Serverless

Gabriel Duessmann[1]

[1]Departamento de Ciência da Computação
Universidade do Estado de Santa Catarina (UDESC) – Joinville, SC - Brasil

gabriel.duessmann@edu.udesc.br

Abstract. *More and more cloud providers are being used to provide solutions in the market and they try to innovate and create new services that help and make it easier for companies and developers to have all the advantages to keep and move their environments to the cloud. Serverless architecture appeared in cloud environments in which it enables the implemenation of applications with no servers setup, as it is managed by the cloud provider itself that abstracts this layer.*

Resumo. *Cada vez mais se utiliza de provedores de nuvem para prover soluções no mercado e estes tentam inovar e criar novos serviços que auxiliem e facilitem empresas e desenvolvedores a aproveitarem o máximo das vantagens de manter ou mover seus ambientes para nuvem. A arquitetura Serverless surgiu nos ambientes em nuvem no qual possibilita implementar aplicações sem precisar configurar servidores, pois é gerenciado pelo próprio provedor de nuvem que abstrae essa camada.*

1. Introdução

Este trabalho tem como objetivo apresentar uma arquitetura recente chamada de Serverless e fazer algumas comparações com arquiteturas mais tradicionais que são utilizadas no mercado há anos. A arquitetura Serverless surgiu para facilitar a implementação de aplicações e neste trabalho é apresentado quando deveria ser utilizado, suas principais vantagens e como implementar essa arquitetura em provedor de nuvem AWS utilizando o serviço Lamda.

2. Conceitos

De forma a esclarecer alguns conceitos e ferramentas, neste capítulo são apresentados ligeiramente conceitos gerais relacionados ao assunto que são mencionados nos próximos capítulos.

2.1. AWS

Amazon Web Services (AWS) é uma plataforma da Amazon que provê mais de 200 serviços completos em nuvem com *data centers* ao redor do mundo [Amazon 2022].

Os serviços ofertados pela empresa podem ser do tipo infraestrutura como um serviço (IaaS), plataforma como um serviço (PaaS) e software como um serviço (SaaS) e foi uma das primeiras empresas a introduzir cobrança no modelo *pay-as-you-go*, no qual o usuário paga baseado em quanto se consome do serviço [Gillis 2020].

Os serviços da AWS ficam espalhados entre as dezenas de zonas de disponibilidade em regiões ao redor do mundo. As zonas de disponibilidade são localizações que concentram *data centers* físicos e uma região é o conjunto de zonas de disponibilidade geograficamente próximas que são conectadas por cabo de rede de baixa latência. Essa infraestrutura permite que os clientes escolham suas regiões e zonas de disponibilidade conforme seus objetivos e prioridades [Gillis 2020].

2.2. AWS Lambda

AWS Lambda é um serviço fornecido pela AWS no qual é baseado na arquitetura Serverless e orientado a evento que permite executar código em forma de funções para praticamente qualquer tipo de aplicativo ou serviço back-end sem provisionar ou gerenciar servidores e paga-se apenas pelo o que utilizar. O serviço pode ser acionado a partir de mais de 200 serviços da AWS e aplicativos de software, como serviços SaaS via internet [Amazon 2022].

2.3 API Gateway

API Gateway pode ser considerado como um ponto de entrada, pois recebe todas as requisições APIs dos clientes. Depois da requisição chegar na API Gateway, esse serviço roteia para o serviço apropriado que precisa ser executado. Para isso, a API Gateway precisa conhecer todos os serviços disponíveis para quais podem encaminhar as requisições. No caminho de volta, onde a resposta é enviada ao cliente, o serviço executado encaminha a resposta para API Gateway e este fica responsável por responder a requisição do cliente [Nginx 2022].

