06.2024
GruPy Blumenau

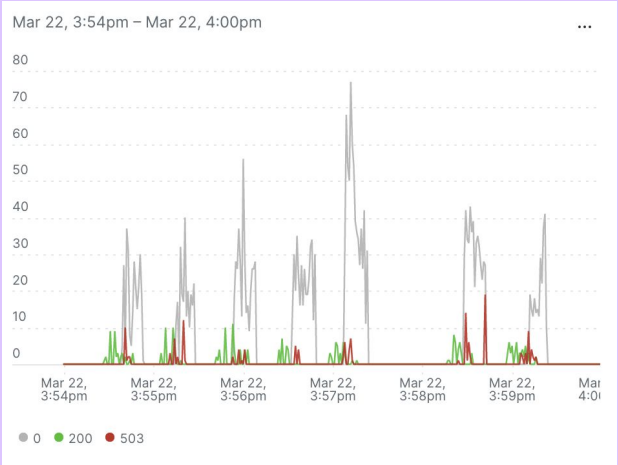
Como deixar a sua aplicação até 20x mais lenta com FastAPI

Júlio César Batista

Como começou

Mar 22, 3:54pm – Mar 22, 4:00pm ...

Response Code	Count
0	2.67 k
200	241
503	149

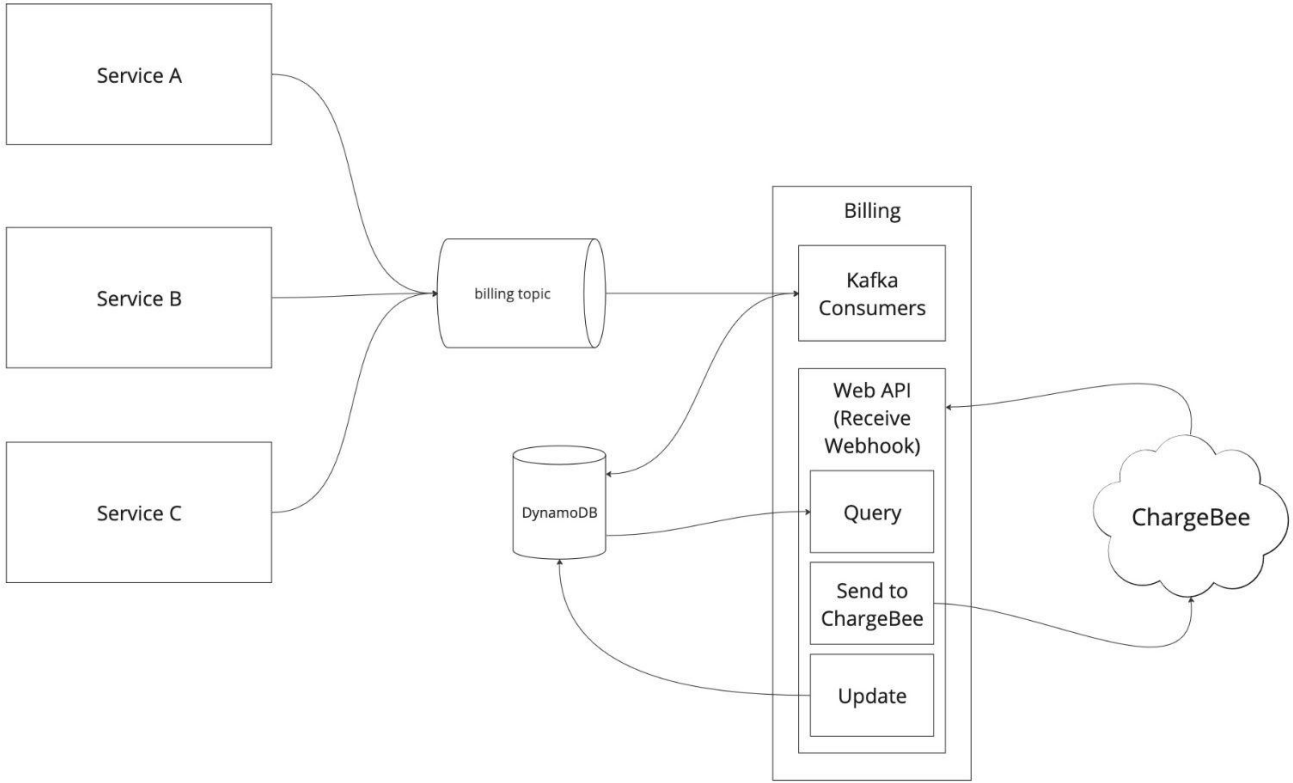


Como está indo

Apr 22, 12:56pm – Apr 22, 1:01pm ...

Response Code	Count
200	48 k

Visão geral


Service A
Service B
Service C
billing topic
Billing
Kafka Consumers
Web API (Receive Webhook)
Query
Send to ChargeBee
Update
DynamoDB
ChargeBee

Configuração dos testes

Sem dependência do Chargebee

Conseguir o mesmo nível de concorrência, restrições de rede (timeouts) e execução de código (acessar DyanmoDB, enviar para o Chargebee, atualizar o Dynamo)

