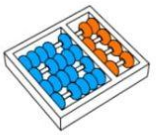


Análise de processamento de notas de corretagem em nuvem

K. C. Takuma *L. F. Bittencourt* *M. F. Olivi*

Relatório Técnico - IC-PFG-22-39
Projeto Final de Graduação
2022 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Resumo

Com a evolução do mercado financeiro[1] e com aumento do número de transações financeiras em circulação, viu-se a necessidade da criação de ferramentas que possibilitem a automatização de operações e a solução de gargalos relacionados à extração e manipulação de notas de corretagem. Aliado a isso, a evolução da *cloud computing* viabilizou maiores possibilidades de gerenciamento de recursos, por meio da criação de serviços, permitindo escalar a automatização de processos para múltiplos clientes nos mais distintos cenários de mercado.

Nesse contexto, foi criado um algoritmo que realiza o parsing do conteúdo da nota de corretagem, utilizando diferentes serviços da AWS de modo a chegar em um sistema econômico, rápido e confiável. Neste trabalho, é detalhado o impacto em desempenho dessa automatização para diferentes ambientes remotos. Buscando a análise crítica para diferentes cenários do mercados e com o uso de diversos serviços na nuvem, focando em desempenho, custo e escalabilidade.

1. Introdução

Na última década, os meios de investimento sofreram grandes melhorias no desempenho e qualidade de seus serviços, com um crescimento de aproximadamente 1000% [2] no que diz respeito ao número de investidores na bolsa de valores brasileira. No decorrer desse crescimento, viu-se a necessidade de melhorar e automatizar processos que até então eram feitos de forma manual.

Uma nota de corretagem é o documento oficial para realização de cálculos e ajustes não só na receita federal e consequentemente no planejamento tributário, mas também para o controle financeiro e avaliação do desempenho na bolsa de valores. Analisando o mercado financeiro, notou-se um grande gargalo relacionado à extração das operações financeiras de cada investidor e ao controle do mesmo, tendo em vista as inúmeras dificuldades em extrair e manipular as notas de corretagem fornecidas por cada corretora. Essa dificuldade, muitas vezes, é decorrente do formato do arquivo enviado, comumente no formato pdf, e as diferentes padronizações do modelo de saída para cada corretora.

Nesse sentido, na medida que novas corretoras surgiram, e considerando as inúmeras variações de operações na bolsa brasileira, surgiu a necessidade de ter um produto que fosse genérico, econômico, rápido e confiável. Pensando em atender todos esses pontos, foi utilizada a ideia de computação em nuvem, ou *cloud computing*, onde é possível gerenciar sob demanda a disponibilidade de recursos para uma determinada aplicação.

Por meio da computação em nuvem, do inglês *cloud computing*, é possível fazer ajustes elásticos na disponibilidade de capacidade de recursos computacionais, seja com foco na escalabilidade vertical ou horizontal. Além disso, devido a facilidade referente ao controle de recursos e a adaptabilidade, às aplicações na nuvem permitem adaptar o negócio a diferentes públicos alvos, desde poucos escritórios com grandes volumes de carga a muitos usuários com pequenas cargas simultâneas.

Com essa motivação, este trabalho apresenta um estudo sobre o impacto no desempenho de diversos serviços na nuvem sobre a realização da análise de notas de corretagem. Buscando a análise empírica para diversas situações de mercados para múltiplas situações de teste de estresse. Comparando para cada modelo, situações de escalabilidade, desempenho e custo.

2. Referencial Teórico

Nesta seção apresentamos conceitos importantes e algumas definições utilizados para o entendimento do desenvolvimento do projeto.

2.1 Virtualização e Cloud computing

Nessa subseção será descrito conceitos referentes ao desenvolvimento no serviços de nuvem, como a utilização de algumas ferramentas da AWS e do Docker.

2.1.1 Conteinerização

Containers[3] são ambientes isolados que proporcionam uma maneira padrão de empacotar códigos, arquivos de configurações e dependência de uma aplicação, isolando todo conteúdo em um único objeto. Por meios dos containers é possível configurar o uso da CPU e da memória de maneira controlada para melhor utilização dos recursos computacionais.

Cada container é acionado a partir de um processo separado que compartilha os recursos do sistema operacional, permitindo que os containers sejam iniciados e desligados rapidamente, sendo dessa forma facilmente escalável.

2.1.2 Docker

Docker[4] é uma plataforma open source que permite a criação, o teste e a implantação de aplicações rapidamente, utilizando os conceitos de container. Por meio do docker, pode-se gerar instâncias de imagens que vão nos permitir construir containers.

A Partir do docker, será possível padronizar operações, utilizar recursos e imagens prontas a partir do próprio Docker Hub, de modo a economizar recursos, tempo e consequentemente dinheiro, nos permitindo a portabilidade e facilitando as diversas simulações realizadas sobre as aplicações da AWS.

2.1.3 Cloud Computing

Cloud Computing[5], ou computação em nuvem, é a entrega de recursos de TI sob demanda por meio da internet com definições de preços em decorrência dos contratos de utilização. A computação em nuvem possibilita o acesso a serviços de tecnologia, como capacidade computacional e armazenamento conforme a necessidade, além de possibilitar maior velocidade e agilidade em infra estrutura como também proporciona redundância de servidores e a elasticidade de acordo com a necessidade.

2.1.4 Amazon AWS

A Amazon Web Services, AWS[6], é uma plataforma de serviços em nuvem oferecida pela Amazon, atualmente a AWS tem mais de 175 serviços completos, dentre os utilizado no projeto estão o Amazon S3, Amazon EC2 , AWS Load Balancer, AWS Lambda e AWS ECR.

2.1.5 Amazon S3

Amazon S3[7] ou Amazon Simple Storage Service é um serviço de armazenamento de objetos, por meio de uma interface de serviço da Amazon, que oferece escalabilidade, disponibilidade de dados, segurança e alto desempenho. Seus preços variam por região, para esse projeto foi utilizado a região Leste dos EUA US East (N.Virginia).

2.1.6 Amazon EC2

Amazon Elastic Compute Cloud ou Amazon EC2 é um serviço web que disponibiliza capacidade computacional segura e redimensionável permitindo que os usuários aluguem computadores virtuais de forma segura, confiável, escalável e com preços variáveis por demanda dependendo do tipo de recurso necessário e da região.

Para esse projeto foi utilizada a região Leste dos EUA US East (N.Virginia) com as instâncias T2.micro e T2.medium.

2.1.7 AWS Load Balancer

O Amazon Elastic Load Balancer ou ELB é o serviço da AWS que distribui o tráfego de rede para melhorar a escalabilidade das aplicações. O ELB distribui o tráfego que entra por meio da rede ou dos aplicativos e envia para outros destinos como containers, endereços de IP ou próprios serviços da AWS como Amazon EC2 e o Lambda.

O ELB consegue dimensionar de forma automática a sua distribuição à medida que o tráfego aumenta ou diminui com o passar do tempo, conseguindo dessa forma alocar recursos e balancear enormes cargas de trabalho. Além disso, é possível configurar o Elastic Load Balancer das mais diferentes maneiras, monitorando as mais variadas necessidades, permitindo apenas que as instâncias internas sejam utilizadas.

Na figura 01, está apresentado um esquema simplificado do Elastic Load Balancer:

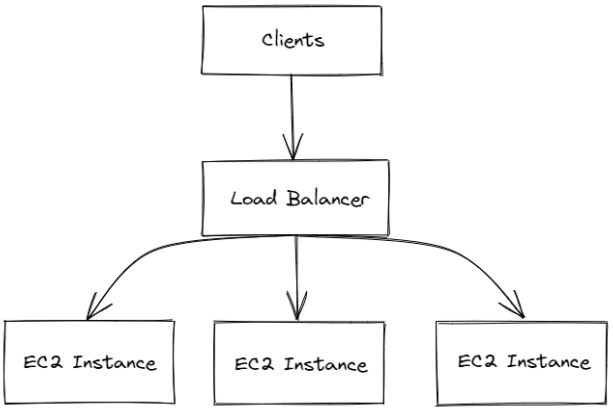


Figura 01: Modelo simplificado ELB.

2.1.8 AWS Lambda

O AWS Lambda[8] é um serviço orientado a eventos sem servidor o que possibilita executar serviços de back end ou qualquer aplicação sem o gerenciamento de servidores, sendo toda camada de infraestrutura do Lambda gerenciada pela AWS.

O modelo de precificação do lambda é do tipo *pay-as-you-go*, portanto os clientes pagam apenas pelos recursos utilizados, no caso do lambda prevalecendo a memória alocada e o tempo de execução. Cada função Lambda é executada em seu próprio container, ao criar uma função o AWS lambda a empacota em um novo container e em seguida executa o container em um cluster multilocatário de máquinas gerenciadas pela AWS .

Antes da execução de cada função, o container recebe a capacidade de CPU necessária e memória RAM para cada função. Ao término da execução das funções, a RAM alocada no início e multiplicada pelo tempo gasto durante a execução da função. Além disso o lambda, oferece suporte a diversas linguagens dentre elas o Node.js, Python, Java, Ruby, C#, Go e PowerShell.

2.1.9 Amazon ECR

O Amazon Elastic Container registry ou Amazon ECR é uma ferramenta da AWS que armazena e registra imagens de containers de forma que seja escalável e seguro.

2.2 Definições referentes à Aplicação

Nesta seção, serão descritos alguns conceitos que foram utilizados especificamente durante o desenvolvimento da aplicação já citada.

2.2.1 Fast API

O Fast API[9] é um Framework Web em python para desenvolvedores de APIs RESTful, auto documentavel, permite uso de APIs assíncronas, promete alto desempenho além de ter uma documentação intuitiva e acessível.

2.2.2 Json

O JavaScript Object Notation[10] ou Json é um arquivo que contém uma série de dados estruturados em formato de texto. Os dados contidos em um JSON são estruturados por meio de uma coleção de pares com chave e valor. Dentre as finalidades do JSON, está a transferência de dados entre aplicações por meio de APIs e a realização de requisições.