3. Serverless

Serverless é um modelo de arquitetura que tem como próposito prover serviços backend sem que haja a configuração de um servidor por parte do cliente. É como se uma aplicação backend, Java por exemplo, seja executada sem servidor algum. Na verdade há um servidor, mas ele é como se fosse uma caixa preta para o cliente, no qual não tem acesso algum. Toda parte de provisionamento e configuração é controlado pelos provedores de nuvem que ofertam esse serviço e garantem o gerenciamento similar a se fosse responsabilidade do cliente. Esse tipo de infraestrutura surgiu em provedores de nuvem pois conseguem intermediar e controlar a camada do servidor afim de facilitar a implementação e resolver problemas que ocorrem em aplicações sob modelos tradicionais, como máquinas virtuais.

Essa arquitetura fornece escabilidade automática, alta disponibilidade e possui modelo de pagamento conforme utilização da aplicação. Dessa forma, otimiza os custos e aumenta a agilidade para o provisionamento, pois os times de desenvolvimento podem se dedicar a escrever códigos e criarem novas funcionalidades sem se preocuparem com o provisionamento e gerenciamento da infraestrutura do servidor em que sua aplicação é executada [Amazon 2022].

A camada de servidor se torna como uma abstração para o desenvolvedor que não precisa mais implementá-la. Apenas internamente os provedores desse serviço que realizam a implementação dessa camada e alocam os recursos necessários do servidor e da máquina para executar a aplicação, o que deixa muito mais simples para a empresa ou desenvolvedor disponibilizar sua implementação para a internet [Cloudfare 2022].

Como mencionado anteriormente, os clientes que utilizam os serviços relacionados a Serverless, pagam pela quantidade de utilização do seu código, ao invés do modelo trandicional de reservar e pagar por uma quantidade fixa de largura de banda ou número de servidores alocados, pois os serviços Serverless já são demensionáveis automaticamente. Empresas e desenvolvedores costumavem ter que alocar unidades fixas na computação em nuvem e muitas vezes precisavam alocar mais recurso do que necessário para prevenir ataques ou picos de tráfego para manter o sistema no ar, resultando em gastos extras e desnecessários pois nem sempre é utilizado todo o recurso alocado. Com os serviços Serverless, o cliente não precisa alocar nenhum recurso físico e o próprio serviço possui um mecanismo de escalabilidade automática que consegue tratar grandes picos de utilização do servidor, inclusive em ataques DDoS [Cloudfare 2022].

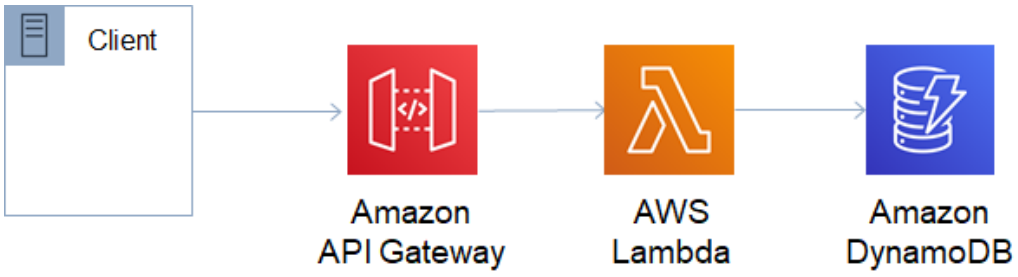
Uma técnica utilizada pelos provedores de nuvem é o *cold start*, ou partida fria, no qual o servidor da função Serverless é interrompido e fica parado como se estivesse "hibernando" quando uma aplicação não é exucutada por um longo período, no qual é configurado pelo próprio provedor. Quando um cliente chama a função novamente, o provedor inicia o servidor e a aplicação volta a ficar disponível, o que faz com que o tempo de resposta dessa requisição seja maior porque a aplicação não estava no ar e teve que ser inicializada. As requisições adjancetes terão tempo de resposta menor se chamadas dentro de um curto período de tempo pois o servidor já está ligado. Essa abordagem é utilizada pelos provedores de nuvem para que os recursos da máquina alocada não sejam consumidos desnecessariamente para provisionar uma aplicação que não é utilizada ou utilizada com pouca frequência, e consequentemente resulta na economia de energia de seus *data centers* [Cloudfare 2022].