Cuidado com os recursos do sistema

Pode ter que aumentar os *file handlers* `ulimit -n`

Script

Usando aiohttp e multiprocessing

Resultado inicial para 1k requisições

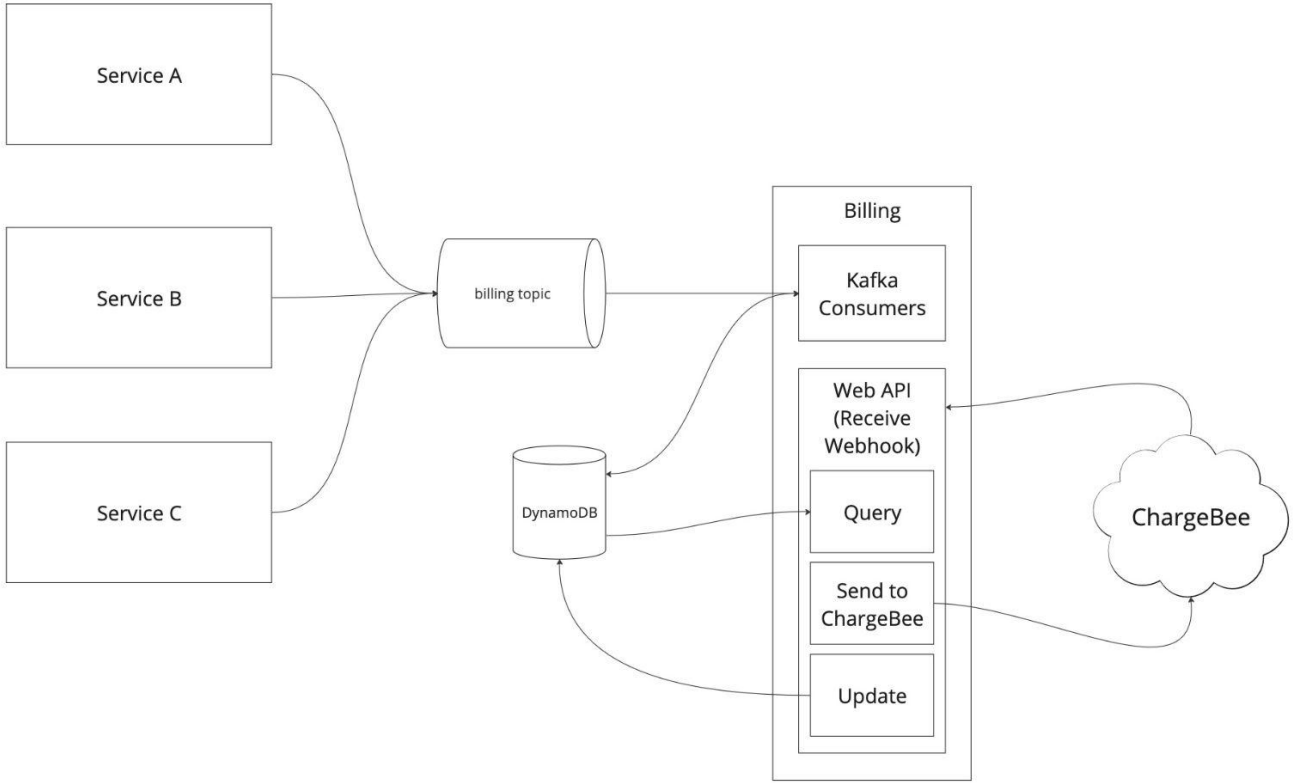
30%

Sucesso
- HTTP 200

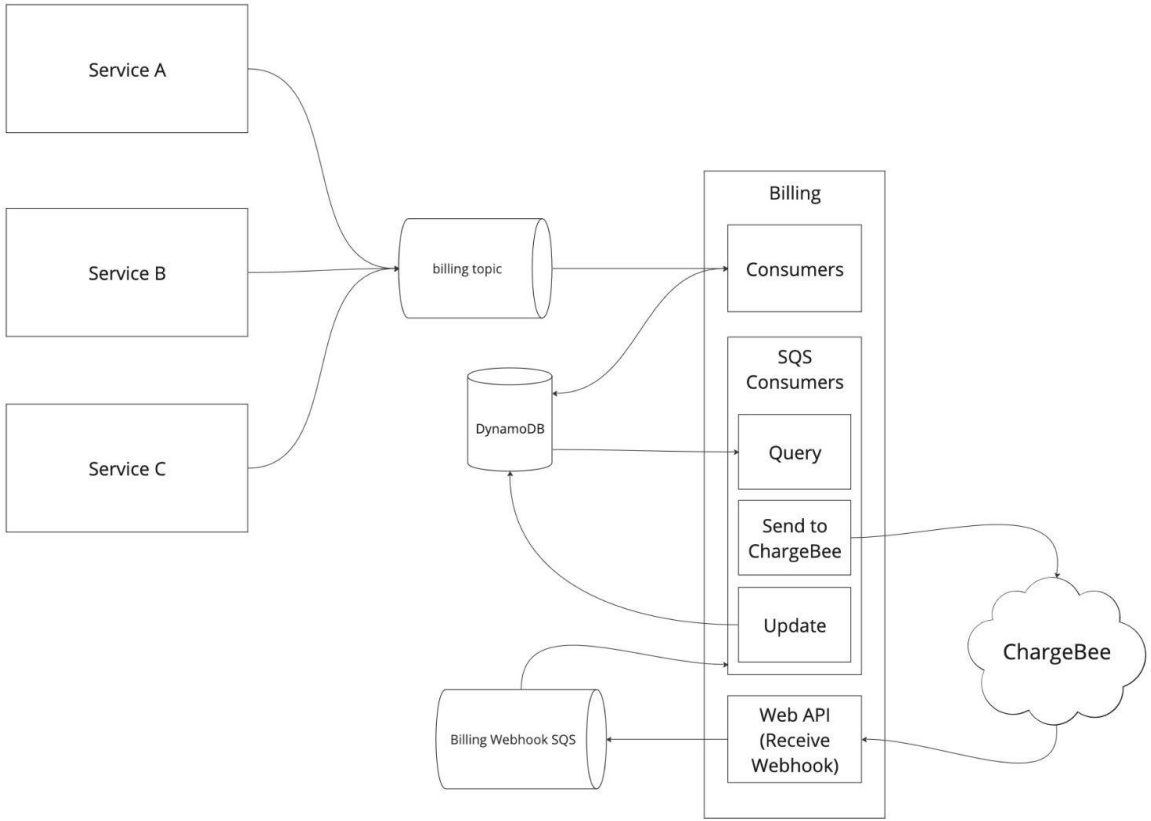
70%

Erro
- Sem resposta / timeout
- HTTP 503

O que podemos melhorar?



Adicione uma fila



1k requisições com SQS

45%

Sucesso
- HTTP 200

55%

Erro
- Sem resposta / timeout
- HTTP 503

50%

de melhora na taxa de sucesso

Outros benefícios de usar SQS

Controle sobre a capacidade de processamento

Controle sobre o poder de processamento

Se queremos aumentar/diminuir o nosso poder de processamento, só precisamos adicionar/remover consumidores.

Processar mais requisições

Como o processamento de requisições envolve apenas adicionar em uma fila, se precisarmos de mais poder de processamento podemos adicionar mais pods sem restrições.

Sem problemas com HTTP 429 (rate limit)

O processamento de requisições não está mais limitado pela quantidade de requisições que podemos enviar para o Chargebee ou pelo tempo de resposta deles.

E agora?

Uma vez que melhoramos a arquitetura do serviço, o que podemos fazer?

- Usar `uvloop`
- Usar `httptools`
- Usar `async` handlers
- Aumentar thread pool
- Evitar validação duplicada de JSON
- Usar `ORJSONResponse`
- Desabilitar logs do `uvicorn` com `--no-access-log`
- Usar `ASGIMiddleware` ao invés de `BaseHTTPMiddleware`

1k requisições com `uvloop` + `httptools`

- Nosso projeto base já usa `uvicorn[standard]` que inclui `uvloop` e `httptools` se possível
  - https://www.uvicorn.org/#quickstart
- Esse foi um teste de verificação com `uvicorn` sem os extras

38%

Sucesso
- HTTP 200

62%

Erro
- Sem resposta / timeout
- HTTP 503

Antes de continuar

Um pouco sobre `uvicorn`

Uvicorn + FastAPI

`uvicorn` é o servidor que usamos para rodar `FastAPI`.
Verifique as configurações aqui

Limite a concorrêncua

`--limit-concurrency <int>`
Quantas requisições concorrentes o seu servidor consegue processar? Por padrão, até esgotar os recursos do sistema.

Fila de requisições

`--backlog <int>`
Quantas requisições são mantidas na *accept queue* antes de retornar 503 para novas requisições?
Por padrão, 2048.

Configuração dos tests [atualizada]

Apenas adiciona uma mensagem no SQS

Não é mais necessário simular todo o processo de acessar o DynamoDB e Chargebee, apenas adicionamos uma mensagem em uma fila.