2.2.3 Notas de corretagem

Notas de corretagem[11] são extratos do mercado financeiro, através do qual é possível verificar gastos, operações, datas, preços de aquisição de ações e emolumentos. As notas de corretagem são geradas no formato PDF seguindo as normas referentes ao padrão Sinacor.

2.2.4 Padrão Sinacor

O sistema integrado de administração de corretoras (SINACOR), criou um padrão (padrão sinacor), de modo garantir a qualidade administrativa das notas de corretagem. O padrão sinacor é o modelo oficial aceito pela receita federal, englobando atualmente 95% das corretoras.

3. Metodologia

Nesta seção, é definido o escopo do projeto, as regras gerais definidas assim como a solução na nuvem.

3.1 Definição do Escopo

Para a listagem dos requisitos iniciais, foi utilizado o brainstorm de modo a estimular a criação de ideias e consequentemente de possíveis soluções. Para realização do filtro dessas soluções utilizamos o método de Matriz Moscow, de modo a organizar as prioridades do projeto e o desenvolvimento da aplicação, tendo assim um maior controle do escopo geral do projeto.

Por meio das informações utilizadas a partir da matriz de Moscow foi utilizado o método da matriz de risco de forma a mapear o que seria possível entregar durante o tempo estipulado para o projeto e quais as relações de risco para cada item.

Dessa forma, foi preparada o seguinte modelo de matriz de Moscow conforme a Figura 02:

DEVEMOS TER	DEVERIAMOS TER
- Custo de Corretagem - Inclusão dos principais tipos de ativo - Generalização p/ vários tipos de nota - Balanço entre preço, escalabilidade e eficiência - Facilidade na inserção de novos tipos de notas de corretagem .	- Transformação do nome do ativo em ticker. - Criação de uma API escalável na Nuvem
PODERIAMOS TER	**NÃO TEREMOS**
- Desenvolvimento de um Front End. - Autenticação de usuário. - Metrificação com uso do grafana. - Leitura de varias notas em Lote	- Ajuste operacional para IR. - Leitura de extratos bancários.

Figura 02: Matriz de Moscow.

A partir da análise da figura 02, decidiu-se, por questão do escopo deste trabalho, priorizar os seguintes itens:

1) Generalização das notas de corretagem de modo a incluir o maior número de notas.
2) Inclusão dos principais tipos de ativos referentes ao Ibovespa.
3) Possibilidade de extração de custos de corretagem.
4) Facilidade na inserção de novos tipos de notas de corretagem.
5) Soluções que ponderam preço, escalabilidade e eficiência.
6) Criação de APIs ajustáveis a diferentes cenários do mercado.

3.2 Detalhamento da solução

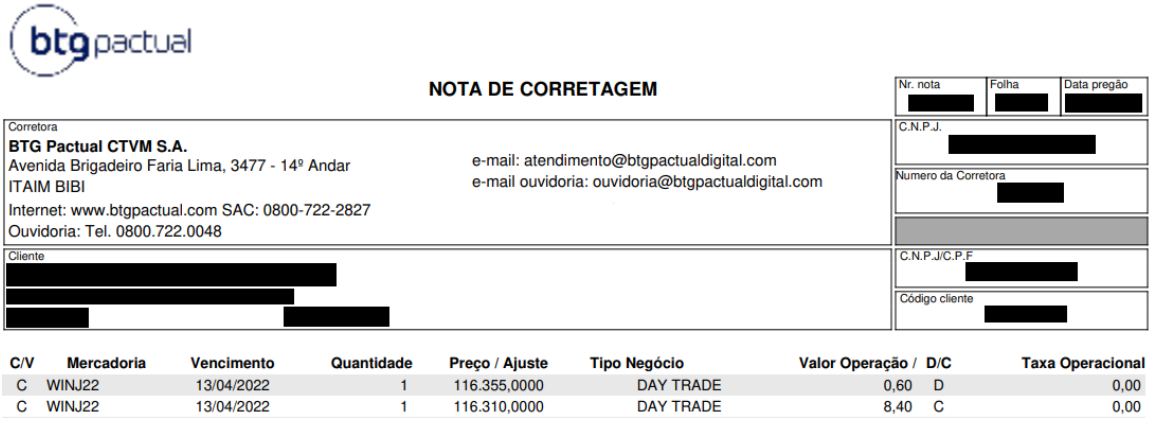
A ferramenta em questão realiza a extração e conversão do conteúdo da nota de corretagem no padrão B3 (sinacor) e retorna dados em um padrão estruturado que permita a leitura e as possíveis automatizações, seja para cálculos de ajustes como também para o planejamento tributário.

Durante o processo, foi possível notar que embora todas as notas estivessem no padrão sinacor, as notas apresentavam campos distintos, podendo variar tanto por corretora, como também por tipo de ativos.

Neste trabalho, utilizamos ferramentas que permitem lidar com os seguintes tipos de ativos: Ações, ETFs, BDR , FIIs, Units, Opções, Exercicio de opções e Futuros. Para o desenvolvimento da ferramenta foram coletadas informações das seguintes corretoras: XP, Clear, Rico, BTG, Genial, Órama e NuInvest.

A ferramenta em questão, é responsável por ler cada página do pdf referente a nota de corretagem, identificar o cabeçalho de cada corretora, extrair, para cada operação o ticket do ativo, usando um conjunto de dados da B3 para a conversão das descrições em ticket, além de extrair outros campos importantes da nota de corretagem.

As figuras 3 e 4 ilustram a variação no padrão de cabeçalho, exemplificando as corretoras BTG e XP, respectivamente. Diferenças podem ser observadas na nomeação das colunas e nos espaçamentos entre os caracteres.



btg pactual

NOTA DE CORRETAGEM

Nr. nota	Folha	Data pregão
[redacted]	[redacted]	[redacted]

Corretora
BTG Pactual CTVM S.A.
Avenida Brigadeiro Faria Lima, 3477 - 14º Andar
ITAIM BIBI
Internet: www.btgpactual.com SAC: 0800-722-2827
Ouvidoria: Tel. 0800.722.0048

e-mail: atendimento@btgpactualdigital.com
e-mail ouvidoria: ouvidoria@btgpactualdigital.com

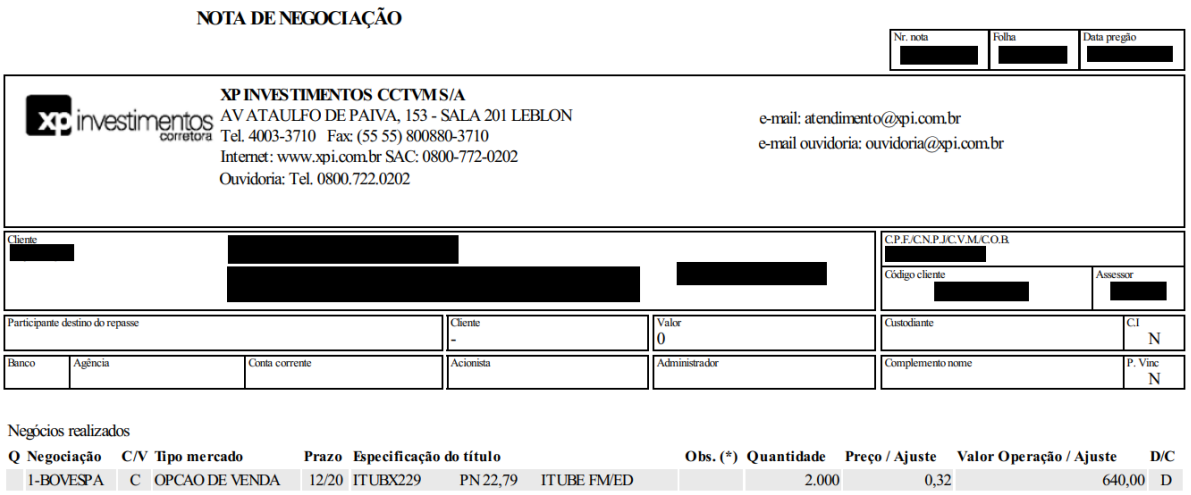
C.N.P.J. [redacted]
Numero da Corretora [redacted]

Cliente [redacted]

C.N.P.J/C.P.F [redacted]
Código cliente [redacted]

C/V	Mercadoria	Vencimento	Quantidade	Preço / Ajuste	Tipo Negócio	Valor Operação /	D/C	Taxa Operacional
C	WINJ22	13/04/2022	1	116.355,0000	DAY TRADE	0,60	D	0,00
C	WINJ22	13/04/2022	1	116.310,0000	DAY TRADE	8,40	C	0,00

Figura 03: Nota de corretagem BTG



NOTA DE NEGOCIAÇÃO

Nr. nota	Folha	Data pregão
[redacted]	[redacted]	[redacted]

XP INVESTIMENTOS CCTVM S/A
AV ATAULFO DE PAIVA, 153 - SALA 201 LEBLON
Tel. 4003-3710 Fax: (55 55) 800880-3710
Internet: www.xpi.com.br SAC: 0800-772-0202
Ouvidoria: Tel. 0800.722.0202

e-mail: atendimento@xpi.com.br
e-mail ouvidoria: ouvidoria@xpi.com.br

Cliente		C.P.F./C.N.P.J/C.V.M./C.O.B.	
[redacted]	[redacted]	[redacted]	Código cliente: [redacted] / Assessor: [redacted]

Participante destino do repasse	Cliente	Valor	Custodiante	C.I
	-	0		N

Banco	Agência	Conta corrente	Acionista	Administrador	Complemento nome	P. Vinc
						N

Negócios realizados

Q	Negociação	C/V	Tipo mercado	Prazo	Especificação do título	Obs. (*)	Quantidade	Preço / Ajuste	Valor Operação / Ajuste	D/C
	1-BOVESPA	C	OPCAO DE VENDA	12/20	ITUBX229 PN 22,79 ITUBE FM/ED		2.000	0,32	640,00	D

Figura 04: Nota de corretagem XP

Para otimizarmos o mapeamento de modo a generalizar a extração do conteúdo das notas, foi fixado os campos de cada cabeçalho. Dessa forma, além de facilitar a extração de cada nota, esse modelo permite expandir o projeto de modo a adicionar mais notas de corretagem, incrementando apenas o campo do cabeçalho da respectiva nota de corretagem.