3.1. Serverless no contexto Web

Para aplicações Web, a arquitetura Serverless também se faz presente e pode ser uma boa escolha para que o desenvolvedor não precise hospedar um servidor Web, como Wildfly ou Tomcat por exemplo.

A Figura 1 representa um diagrama de como seria uma arquitetura no provedor de nuvem AWS para disponibilizar um ambiente Serverless que possa ser utilizado na internet via chamadas API.

Figura 1. Diagrama de arquitetura Web com aplicação Serverless



Fonte: Amazon, 2022.

Como apresentando no diagrama da Figura 1, o cliente pode fazer uma chamada via API que entra em contato primeiramente com o serviço Amazon API Gateway. Esse serviço

é responsável por receber as requisições e redirecionar para o serviço correspondente que o cliente deseja, neste caso uma função Lamda. Enquanto a função está sendo executada, pode utilizar de outros serviços, como mostra no diagrama que a função utiliza um serviço de banco de dados, o Amazon DynamoDB. Ao finalizar o processamento, a reposta passa para o Amazon API Gateway novamente e por fim esse serviço retorna a resposta da requisição ao cliente.

3.2. Vantagens

A arquitetura Serverless surgiu para facilitar a implementação de aplicações sem configurar os servidores, trazendo benefícios em se utilizar serviços baseados nessa arquitetura. A seguir são apresentados as principais vantagens dessa arquitetura:

- Camada de servidor abstrata para o usuário;
- Agilidade nas implementações e atualizações;
- Funções independentes.
- Inerentemente escaláveis;
- Redução nos custos.

Umas das princpais vantagens está relacionada a configuração do sistema, pois nenhuma configuração no ambiente e do servidor são necessárias, permitindo os desenvolvedores não se preocuparem com a implementação dessa camada de servidor, pois isso fica abstrato para eles e é gerencido pelo provedor de nuvem onde a aplicação é hospedada. As empresas de software também se beneficiam pois conseguem reduzir investimentos e custos com equipes específicas para realizar as configurações e gerenciamento de servidores, como profissionais na área de DevOps, e deixam seus desenvolvedores muito mais focado em criarem novas funcionalidades e expandirem as aplicações sem limitações do servidor [Cloudfare 2022].

Tem-se um ganho significativo em implementações e atualizações do ambiente pois pode ser liberado apenas funcionalidades pequenas e rápidas de serem desenvolvidas. Por não se tratar de uma aplicação única monolítica, mas de uma coleção de funções indepenentes que compõem toda a aplicação, novas funcionalidades desenvolvidas podem ser liberadas facilmente e sem impactar as outras funções que já estão sendo executadas e funcionam. Isso também resulta em maior estabilidade em toda a aplicação pois as alterações e correções necessárias são implementadas e liberadas em apenas uma função por vez [Cloudfare 2022].

Com uma infraestrutura Serverless, as aplicações são inerentemente escaláveis pois possum escalabilidade automática sem ser necessário nenhuma configuração pelo usuário, como acontence em aplicações de máquina virtual que é necessário dimensionar os recursos físicos da máquina e *load balancers* para escalarem a aplicação em mais instâncias. A escalabilidade de uma aplicação é extremamente importante para que o servidor não deixe de funcionar. Por exemplo, se a quantidade de requisições de um servidor aumentar drasticamente, apenas uma imagem ou instância da aplicação pode ser que não consiga dar conta de atender todas as requisições e por isso novas instâncias devem ser criadas e inicializadas conforme a demanda. E caso uma aplicação não tenha essa gerenciamento de instâncias, o servidor corre o risco de atingir seu armazenamento interno ou consumo de memória e pode simplesmente sair do ar porque parou [Cloudfare 2022].

