PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA E DE ARTES
CIÊNCIAS DA COMPUTAÇÃO

ARQUITETURA *SERVERLESS*

RONALDO FERNANDES NOGUEIRA FILHO

GOIÂNIA

2023

RONALDO FERNANDES NOGUEIRA FILHO

ARQUITETURA *SERVERLESS*

Trabalho de Conclusão de Curso apresentado à Escola Politécnica, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Ciências da Computação. Orientadora: Profa. Dra. Solange Da Silva

GOIÂNIA

2023

RONALDO FERNANDES NOGUEIRA FILHO

ARQUITETURA SERVERLESS

Este Trabalho de Conclusão de Curso julgado adequado para obtenção o título de Bacharel em Ciências da Computação, e aprovado em sua forma final pela Escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, em 15/06/2023.
.

Banca Examinadora:

Orientadora: Profa. Dra. Solange da Silva

Prof. Me. Aníbal dos Santos Jukemura

Professor Dr. Vicente Paulo de Camargo

GOIÂNIA

2023

AGRADECIMENTOS

Agradeço meus pais por sempre me dar apoio.

Agradeço à minha orientadora Solange da Silva, pois durante toda a construção do meu trabalho de conclusão de curso (TCC), teve paciência e deu todo apoio possível.

Agradeço meu chefe Tiago Jorge, pelo apoio e me ensinar todos os dias sobre nuvem e a área de tecnologia da informação.

RESUMO

Os objetivos deste trabalho foram de pesquisar e escrever sobre a computação em nuvem e arquitetura serverless e usar esses conhecimentos novos para construir uma aplicação através desta arquitetura *serverless*, utilizando um provedor de computação em nuvem. Segundo os procedimentos técnicos esta é uma pesquisa bibliográfica e experimental. Como resultado foi construído o CloudBot, ou seja, um chatbot baseado em arquitetura serverless, hospedado na nuvem AWS. Ele utiliza os serviços AWS Lambda e Amazon API Gateway. O CloudBot, desenvolvido cumpriu os objetivos propostos, apresentando um desempenho adequado, escalabilidade eficiente, confiabilidade na detecção e resolução de erros, além de implementar medidas de segurança para proteção dos dados. Com isso, o CloudBot se mostra como uma solução viável para a criação de aplicativos e pode ser aplicado em diferentes cenários. Este estudo permitiu concluir que a arquitetura serverless é, de fato, uma opção válida e eficiente para a construção de aplicativos.

Palavras Chaves: *Serverless*. Computação em nuvem. Amazon Web Services (AWS). Nuvem. Chatbot.

ABSTRACT

The objective of this work is to research and write about cloud computing and serverless architecture and use this newfound knowledge to build an application with serverless architecture using a cloud computing provider. According to technical procedures, this is a bibliographic and experimental research. As a result, the CloudBot was built, which is a chatbot based on serverless architecture, hosted on the AWS cloud. It utilizes AWS Lambda and Amazon API Gateway services. The developed CloudBot has fulfilled the proposed objectives by demonstrating adequate performance, efficient scalability, reliability in error detection and resolution, as well as implementing security measures for data protection. Therefore, the CloudBot proves to be a viable solution for application development and can be applied in different scenarios. This study has led to the conclusion that serverless architecture is indeed a valid and efficient option for building applications.

Key Words: Serverless. Cloud Computing. Amazon Web Services (AWS). Cloud. Chatbot

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API	*Application Programming Interface* ou Interface de Programação de Aplicativ
AWS	*Amazon Web Services*
BD	Banco de Dados
IA	Inteligência Artificial
IaaS	*Infrastructure as a Service*
IP	*Internet Protocol*
NoSQL	Não Relacional
PaaS	*Platform as a Service*
SaaS	*Software as a Service*
SOAP	*Simple Object Access Protocol*
REST	*Representacional State Transfer*
RPC	*Remote Procedure Calls*
TI	Tecnologia de Informação
DNS	Domain Name System

SUMÁRIO

SUMÁRIO

1 INTRODUÇÃO

Desde sua criação, a Internet vem tendo grandes avanços tecnológicos. Surgem novos recursos para refinar e fortalecer cada vez mais determinados serviços para as empresas e consumidores. Uma dessas tecnologias que surgiu nos últimos anos seria a computação em nuvem. Nos últimos anos, a computação em nuvem apresentou um avanço considerável. Isso foi provocado devido a questões de segurança, economia, confiabilidade, privacidade, sustentabilidade e acesso por meio da rede. Ao repartir os recursos de Tecnologia de Informação (TI) com os clientes e aprimorar a utilização desses recursos, a computação em nuvem contribui com um crescimento sustentável para as empresas (SALATI et al., 2020).

Computação em nuvem possui uma ampla quantidade de soluções tecnológicas, suportadas por três modelos de implantação: *Software as a Service* (SaaS), *Platform as a Service* (PaaS) e *Infrastructure as a Service* (IaaS), para a indústria, instituições, organizações ou até mesmo *startups*. Além disso, atende a centenas de milhões de usuários individuais. Esses modelos facilitam o acesso sob demanda, na modalidade pública ou privada, gratuita ou paga. Todos esses recursos são fornecidos sem nenhum serviço instalado no ambiente local do usuário, eliminando o custo e esforço para instalação, manutenção e atualização dos serviços (PAZ; SALGADO & DIAZ, 2017).

Uma consequência da criação da computação em nuvem foi a arquitetura *serverless*. A arquitetura *serverless* é um modelo de desenvolvimento nativo em nuvem para a criação e execução de aplicações. Nesse modelo, os servidores são provisionados, mantidos e escalados pelo provedor de nuvem. O desenvolvedor apenas precisa se preocupar em empacotar o código em *containers* para fazer a implantação. Na computação em nuvem padrão, baseada em IaaS, o usuário precisa aumentar a capacidade do servidor nos momentos de alta demanda e diminuí-la quando a capacidade alta não é mais necessária. As aplicações são iniciadas apenas quando necessárias, de forma dinâmica, assim os usuários só pagam pelo tempo de execução (REDHAT, 2017).

Justifica estudar este assunto porque as empresas buscam alternativas para serem mais competitivas e sobreviverem no mundo globalizado. O tempo, custo e manutenção associados ao gerenciar sua própria infraestrutura podem ser exponencialmente mais altos do que um ambiente na nuvem. A computação em nuvem

tem foco em agilidade e desempenho, permitindo que as empresas tenham um controle maior das suas operações, inserindo neste controle o diferencial da mobilidade, fazendo com que os gestores acompanhem o andamento das demandas e atuem em tempo real. A utilização da computação em nuvem possibilita o acesso a recursos de computação praticamente ilimitados, permitindo que seus usuários atinjam novos patamares de eficiência, produtividade e qualidade (PAZ & LOOS, 2020).

Diante deste contexto, esse projeto visa responder a seguinte questão de pesquisa: **- A arquitetura serverless é um padrão válido para criar aplicativos?**

O objetivo geral é:

- Desenvolver um aplicativo utilizando serviços da plataforma AWS com uma arquitetura serverless

Os objetivos específicos são:

- Identificar e descrever como os serviços em nuvem podem facilitar o desenvolvimento de projetos.
- Aprofundar o conhecimento da arquitetura serverless
- Criar aplicações ou *Application Programming Interface* (API) dentro da plataforma AWS

A plataforma Amazon Web Services (AWS, 2023) foi escolhida para ser utilizada pois é uma provedora líder em serviços de nuvem que oferece suporte para a construção de projetos com arquitetura serverless. Com a AWS, é possível desenvolver aplicações sem a necessidade de gerenciar servidores, garantindo escalabilidade automática, redução de custos operacionais e flexibilidade para focar no desenvolvimento da lógica de negócio, proporcionando uma solução escalável e ágil para atender às demandas do mercado e oferecer uma experiência excepcional aos usuários.

Espera-se que os resultados deste trabalho possam contribuir:

- Com informações importantes para os desenvolvedores não utilizando arquitetura *serverless*;
- Com a disseminação do uso da computação em nuvem;
- Trazendo informações sobre os serviços da plataforma AWS, para os desenvolvedores;

Esta pesquisa, segundo sua natureza, é um resumo de assunto. Segundo, seus objetivos é uma pesquisa exploratória. Quanto aos seus procedimentos técnicos é uma pesquisa bibliográfica e experimental.

Esta pesquisa está estruturada da seguinte maneira: neste capítulo é apresentado o contexto do trabalho, a questão de pesquisa, objetivo e resultados esperados. O capítulo 2 traz o referencial teórico com conceitos, definições e trabalhos relacionados com o tema. No capítulo 3 é descrito o método, mostrando como o trabalho será desenvolvido e o que será para atingir objetivo geral. O Capítulo 4 é dedicado à apresentação dos serviços utilizados na construção da API com arquitetura serverless em um ambiente em nuvem, bem como aos testes realizados para validar o funcionamento e desempenho do sistema. No Capítulo 5 são apresentadas informações sobre o funcionamento da API, abordando sua arquitetura, fluxo de dados e interação com os usuários. O Capítulo 6 traz os resultados dos testes realizados, fornecendo uma análise aprofundada do desempenho, escalabilidade, confiabilidade e segurança do sistema, com base nos dados coletados durante os experimentos. Finalmente, o capítulo 7 apresenta as consideracoes finais e sugestão de trabalho futuro.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados alguns conceitos e definições, além de alguns trabalhos relacionados ao tema.

2.1 Computação em Nuvem

A computação em nuvem é a entrega de recursos de TI sob demanda por meio da Internet. Esses recursos costumam ter as seguintes características: conveniência e sob demanda para um conjunto de recursos de computação compartilhados e configuráveis. Estes recursos podem ser redes, servidores, armazenamento, aplicativos e serviços que podem ser rapidamente provisionados e liberados, com o mínimo de esforço de gerenciamento ou a interação do provedor de serviços (PAZ & LOOS, 2020).

Com *datacenters* localizados por todo o mundo, a computação em nuvem tornou-se flexível e acessível de acordo com a necessidade dos usuários (PAZ & LOOS, 2020).

2.2 AWS

O aplicativo foi desenvolvido dentro da plataforma da AWS, que é uma plataforma de computação em nuvem abrangente e em evolução fornecida pela Amazon que inclui uma mistura de IaaS, PaaS e SaaS. Os serviços da AWS podem oferecer ferramentas de organização, como poder de computação, armazenamento de banco de dados e serviços de entrega de conteúdo (AWS, 2022).