A resposta do parser é um JSON para cada operação da nota, um exemplo de resposta para uma operação pode ser representada a partir da figura 05:


```
{
    "number": "0001",
    "date": "2020-10-20",
    "brokerName": "BANCO PACTUAL DTVM",
    "tradeType": "buy",
    "market": "full",
    "tradingName": "PETROBRAS     ON  NM",
    "quantity": 8400,
    "price": 23.65,
    "totalValue": 198660,
    "dayTrade": false,
    "symbol": "PETR3",
    "type": "stocks",
    "clearingCosts": 232.27,
    "otherCosts": 46.56
}
```


Figura 05: Resposta da API da Aplicação.

Sobre o JSON resultante apresentado por meio da figura 05, veja por meio da tabela 01 o significado de cada campo, detalhando para cada chave do JSON seu significado:

Campo	Descrição
number	Identificador da nota.
date	Data do pregão.
brokerName	Nome da corretora.
tradeType	Tipo da transação (buy : Compra, sell:venda)
market	Tipo de mercado.
tradingName	Especificação do título.
quantity	Quantidade movimentado
price	Valor unitário do ativo
totalValue	Valor total da movimentação.
dayTrade	Sinaliza se foi, ou não, day Trade.
symbol	Símbolo referente ao ativo
type	Tipo de operação.
clearingCosts	Custo de corretagem.
otherCosts	Outros custos.

Tabela 01: Detalhamento dos campos referente a resposta da API.

3.3 Etapas de solução na nuvem

Inicialmente, de modo a atender os requisitos referentes ao item 3.2, foi desenvolvido um script local, com foco em realizar o parsing do conteúdo da nota de corretagem no padrão B3 (sinacor). Ao confirmar a corretude do código, foi utilizado o framework FastAPI, de modo a criar APIs assíncronas a fim de considerar vários usuários simultâneos.

Em seguida, buscando otimizar o tempo de resposta e a escala do projeto, criou-se uma instância do Amazon EC2, coletando e comparando os tempos de respostas e preços para um conjunto de notas de corretagem definidas.

A fim de entender melhor o serviço do Amazon EC2 variou-se os tipos de máquinas de modo a ter um comparativo prático entre tempo de resposta e preços. Procurando evitar gargalos através do uso de apenas uma instância do EC2 e a fim de testar uma solução mais escalável pensando em um número maior de cliente foi criado grupos de Auto Scaling com Load Balancer.

Além disso, buscando variar os cenários de modo a achar uma solução que leve em conta o balanço entre custo e eficiência da aplicação, utilizamos o AWS Lambda para criação e validação de diferentes esquemas de arquitetura.

4. Resultados Experimentais

Levando em conta o contexto apresentado na sessão anterior e pensando em vários usuários simultâneos, foi feito uma análise de como levar essa aplicação para a nuvem pensando tanto em preço, quanto na escalabilidade e eficiência.

A análise foi iniciada de maneira simples e foi se escalando até as maneiras mais robustas formas de levar essa aplicação para a nuvem.

4.1 Máquina no EC2

Como iniciado no tópico anterior, vamos começar da maneira mais simples, utilizando somente uma instância no EC2.

4.1.1 Máquina t2.micro

O estudo foi iniciado com uma máquina t2.micro, em que se utilizou o nginx para seguir como *reverse proxy* e o uvicorn[12] para rodar o FastAPI, uma ilustração dessa arquitetura se encontra na figura 06:

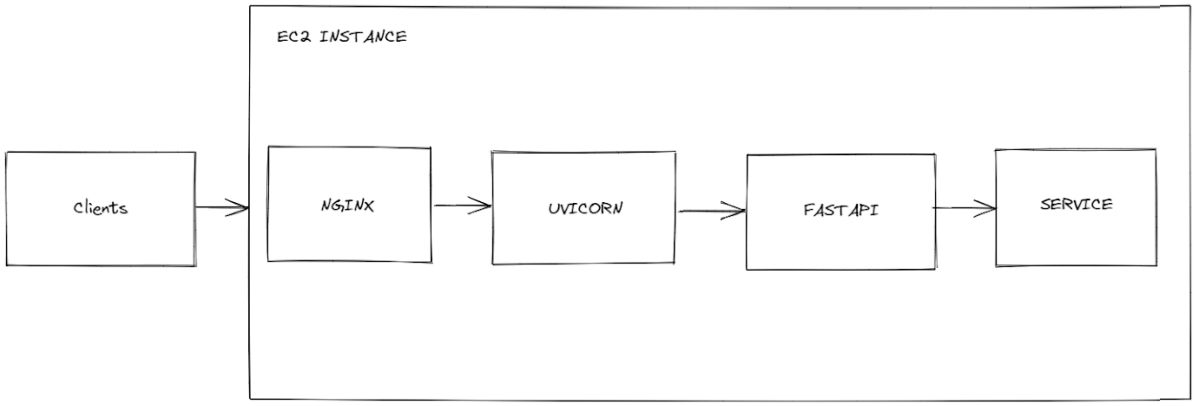


Figura 06: Arquitetura aplicação máquina t2.micro

A Figura 07 apresenta o resultado da análise de estresse executada considerando a arquitetura apresentada. Para obtê-la variou-se o número de clientes simultâneos e obteve-se o tempo médio de resposta de 5 execuções.

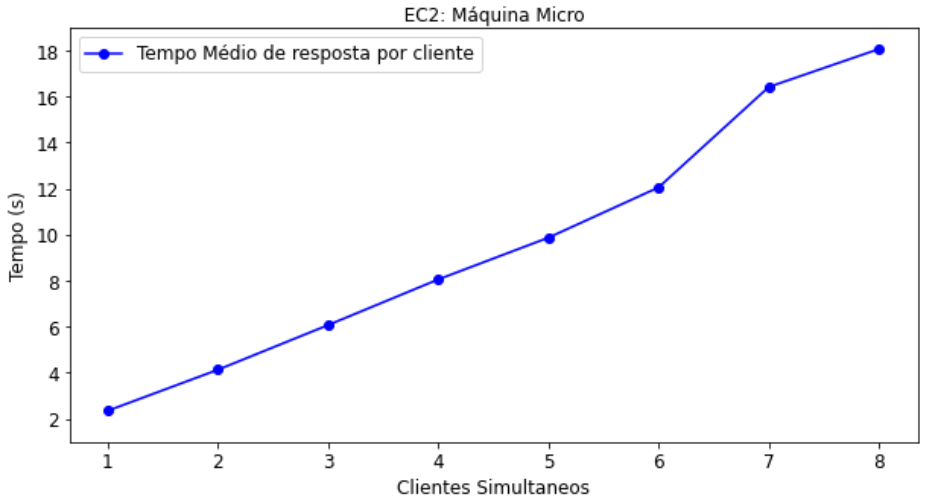


Figura 07: Teste de Estresse EC2 - t2.micro

No gráfico da figura 07, pode-se observar que o número de clientes simultâneos era proporcional ao tempo de retorno da requisição, com aproximadamente 2s para cada novo cliente. O teste encerrou com 8 clientes simultâneos porque a máquina parou de responder ao fazer o teste com um número maior.

4.1.2 Máquina t2.medium

Utilizando uma máquina t2.medium, seguiu-se com o mesmo esquema de arquitetura: nginx, uvicorn e FastApi. Entretanto, como a máquina t2.medium tem 2 núcleos, foram colocados 2 workers para o servidor do uvicorn, tendo uma arquitetura representada pela figura 08.

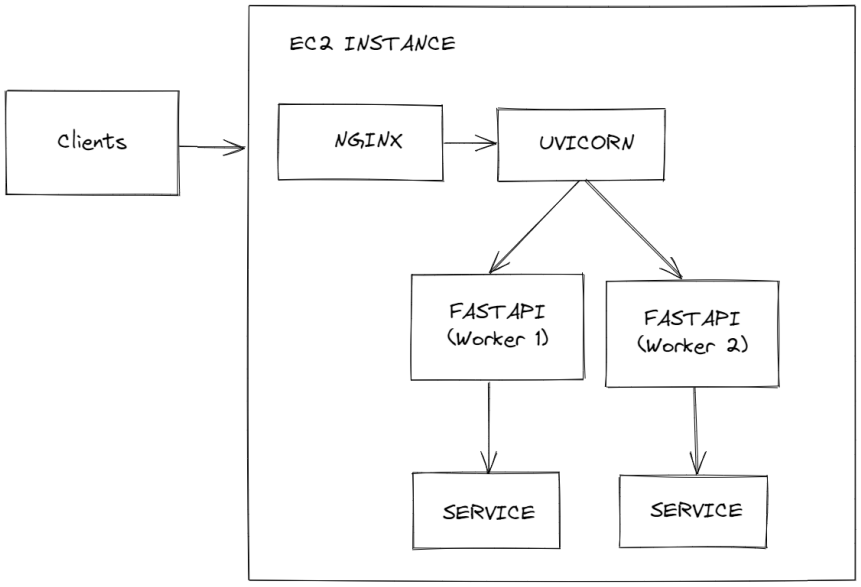


Figura 08: Arquitetura da aplicação para máquina t2.medium

Da mesma forma que foi feito para a máquina t2.micro, tentou-se estressar a aplicação, aplicando um número variável de clientes simultâneos e calculando o tempo médio de resposta de 5 execuções, obtendo-se o gráfico da figura 09.