Sem necessidade de dados reais

Como apenas adicionamos uma mensagem no SQS, não importa mais se os IDs existem e se existem dados para processar.

O objetivo é o servidor web

Nós queremos melhorar o desempenho do servidor web, não necessariamente a capacidade de retornar dados para o Chargebee.

Usar uma ferramenta padrão

Existem muitas opções.
Usamos vegeta.

vegeta attack -rate 500 -duration 5s

59% 932ms 42rps

Sucesso | Resposta mais rápida | Throughput

```
Requests      [total, rate, throughput]         2500, 500.21, 41.92
Duration      [total, attack, wait]             34.998s, 4.998s, 30s
Latencies     [min, mean, 50, 90, 95, 99, max]  932.236ms, 21.384s, 25.799s, 30s, 30s, 30.001s, 30.006s
Bytes In      [total, mean]                     64548, 25.82
Bytes Out     [total, mean]                     8347230, 3338.89
Success       [ratio]                           58.68%
Status Codes  [code:count]                      0:1033  200:1467
```

Uma espiada no código

O que tem de errado aqui?

```python
@router.post(PENDING_INVOICE_CREATED_PATH)
async def pending_invoice_created_webhook_handler(
    request: Request,
    producer: Producer = Depends(Producer),
) -> JSONResponse:
    body = await request.body()
    body_str = body.decode("utf-8")
    try:
        PendingInvoiceCreated.model_validate_json(body_str)
        producer.send_message(body_str)
    except ValidationError as e:
        pass
    except Exception as e:
        pass
    return JSONResponse(status_code=200, content={...})
```

Como FastAPI lida com concorrência?

Threading

A forma mais comum para lidar com concorrência é usar threads.

A GIL

Python a famosa GIL que limita a capacidade de processamento concorrente.

CPU ou IO

O problema da GIL é “apenas” para tarefas que exigem CPU, para tarefas com IO, é ok usar threads.

async não é thread

Funções async não são threads. Existe apenas a thread principal e um gerenciador de eventos (*event loop*)coordenando tarefas cooperativas, não preemptivas (OS threads).

Como FastAPI lida com concorrência?

async handlers

Rodam na thread principal e não devem ser bloqueantes, caso contrário a thread para por completo e o servidor para de processar requisições.

sync handlers

Rodam em um thread pool. Podem bloquear, mas se o pool estiver cheio a thread principal será bloqueada até que um espaço seja liberado.