2.2.1 Como surgiu a AWS?

A AWS lançou seus primeiros serviços da web em 2002 a partir da infraestrutura interna que a Amazon construiu para lidar com suas operações de varejo online. Em 2006 começou a oferecer os serviços IaaS a seus usuários. A AWS foi uma das primeiras empresas a introduzir um modelo de computação em nuvem com pagamento conforme o uso, que pode ser dimensionado, para fornecer aos usuários computação, armazenamento ou taxa de transferência conforme necessário (AWS, 2020).

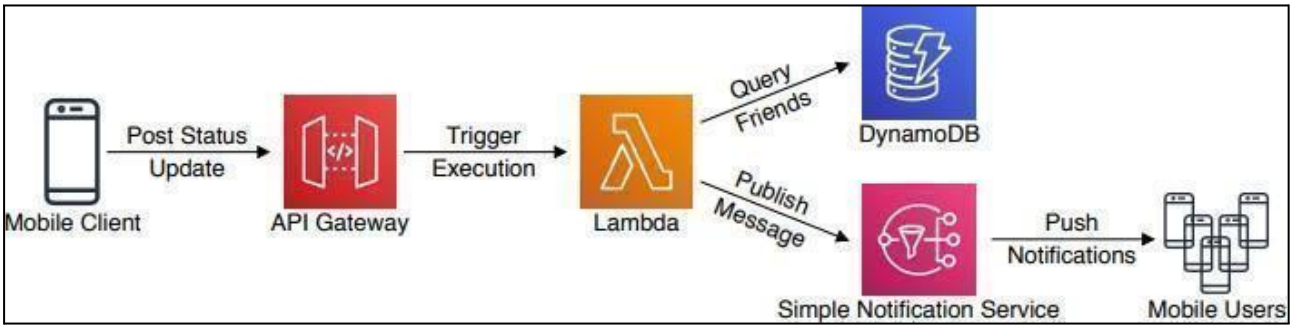
2.2.2 O que a AWS faz?

A AWS oferece muitas ferramentas e soluções diferentes para empresas e desenvolvedores de software que podem ser usadas em *datacenters* em até 245 países. Grupos como agências governamentais, instituições de ensino, organizações sem fins lucrativos e privadas podem usar os serviços da AWS. Mais de 200 serviços compõem o portfólio da AWS, incluindo aqueles para computação, bancos de dados, gerenciamento de infraestrutura, segurança e desenvolvimento de aplicativos, conforme mostrado na Figura 1 (AWS, 2022).

Na Figura 1 existem quatro (4) serviços da AWS que formam uma arquitetura serverless:

- **API Gateway** que foi utilizado para criar uma API.
- O **Lambda** ta sendo seu poder computacional, executando o código toda vez que alguém bater na URL da API.
- Dados são armazenados e buscados dentro do **DynamoDB**, um banco de dados NoSQL.
- ***Simple Notification Service*** que envia notificação para os usuários toda vez que o código termina de ser executado.

Figura 1 - Arquitetura de um aplicativo criado completamente usando serviços da AWS



Fonte: EISMAN et al., 2020.

2.3 Comparação do *On-premise* e a nuvem

O software *On-Premise* é instalado localmente nos computadores e servidores

de uma empresa. Na prática, ele representa um datacenter local com toda a infraestrutura e sistemas instalados utilizados pelo negócio. Nesse modelo de ambiente, os sistemas são instalados nos servidores locais e acessados pelos usuários por meio da rede da empresa. Outra característica desse modelo é que toda a manutenção do ambiente, tanto do hardware quanto do software, é de responsabilidade da empresa (HAYEK & ODEH, 2020).

Em geral, os sistemas *On-Premise* são muito mais fáceis de modificar. A capacidade de personalizar de acordo com suas necessidades e requisitos específicos é fundamental para muitas organizações, especialmente em setores de nicho, como fabricantes especializados com processos exclusivos. Isso também deixa possível que você é responsável por todo aspecto da segurança dos aplicativos, um grande fator quando está escolhendo o ambiente para construir um aplicativo (HALE & COX, 2020).

A desvantagem maior dos ambientes *On-premise* é que os custos associados ao gerenciamento e à manutenção de todas as implicações da solução são mais altos do que em nuvem. Uma configuração *On-Premise* requer hardware de servidor interno, licenças de software, recursos de integração e funcionários de TI disponíveis para dar suporte e gerenciar possíveis problemas que possam surgir. Isso nem leva em consideração a quantidade de manutenção pela qual uma empresa é responsável quando algo quebra ou não funciona (CLEO, 2020).

Um ambiente baseado em nuvem utiliza tecnologia virtual para hospedar os aplicativos de uma empresa fora do local. Isso já elimina a necessidade de gastar dinheiro em *hardware* como servidores, O único gasto vai ser com os serviços de nuvem (CLEO, 2020).

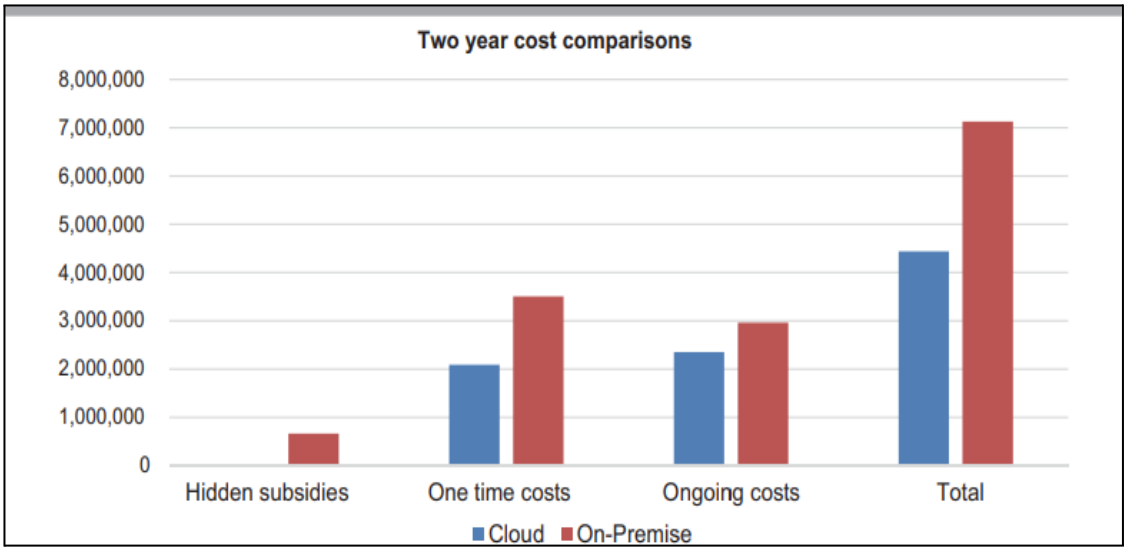
Além disso, a computação em nuvem apresenta provisionamento quase instantâneo porque tudo já está configurado. Assim, qualquer novo software integrado ao seu ambiente está pronto para uso imediatamente após a assinatura da empresa. Com o provisionamento instantâneo, o tempo gasto na instalação e configuração é eliminado e os usuários podem acessar o aplicativo imediatamente (CLEO, 2020).

Como todas as tecnologias, a computação em nuvem também tem suas desvantagens. Um de seus maiores desafios é que necessita de uma conexão com a Internet para funcionar. Sem uma conexão talvez não seja possível acessar seu aplicativo e informações. Outro aspecto contra a computação em nuvem é a perda de controle sobre a infraestrutura, disponibilidade de serviço e complexidade de integração. Como tudo é gerenciado pelo provedor, no caso que alguma coisa não satisfaz os requisitos do aplicativo que está construindo, vai ter de buscar por soluções

em outros lugares (GOOGLE CLOUD, 2022).

Na Figura 2 está ilustrado uma comparação entre um ambiente *On-Premise* versus ambiente em Nuvem, durante o período de 2 anos em uma mesma empresa. Nos subsídios ocultos tem gasto de quase em 1 milhão com o ambiente *On-Premise,* enquanto no ambiente de nuvem não tem nenhum. No custo único, o ambiente em nuvem salvou 1 milhão e meio a mais. Nos custos contínuos, o ambiente *On-Premise* tem quase um milhão em despesa a mais.

Figura 2 - ambiente *On-Premise* Vs ambiente Nuvem



Fonte: JOHNSON et al., 2019.

2.4 Arquitetura Serverless

As Soluções de computação *serverless* costumam ser divididas em *Back-end as a Service* (BaaS) e *Function as a Service* (FaaS). Os desenvolvedores têm acesso a vários serviços e aplicações de terceiros com o BaaS. As funções *serverless* para essas aplicações são normalmente chamadas por meio de interfaces de programação de aplicações ou API (REDHAT, 2017).

Já com o FaaS, os desenvolvedores ainda gravam uma lógica personalizada no lado do servidor, mas ela é executada em containers totalmente gerenciados por um provedor de serviços de nuvem. Todos os principais provedores oferecem pelo menos uma solução FaaS. Eles incluem a *Amazon Web Services* (AWS) com o AWS Lambda,

o *Microsoft Azure* com o *Azure Functions*, o *Google Cloud* com várias opções, o *IBM Cloud* com o *IBM Cloud Functions* e muitos outros (REDHAT, 2017).

2.5 API

Uma API é um mecanismo que possibilita que dois componentes de software se comuniquem usando um conjunto de definições e protocolos. A arquitetura da API é definida como cliente e servidor. A parte que envia a solicitação é chamada de cliente, enquanto a que envia resposta é chamada de servidor (AWS, 2022).

Existem quatro tipos de APIs; *Simple Access Protocol* (SOAP) ou Protocolo de Acesso a Objetos Simples, *Remote Procedure Calls* (RPC) ou Chamadas de Procedimento Remoto, *WebSocket* e *Representacional State Transfer* (REST) ou Transferência Representacional de Estado. As APIs REST são as mais utilizadas devido a sua flexibilidade. O cliente simplesmente envia sua solicitação ao servidor como dados. O servidor usa essa entrada para iniciar funções internas e retorna os dados (AWS, 2022).

2.6 Trabalhos Relacionados

Paz e Loos (2020) tiveram como objetivo apresentar a importância da computação em nuvem para a Indústria 4.0. Mostraram que a indústria atual está passando por uma transformação e que a computação em nuvem é um dos seus grandes pilares. Além disso, com a utilização correta de suas ferramentas as unidades fabris tornam-se mais ágeis e adaptáveis às variações da produção, evitando perdas e retrabalho.