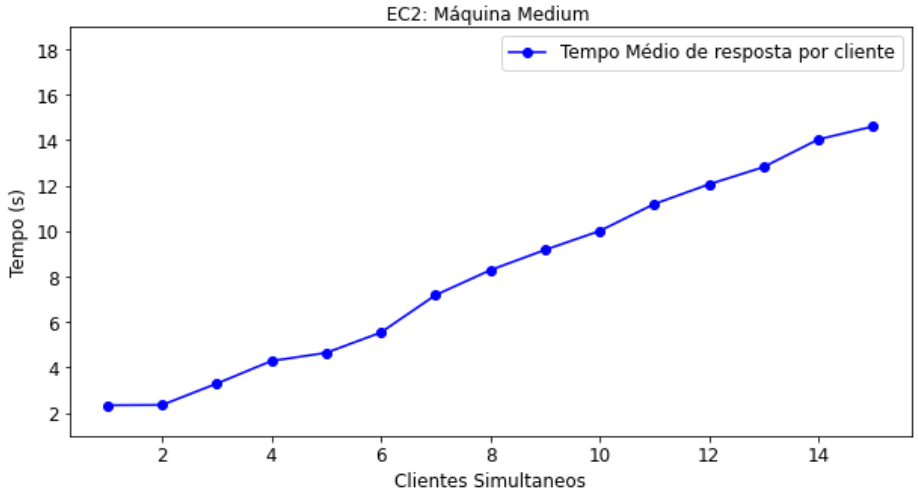


Figura 09: Teste de Estresse EC2 - t2.medium

Neste teste, também pode-se observar que o número de clientes simultâneos era proporcional ao tempo de retorno da requisição, com aproximadamente 1s para cada novo cliente.

4.1.3 Análise de preços

Por meio do simulador de preços da AWS [13], obteve-se os seguintes resultados, definindo a região Leste dos EUA US East (N.Virginia) e a aplicação Amazon EC2.

Nome	Memória alocada	Storage alocado	Valor Mensal	Valor no Ano
T2.micro	1 GB	8 GB	$6,06	$72,72
T2.medium	4 GB	8 GB	$21,75	$261,00

Tabela 02: Billing máquinas no EC2.

4.1.4 Discussão

Analisando os resultados obtidos, é possível notar que na máquina média foi obtido um tempo médio de requisição melhor do que para máquina micro, o que era esperado. Com um aumento de 2s para cada cliente simultâneo para a micro e 1s para maquina média.

Entretanto, apesar da máquina média ter se saído melhor, é possível notar que o resultado não é escalável, visto que a cada novo cliente simultâneo é perdido muito em desempenho e, portanto, para um número muito grande de clientes simultâneos, cada requisição demoraria um tempo elevado para obter a resposta.

4.2 Grupo de AutoScaling com Load Balancer

No tópico 4.1, foi verificado que a utilização de uma única máquina no EC2 não é uma solução escalável. Com isso em mente, o próximo passo foi utilizar um load balancer aliado a um grupo de autoscaling, serviços também encontrados no EC2 do AWS.

4.2.1 Arquitetura

Para criação desse sistema, foi considerada como imagem base a que foi configurada para a máquina do EC2 da Seção 4.1. A partir dessa imagem base, foi criado um grupo de autoscaling, com no máximo cinco máquinas e no mínimo uma, com parâmetro de criação de uma nova máquina a utilização de 50% do total de CPU. Então, fez-se um load balancer, acoplando o grupo de autoscaling configurado e adicionando um health checker para verificar a integridade da aplicação. Um desenho simplificado dessa arquitetura pode ser representado pela figura 10.

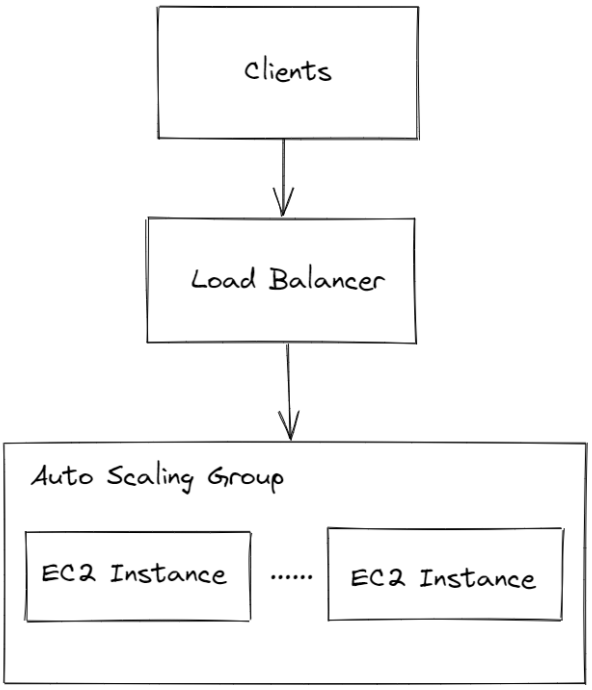


Figura 10: Arquitetura Load Balancer com grupo de AutoScaling

Da mesma forma que foi feito para as máquinas do EC2, foi feito um teste de estresse em cima da aplicação, conforme serão apresentados pelas seções seguintes.

4.2.2 Fixando número de máquinas

Para o teste inicial, foi configurado um número fixo de máquinas no grupo de instâncias, começando com uma máquina e indo até cinco. Além disso, o teste seguiu-se como anteriormente, obtendo o tempo médio de resposta por cliente, variando o número de clientes simultâneos, obtendo-se o gráfico da figura 11.

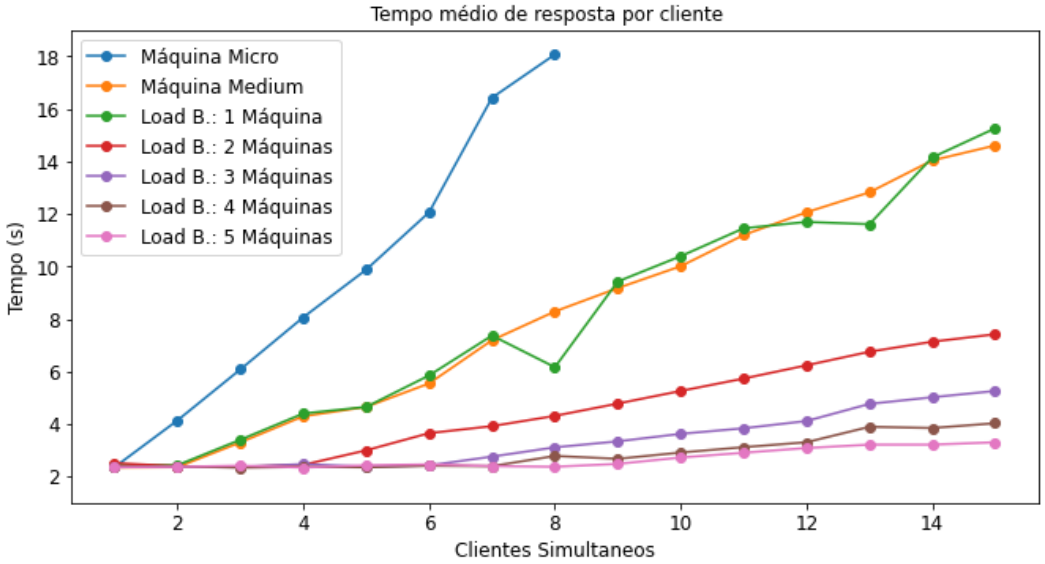


Figura 11: Tempo médio de resposta considerando varios cenários

Analisando as curvas da figura 11, é possível notar que quanto maior o número de máquinas, mais rápido é o tempo de resposta médio para cada cliente, como era de se esperar.

Além disso, à medida que o número de máquinas aumenta, é possível notar o surgimento de um patamar no início do gráfico. Esse patamar está relacionado com o número de núcleos que cada máquina possui e o número de máquinas existentes. Para melhor visualização, a figura 12 apresenta os gráficos relativos ao Load Balancer com duas e três máquinas, respectivamente.

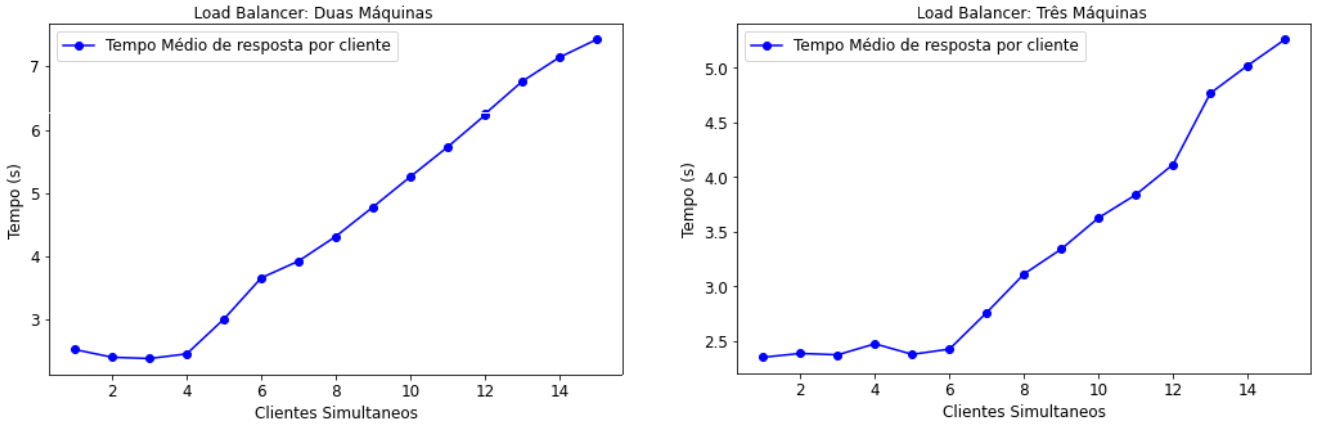


Figura 12: Teste de estresse para Load Balancer considerando duas e cinco máquinas

Na figura 12, observando o gráfico cinco, temos 3 máquinas com 2 núcleos cada, assim teremos núcleos disponíveis até 6 clientes, o que gera o surgimento de um patamar até aproximadamente 6 clientes simultâneos.