Injeção de dependência

As mesmas regras se aplicam para a injeção de dependências (`Depends(some_function)`).

O que há de errado aqui?

Uma *blocking call* em um *async handler*

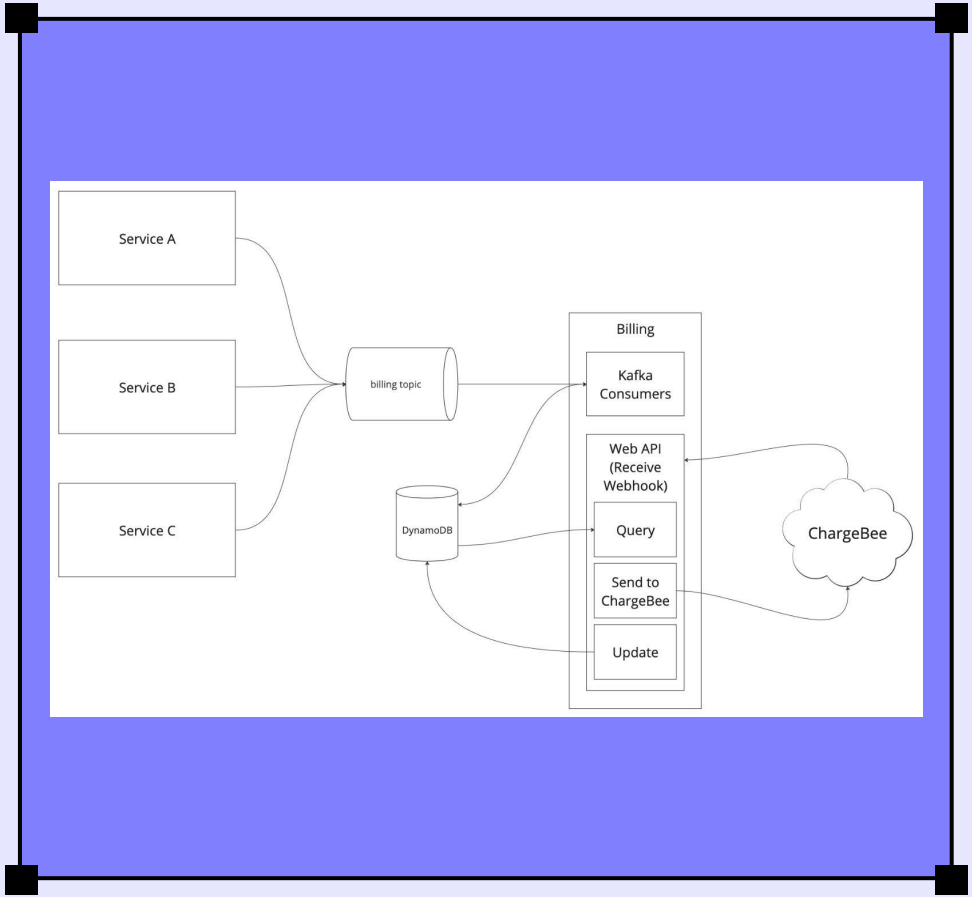
- A chamada do SQS é bloqueante (network IO)
- Como é um `async` handler, está rodando na thread principal
- Durante a chamada bloqueante, o handler não devolve o controle para o *event loop*
- A thread principal fica incapacitada de processar/aceitar novas requisições até que o SQS retorne

```python
@router.post(PENDING_INVOICE_CREATED_PATH)
async def pending_invoice_created_webhook_handler(
    request: Request,
    producer: Producer = Depends(Producer),
) -> JSONResponse:
    body = await request.body()
    body_str = body.decode("utf-8")
    try:
        PendingInvoiceCreated.model_validate_json(body_str)
        producer.send_message(body_str)
    except ValidationError as e:
        pass
    except Exception as e:
        pass
    return JSONResponse(status_code=200, content={...})
```

De volta ao início

Multiplos bloqueios no async handler

- Não era apenas sobre mover para o SQS, mas remover multiplas *blocking calls* em um *async handler*



Execute no threadpool

Mover as blocking calls para o threadpool

- FastAPI possui `run_in_threadpool` que executa funções no mesmo threadpool dos *sync handlers*

```python
from fastapi.concurrency import run_in_threadpool

@router.post(PENDING_INVOICE_CREATED_PATH)
async def pending_invoice_created_webhook_handler(
    request: Request,
    producer: Producer = Depends(Producer),
) -> JSONResponse:
    body = await request.body()
    body_str = body.decode("utf-8")
    try:
        PendingInvoiceCreated.model_validate_json(body_str)
        await run_in_threadpool(producer.send_message,
body_str)
    except ValidationError as e:
        pass
    except Exception as e:
        pass
    return JSONResponse(status_code=200, content={...})
```

vegeta attack -rate 500 -duration 5s

78% 894ms 55rps

Sucesso | Resposta mais rápida | Throughput

```
Requests      [total, rate, throughput]         2500, 500.21, 55.92
Duration      [total, attack, wait]             34.996s, 4.998s, 29.998s
Latencies     [min, mean, 50, 90, 95, 99, max]  894.502ms, 18.744s, 20.421s, 30s, 30s, 30.001s, 30.004s
Bytes In      [total, mean]                     86108, 34.44
Bytes Out     [total, mean]                     11135330, 4454.13
Success       [ratio]                           78.28%
Status Codes  [code:count]                      0:543  200:1957
```