Ao final de sua análise concluíram que inúmeras grandes empresas já estão utilizando a computação em nuvem para automatizar, acelerar seus processos e com uma excelente capacidade de aumentar a escalabilidade com seus recursos sob demanda. Além disso, observaram que o potencial da computação em nuvem pode ser alcançado através da integração das plantas fabris com as plataformas de TI na nuvem. A tecnologia da nuvem é um elemento chave da próxima revolução industrial, proporcionando às empresas os meios para inovar em torno dessas tecnologias. Com a computação em nuvem as empresas de médio e grande porte tornam-se competitivas, através da inovação.

Salati et al. (2020) tiveram o objetivo de analisar a satisfação, lealdade e

intenção de uso de fornecedores de vídeo que usavam a computação em nuvem. Como a Netflix é um dos maiores provedores de *streaming,* a pesquisa foi realizada baseada nele, para verificar o quanto a computação em nuvem afetou seu crescimento. Conseguiram apresentar muitas informações sobre a utilização da computação em nuvem pela Netflix. Como o preço era justo acabou tendo uma satisfação alta e também ajudou muito na parte da lealdade. A parte de intenção de uso teve um alto índice, devido a facilidade de uso e diversão de seus clientes. Com isso, concluíram que a computação em nuvem teve um impacto positivo para a Netflix.

Hayek e Odeh (2020) tiveram o objetivo de encontrar a melhor forma de desenvolver um *Enterprise Resource Planning* (ERP), comparando-o com o On-Premise ERP. Analisaram o desenvolvimento no ambiente On-premise e em nuvem, para que pudessem encontrar qual forma seria melhor para as empresas serem mais competitivas no mercado. Mostraram que as ERP utilizando nuvem tiveram mais vantagens comparadas às ERP utilizando a *On-Premise*. Em termos de custo benefício, implementação, escalabilidade e manutenção todos mostraram mais benefício em um ambiente de nuvem. O único aspecto que o *On-Premise* teve de superior a de nuvem foi o de sua customização, pois o ambiente de nuvem é limitado pelas opções oferecidas por seu provedor. Concluíram que as empresas devem analisar alguns aspectos antes de escolher o ambiente. Apesar da nuvem ser geralmente mais preparada para lidar com um ERP, pode ser que o controle e customização de um ambiente *On-Premise* seja mais adequado.

Conforme Eisman et al. (2020), apesar da computação *serverless* demonstrar um bom índice de usabilidade e eficiência, ainda existem reportagens que conflitam com um ou outro, sobre a viabilidade do *serverless*. Seu trabalho buscou responder as seguintes três perguntas: Porque tantas empresas adotam o serverless? Quando aplicações *serverless* são adequadas? Como aplicações *serverless* são aplicadas corretamente?

O estudo destes autores perimitiu responderem que o motivo mais comum relatado para a adoção de *serverless* é para economizar custos para cargas de trabalho irregulares ou intermitentes, para evitar preocupações operacionais e para a escalabilidade integrada. Os aplicativos *serverless* são mais comumente usados para tarefas de curta duração, com baixo volume de dados e cargas de trabalho em rajadas, mas também são frequentemente usados para latência crítica, alto volume de funcionalidade principal. Os aplicativos *serverless* são implementados principalmente na AWS, em Python ou JavaScript e fazem uso intenso de BaaS. Concluíram que esse

catálogo de aplicativos *serverless* pode estimular uma nova onda de aplicativos *serverless,* que facilitam ajustes significativos de desempenho e, em geral, são úteis tanto no trabalho quanto nas escolas.

3 PROCEDIMENTOS METODOLÓGICOS

Esta pesquisa, segundo sua natureza, é um resumo de assunto, buscando explicar a área do conhecimento do projeto, indicando sua evolução histórica, como resultado da investigação das informações obtidas, levando ao entendimento de suas causas e explicações (WAZLAWICK, 2014).

Segundo os objetivos é uma pesquisa exploratória. A pesquisa exploratória muitas vezes é considerada como a primeira parte do processo de pesquisa, porque não necessariamente o autor tem um objetivo ou hipótese definida (WAZLAWICK, 2014). Essa pesquisa tem como objetivo a maior familiaridade do autor com o problema, tornando mais explícito ou facilitar a construção de hipóteses. Geralmente é uma pesquisa flexível, porque considera os variados aspectos referentes aos fatos ou fenômenos (GIL, 2017).

Quanto aos procedimentos técnicos, é uma pesquisa bibliográfica e experimental. A pesquisa bibliográfica requer o estudo de teses, artigos, entre outros. A pesquisa experimental é caracterizada por ter uma ou mais variáveis experimentais que podem ser coordenadas pelo pesquisador (WAZLAWICK, 2014).

A pesquisa bibliográfica foi elaborada a partir de materiais já publicados, podendo incluir livros, teses, materiais disponibilizados na Internet, revistas, entre outros. A principal vantagem é permitir uma sucessão de fenômenos maior do que seria capaz de pesquisar diretamente (GIL, 2017).

A pesquisa experimental consiste em estabelecer um objeto de estudo, escolher as variáveis que a influenciam e determinam as formas de controle e observar os efeitos que a variável gera no objeto. Realiza pelo menos um dos elementos que julga ser responsável pela circunstância que está sendo pesquisado (GIL, 2017).

A pesquisa experimental é composta das seguintes etapas, conforme Gil (2017):

a) Formulação do problema: **A arquitetura serverless é um padrão válido para criar aplicativos?**

b) Definição do plano experimental: foi construída uma API utilizando vários serviços da plataforma AWS em conjunção com um ou outro. Essa API foi criada dentro do AWS API *Gateway* e executada com o potencial computacional do AWS *Lambda*. A API guarda as informações dentro de um banco de dados (BD) não relacional (NoSQL), chamado DynamoDB.

c) Determinação do ambiente: Os testes da API foram realizados dentro da plataforma de API Postman, versão 10.2.2. O computador utilizado foi um notebook Dell, com Sistema Operacional (SO) Windows, com 3,5 GigaHertz (GHz), 8 GigaByte (GB) de Memória de Acesso Randômico (RAM) e com processador AMD Ryzen 5.

d) Coleta de dados: foram realizados testes no *postman* e com serviços da AWS como o AWS *Cloudwatch* e AWS *EventBridge* para verificar a segurança, confiabilidade e escalabilidade de uma API na nuvem para determinar se é viável a construção de aplicações completamente dentro da nuvem. Detalhes dos testes estão no capítulo 4.

e) Análise e interpretação dos dados: foram identificados e explicados os possíveis desafios encontrados na criação de uma API na nuvem. Foram analisados os resultados obtidos de acordo com os testes, realizando a comparação com trabalhos relacionados e a pesquisa feita, sendo apresentados no capítulo 5.

f) Redação do relatório: escrita de um TCC.

4 ARQUITETURA DA API DESENVOLVIDA

Neste capítulo são descritos alguns serviços da AWS que foram utilizados para construir a API proposta no objetivo geral deste trabalho com uma arquitetura *serverless*. Além disso, também descreve como a API funciona e os testes a serem feitos com ela.

4.1 Serviços

A seguir será descrito como funcionam os serviços da AWS.

4.1.1 AWS Lambda

Lambda é um serviço de computação que habilita o usuário a executar códigos sem provisionar ou manter servidores. O lambda será o serviço de computação em nuvem será executado no código da API desenvolvida neste trabalho. Como o Lambda só cobra para o tempo que o código é executado e é altamente escalável, é o serviço de computação serverless para a execução de testes (AWS, 2022).

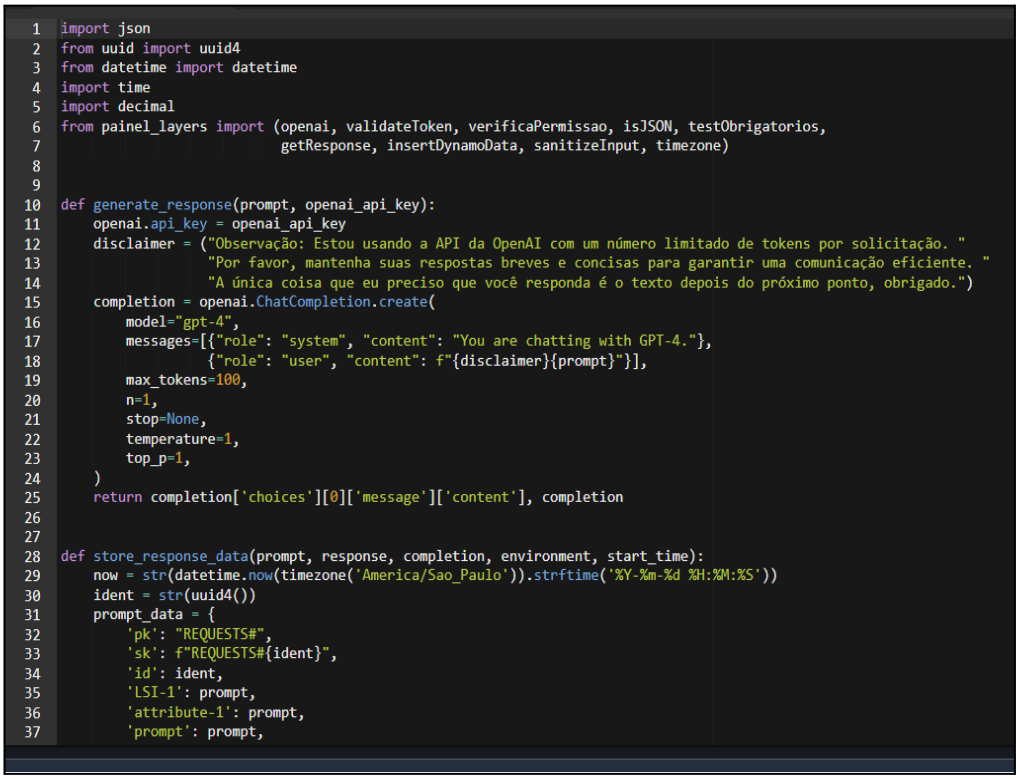
Na Figura 3 é apresentado o console do AWS Lambda, que oferece uma visão geral de todas as funções associadas à conta, além de fornecer informações relevantes sobre o desempenho dessas funções. O console é uma ferramenta essencial para gerenciar e monitorar o uso dos recursos do AWS Lambda.

Figura 3 - Funções do console

Fonte: AWS, 2023.

A Figura 4 apresenta o conteúdo de uma função Lambda, na qual é possível fazer alterações no código e realizar testes na nuvem antes de fazer o *deploy* para produção. Essa funcionalidade é extremamente útil para garantir que a função esteja funcionando corretamente e que não haja erros na implementação antes de disponibilizá-la para o ambiente de produção. O ambiente de testes na nuvem do AWS Lambda proporciona uma maior agilidade e eficiência no desenvolvimento de aplicações na nuvem.