As aplicações executadas sob serviços Serverless são cobradas apenas quando utilizadas, chamado de plano *pay-as-you-go*. A empresa ou o desenvolvedor paga apenas pela utilização do seu sistema, ou seja, quanto mais a aplicação é utilizada, mais se paga por ela, mas por outro lado, se a utiliza pouco, paga-se menos. O modelo de cobrança se difere bastante em arquiteturas tradicionais, onde os desenvolvedores precisam analisar e projetar com antecedência os recursos que precisam ser alocados, como capacidade de memória e armazenamento, e serviços de escalabilidade, no qual são cobrados pelo provedor de nuvem toda vez, independente da sua utilização ou não. E se não for projetado por um profissional experiente, há riscos das configurações estarem incorretas ou não serem suficientes, podendo gerar problemas com o servidor, como sair do ar. Essa forma de cobrança faz com que em muitos casos se pague menos pelo serviço Serverless do que em outros modelos tradicionais, como máquinas virtuais e Kubernetes onde o pagamento é conforme alocação de recursos e máquinas, nos quais estão alocados a todo momento [Cloudfare 2022].

3.3. Desvantagens

Apesar dos recursos presentes nessa arquitetura, possui algumas desvantagens em alguns casos e cenários, conforme apresentadas a seguir:

- Testes e depuração;
- Segurança da infraestrutura;
- Custos a longo prazo;
- Inicialização da aplicação.

Em situações de problemas ou erros em que precisa testar e depurar o código, é mais complicado de conseguir replicar um ambiente sob arquitetura Serverless para analisar como o código se comporta. A depuração do código é mais difícil de ser realizada porque os desenvolvedores não conseguem ter nenhum acesso aos processos do servidor e também porque a aplicação é separada em diversas funções independentes menores e talvez seja necessário ir de função em função para encontrar o problema [Cloudfare 2022].

Como a camada de servidor é uma caixa preta que os desenvolvedores não conseguem ter acesso, há algumas preocupações referente a segurança por não ser possível verificar completamente toda a segurança da infraestrutura, o que pode ser um problema em casos bem sensíveis onde precisa lidar com dados pessoais ou dados confidenciais. Como nesse tipo de arquitetura não é alocado uma máquina inteira para um cliente, geralmente o servidor é alocado em uma máquina que executa outros serviços e aplicações compartilhadas com outros clientes, conhecido como *multitenancy*. O compartilhamento entre clientes do mesmo recurso pode afetar a performance e se não configurado corretamente pelo provedor, pode causar exposição de dados indevidamente [Cloudfare 2022].

Apesar de atenderem bem em configurações inicias para executar uma aplicação, a arquitetura Serverless não é pensada para sistemas e aplicações de longo prazo. Como os provedores cobram pelo tempo de utilização e execução do código, ao longo do prazo pode ser que fique mais caro executar a aplicação nesse tipo de infraestutura do que em uma infraestrutura tradicional [Cloudfare 2022].

Para economia de custos e consumos de recursos ociosos, o servidores são desativados quando uma aplicação não é utilizada por um longo período, e assim que for chamado o servidor precisa ser reiniciado. Essa técnica é conhecida de *cold start* e pode afetar o tempo de resposta da requisição, no qual pode gerar impactos caso o tempo de resposta seja muito relevante para a aplicação e a mesma costuma ficar inativa frequentemente [Cloudfare 2022].

3.4. Quando deveria ser adotado

Quando se deseja diminuir o tempo de implementação da aplicação e que esssa possa ser expandida e atualizada rapidamente, a arquitetura Serverless é uma boa opção. Esse modelo pode reduzir custos e evitar desperdícios de recursos para executar aplicações de uso inconsistente, onde há períodos de pico e horários de baixo ou nenhum tráfego por possuir escalabilidade automática e *cold start* [Cloudfare 2022].

3.5. Quando deveria ser evitado

Para aplicações de grande porte, com uma carga de trabalho razoavelmente constante e previsível, as configurações tradicionais provavelmente tem um menor custo para implementar e gerenciar, mas será necessário profissionais na área para realizar as configurações necessárias. Outro ponto é a dificuldade de migrar códigos legados para esse novo tipo de infraestrutura pois utiliza-se funções independentes para executar toda a aplicação, ou seja, dificilmente consiga implementar um monolítico nessa arquitetura porque muito código teria que ser reescrito ou refatorado [Cloudfare 2022].