Completando a análise do gráfico da figura 12, nota-se que, quando foi aumentado o número de máquinas de 4 para 5, a melhora de tempo não foi tão significativa quanto o aumento de uma para duas máquinas. O que sugere que, para um grande número de máquinas, o maior limite está na aplicação em si, e não na concorrência entre os diversos clientes.

4.2.3 Utilizando Auto Scaling

Como apresentado anteriormente, foi utilizado um grupo de auto scaling para fazer a criação ou destruição automática das instâncias. Entretanto, até agora, ainda não foi mostrado isso na prática visto que, no tópico 4.1, foi fixado o número de máquinas para testar o comportamento da aplicação.

Como dito anteriormente, esse grupo de autoscaling foi configurado para operar com no mínimo uma e no máximo cinco máquinas, sendo o parâmetro de criação de uma nova máquina a utilização de 50% do total de CPU.

Para os testes de estresse, vale ressaltar que, fixamos como uma iteração 5 chamadas simultâneas para o load balancer. A medida que cada iteração era realizada, eram obtidos os valores de tempo de resposta e calculada a mediana.

No primeiro teste, foi fixada uma nota de corretagem pequena, ou seja, a carga que a aplicação receberia era baixa. Para cada iteração, foi obtida a mediana do tempo de resposta, obtendo-se o gráfico da figura 13.

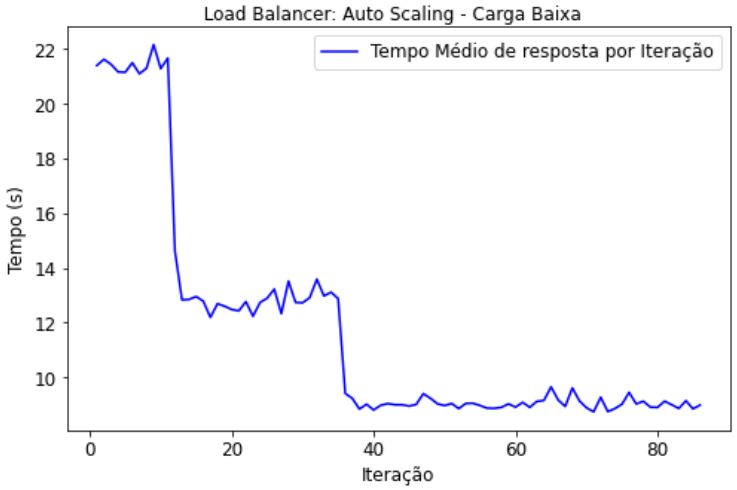


Figura 13: Load Balancer com Auto Scaling: Carga Baixa

A partir do gráfico da figura 13, é possível notar 3 patamares, que representam a criação de novas máquinas, isso aconteceu porque, nesses momentos, o uso total da CPU chegou em 50% e foram invocadas novas instâncias, diminuindo o tempo de resposta drasticamente.

Na segunda etapa do teste, foi fixada uma nota de corretagem maior, ou seja, a aplicação receberia uma carga maior, obtendo-se a figura 14.

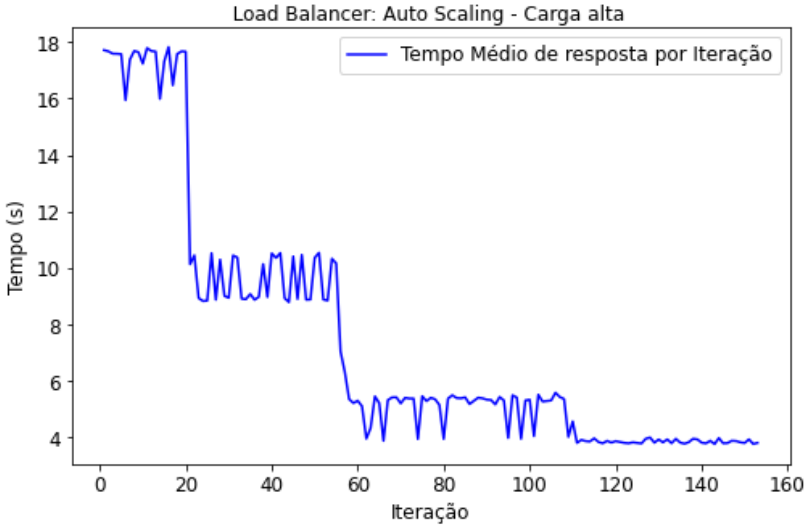


Figura 14: Load Balancer com Auto Scaling: Carga Alta

Observando o gráfico da figura 14, é possível observar um comportamento semelhante de quando foi testado com uma carga menor. Entretanto, apesar do comportamento semelhante, é possível ver que agora existe um patamar extra, isso aconteceu porque, como a carga é maior, se exigiu um consumo maior da CPU das máquinas, levando a criação de novas instâncias pelo grupo de autoscaling.

4.2.4 Análise de Preços

Da mesma forma que foi feita para as máquinas individuais do EC2, foi utilizado o simulador de preços da AWS.

Como o número de máquinas é variável, no grupo de autoscaling, será considerado todos os cenários, ou seja, considerando de uma a cinco máquinas em operação.

Além disso, será considerado que o Load Balancer processará 100 GB de dados por mês, o que é mais do que suficiente tendo em vista que o tamanho médio de uma nota de corretagem é de aproximadamente 300 KB, ou seja, processar 100 GB de notas de corretagem seria o equivalente a processar 200 mil notas por mês.

Portanto, considerando os parâmetros, a tabela 3 apresenta os custos obtidos.

Número de Máquinas	Máquinas - Custo Mensal	Load Balancer Custo Mensal	Custo Total Mensal
1	$21,75	$17,73	$39,48
2	$43,50	$17,73	$61,23
3	$65,25	$17,73	$82,98
4	$87,00	$17,73	$104,73
5	$108,75	$17,73	$126,48

Tabela 03: Billing EC2 + Load Balancer.

Ou seja, considerando o pior cenário, onde existe a necessidade do número máximo de máquinas, que nesse caso são 5, o gasto total seria de aproximadamente $127/mês.

4.2.5 Discussão

Baseando-se no que foi observado, é possível notar que a solução utilizando um load balancer melhorou significativamente os tempos de resposta médio para clientes simultâneos, isso devido ao load balancer redirecionar o tráfego de requisições para as diversas máquinas que serão geradas pelo grupo de auto scaling.

Ainda, é possível notar que essa solução é escalável, visto que o grupo de autoscaling cria e destrói as máquinas automaticamente e o load balancer distribui o tráfego para essas diferentes máquinas.

Considerando os custos, é visível que cada máquina operando 100% do mês tem um preço relativamente alto de $20/mês, o que indica que se o grupo de autoscaling não administrar bem a criação e destruição das máquinas e se não for definido um limite preciso para essa, os custos podem aumentar consideravelmente.

4.3 Utilizando Lambdas

No tópico 4.2, foi verificado que a utilização do Load Balancer fez com que a aplicação fosse escalável, entretanto, a criação e destruição das máquinas pelo grupo de autoscaling pode tornar os custos elevados em um curto intervalo de tempo.

Assim, continuando o estudo proposto, foi utilizado o AWS Lambda para criar diferentes esquemas de arquitetura, a fim de achar uma solução que leva em conta o balanço entre preço, escalabilidade e eficiência da aplicação.

4.3.1 Dockerização da Aplicação

O primeiro passo para a utilização do Lambda foi a criação de uma imagem da aplicação, isso foi necessário porque subir algumas dependências manualmente ao Lambda, além de ser trabalhoso, aumenta a margem de erro e diminui a replicabilidade da aplicação.

O processo foi simples, foi criado um Dockerfile, utilizando a imagem base do python para o lambda[14], além de serem adicionadas todas as dependências da aplicação.

Assim, preparado o dockerfile, foi criada uma imagem que foi enviada a um repositório do ECR no AWS.

Por fim, utilizando esta imagem, pode-se criar o Lambda diretamente, já que a validade da aplicação já tinha sido feita utilizando a imagem do Docker.

4.3.2 Utilizando um único Lambda

Como introduzido no início da seção 4.3, foram utilizadas algumas arquiteturas para a implementação do Lambda, começamos da maneira mais simples possível, adicionando toda aplicação em um único lambda, como apresentado na figura 15.

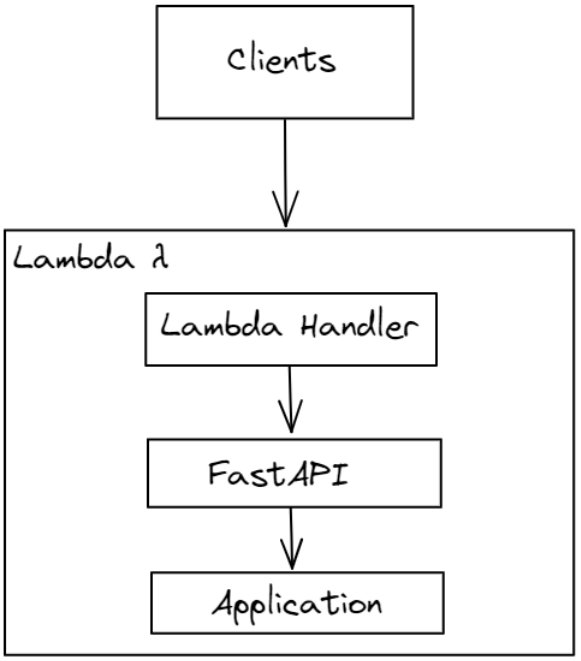


Figura 15: Arquitetura Lambda com toda aplicação.

Fazendo um teste de estresse, seguindo o mesmo procedimento das seções anteriores, variando o número de clientes simultâneos e obtendo o valor médio do tempo de resposta para 5 execuções, foi obtido o gráfico da figura 16.