Figura 4 - Código de uma função



```python
import json
from uuid import uuid4
from datetime import datetime
import time
import decimal
from painel_layers import (openai, validateToken, verificaPermissao, isJSON, testObrigatorios,
                           getResponse, insertDynamoData, sanitizeInput, timezone)


def generate_response(prompt, openai_api_key):
    openai.api_key = openai_api_key
    disclaimer = ("Observação: Estou usando a API da OpenAI com um número limitado de tokens por solicitação. "
                  "Por favor, mantenha suas respostas breves e concisas para garantir uma comunicação eficiente. "
                  "A única coisa que eu preciso que você responda é o texto depois do próximo ponto, obrigado.")
    completion = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[{"role": "system", "content": "You are chatting with GPT-4."},
                  {"role": "user", "content": f"{disclaimer}{prompt}"}],
        max_tokens=100,
        n=1,
        stop=None,
        temperature=1,
        top_p=1,
    )
    return completion['choices'][0]['message']['content'], completion


def store_response_data(prompt, response, completion, environment, start_time):
    now = str(datetime.now(timezone('America/Sao_Paulo')).strftime('%Y-%m-%d %H:%M:%S'))
    ident = str(uuid4())
    prompt_data = {
        'pk': "REQUESTS#",
        'sk': f"REQUESTS#{ident}",
        'id': ident,
        'LSI-1': prompt,
        'attribute-1': prompt,
        'prompt': prompt,
```

Fonte: AWS,2023

4.1.2 AWS API *Gateway*

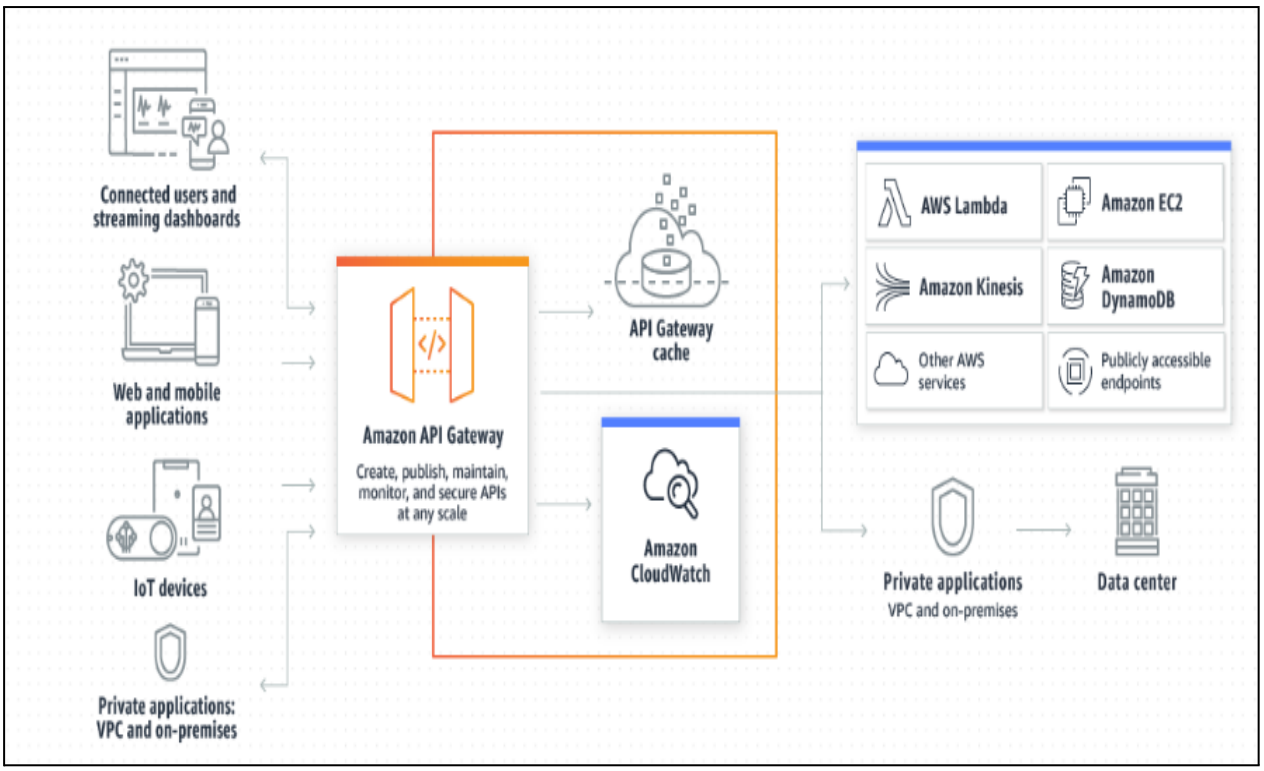
AWS API *Gateway* é o serviço para criar, publicar, manter, monitorar e proteger APIs de REST, HTTP e WebSocket. Junto com o AWS *Lambda*, o API *Gateway* forma a parte voltada para o aplicativo da infraestrutura serverless da AWS (AWS, 2022).

O *API Gateway* vai funcionar como uma porta de entrada para acessar a API. Para fazer isso, preciso criar um endpoint único usando o *domain* gerado pelo API

Gateway ou usar o *Route* 53 para gerar seu próprio. Quando algum usuário bater nesse *endpoint* vai executar a função *Lambda* que está conectado a ele.

A Figura 5 ilustra um diagrama apresentando como as APIs criadas no *Amazon API Gateway* fornecem aos usuários uma experiência de desenvolvedor integrada e consistente para criar aplicativos sem servidor da AWS. No lado esquerdo pode se observar as formas de se conectar com o *API Gateway*, no centro tem o *API Gateway* e os serviços que funciona junto com ele. Finalmente, tem os outros serviços da AWS que podem funcionar junto com o *API Gateway* e que tem a possibilidade de utiliza-lo com aplicativos particulares (AWS, 2022).

Figura 5 – Arquitetura do API do Gateway



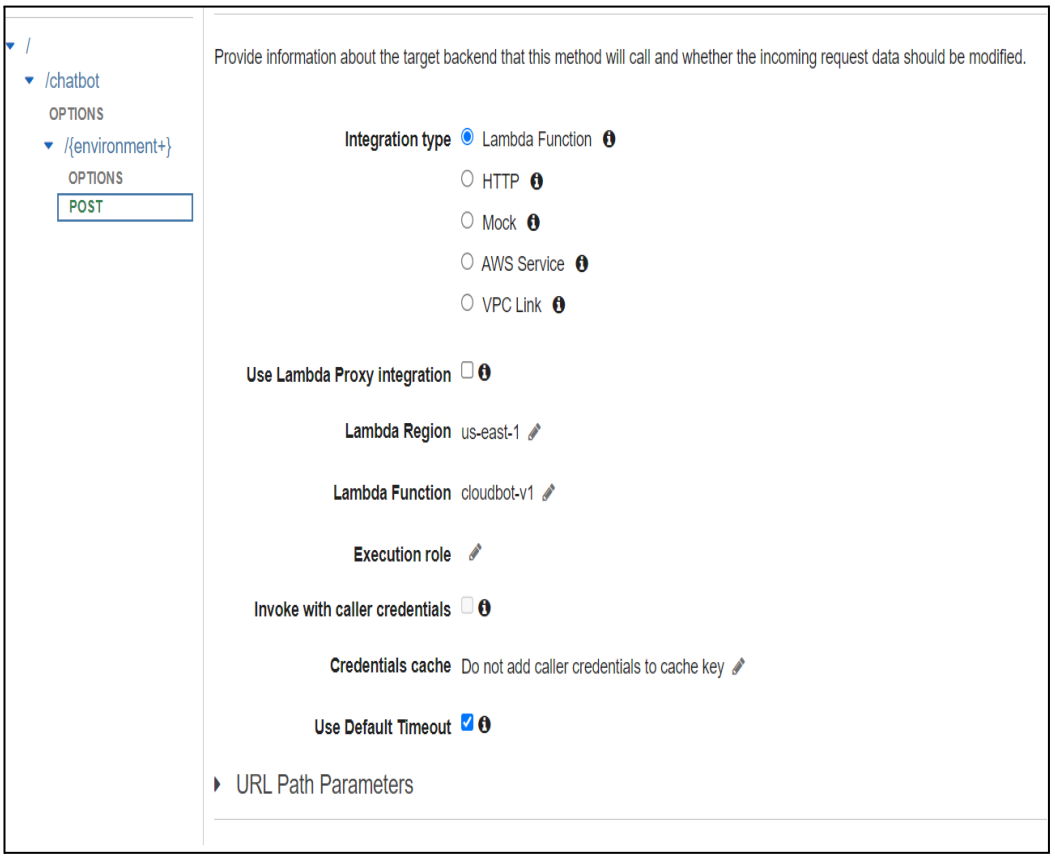
Fonte: AWS, 2022

A Figura 6 apresenta uma API criada no *API Gateway*, na qual é possível observar o recurso "chatbot" dentro de "*Resources*", o que vai determinar o caminho do URL da API. Além disso, é possível notar que o ambiente está configurado como "{*environment*+}" e, dentro deste ambiente, há um método HTTP definido como POST. Esse método HTTP está vinculado à função do Lambda chamada "*cloudbot*-v1".

Essa configuração permite que a API Gateway execute a função Lambda sempre que a rota da API for acionada pelo método HTTP definido. O API Gateway é

uma ferramenta poderosa para criar e gerenciar APIs na nuvem e é amplamente utilizado em aplicações que envolvem comunicação entre sistemas distribuídos.

Figura 6 – API Configurada



Fonte: AWS, 2023

4.1.3 DynamoDB

O DynamoDB é um banco de dados não relacional (NoSQL) oferecido pela AWS. Diferente dos bancos de dados relacionais tradicionais, o DynamoDB não segue o modelo de tabelas com esquemas fixos. Em vez disso, ele permite armazenar e recuperar dados de forma flexível, sem a necessidade de definição de esquemas rígidos. Isso torna o DynamoDB altamente escalável e adequado para cenários em que a estrutura dos dados pode variar ou evoluir ao longo do tempo.

Uma das principais vantagens do DynamoDB é sua natureza *serverless*. Isso significa que o usuario não precisa provisionar ou gerenciar infraestrutura de banco de dados, pois a AWS cuida desses aspectos. O DynamoDB escala automaticamente

para lidar com a carga de dados e o número de solicitações, permitindo um dimensionamento eficiente e sem interrupções. Além disso, ele oferece recursos de replicação e alta disponibilidade, garantindo a durabilidade e a resiliência dos dados.

