositório GITHUB:

https://github.com/henricobela/api-ingest

Commits



MUITO OBRIGADO!