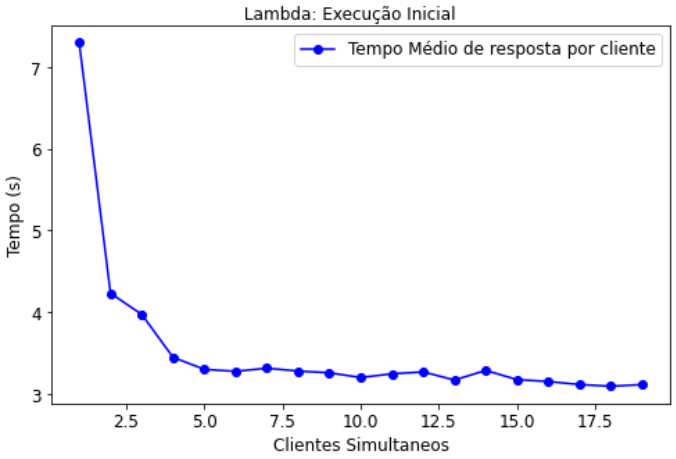


Figura 16: Teste de estresse inicial para o Lambda

No gráfico da figura 16, pode-se notar que o tempo começou elevado para um baixo número de clientes e foi diminuindo à medida que os clientes simultâneos foram aumentando. Apesar de o resultado parecer incorreto, vale lembrar que existe um tempo de inicialização do Lambda, onde o AWS faz o deploy da aplicação e inicializa os recursos a serem utilizados.

Para provar esse fato, foi deixado o Lambda rodando por alguns segundos e depois se iniciou a captura dos tempos médios, como foi feito anteriormente, obtendo-se o gráfico da figura 17.

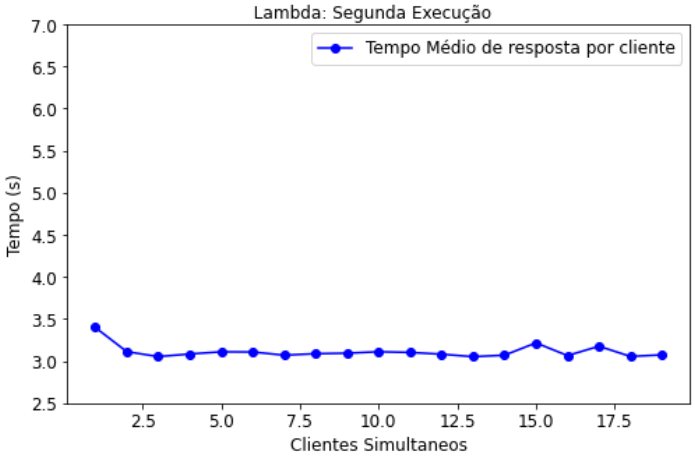


Figura 17: Segundo teste de estresse com o lambda

No gráfico da figura 17, pode-se notar que, diferentemente do gráfico 11, não existe o tempo de inicialização citado, o que comprova o que foi citado nos parágrafos anteriores.

Além disso, é possível notar que o tempo de resposta médio varia muito pouco, e que a simultaneidade de clientes não interfere no desempenho da aplicação, mostrando que existe uma auto escalabilidade do Lambda por parte do AWS.

4.3.3 Análise por páginas

Na seção anterior, foi mostrado que a solução utilizando um único lambda para toda a aplicação é escalável e atende muito bem a simultaneidade de clientes. Entretanto, ainda é necessário verificar o desempenho da aplicação para diferentes cargas de entrada.

Seguindo para análise, foi feito um teste de estresse, utilizando o lambda contendo toda aplicação, variando o número de páginas do PDF da nota de corretagem de entrada. O gráfico obtido pode ser encontrado na figura 18.

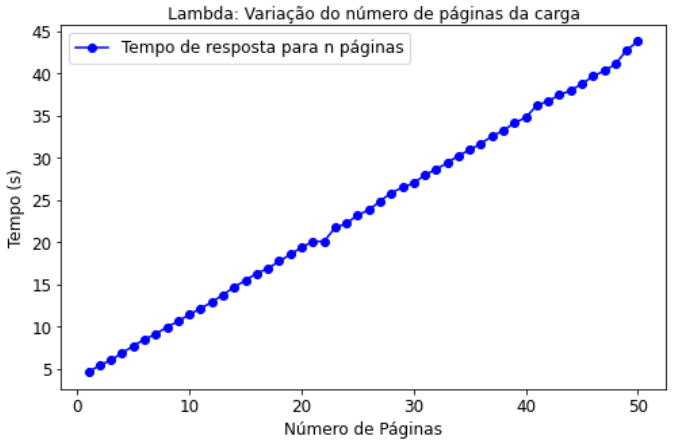


Figura 18: Variação da carga para o lambda com toda aplicação

No gráfico da figura 18, pode-se ver que o tempo de resposta varia com o número de páginas da carga, o que era esperado, já que, quanto mais páginas na nota de corretagem, mais ativos precisam ser extraídos e identificados, e portanto, mais processamento é necessário, aumentando o tempo de resposta.

Note, porém, que para uma carga com excessivas páginas, a aplicação pode demorar muito para responder, o que seria desagradavel para o cliente.

Uma forma de contornar esse problema seria criar dois serviços para a aplicação, um responsável por dividir o PDF da nota de corretagem em várias páginas, e outro serviço que seria especializado em extrair os dados de uma página do PDF. Assim, o primeiro serviço dividiria o PDF em suas *n* páginas e chamaria o outro serviço *n* vezes, conforme a seguinte arquitetura.

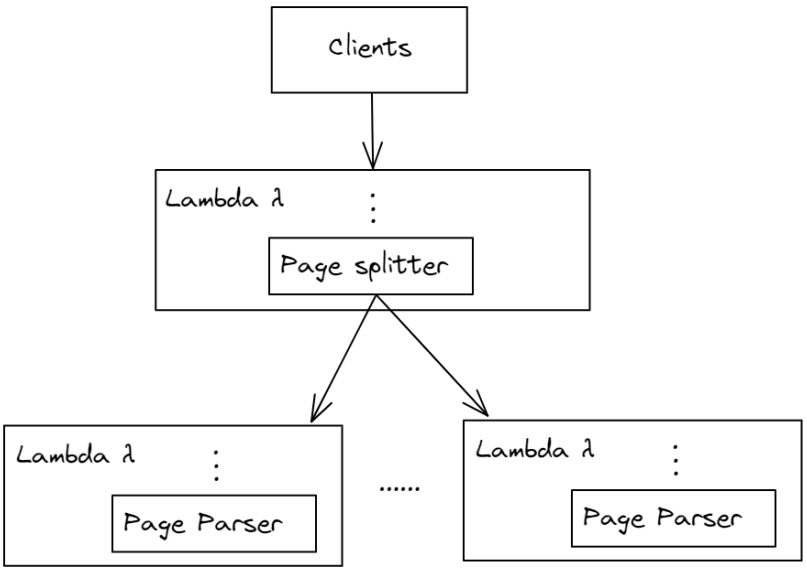


Figura 19: Arquitetura da aplicação com serviço divisor de páginas

Note que, na figura 19, para fins de exibição, foram omitidas algumas partes da arquitetura, como o FastAPI.

Aplicando o mesmo teste que foi feito anteriormente, isso é, foi variado o número de páginas do PDF da nota de corretagem de entrada, obtendo-se o gráfico da figura 20.

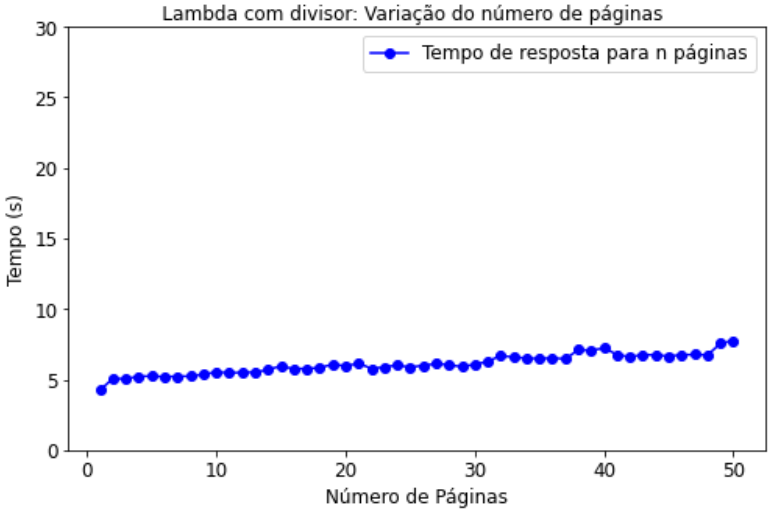


Figura 20: Variação da carga para o lambda com divisor de páginas

Para fins de análise, os gráficos referentes a figura 19 e 20 foram sobrepostos, obtendo o gráfico da figura 21.

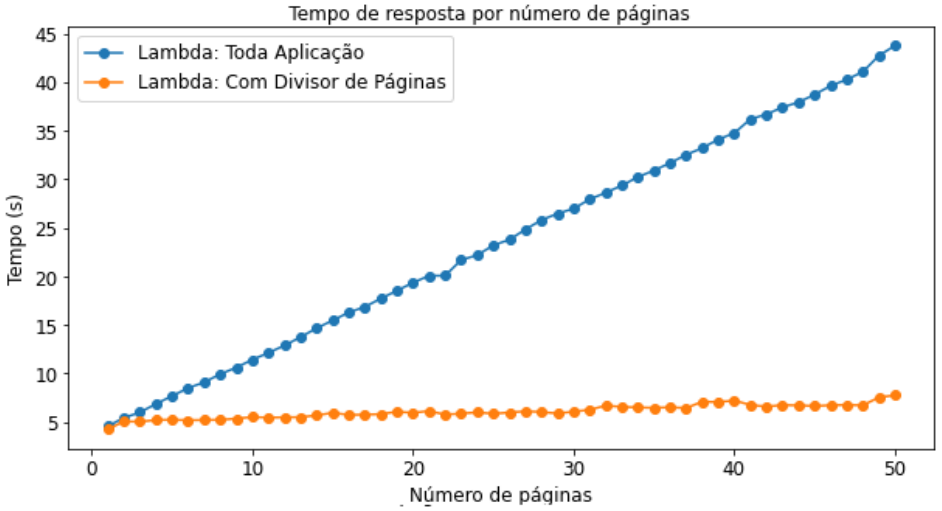


Figura 21: Comparação entre lambda com toda aplicação e com divisor de páginas

No gráfico da figura 21, é possível observar que a divisão da aplicação em dois serviços melhorou consideravelmente o desempenho da aplicação, mantendo o tempo de resposta praticamente constante, se tornando praticamente independente do número de páginas da carga.