Ajuste o threadpool

Aumente a quantidade de workers no thread pool

- FastAPI usa AnyIO como uma abstração para `asyncio` e `trio`
- Por padrão, usa 40 workers

```python
from anyio import to_thread

DEFAULT_FASTAPI_THREAD_POOL_WORKERS = 100

def create_app() -> FastAPI:
    ...

async def lifespan(app: FastAPI) -> AsyncGenerator:
    workers = settings.get_int(
        "FASTAPI_THREAD_POOL_WORKERS",
        DEFAULT_FASTAPI_THREAD_POOL_WORKERS
    )
    to_thread.current_default_thread_limiter().total_tokens = workers

    ...
```

vegeta attack -rate 500 -duration 5s

85% 886ms 61rps

Sucesso | Resposta mais rápida | Throughput

```
Requests      [total, rate, throughput]         2500, 500.20, 61.12
Duration      [total, attack, wait]             34.979s, 4.998s, 29.981s
Latencies     [min, mean, 50, 90, 95, 99, max]  886.882ms, 17.511s, 18.271s, 30s, 30s, 30.001s, 30.003s
Bytes In      [total, mean]                     94072, 37.63
Bytes Out     [total, mean]                     12165220, 4866.09
Success       [ratio]                           85.52%
Status Codes  [code:count]                      0:362  200:2138
```

Ainda tem algo errado, 800ms para postar uma mensagem no SQS parece demais

- Profile com [pyinstrument]
- O tempo foi quase o mesmo para sync e async (~50ms)

```
SYNC

0.058 Runner.run  asyncio/runners.py:86
└─ 0.058 coro  starlette/middleware/base.py:65
       [20 frames hidden]  starlette, fastapi, anyio
       0.043 run_sync_in_worker_thread  anyio/_backends/_asyncio.py:834
       └─ 0.043 [await]  anyio/_backends/_asyncio.py
       0.013 run_endpoint_function  fastapi/routing.py:182
       └─ 0.013 pending_invoice_created_webhook_handler
billable_event_consumer/routers/webhooks/chargebee/pending_invoice_created
.py:19
              └─ 0.013 Producer.send_message
billable_event_consumer/sqs/chargebee_pending_invoice_created_webhook.py:4
5
```

```
ASYNC

0.050 Runner.run  asyncio/runners.py:86
├─ 0.049 coro  starlette/middleware/base.py:65
│       [24 frames hidden]  starlette, fastapi, anyio, shippo_cor...
│       0.037 run_sync_in_worker_thread  anyio/_backends/_asyncio.py:834
│       └─ 0.037 [await]  anyio/_backends/_asyncio.py
│       0.010 run_endpoint_function  fastapi/routing.py:182
│       └─ 0.010 pending_invoice_created_webhook_handler_async
billable_event_consumer/routers/webhooks/chargebee/pending_invoice_created
.py:74
│              └─ 0.010 run_in_threadpool  starlette/concurrency.py:35
│              [4 frames hidden]  starlette, anyio
└─ 0.001 [self]  asyncio/runners.py
```

E quando precisa processar mais requisições?

- Os handlers precisam de < 150ms
- Existe algo antes dos handlers que leva > 150ms
- Por que tem algo rodando no threadpool para um async handler?

```
0.464 Runner.run  asyncio/runners.py:86
├─ 0.427 coro  starlette/middleware/base.py:65
│       [45 frames hidden]  starlette, fastapi, anyio, asyncio
│       0.158 run_sync_in_worker_thread  anyio/_backends/_asyncio.py:834
│       ├─ 0.128 [await]  anyio/_backends/_asyncio.py
│       0.105 run endpoint function  fastapi/routing.py:182
│       └─ 0.105 pending invoice created webhook handler
billable_event_consumer/routers/webhooks/chargebee/pending_invoice_created
.py:19
│               [6 frames hidden]  loguru, shippo_core, <built-in>
│       0.055 ProfilingMiddleware.__call__  starlette/middleware/base.py:24
│       └─ 0.053 ProfilingMiddleware.dispatch
└─ 0.006 RequestResponseCycle.run_asgi
uvicorn/protocols/http/httptools_impl.py:417
        [15 frames hidden]  uvicorn, fastapi, starlette, opentele...
```

```
0.531 Runner.run  asyncio/runners.py:86
├─ 0.480 coro  starlette/middleware/base.py:65
│       [45 frames hidden]  starlette, fastapi, anyio, asyncio
│       0.180 run_sync_in_worker_thread  anyio/_backends/_asyncio.py:834
│       ├─ 0.158 [await]  anyio/_backends/_asyncio.py
│       0.124 run endpoint function  fastapi/routing.py:182
│       └─ 0.124 pending invoice created webhook handler
billable_event_consumer/routers/webhooks/chargebee/pending_invoice_created
.py:19
│               ├─ 0.082 run_in_threadpool  starlette/concurrency.py:35
│               [6 frames hidden]  loguru, shippo_core, <built-in>
│       0.046 ProfilingMiddleware.__call__  starlette/middleware/base.py:24
│       └─ 0.045 ProfilingMiddleware.dispatch
billable_event_consumer/main.py:227
│               └─ 0.043 call_next  starlette/middleware/base.py:31
│               [9 frames hidden]  starlette, anyio, asyncio
```