4. Aplicação Serverless

Para aplicar o conhecimento adquirido com este trabalho, foi desenvolvido uma aplicação simples na linguagem de programção Java e implementado em arquitetura Serverless no serviço AWS Lambda. O objetivo principal desa prática foi aprender e utilizar um provedor de nuvem, neste caso AWS, para implementar e disponibilizar uma aplicação com as vantanges e facilidades que a arquitetura Serverless oferece.

O desenvolvimento se deu por partes, onde inicialmente foi desenvolvido o projeto em código e gerado um arquivo executável e depois configurado o ambiente do serviço AWS Lambda e implementação da aplicação.

4.1. Contexto da aplicação

O contexto geral da aplicação é retornar a data e hora conforme a região informada pelo requisitador. Em formato JSON, o usuário precisa passar o parâmetro ‘region’ e a respectiva região. A aplicação foi desenvolvida na linguagem Java 11 e configurado para ser utilizado no serviço AWS Lambda, conforme abordado na seções seguintes deste capítulo.

4.2. Interface Java

O serviço AWS Lambda para linguagem Java requer um ponto de entrada na aplicação para chamar o método no código e executar a requisição. Para isso, precisa adicionar um pacote Maven do serviço Lambda e implemenar a interface ‘RequestHandler’ na classe onde deseja ser chamada ao receber requisições. E para que o serviço Lambda chame o

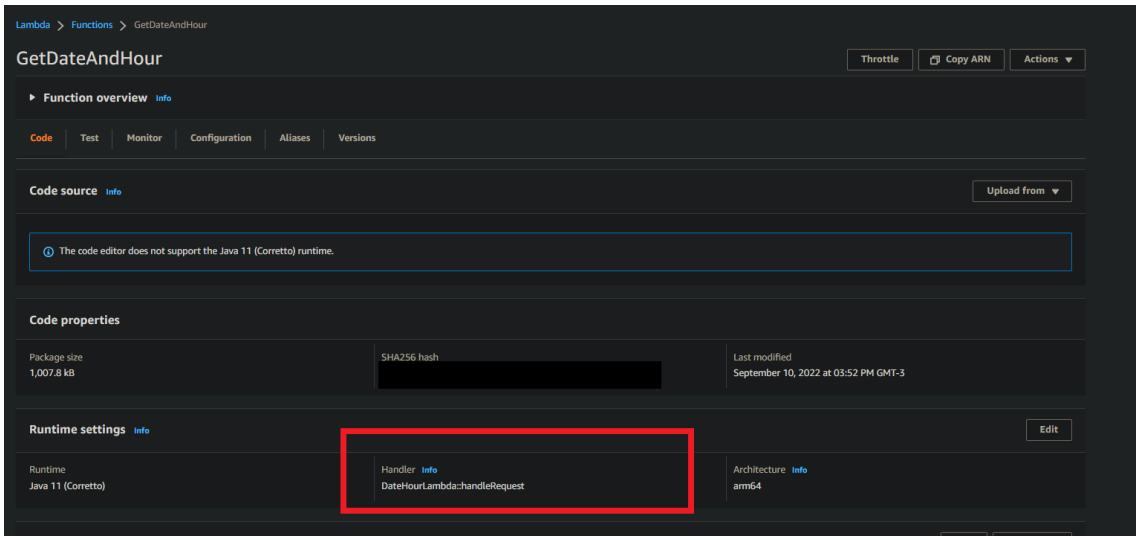
método dessa classe, é necessário implementar o método ‘handleRequest’, onde nele é desenvolvido a lógica e retorno conforme contexto da aplicação.

4.3. Ambiente na AWS Lamda

No serviço AWS Lambda, foi criado uma nova função com arquitetura ‘arm64’ e linguagem de programação Java para executar a aplicação desenvolvida. Com o arquivo .jar executável do projeto desenvolvido, foi realizado o upload deste para a função Lambda e a partir disso, a aplicação já está sendo provisionada pela AWS na arquitetura Serverless.