4.3.4 Análise por Operação

Na seção 4.3.3, foi apresentado uma análise variando a carga de entrada, em que se criou um outro serviço para fazer uma análise por páginas, o que apresentou uma melhoria de desempenho.

Alternativamente, pode-se fazer uma análise por operação, que é uma fonte de processamento quando está se fazendo análise nas notas de corretagem. Mais especificamente, a ideia é criar um outro serviço, que é especializado em identificar informações do ativo de uma operação, assim, dada uma operação, ele é capaz de identificar seu ticker, tipo de ativo, mercado, entre outras características.

Portanto, existiria um serviço que extrairia as informações dos PDFs e outro serviço que seria responsável por fazer a análise das operações que foram extraídas, conforme a arquitetura simplificada da figura 22.

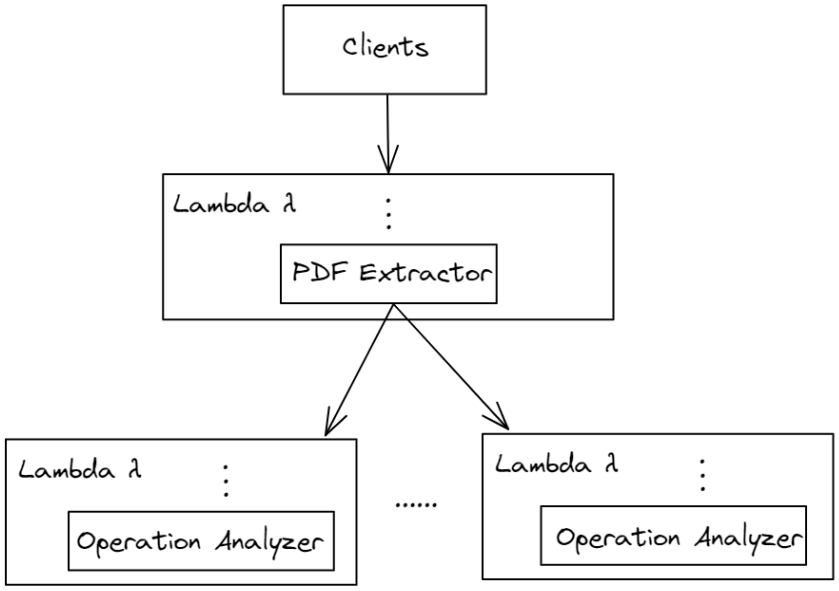


Figura 22: Arquitetura da aplicação com serviço identificador de ativos

Seguindo para análise, foi feito um teste de estresse, utilizando o lambda com toda a aplicação, e o sistema de lambdas apresentado por meio da figura 22. Nesse teste, foi variado o número de operações do PDF da nota de corretagem de entrada, em que se obteve os seguintes resultados:

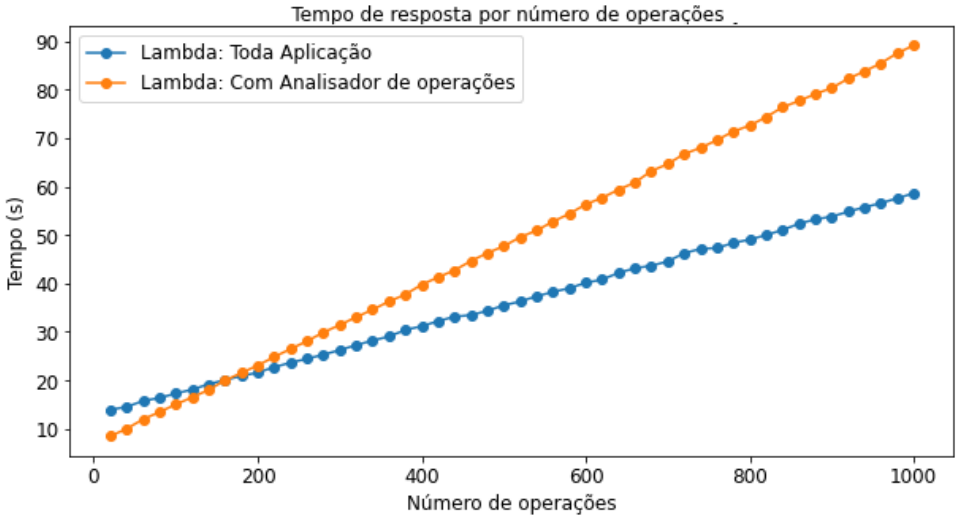


Figura 23: Comparação entre lambda com toda aplicação e lambda com identificador de ativos

Analisando o gráfico da figura 23, pode-se notar que o tempo de resposta médio piorou, para um número de operações maior que aproximadamente 200, em relação ao lambda com toda aplicação, isso acontece por causa de uma otimização que existe na aplicação original que foi perdida quando dividida em serviços.

Mais especificamente, durante a execução do código, quando é feita a identificação e análise dos ativos em uma operação, existe uma consulta em um banco de dados, que pode ser custosa se for feita para todos os ativos individualmente.

Então, quando é identificado um ativo de uma operação, a aplicação salva esse resultado em memória, e assim, durante o processo de análise, antes de se buscar no banco, é verificado primeiro se o resultado já está na memória, que é muito menos custoso para se acessar.

Vale lembrar que uma pessoa, em geral, não costuma ter um número absurdamente grande de tipos de ativos diferentes, o que inviabiliza um pior caso do código, em que o desempenho seria muito pior do que consultar o banco para cada operação individualmente.

Quando foi criado o outro serviço, como ele lida apenas com a identificação de uma operação, não tem como se aproveitar dessa otimização. Então, quanto maior o número de ativos, mais falta essa otimização vai fazer, o que justifica o melhor desempenho para um número baixo de operações e uma piora de desempenho para um número alto de operações.

Um fluxograma mostrando a lógica por trás dessa otimização pode ser representado pela seguinte imagem:

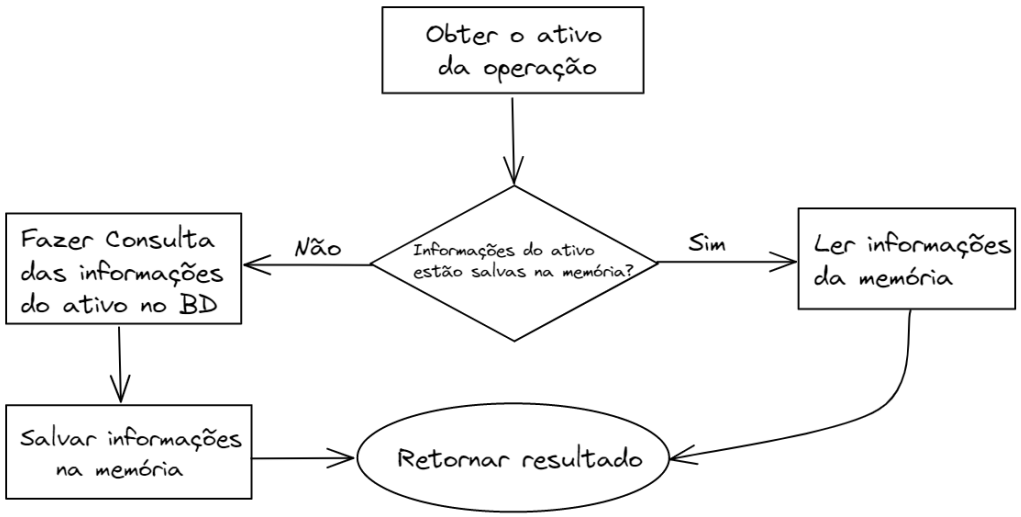


Figura 24: Fluxograma do processo de otimização

Uma maneira que pode melhorar o serviço identificador de ativos da operação é alterá-lo para aceitar uma lista de operações. Então, o serviço que extrai as informações da nota de corretagem, enviaria lotes de listas de operações que contém o mesmo ativo para o outro serviço.

Assim, como o serviço identificador de ativos seria invocado para diversas listas de ativos iguais, a otimização apresentada funcionaria, isso por que só seria necessário um acesso ao banco para cada invocação do serviço, que seria o mesmo número de acessos da versão otimizada para toda aplicação.

Sendo assim, foi criado um sistema de lambdas conforme descrito acima. Gerando o resultado da figura 25.