De volta ao código

Tem algo errado?

```python
from fastapi.concurrency import run_in_threadpool


@router.post(PENDING_INVOICE_CREATED_PATH)
async def pending_invoice_created_webhook_handler(
    request: Request,
    producer: Producer = Depends(Producer),
) -> JSONResponse:
    body = await request.body()
    body_str = body.decode("utf-8")
    try:
        PendingInvoiceCreated.model_validate_json(body_str)
        await run_in_threadpool(producer.send_message,
body_str)
    except ValidationError as e:
        pass
    except Exception as e:
        pass
    return JSONResponse(status_code=200, content={...})
```

Resolver o `Producer` é sync

- Suposição: com alta demanda, o threadpool está esgotado e bloqueava novas requisições
- Mover para uma função async, já que ele está apenas criando uma instância do SQS

```python
def get_queue() -> SqsSimpleQueue:
    return SqsSimpleQueue(
        SqsQueueConfig(
            name=settings...,
            processor_thread_count=settings...,
            visibility_timeout_seconds=settings...,
        )
    )

class Producer:
    def __init__(self, queue: SqsSimpleQueue = Depends(get_queue)):
        self.queue = queue
```

```
0.531 Runner.run  asyncio/runners.py:86
├─ 0.480 coro  starlette/middleware/base.py:65
│  ├─ 0.433 ExceptionMiddleware.__call__  starlette/middleware/exceptions.py:53
│  │  └─ 0.433 AsyncExitStackMiddleware.__call__  fastapi/middleware/asyncexitstack.py:12
│  │     └─ 0.433 APIRouter.__call__  starlette/routing.py:697
│  │     └─ 0.433 APIRoute.handle  starlette/routing.py:265
│  │           └─ 0.433 app  starlette/routing.py:63
│  │           ├─ 0.402 app  fastapi/routing.py:217
│  │           │  ├─ 0.277 solve_dependencies  fastapi/dependencies/utils.py:508
│  │           │  │  ├─ 0.180 solve_dependencies  fastapi/dependencies/utils.py:508
│  │           │  │  │  └─ 0.180 run_in_threadpool  starlette/concurrency.py:35
│  │           │  │  └─ 0.180 run_sync  anyio/to_thread.py:12
│  │           │  │           └─ 0.180 run_sync_in_worker_thread
```

Depois de mover para uma função async

Criar SqsSimpleQueue faz uma requisição para a AWS (blocking)

Movendo a fila (SQS) para um singleton

```
0.212 Runner.run  asyncio/runners.py:86
├─ 0.193 coro  starlette/middleware/base.py:65
│  ├─ 0.186 ExceptionMiddleware.__call__  starlette/middleware/exceptions.py:53
│  │  └─ 0.186 AsyncExitStackMiddleware.__call__  fastapi/middleware/asyncexitstack.py:12
│  │     └─ 0.186 APIRouter.__call__  starlette/routing.py:697
│  │           └─ 0.186 APIRoute.handle  starlette/routing.py:265
│  │           └─ 0.186 app  starlette/routing.py:63
│  │           ├─ 0.155 app  fastapi/routing.py:217
│  │           │  └─ 0.155 run_endpoint_function  fastapi/routing.py:182
│  │           │     └─ 0.155 pending_invoice_created_webhook_handler
billable_event_consumer/routers/webhooks/chargebee/pending_invoice_created.py:23
│  │           │     ├─ 0.137 run_in_threadpool  starlette/concurrency.py:35
│  │           │     │  └─ 0.137 run_sync  anyio/to_thread.py:12
│  │           │     │     └─ 0.137 run_sync_in_worker_thread  anyio/_backends/_asyncio.py:834
│  │           │     │     ├─ 0.120 [await]  anyio/_backends/_asyncio.py
│  │           │     │     ├─ 0.007 CapacityLimiter.__aenter__  anyio/_backends/_asyncio.py:1796
│  │           │     │     │  └─ 0.007 CapacityLimiter.acquire  anyio/_backends/_asyncio.py:1857
│  │           │     │     │     └─ 0.007 CapacityLimiter.acquire_on_behalf_of  anyio/_backends/_asyncio.py:1860
│  │           │     │     │     └─ 0.007 cancel_shielded_checkpoint  anyio/_backends/_asyncio.py:469
│  │           │     │     │           └─ 0.007 sleep  asyncio/tasks.py:627
│  │           │     │     │           ├─ 0.006 [await]  asyncio/tasks.py
│  │           │     │     │           └─ 0.001 [self]  asyncio/tasks.py
│  │           │     │     ├─ 0.006 checkpoint  anyio/_backends/_asyncio.py:446
│  │           │     │     │  └─ 0.006 sleep  asyncio/tasks.py:627
│  │           │     │     │     └─ 0.006 [await]  asyncio/tasks.py
│  │           │     │     └─ 0.003 WorkerThread.start  threading.py:938
│  │           │     │           └─ 0.003 Event.wait  threading.py:604
│  │           │     │           └─ 0.003 Condition.wait  threading.py:288
│  │           │     │           └─ 0.003 lock.acquire  <built-in>
│  │
```

```
vegeta attack -rate 500 -duration 5s
```

99% 258ms 150rps

Sucesso | Resposta mais rápida | Throughput

```
Requests      [total, rate, throughput]         2500, 500.22, 158.58
Duration      [total, attack, wait]             15.734s, 4.998s, 10.736s
Latencies     [min, mean, 50, 90, 95, 99, max]  258.44ms, 5.595s, 6.684s, 10.11s, 10.551s, 10.795s, 10.933s
Bytes In      [total, mean]                     100575, 40.23
Bytes Out     [total, mean]                     14215000, 5686.00
Success       [ratio]                           99.80%
Status Codes  [code:count]                      200:2495  502:5
Error Set:
502 Bad Gateway
```