Após o upload, é preciso configurar o caminho da classe e método que o serviço deve se comunicar ao ser requisitado. Sem essa configuração, a função Lambda criada não consegue se comunicar com a aplicação e irá dar erro ao tentar executar a função. A Figura 2 é uma captura dessa tela de configuração.

Figura 2. Tela de configuração do código da função Lambda



Fonte: Elaborado pelo autor.

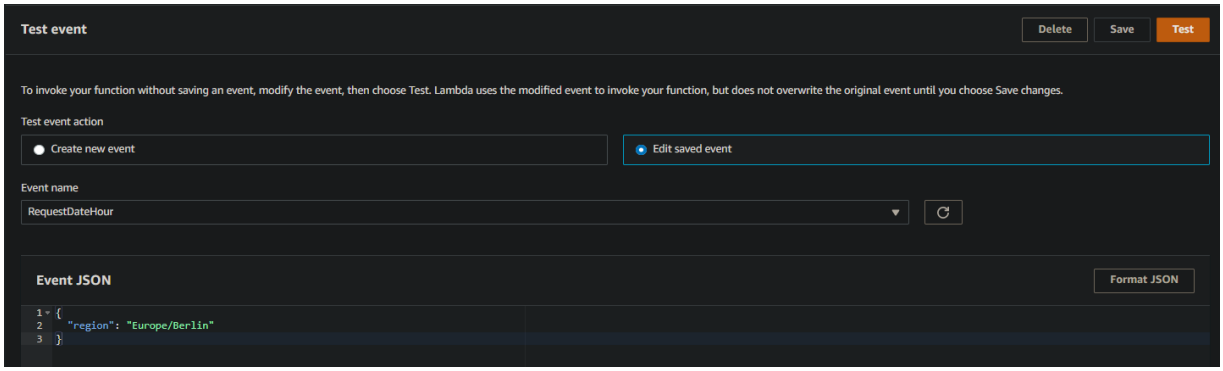
Conforme destacado em vermelho na Figura 2, na configuração ‘Handler’ é infomado o caminho de entrada na aplicação onde a função Lambda consegue se comunicar. ‘DateHourLambda’ é a classe da aplicação e ‘handleRequest’ o método implementado da interface.

4.4. Testes

Ainda nas funções Lambda, tem uma aba chamada de testes onde é possível realizer testes na aplicação enviando eventos no formato JSON e seus respectivos parâmetros esperados, similar a requisições que clientes poderiam fazer em uma aplicação Web, por exemplo.

No teste da Figura 3, o evento JSON tem informado o parâmetro ‘region’ com o valor da região ‘Europe/Berlin’ que é o parâmetro que a aplicação desenvolvida espera receber de argumento.

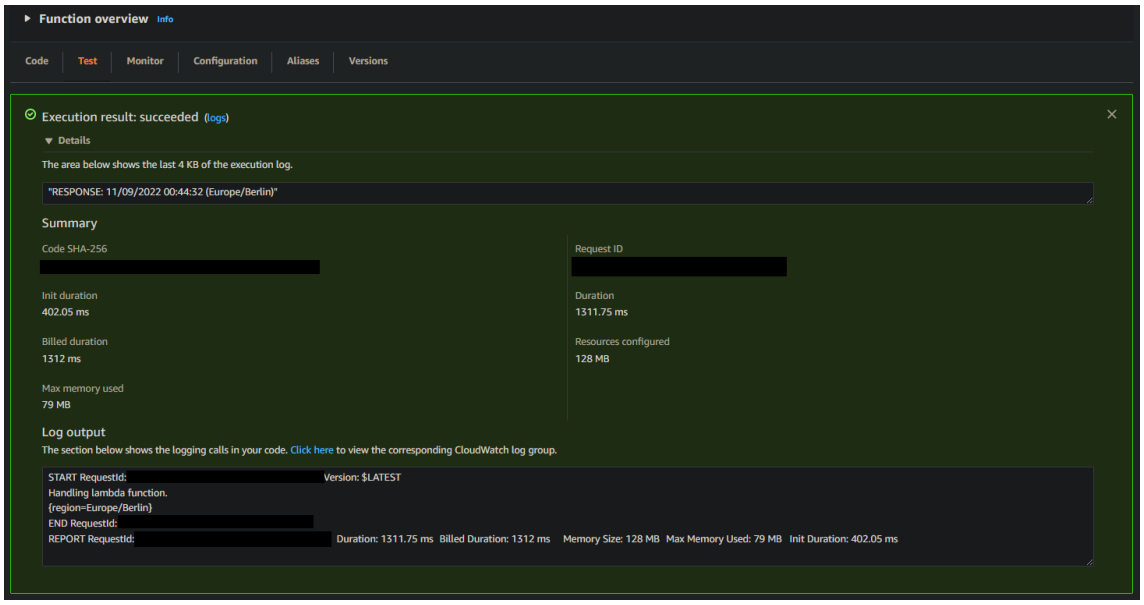
Figura 3. Teste de evento da função Lambda