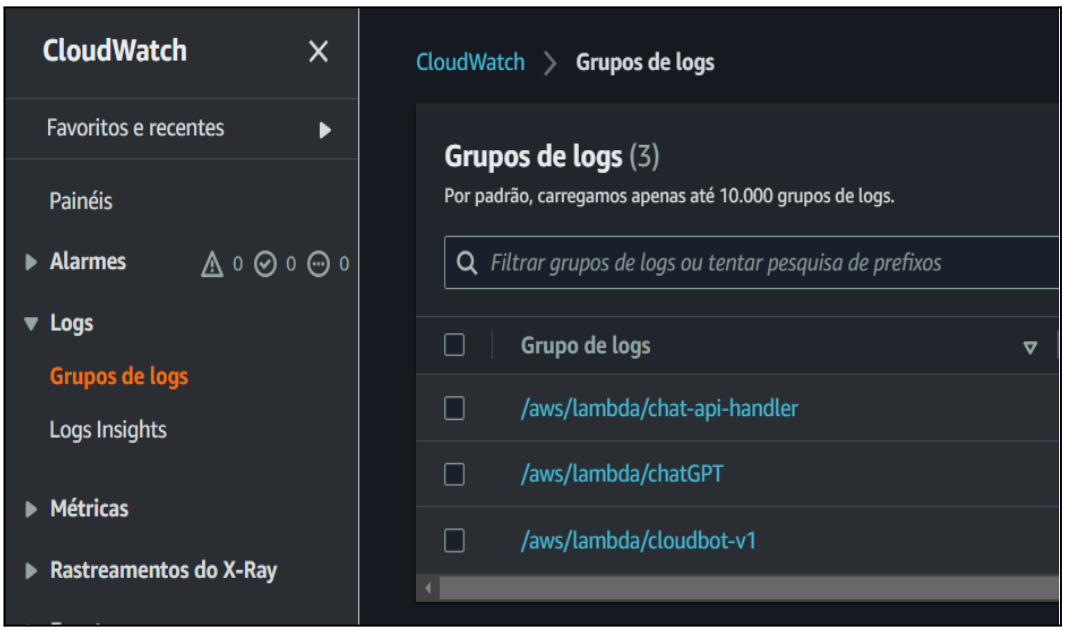
No contexto deste trabalho, o DynamoDB foi escolhido como o banco de dados para armazenar informações geradas durante a execução da API do Chatbot, pois sua natureza serverless se alinha com a arquitetura adotada, na qual os serviços AWS utilizados são serverless, como o AWS Lambda. Isso contribui para a simplicidade e a escalabilidade da solução, eliminando a necessidade de gerenciar a infraestrutura do banco de dados e permitindo que o foco seja direcionado para o desenvolvimento do Chatbot e a análise dos dados armazenados.

4.1.4 CloudWatch

O *Amazon CloudWatch* serve para monitorar os recursos e aplicativos dentro de uma conta da AWS em tempo real. Dentro desse serviço pode ser visto uma grande variedade de dados e sobre seu projeto como, por exemplo, o tempo de execução de uma API e o que está retornando (AWS, 2022).

Ao observar a Figura 7 é possível visualizar o console do CloudWatch destinado aos registros de atividades. Nessa Figura encontram-se os registros de todos os recursos do serviço que tiveram o monitoramento habilitado.

Figura 7 – Console de *CloudWatch Logs*



Fonte: AWS, 2023

4.1.5 EventBridge

O *Amazon EventBridge* usa eventos para interligar aplicativos com o outro. Aqui dentro pode colocar uma API para ser executada em um horário específico com um payload específico (AWS, 2022).

4.2 APLICAÇÃO CONSTRUÍDA – UM CHATBOT

O *Chat Bot* é uma ferramenta que empresas utilizam para conversar com seus clientes por meio de aplicativos de mensagens, sites e outras plataformas digitais. Os *Chat Bots* costumam funcionar de duas formas: o primeiro usando diretrizes pré-programadas, e a segunda forma utilizando inteligência artificial (IA) (TAKE BLIP, 2021).

O *Chat Bot* será construído utilizando o *framework serverless* 3.0 no *Visual Studio Code,* na linguagem *Python* e mantido pela *A*WS, com uma arquitetura *serverless*. Dentro da plataforma AWS, o código será executado pelo AWS *Lambda*. A AWS *API Gateway* será utilizada para criar uma API de REST e o *Route* 53 será usado como seu serviço de DNS. Esses serviços, em conjunto com um ou outra forma a arquitetura *serverless* e servirá como o substituto de um ambiente *On-Premise.*

4.2.1 Funcionamento da API

Para iniciar o funcionamento da API, ou seja, realizar uma chamada de API, vai iniciar o atendimento do *Chat Bot*. O usuário vai digitar uma pergunta qualquer para o Chat Bot e ele lhe responderá. Como o Chat Bot está utilizando a API do ChatGPT, da OpenAI, o Chat não tem limite de assunto. Durante a execução do aplicativo desenvolvido, o Chat Bot estará guardando informações sobre o atendimento dentro de um BD *NoSQL* da AWS, chamado AWS *DynamoDB*. Informações tais como o tempo de atendimento, quantas chamadas de API foram feitas e se o usuário ficou satisfeito com o atendimento, será armazenado no BD.

Como o AWS *Lambda* cobra pelo tempo de execução do código, será colocado um tempo de *timeout*. Se esse tempo de *timeout* for ultrapassado o atendimento será encerrado automaticamente. Caso acontecer *timeout,* este também será armazenado no BD.

A necessidade desse timeout seria uma das limitações do *API Gateway*. Esse serviço tem um tempo máximo para receber resposta. Caso não receba nenhuma resposta, simplesmente vai fechar a conexão.

A intenção de construir um Chat Bot na nuvem é para mostrar que é possível criar algo que os desenvolvedores sempre associam com um ambiente On-Premise. Além disso, o Chat Bot vai responder dúvidas dos usuários interessados sobre a *cloud* AWS e a arquitetura *serverless*.

4.2.2 ChatGPT

Este item terá como referência o site da OpenAI (2023). O ChatGPT é um modelo de linguagem natural desenvolvido pela empresa americana chamada OpenAI, que utiliza redes neurais de última geração para gerar respostas coerentes e humanas em diálogos em linguagem natural.

O "GPT" significa "*Generative Pre-trained Transformer*", que em tradução livre seria "transformador Pré-Treinado Generativo" e se refere à tecnologia de treinamento de redes neurais que o ChatGPT utiliza para gerar suas respostas. O GPT é um algoritmo baseado em modelos de linguagem pré-treinados em grandes quantidades de dados, que lhe permitem "entender" a estrutura da linguagem e produzir respostas coerentes a partir de entradas em linguagem natural.

O ChatGPT é capaz de realizar conversas em diversos tópicos e contextos, desde perguntas simples até diálogos mais complexos, com uma grande variedade de palavras e frases. Ele pode ser utilizado em diversas aplicações, como assistentes virtuais, chatbots, sistemas de suporte ao cliente e outros tipos de interação com usuários.

A API do ChatGPT é semelhante à versão disponível no site, permitindo que os usuários enviem perguntas em linguagem natural para o modelo de inteligência artificial. Essa API vai servir com o fundamento do CloudBot. O CloudBot vai utilizar uma versão configurada da API do ChatGPT onde as respostas são mais curtas e mais relevantes a assuntos de nuvem e arquitetura serverless.

4.3 Testes realizados

Esta seção descreve como os testes foram feitos e quais informações pode extrair deles.

4.3.1 Teste de Performance

Esse teste foi para verificar o desempenho de uma API na nuvem. Para esse teste foi utilizado o *Postman* para verificar o tempo de resposta da API junto com os dados gravados no *DynamoDB.*

Este teste vai estar principalmente sendo comparado ao ChatGPT em se, quando um usuário entra nele uma resposta pode demorar entre alguns segundos até alguns minutos para completar.

4.3.2 Teste de Escalabilidade

Um dos pontos fortes da nuvem e a arquitetura *serverless* é sua capacidade de escalar praticamente infinitamente automaticamente. Para verificar isso foi simulado o horário pico com muitos usuários tentando utilizar a API ao mesmo tempo.

Esse teste foi feito com o AWS *EventBridge* e o *Cloudwatch*. Este serviço consegue executar o Lambda sozinho, com comandos específicos a serem executados no Lambda. Foi usado o *Cloudwatch* para verificar como a API reagiria com todo esse serviço de uma vez.

4.3.3 Teste de Confiabilidade

Esse teste foi feito com a ajuda do AWS *Cloudwatch* em conjunção com o *Postman*. Quando um serviço falha é importante que os outros serviços continuem processando sua ação ou a falha é rapidamente detectada. O *Cloudwatch* tem a função de monitorar as execuções do *Lambda*.

Várias falhas propositais foram causadas nos seguintes lugares: Lambda, API Gateway e no código da API em si. Após a execução, foi consultado o *Postman* e o *Cloudwatch* para conferir se o erro foi identificado.

4.3.4 Teste de Segurança

Este teste serviu para verificar se uma API na nuvem é realmente segura. Esse teste foi executado dentro do *postman*. A API foi executada com várias combinações

de HTTPs para verificar se conseguiria obter algum dado sensível da API.