* 500s: the pod restarted because liveness probes failed (timeout exceeded because the server was busy)

Uma surpresa ao refatorar

```python
async def get_body(request: Request) -> str:
    body = await request.body()
    return body.decode("utf-8")

@router.post(PENDING_INVOICE_CREATED_PATH)
def pending_invoice_created_webhook_handler(
    request: str = Depends(get_body),
    producer: Producer = Depends(Producer),
) -> JSONResponse:
    try:
        PendingInvoiceCreated.model_validate_json(body_str)
        producer.send_message(body_str)
    except ValidationError as e:
        pass
    except Exception as e:
        pass
    return JSONResponse(status_code=200, content={...})
```

```
vegeta attack -rate 500 -duration 5s
```

100% 437ms 195rps

Sucesso | Resposta mais rápida | Throughput

```
Requests      [total, rate, throughput]         2500, 500.21, 195.40
Duration      [total, attack, wait]             12.794s, 4.998s, 7.796s
Latencies     [min, mean, 50, 90, 95, 99, max]  437.754ms, 4.427s, 5.879s, 7.601s, 7.78s, 8.011s, 8.097s
Bytes In      [total, mean]                     100000, 40.00
Bytes Out     [total, mean]                     14215000, 5686.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:2500
Error Set:
```

vegeta attack -rate 200 -duration 4m (sustentar 200 RPS) - Total de 48k requisições em 4 minutos

100%

Sucesso

187ms

Resposta mais rápida

198rps

Throughput

```
Requests      [total, rate, throughput]         48000, 200.00, 198.75
Duration      [total, attack, wait]             4m2s, 4m0s, 1.509s
Latencies     [min, mean, 50, 90, 95, 99, max]  187.879ms, 363.993ms, 248.382ms, 587.687ms, 990.065ms, 1.738s,
6.794s
Bytes In      [total, mean]                     1920000, 40.00
Bytes Out     [total, mean]                     272928000, 5686.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:48000
Error Set:
```

vegeta attack -rate 200 -duration 4m (sustentar 200 RPS) - Total de 48k requisições em 4 minutos

“Uma pessoa que nunca cometeu um erro nunca tentou algo novo”

- Albert Einstein

Suposições

Às vezes nossas suposições nos confundem, como assumir que requisições eram processadas em threads.

Fique de olho ao trabalhar com async

Async é legal, mas requer atenção extra para não bloquear o event loop e travar a aplicação, algo que não ocorre com threads.

200 rps

Cada requisição usa ~5ms do tempo de CPU. É possível que possamos otimizar ainda mais.

Apêndice

200 RPS é bom?

O que podemos esperar?

FastAPI diz que pode processar milhares de RPS

Veja benchmarks.

Qual a configuração para processar tantas requisições?

Qual o hardware? Precisa de multiplos workers?

Como isso se compara com a nossa infra?

Benchmarks podem mudar em comparação com a nossa infra e como isso afeta o throughput?

Recursos (kubernetes)

Requests 0.25 CPU and 256Mi RAM. Limits 512Mi RAM, sem limite para CPU.

200 worker threads

vegeta attack -rate 500 -duration 5s

```python
@router.post("/do-nothing-sync")
def do_nothing_sync() -> JSONResponse:
    return JSONResponse(status_code=200, content={"success": "Do nothing sync"})
```

```
Requests      [total, rate, throughput]         2500, 500.21, 461.53
Duration      [total, attack, wait]             5.417s, 4.998s, 418.858ms
Latencies     [min, mean, 50, 90, 95, 99, max]  186.846ms, 438.158ms, 389.384ms, 674.563ms, 857.468ms, 1.014s,
1.506s
Bytes In      [total, mean]                     75000, 30.00
Bytes Out     [total, mean]                     14215000, 5686.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:2500
Error Set:
```

vegeta attack -rate 500 -duration 5s

```python
@router.post("/do-nothing-async")
async def do_nothing_async() -> JSONResponse:
    return JSONResponse(status_code=200, content={"success": "Do nothing async"})
```

```
Requests      [total, rate, throughput]         2500, 500.22, 432.16
Duration      [total, attack, wait]             5.785s, 4.998s, 787.044ms
Latencies     [min, mean, 50, 90, 95, 99, max]  185.269ms, 690.328ms, 524.101ms, 1.193s, 2.201s, 2.939s, 3.4s
Bytes In      [total, mean]                     72500, 29.00
Bytes Out     [total, mean]                     14215000, 5686.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:2500
Error Set:
```

vegeta attack -rate 500 -duration 5s

```python
@router.post("/do-nothing-sync-body")
def do_nothing_sync_body(body_str: str = Depends(get_body)) -> JSONResponse:
    return JSONResponse(status_code=200, content={"success": f"Do nothing sync {body_str[:10]}"})
```

```
Requests      [total, rate, throughput]         2500, 500.22, 408.78
Duration      [total, attack, wait]             6.116s, 4.998s, 1.118s
Latencies     [min, mean, 50, 90, 95, 99, max]  185.479ms, 867.97ms, 616.135ms, 2.118s, 2.866s, 3.585s, 3.774s
Bytes In      [total, mean]                     107500, 43.00
Bytes Out     [total, mean]                     14215000, 5686.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:2500
Error Set:
```