Fonte: Elaborado pelo autor.

Ao clicar no botão ‘Test’, é realizado a requisição para a função Lambda, na qual chama e executa a aplicação, que por sua vez retorna com sucesso para o cliente a data e a hora da região informada, conforme mostra a Figura 4.

Figura 4. Reposta de sucesso da função Lambda



Fonte: Elaborado pelo autor.

Além do conteúdo da reposta na parte superior da Figura 4, outra informações são apresentadas durante os testes, como o tempo de reposta e a quantidade de memória utilizada. Essas informações são importante pois auxiliam os desenvolvedores a saberem da saúde de sua aplicação e se está sendo executada normalmente.

Caso seja enviado algum dado incorreto ou acontence alguma erro ou exceção durante a execução do código, o console de testes também irá retornar uma mensagem de erro, conforme mostra a Figura 5.

Figura 5. Reposta de erro da função Lambda

Fonte: Elaborado pelo autor.

No caso da Figura 5, foi informado uma região incorreta ‘Europe/Berli’, na qual a aplicação não consegue encontrar e consequentemente emite uma exceção de erro. Os erros que acontencem na aplicação também são retornados com suas mensagens a fim de informar o cliente qual foi o problema que aconteceu.

5. Conclusão

Como citado, a arquitetura Serverless tem o objetivo de facilitar a etapa de implementação de uma aplicação em um provedor em nuvem para tornar a etapa mais rápida. É uma nova abordagem de infraestrutura que pode ser escolhida pelos desenvolvedores ou empresas e se adapta melhor em serviços menores e que sejam escritos já pensados na implementação de arquitetura Serverless. Sua forma de pagamento também se difere um pouco dos serviços tradicionais o que pode atrair clientes em casos de redução de custo.

Há cenários em que essa arquitetura vai ser uma boa esolha e em outros pode ser que não atenda aos requisitos e considerações do projeto ou da empresa. Com o desenvolvimento deste trabalho, a empresa, o desenvolvedor ou time responsável pela implementação das aplicações poderá analisar cada ponto minuciosamente e decidir a melhor decisão arquitetural.

Referências

Amazon (2022) “AWS Lambda”, https://aws.amazon.com/lambda, September.

Amazon (2022) “Cloud computing with AWS”, https://aws.amazon.com/what-is-aws, September.

Amazon (2022) “Serverless on AWS”, https://aws.amazon.com/serverless, September.

Cloudfare (2022) “What is serverless computing? Serverless definition”, https://www.cloudflare.com/learning/serverless/what-is-serverless, September.

Cloudfare (2022) “Why use serverless computing? Pros and cons of serverless”, https://www.cloudflare.com/learning/serverless/why-use-serverless, September.

Gillis, A. S. (2020) “Amazon Web Services (AWS)”, https://www.techtarget.com/searchaws/definition/Amazon-Web-Services, April.

Nginx (2022) “API Gateway”, https://www.nginx.com/learn/api-gateway, September.