5 ARQUITETURA DA API DESENVOLVIDA

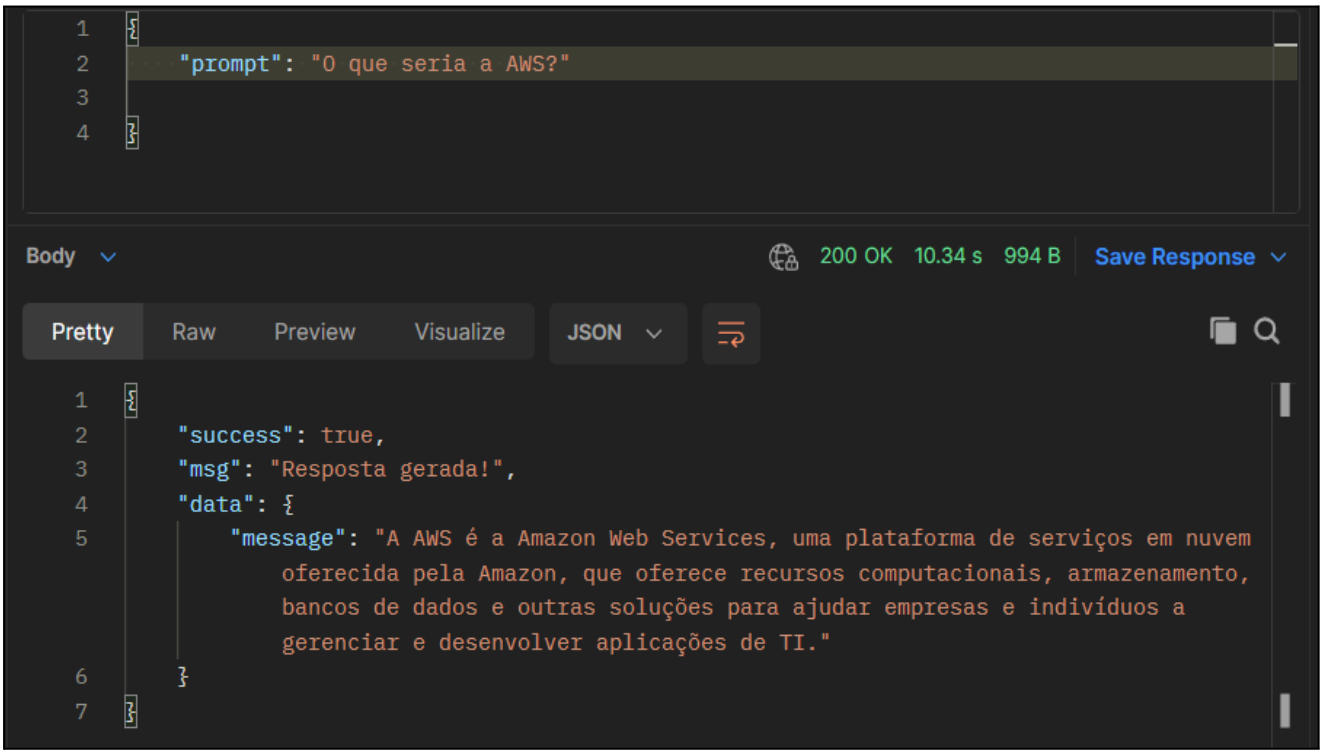
Neste capítulo são apresentadas informações detalhadas sobre o funcionamento do aplicativo desenvolvido, bem como o processo que foi necessário para fazê-lo funcionar. Além disso, são fornecidas as configurações gerais do aplicativo, com o objetivo de esclarecer o seu uso e garantir um bom desempenho.

A API desenvolvida foi denominada de CloudBot, por ser um chatbot na nuvem e sempre será referida assim neste trabalho.

5.1 Entrada e saída do aplicativo CloudBot

Caso o usuário queira fazer uma pergunta ao CloudBot, vai digitar sua questão no campo de consulta (*querystring*) do aplicativo. Em seguida, o aplicativo envia a requisição para a API do ChatGPT e aguarda a resposta. Um exemplo deste processo pode ser visualizado na Figura 8.

Figura 8 - Execução da API



Fonte: Postman, 2023.

No campo "*Body*" é possível observar o campo prompt, que contém a mensagem ou pergunta que o usuário deseja enviar ao CloudBot. No campo *Response*, são apresentadas todas as informações retornadas pela API como resultado da interação com o CloudBot.

O termo {{endpoint}} representa uma variável de URL fornecido pelo API Gateway para realizar chamadas de API. À direita, o URL completo pode ser visto ao lado de curl –location. Abaixo de curl –location, encontra-se o parâmetro –header, no qual a chave de API, também gerada pelo API Gateway, é inserida.

5.2 Código do *Lambda Function*

Dentro do *Lambda*, a primeira etapa é realizar verificações. Em primeiro lugar, é verificado se o chamado HTTP é do tipo "POST". Em seguida, é verificado se as variáveis de ambiente estão configuradas corretamente. Uma vez que essas verificações tenham sido concluídas com sucesso, o próximo passo é verificar se o usuário enviou as informações necessárias para chamar a API do chatbot. Após essas verificações, as informações retornadas pelo chatbot são armazenadas no DynamoDB.

5.3 Configurações do *DynamoDB*

O *DynamoDB* é configurado com capacidade sob demanda, ou seja, ele escala automaticamente para atender à quantidade de dados que estão sendo utilizados. O nome da chave de partição é "pk" e a chave de classificação é "sk". É importante notar que como o DynamoDB é um banco de dados não relacional, a chave de partição "pk" é equivalente a uma tabela em um banco de dados relacional e a chave de classificação "sk" serve como a chave primária desta tabela. Embora seja possível que os valores de "pk" e "sk" de um item sejam iguais, eles não podem ser iguais, simultaneamente.

Além dessas chaves principais, também foram definidos três atributos de Índice Secundário Local ou LSI, que receberam os nomes de LSI-1, LSI-2 e LSI-3, todos eles do tipo *string*. No caso do LSI-1, ele será utilizado para armazenar informações de um outro atributo do tipo *string*, como a data de inclusão dos dados. Isso permite que os dados sejam ordenados com base no LSI, ao invés da chave de classificação "sk". Com esses índices secundários adicionais, o DynamoDB oferece mais flexibilidade para realizar consultas de dados, possibilitando uma melhor otimização do acesso aos

dados armazenados no banco.

Os outros atributos são:

- tempoExecucao: Vai definir o tempo de execução da chamada de API
- dataInclusao: A data e hora que foi feito a chamada de API
- prompt: Será o que foi perguntado/dito ao chatbot
- resposta: Resposta retornado pelo chatbot

5.4 Configurações do API Gateway

Foi criada uma base no API Gateway que permite criar APIs RESTful. Foi incluso um *endpoint* do tipo *edge* para reduzir o tempo de execução, em caso de acesso de outras regiões geográficas.

A API possui um recurso, um ambiente e um método HTTP do tipo "POST". A URL principal foi gerada pelo API Gateway, mas como ela foi criada com um nome aleatório, uma outra URL foi configurada utilizando o Route 53.

O número de requisições à API é ilimitado, mas somente usuários com a URL correta, o recurso, o ambiente, o método e a chave de API poderão utilizá-la. Todas essas medidas foram implementadas como formas de segurança, para garantir que somente usuários autorizados possam acessar a API.

6 ANÁLISE DOS RESULTADOS OBTIDOS

Neste capitulo serão apresentadas as informações detalhadas sobre os testes realizados com o ChatBot (API dsenvolvida neste trabalho). Ao final, serão comparados os resultados obtidos com os trabalhos relacionados, a fim de avaliar o desempenho e a eficácia do CloudBot.

6.1 Resultado Teste de Performance

Conforme mostrado na Figura 9, o tempo de execução ou ‘executionTime’ da requisição até a obtenção da resposta do CloudBot foi de 10,35 segundos. Esse intervalo muito longo se deve ao fato de que a execução ocorreu em um lambda frio. Nesses casos, é necessário que a AWS realize a alocação prévia de um lambda antes que o código possa ser efetivamente executado, o que pode impactar significativamente no tempo de resposta. Descricao de outros atributos da tabela: ‘pk’ seria o nome desta tabela, ‘sk’ o id identificador do registro, ‘dateAdded’ seria a data que foi incluído o registro, ‘id’ seria uma versão mais simplificada do id para facilidade na busca e o ‘info’ tem várias informações sobre a execução da API.

Figura 9 – BD Cloudbot

pk	sk	dateAd...	executionTime	id	info
REQUESTS#	REQUEST...	2023-03-...	7.41636347770...	098beadd-...	{ "created" :...
REQUESTS#	REQUEST...	2023-03-...	5.65550637245...	2f73a914-e...	{ "created" :...
REQUESTS#	REQUEST...	2023-03-...	4.78674578666...	4504faa1-2...	{ "created" :...
REQUESTS#	REQUEST...	2023-03-...	4.07890725135...	935c90e1-...	{ "created" :...

Fonte: AWS, 2023

Ao observar a Figura 9 é possível verificar alguns dados armazenados no banco de dados. No que se refere ao teste de performance, o atributo *executionTime* se apresenta como o mais relevante, pois revela o tempo gasto na execução de cada chamada da API. Conforme os dados exibidos, nota-se que a média de tempo de

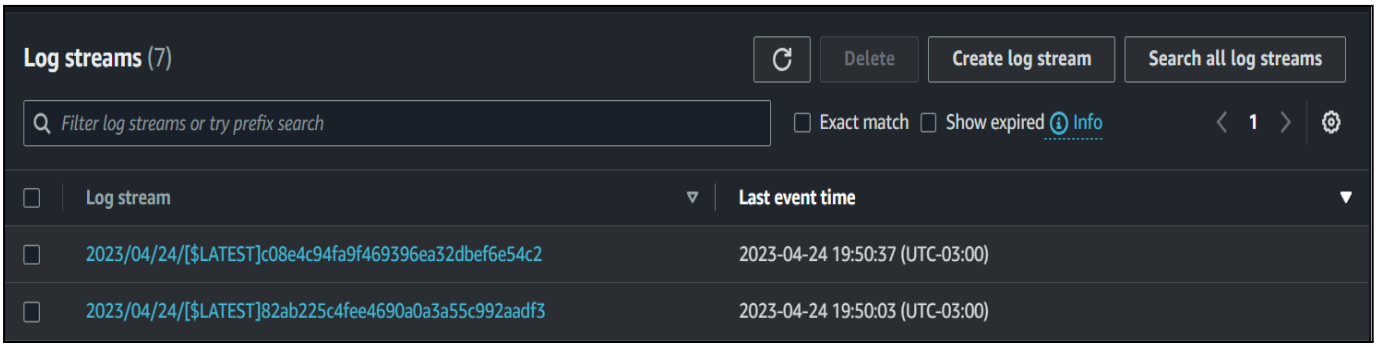
execução fica de 3 a 5 segundos, sendo o tempo mais elevado registrado quando o lambda se encontra em estado frio.

6.2 Resultado Teste de Escalabilidade

Foram configuradas três regras no EventBridge, cada uma delas fazendo uma requisição ao Cloudbot, simultaneamente. Cada regra realiza uma pergunta distinta: uma sobre o que é o EventBridge, outra sobre o que é uma regra no EventBridge e, por fim, uma pergunta sobre o cron. O CloudBot processou essas 3 perguntas em todos os segundos durante 1 minuto e simulou várias pessoas utilizando, ao mesmo tempo.

Na Figura 10 são apresentados os logs do CloudWatch. Foram gerados apenas dois logs porque, mesmo havendo três requisições tão próximas uma da outra, o CloudWatch agregou as informações de duas ou mais requisições em um mesmo log.