vegeta attack -rate 500 -duration 5s

```python
@router.post("/do-nothing-async-body")
async def do_nothing_async_body(request: Request) -> JSONResponse:
    body = await request.body()
    body_str = body.decode("utf-8")
    return JSONResponse(status_code=200, content={"success": f"Do nothing async {body_str[:10]}"})
```

```
Requests      [total, rate, throughput]         2500, 500.22, 411.22
Duration      [total, attack, wait]             6.079s, 4.998s, 1.082s
Latencies     [min, mean, 50, 90, 95, 99, max]  183.287ms, 682.225ms, 530.176ms, 1.358s, 1.982s, 2.603s, 2.718s
Bytes In      [total, mean]                     110000, 44.00
Bytes Out     [total, mean]                     14215000, 5686.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:2500
Error Set:
```

vegeta attack -rate 500 -duration 5s

```python
@router.post("/do-nothing-sync-model-body")
def do_nothing_sync_model_body(payload: PendingInvoiceCreated) -> JSONResponse:
    return JSONResponse(status_code=200, content={"success": f"Do nothing sync model {payload.id}"})
```

```
Requests      [total, rate, throughput]         2500, 500.22, 354.59
Duration      [total, attack, wait]             7.05s, 4.998s, 2.053s
Latencies     [min, mean, 50, 90, 95, 99, max]  198.023ms, 1.246s, 735.892ms, 3.165s, 3.743s, 4.232s, 4.287s
Bytes In      [total, mean]                     110000, 44.00
Bytes Out     [total, mean]                     14215000, 5686.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:2500
Error Set:
```

vegeta attack -rate 500 -duration 5s

```python
@router.post("/do-nothing-async-model-body")
async def do_nothing_async_model_body(payload: PendingInvoiceCreated) -> JSONResponse:
    return JSONResponse(status_code=200, content={"success": f"Do nothing async model {payload.id}"})
```

```
Requests      [total, rate, throughput]         2500, 500.22, 374.48
Duration      [total, attack, wait]             6.676s, 4.998s, 1.678s
Latencies     [min, mean, 50, 90, 95, 99, max]  186.523ms, 1.006s, 653.306ms, 2.524s, 3.086s, 3.627s, 3.664s
Bytes In      [total, mean]                     112500, 45.00
Bytes Out     [total, mean]                     14215000, 5686.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:2500
Error Set:
```

vegeta attack -rate 500 -duration 5s

```python
@router.post("/do-nothing-sync-http")
def do_nothing_sync_http() -> JSONResponse:
    response = requests.get("https://alguma-url-interna")
    return JSONResponse(status_code=200, content={"success": f"Do nothing sync http {response.status_code}"})
```

```
Requests      [total, rate, throughput]         2500, 500.22, 159.97
Duration      [total, attack, wait]             15.628s, 4.998s, 10.63s
Latencies     [min, mean, 50, 90, 95, 99, max]  489.027ms, 5.738s, 6.848s, 9.95s, 10.368s, 10.693s, 10.805s
Bytes In      [total, mean]                     95000, 38.00
Bytes Out     [total, mean]                     14215000, 5686.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:2500
Error Set:
```

vegeta attack -rate 500 -duration 5s

```python
@router.post("/do-nothing-async-http")
async def do_nothing_async_http() -> JSONResponse:
    async with aiohttp.ClientSession() as session:
        async with session.get("https://alguma-url-interna") as resp:
            return JSONResponse(status_code=200, content={"success": f"Do nothing async http {resp.status}"})
```

```
Requests      [total, rate, throughput]         2500, 500.21, 252.25
Duration      [total, attack, wait]             9.911s, 4.998s, 4.913s
Latencies     [min, mean, 50, 90, 95, 99, max]  231.306ms, 2.683s, 1.231s, 5.426s, 5.462s, 5.623s, 5.931s
Bytes In      [total, mean]                     97500, 39.00
Bytes Out     [total, mean]                     14215000, 5686.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:2500
Error Set:
```

vegeta attack -rate 500 -duration 5s

```python
@router.post("/do-something-sync")
def do_something(key: str, body_str: str = Depends(get_body)) -> JSONResponse:
    response = requests.get('https://alguma-url-interna')
    return JSONResponse(
        status_code=200,
        content={"success": f"Do something sync {response.status_code} {key} {body_str[:10]}"},
    )
```

```
Requests      [total, rate, throughput]         2500, 500.22, 152.57
Duration      [total, attack, wait]             16.386s, 4.998s, 11.389s
Latencies     [min, mean, 50, 90, 95, 99, max]  533.723ms, 6.133s, 7.113s, 10.587s, 11.124s, 11.378s, 11.42s
Bytes In      [total, mean]                     205000, 82.00
Bytes Out     [total, mean]                     14215000, 5686.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:2500
Error Set:
```

vegeta attack -rate 500 -duration 5s

```python
@router.post("/do-something-async")
async def do_something_async(request: Request, key: str) -> JSONResponse:
    body = await request.body()
    body_str = body.decode("utf-8")
    async with aiohttp.ClientSession() as session:
        async with session.get("https://alguma-url-interna") as resp:
            return JSONResponse(
                status_code=200,
                content={"success": f"Do something async {resp.status} {key} {body_str[:10]}"},
            )
```

```
Requests       [total, rate, throughput]         2500, 500.22, 212.33
Duration       [total, attack, wait]             11.774s, 4.998s, 6.776s
Latencies      [min, mean, 50, 90, 95, 99, max]  417.271ms, 3.65s, 2.4s, 6.78s, 6.925s, 7.145s, 7.212s
Bytes In       [total, mean]                     207500, 83.00
Bytes Out      [total, mean]                     14215000, 5686.00
Success        [ratio]                           100.00%
Status Codes   [code:count]                      200:2500
Error Set:
```

Fim.