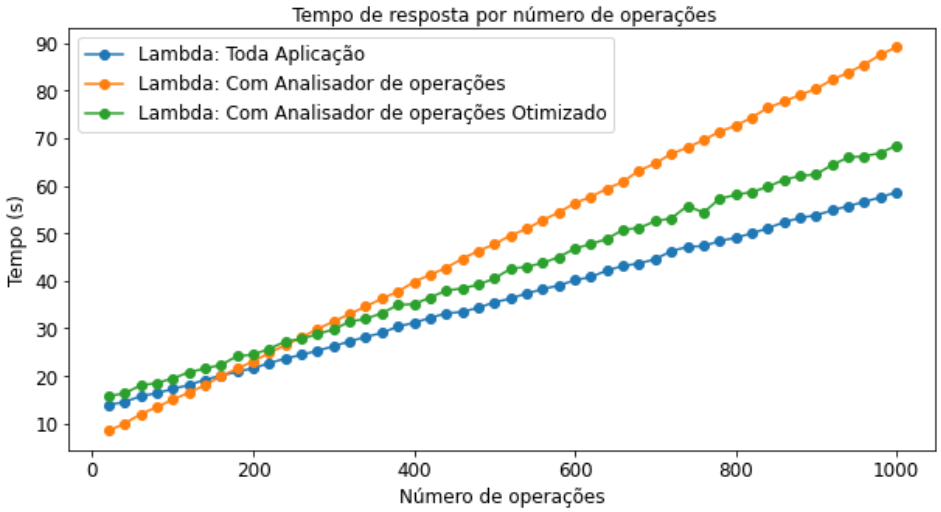


Figura 25: Comparação entre Lambdas

Analisando o gráfico da figura 25 nota-se que, a aplicação utilizando o serviço que analisa as operações com a otimização citada anteriormente, possui um tempo de resposta médio muito parecido com o lambda com toda a aplicação. O tempo era muito próximo para um baixo número de operações, aumentando gradativamente à medida que esse número ia subindo.

Isso acontece porque, como dito anteriormente, a versão utilizando o serviço identificador de ativos otimizado faz o mesmo número de acessos ao banco que a versão contendo toda a aplicação.

A pequena piora para casos com mais operações acontece porque, quanto maior número de operações, maior a quantidade de dados que os serviços devem trocar entre si, algo que não acontece na versão que contém apenas um serviço.

4.4.4 Análise de Preços

Da mesma forma que foi feito para os casos anteriores, foi utilizado o simulador de preços da AWS, para estimar os valores gastos.

Para todos os casos, será considerado um número de 1000 requisições por dia para o lambda do serviço principal e 1 GB de memória disponível. Além disso, será considerado que uma nota tem 20 páginas, 200 operações e 30 tipos de ativos, o que é um caso além da média, e será bom para analisar os cenários extremos.

Nos itens a, b e c, serão analisados os casos apresentados nas seções 4.3.1, 4.3.3 e 4.3.4

a. Lambda contendo toda aplicação: Nesse caso, considerando a nota com 20 páginas e 200 operações, proposta no parágrafo anterior, e analisando os gráficos das figuras 18 e 21, percebe-se que o tempo de resposta é aproximadamente 20s.

Ainda, considerando 1GB de memória disponível e considerando as 1000 execuções diárias, chega-se a um valor de aproximadamente $10 mensais, utilizando o avaliador de custos do AWS.

b. Lambda utilizando divisor de páginas: Nesse caso, existem dois serviços, um para a aplicação principal, e o outro, que é chamado para cada página da nota.

No primeiro serviço será considerado 1000 solicitações diárias, no segundo serão consideradas 1000 vezes o número de páginas, que nesse caso são 20, ou seja, 20000 vezes.

Para o tempo de requisição, analisando o gráfico da figura 20, nota-se que o tempo médio é de aproximadamente 5 segundos, considerando as 20 páginas propostas. Como as páginas são extraídas de maneira paralela, é uma boa aproximação considerar aproximadamente 5 segundos para os dois serviços.

Assim, considerando 1GB de memória disponível, as 1000 solicitações/dia de 5 segundos do primeiro serviço e as 20000 solicitações/dia de 5 segundos para o segundo serviço, obtém-se um valor de $2,5 mensais para o primeiro, e $50 mensais para o segundo, totalizando $52.

c. Lambda utilizando analisador de operações: Nessa análise, será considerado o serviço que analisa os ativos com a otimização citada na seção 4.3.4, o serviço sem otimização perdeu muito em desempenho e será desconsiderado.

Então, para o caso citado, existem dois serviços, o que extrai os dados do PDF, e o que analisa as operações extraídas pelo primeiro serviço. No primeiro serviço será considerado 1000 solicitações diárias, no segundo serão consideradas 1000 vezes o número de tipos de ativos, que nesse caso são 30, ou seja, 30000 vezes.

Para o tempo de requisição, analisando o gráfico da figura 25, nota-se que o tempo médio de resposta é de aproximadamente 25 segundos, considerando as 200 operações propostas. O tempo de extração dos ativos, como visto anteriormente, é de 20 segundos, restando sendo 5 segundos o tempo de análise, que é feita de maneira paralela.

Assim, considerando 1GB de memória disponível, as 1000 solicitações/dia de 25 segundos do primeiro serviço e as 30000 solicitações/dia de 5 segundos para o segundo serviço, obtém-se um valor de $12,51 mensais para o primeiro, e $75,18 mensais para o segundo, totalizando $87,69.

A Tabela 4 apresenta um resumo dos resultados obtidos.

Caso	N. Chamadas (Serviço 1)	N. Chamadas (Serviço 2)	Duração Solicitação (Serviço 1)	Duração Solicitação (Serviço 2)	Custo Total
Toda Aplicação	30.000/mês	-	20s	-	$10,00
Divisor de Páginas	30.000/mês	600.000/mês	5s	5s	$52,00
Analisador de Ativos	30.000/mês	900.000/mês	25s	5s	$87,69

Tabela 04: Billing AWS Lambda.

.

4.4.5 Discussão

É possível notar, a partir dos resultados obtidos, que a solução utilizando um único lambda melhorou significativamente os tempos de resposta médio para clientes simultâneos, isso devido ao AWS Lambda resolver simultaneidade automaticamente, não se limitando ao número de núcleos da máquina.

Além disso, as análises baseadas nas cargas mostraram uma melhora significativa em desempenho ao criar uma arquitetura de serviços que faz a análise das páginas de maneira concorrente.

Em contrapartida, tornar a análise dos ativos de forma concorrente, também através de um novo serviço, não mostrou melhora de desempenho, o que demonstra que a análise inicial do PDF é o processo mais significativo em relação ao desempenho do código.

Considerando os custos, percebe-se que a melhor opção, em relação ao custo e ao desempenho, é utilizando o divisor de páginas, que possui um desempenho muito alto e um preço mediano em relação às opções apresentadas.

Assim, se considerar o balanço entre preço, escalabilidade e desempenho, percebe-se que a utilização do AWS Lambda é uma opção viável, e que, no caso apresentado, a arquitetura envolvendo a divisão de páginas foi a que se saiu melhor.

5. Conclusão

Neste projeto, avaliamos o desempenho e custo do uso de serviços de nuvem computacional para extração de informações de notas de corretagem de arquivos Portable Document Format (PDF). Realizamos experimentos para avaliar diferentes implementações, buscando um balanço entre escalabilidade, desempenho e preços.

Seguindo a proposta inicial, procurou-se começar da forma mais simples, escalando gradativamente a arquitetura do projeto, bem como a aplicação dos serviços do AWS e o algoritmos da aplicação principal. Assim, iniciou-se utilizando uma máquina do EC2, onde notou-se que uma máquina, apesar de levar a aplicação a nuvem, não garantia escalabilidade e desempenho. Como os clientes simultâneos dividem o poder de processamento da máquina, a aplicação perder desempenho, o que gera um maior tempo de resposta às requisições aos clientes.

Para resolver este problema, decidiu-se utilizar um grupo de autoscaling associado a um load balancer. Isso garantiu que a aplicação se tornasse escalável e com bom desempenho, isso porque o grupo de autoscaling criava novas máquinas quando necessário, enquanto o load balancer redirecionava o tráfego para as máquinas criadas. Entretanto, apesar de ganharem em desempenho e escalabilidade, os preços podem aumentar rapidamente, já que a aplicação demanda bastante CPU e, em um caso com vários clientes simultâneos, levaria ao grupo de autoscaling a criar muitas máquinas.

Em seguida, para tentar melhorar o desempenho e obter um custo melhor, optou-se por utilizar o lambda, onde foram analisadas diferentes arquiteturas, levando em conta os mais variados tipos de carga. Chegou-se à conclusão de que, dividindo a arquitetura em dois serviços, foi possível obter um desempenho excelente, onde o tamanho da carga não interfere significativamente no tempo de resposta; uma escalabilidade muito boa, que é feita pelo próprio AWS ao usar o lambda; e um preço significativamente bom, já que o lambda é precificado por requisições.

Referências:

[1]Mendes, Enrico Rondinelli de Oliveira. A evolução da B3 e fatores que motivaram o aumento recente no número de investidores pessoas físicas na Bolsa de Valores, 2021

[2] Relação de pessoas físicas na Bolsa de valores brasileira. Perfil pessoa física | B3

[3] Container, Amazon AWS, 2022. Disponível em: https://aws.amazon.com/pt/containers/

[4] Docker, Inc. Docker. Disponível em: https://www. docker. com/what-docker, 2020

[5]Erl, Thomas; Puttini, Ricardo; Mahmood, Zaigham. Cloud Computing: Concepts, Technology & Architecture**.** Prentice Hall: New Jersey, 2013.

[6]Wittig, Andreas; Wittig, Michael. Amazon Web Services in Action, Second Edition. New York: Manning Publications, 2018

[7] Amazon S3, Amazon AWS, 2022. Disponível em: https://aws.amazon.com/pt/s3/

[8] Sbarski, Peter, and Sam Kroonenburg. Serverless architectures on AWS: with examples using Aws Lambda. Simon and Schuster, 2017.

[9] Fast API. Tiangolo, 2018. Disponível em: https://fastapi.tiangolo.com/

[10] Sousa, Ivan. Afinal, o que é JSON e para que ele serve? Rock Content, 2020. Disponível em: https://rockcontent.com/br/blog/json

[11] Nota de corretagem: como entender e interpretar. Leoa Blog, 2022. Disponível: https://www.leoa.com.br/blog/nota-de-corretagem

[12] Deployment. Uvicorn, 2020. Disponível em: https://www.uvicorn.org/deployment/

[13] AWS Pricing Calculator. Amazon AWS, 2022. Disponível em: https://calculator.aws/

[14] Amazon ECR Public Gallery: lambda/python. Amazon AWS, 2020. Disponível em: https://gallery.ecr.aws/lambda/python/