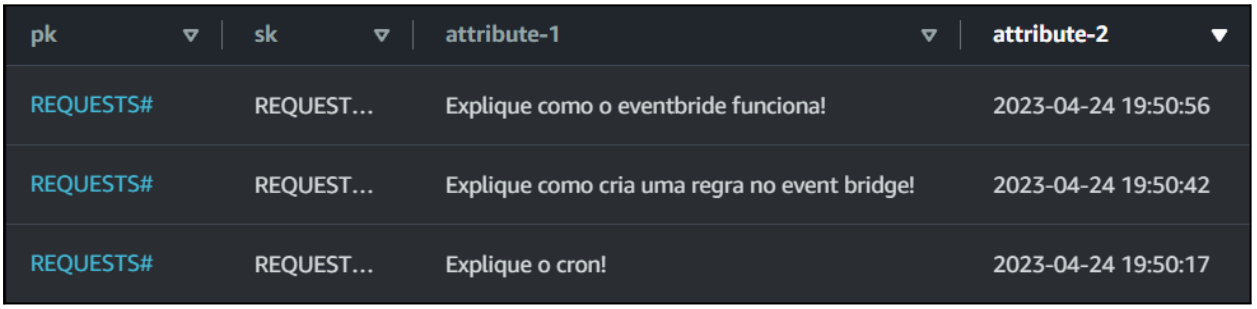
Figura 10 – Log EventBridge



Fonte: AWS, 2023

Na figura 11 é possível observar os resultados dessas requisições no DynamoDB. É possível visualizar a pergunta feita e o tempo em que a execução da API foi concluída.

Figura 11 – DynamoDB EventBridge

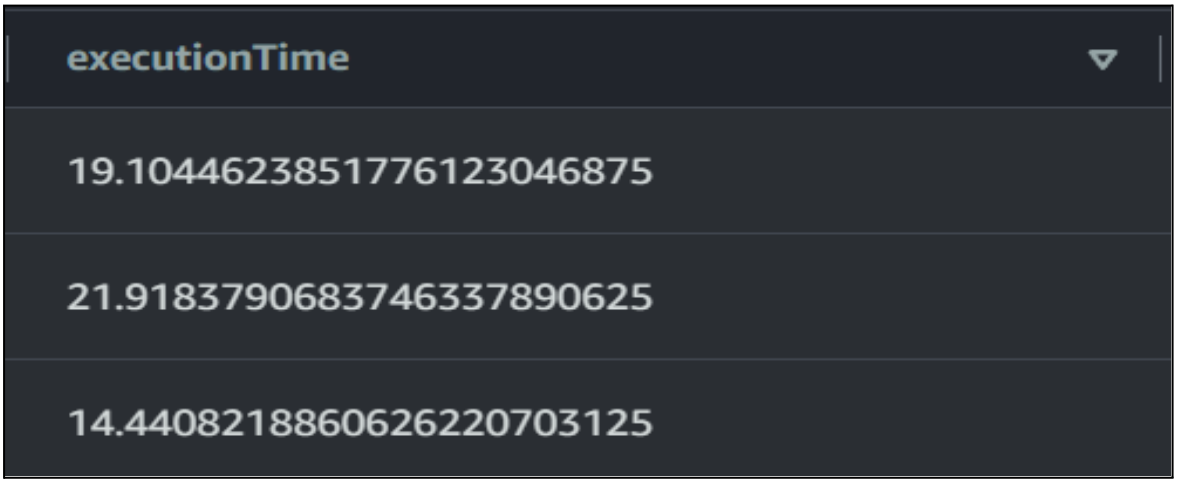


pk	sk	attribute-1	attribute-2
REQUESTS#	REQUEST...	Explique como o eventbride funciona!	2023-04-24 19:50:56
REQUESTS#	REQUEST...	Explique como cria uma regra no event bridge!	2023-04-24 19:50:42
REQUESTS#	REQUEST...	Explique o cron!	2023-04-24 19:50:17

Fonte: AWS, 2023

A figura 12 apresenta o tempo de execução dessas requisições pelo EventBridge. Nesse caso, houve um tempo de execução maior devido ao aumento do tráfego.

Figura 12 – Tempo de Execução EventBridge



executionTime
19.104462385177612304687 5
21.918379068374633789062 5
14.440821886062622070312 5

Fonte: AWS, 2023

6.3 Resultado Teste de Confiabilidade

Nesta secao estão apresentados todos os resultados dos testes de confiabilidade.

6.3.1 Erro no Lambda

Neste teste, foi modificado as configurações gerais do Lambda. Considerando que, em média, o CloudBot precisa de 3 a 7 segundos para executar, estabelece o tempo máximo de execução do Lambda em apenas 1 segundo.

Uma vez que se trata de um erro relacionado ao Lambda, pode-se acessar o CloudWatch para analisar detalhadamente o que ocorreu durante a execução do Lambda. Na Figura 13, observa-se um erro 500 no Postman.

Figura 13 - Resultado Lambda

Fonte: Postman, 2023

Na Figura 14 observa-se o log específico para o erro Lambda. Os logs apresentam o passo a passo do que ocorreu no código, desde o início até o término da execução. Entre o começo e o fim, pode-se identificar um log que menciona "Task timed out after 1.00 seconds". Isso indica que o erro ocorrido foi devido ao tempo limite excedido (*timeout*). Essa situação demonstra como o CloudWatch pode ser uma ferramenta valiosa para identificar e solucionar erros relacionados ao Lambda.

Figura 14: Logs Erro Lambda

```
2023-04-10T03:41:11.276Z 3b5a95b1-9eea-40c5-87ab-f6350838d03f Task timed out after 1.00 seconds
```

Fonte: AWS, 2023

6.3.2 Erro no API Gateway

Na Figura 15 pode ser observado o erro do API Gateway, que apresenta o erro 500 sem fornecer detalhes adicionais para identificar a origem do problema.

Figura 15: Erro API Gateway

Fonte: Postman, 2023

Para um desenvolvedor investigar a causa, seria necessário verificar se há algum log no CloudWatch. No entanto, como o erro está no API Gateway, o processo não chegou ao ponto de iniciar a geração de logs.

Isso deixa o usuário com apenas uma alternativa para identificar a falha, que é refazer todos os passos anteriores: primeiro, testar o código localmente, depois executá-lo no Lambda, em seguida, reconstruir o caminho dentro do API Gateway e, finalmente, tentar executar o código pelo Postman novamente.

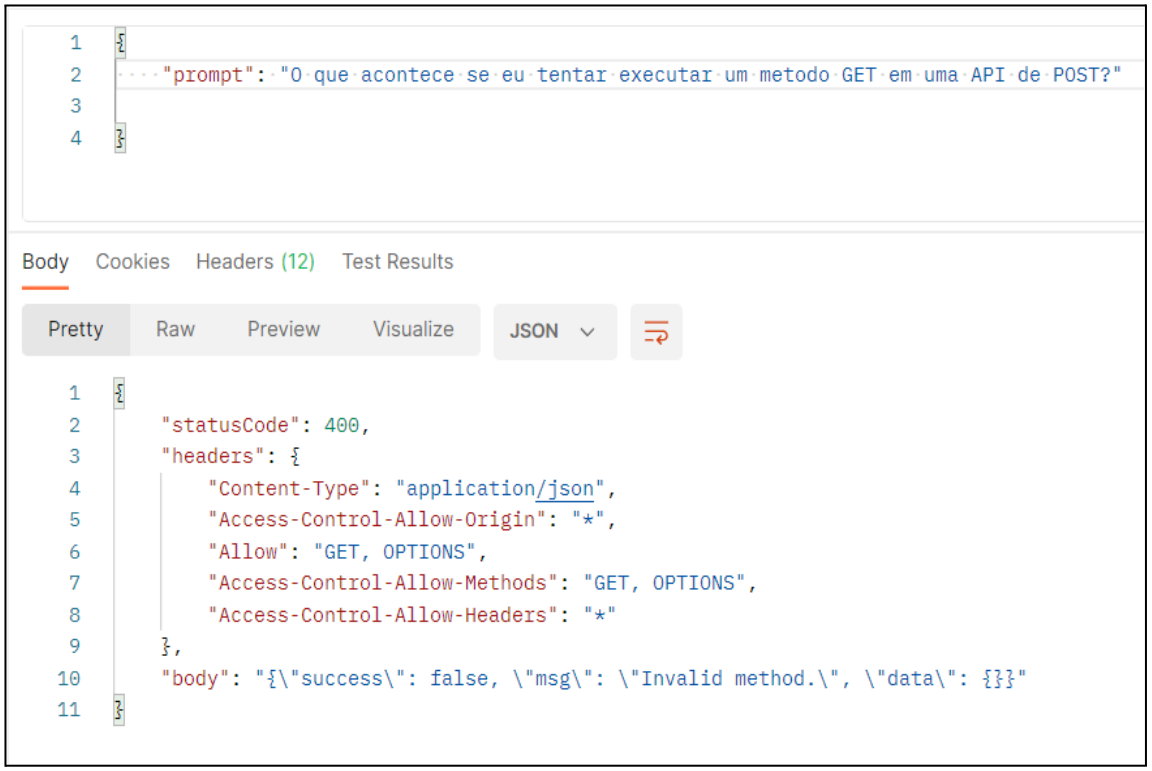
Com isso, pode-se concluir que se o erro estiver no API Gateway não há uma maneira fácil e rápida de identificar a causa do problema.

6.3.3 Erro no Código

Neste teste, foi inserido um erro proposital no código, em forma do método que o método que ele aceita fosse diferente do que pede no API Gateway. Esse erro pode ocorrer quando um desenvolvedor desatento realiza o *deploy* de sua função Lambda sem testá-la previamente.

Na Figura 16, é possível observar que o *Postman* retorna um erro originado do próprio código. Isso significa que o erro pode ser identificado de maneira rápida e eficiente, permitindo que o desenvolvedor direcione seu tempo para corrigi-lo.

Figura 16: Erro Código



Fonte: Postman, 2023

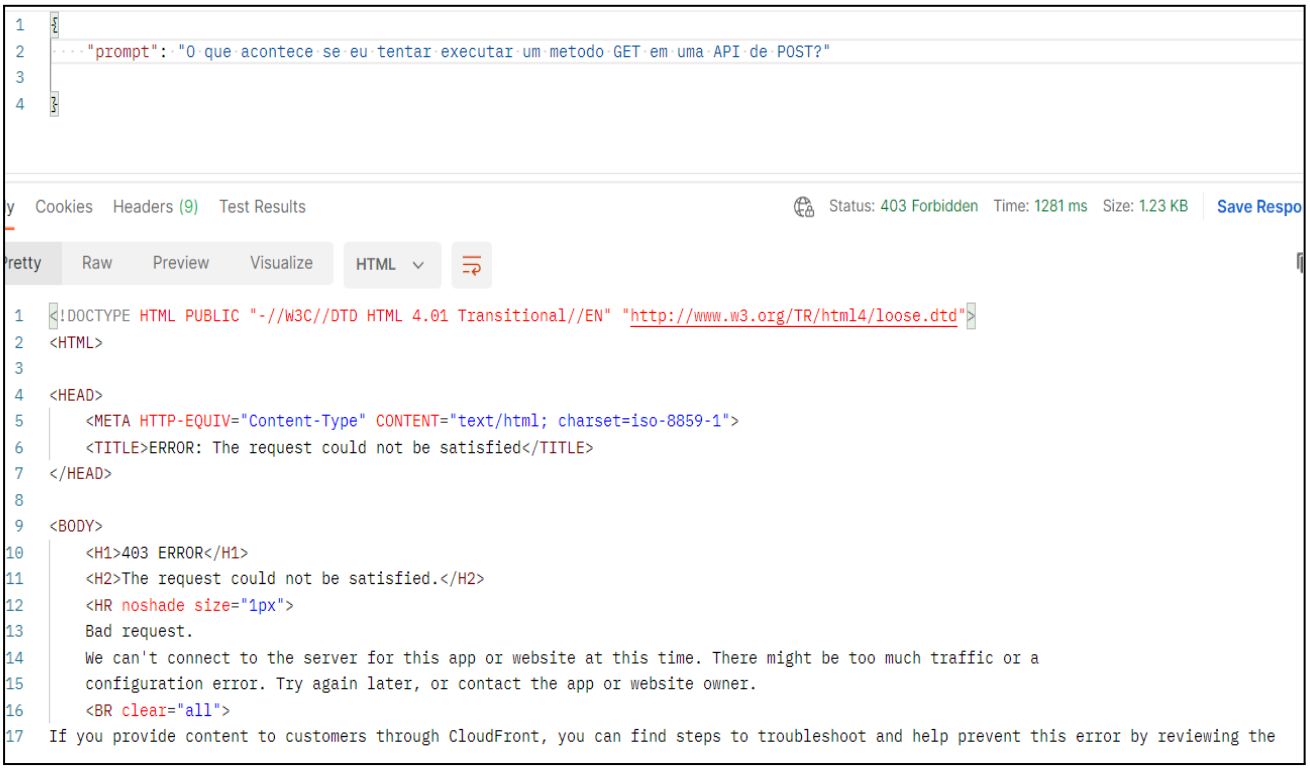
6.4 Resultado Teste de Segurança

Neste teste, foi realizada a alteração do método utilizado dentro do Postman. O Postman vai passar um HTTP de GET enquanto ainda cobra um HTTP de POST no API Gateway. O objetivo era verificar se, ao seguir os padrões estabelecidos pela AWS, seria possível evitar a perda de informações confidenciais.

Na Figura 17, é possível observar o retorno de um erro relacionado à API. O

detalhamento do erro é insuficiente para compreender sua causa. Isso indica que, caso alguém tente extrair informações sensíveis dessa maneira, nem mesmo o retorno da API permitirá a visualização desses dados.

Figura 17 - Teste Segurança



Fonte: Postman, 2023

6.5 Comparação com os Trabalhos Relacionados

Ao adotar a arquitetura serverless, foi possível constatar os benefícios relatados nos estudos anteriores nos trabalhos relacionados.

Paz e Loos (2020) destacaram que a computação em nuvem, um dos principais pilares da Indústria 4.0, proporciona agilidade e adaptabilidade às variações da produção. A arquitetura serverless da API dsenvolvida chamada CloudBot permite a escalabilidade sob demanda, garantindo que a capacidade de processamento se ajuste automaticamente às necessidades do sistema, evitando perdas e retrabalho.

Eisman et al. (2020) abordaram a adoção da computação serverless, destacando sua usabilidade e eficiência. Este trabalho investigou as razões pelas quais as empresas optam por soluções serverless e em quais cenários essas aplicações são adequadas. O CloudBot, ao utilizar o AWS Lambda e o Amazon API Gateway, segue a tendência de adoção do serverless para economizar custos, lidar com cargas de trabalho intermitentes e obter escalabilidade integrada.

7 CONCLUSÃO

Este trabalho teve o intuito de responder a seguinte questão de pesquisa – **A arquitetura serverless é um padrão válido para criar aplicativos**?

O objetivo geral foi de buscar informações sobre a computação em nuvem e arquitetura serverless para a construção de um aplicativo. O resultado disso foi a construção de uma API chamada de CloudBot, ou seja, um chatbot baseado em arquitetura *serverless*, hospedado na nuvem AWS e utiliza os serviços AWS Lambda e Amazon API Gateway.

O CloudBot, desenvolvido com base na arquitetura serverless, cumpriu os objetivos propostos, apresentando um desempenho adequado, escalabilidade eficiente, confiabilidade na detecção e resolução de erros, além de implementar medidas de segurança para proteção dos dados. Com isso, o CloudBot se mostra como uma solução viável para a criação de aplicativos e pode ser aplicado em diferentes cenários.

Ao finalizar o trabalho e analisando os resultados obtidos nos testes de desempenho, escalabilidade, confiabilidade e segurança, foi possível concluir que a arquitetura serverless é, de fato, uma opção válida e eficiente para a construção de aplicativos.

Ao estudar este trabalho, espera-se que os leitores possam adquirir um conhecimento básico sólido sobre a arquitetura serverless, capacitando-os a realizar pesquisas futuras e até mesmo desenvolver aplicativos seguindo esse padrão.

Para continuidade desta pesquisa, sugere-se, como trabalho futuro:

- Construção de outros aplicativos mais complexos utilizando a arquitetura serverless, desenvolvendo e integrando diversas APIs para ampliar suas funcionalidades e permitir uma interação mais abrangente com os usuários. Isso possibilitará explorar ainda mais o potencial da arquitetura serverless na construção de aplicativos versáteis e completos.

8 REFERÊNCIAS

AKE BLIP. **Chatbot: o que é, como funciona, benefícios e cases.** 2022. Take Blip. Disponível em: https://www.take.net/blog/chatbots/chatbot/ Acesso em: 20/11/2022

AMAZON WEB SERVICES LATIN AMERICA. **O que é a AWS?.** YouTube, 24 de maio de 2021. Disponível em: < https://www.youtube.com/watch?v=8JI9wQ8sUdQ > acesso em: 04/09/2022

AWS. ***What is AWS*?.** 2022. AWS. Disponível em: https://aws.amazon.com/what-is-aws/ Acesso em: 15/11/2022

AWS. ***What is API?*.** 2022. AWS. Diponivel em: https://aws.amazon.com/pt/what-is/api/ Acesso em: 18/11/2022

AWS. ***What is AWS Lambda?*.** 2022. AWS Diponivel em: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html Acesso em: 18/11/2022

AWS. ***What is AWS Route 53?*.** 2022. AWS. Diponivel em: https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/Welcome.html Acesso em: 18/11/2022

AWS. ***What is AWS API Gateway?*.** 2022. AWS. Diponivel em: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html Acesso em: 18/11/2022

AWS. ***What is AWS DynamoDB?*.** 2022. AWS. Diponivel em: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html Acesso em: 18/11/2022

CLEO ***On Premise vs. Cloud: Key Differences, Benefits and Risks*.** Cleo Disponível em: https://www.cleo.com/blog/knowledge-base-on-premise-vs-cloud acesso em: 18/10/2022

EISMAN et al. ***Serverless Applications: Why, When, and How*?.** IEEE Software, Vol 38, No 1, 09 September 2020. Disponível em: https://ieeexplore.ieee.org/document/9190031 acesso em: 18/10/2022

GIL, Antônio Carlos **Como Elaborar Projetos de Pesquisa.**. 6. ed. São Paulo: Editora Atlas Ltda., 2017.

GOOGLE CLOUD, ***Advantages and Disadvantages of Cloud Computing.*** Disponível em: https://cloud.google.com/learn/advantages-of-cloud-computing acesso em: 16/11/2022

HALE & COX ***Cloud ERP vs. On-Premise ERP Software Advice.*** 4 de Dezembro de 2020. Disponível em: https://www.softwareadvice.com/resources/cloud-erp-vs-on-premise/

HAYEK, ODEH ***Cloud ERP VS On-Premise ERP.*** International Journal of Applied

JOHNSON et al. ***Cost Comparison of an On-Premise IT Solution with a Cloud-Based Solution for Electronic Health Records in a Dental School Clinic Journal of Dental Education.*** Volume 83, Nu 8 Agosto 2019. Disponível em: https://www.researchgate.net/publication/334946271 acesso em: 18/10/2022

OpenAI. OpenAI: **Artificial Intelligence & Human Collaboration.** Disponível em: https://www.openai.com/. Acesso em: 13 maio 2023.

PAZ, A.C. M. , LOOS, M. J**. A importância da computação em nuvem para a indústria 4.0..** Revista Gestão Industrial, Ponta Grossa, v. 16, n. 2, p. 166-185, Abr./Jun. 2020. Disponível em: https://periodicos.utfpr.edu.br/revistagi. acesso em: 16/08/2022

PAZ, S.F., SALGADO, R. S. E DIAZ, O. G. F. **Model of dynamic orchestration for SaaS.** Revista Ingenierías Universidad de Medellín, Medellín Colômbia, vol. 16, No. 31 pp. 143-153 ISSN 1692 - 3324 Jul./Dez. 2017.

REDHAT. **O que é serverless?**. [S. l.], 31 out. 2017. Disponível em: https://www.redhat.com/pt-br/topics/cloud-native-apps/what-is-serverless. Acesso em: 11 set. 2022.

SALATI, G.H. et.al. **Como é estar nas nuvens? Satisfação, lealdade e intenção de uso dos usuários de serviços de computação em nuvem da Netflix.** Revista Tecnologia e Sociedade, Curitiba, v. 16, n. 40, p. 1- 23, abr/jun. 2020. Disponível em: https://periodicos.utfpr.edu.br/rts/article/view/9355. acesso em: 16/08/10

WAZLAWICK, R. S. **Metodologia da Pesquisa para Ciência da Computação.** 2ª. ed. [S.l.]: Campus, 2014.