UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
COORDENADORIA DO CURSO DE ENGENHARIA DE SOFTWARE

GABRIEL SOUZA DE PAULA

AVALIAÇÃO DE SERVIÇOS SERVERLESS: UM EXPERIMENTO PILOTO

TRABALHO DE CONCLUSÃO DE CURSO

DOIS VIZINHOS

2018

GABRIEL SOUZA DE PAULA

AVALIAÇÃO DE SERVIÇOS SERVERLESS: UM EXPERIMENTO PILOTO

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia de Software, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Rafael Alves Paes de Oliveira

Co-orientador: Prof. Dr. Gabriel Costa Silva

DOIS VIZINHOS

2018

TERMO DE APROVAÇÃO

Avaliação de Serviços Serverless: Um Experimento Piloto

por

Gabriel Souza De Paula

Este Trabalho de Conclusão de Curso foi apresentado em 22 de Junho de 2018 como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software. O(a) candidato(a) foi arguido(a) pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Rafael Alves Paes de Oliveira
Presidente da Banca

Andre Roberto Ortoncelli
Membro Titular

Fernando Luiz Prochnow Ramme
Membro Titular

* A Folha de Aprovação assinada encontra-se na Coordenação do Curso

RESUMO

Souza de Paula, Gabriel. AVALIAÇÃO DE SERVIÇOS SERVERLESS: UM EXPERIMENTO PILOTO. 57 f. Trabalho de Conclusão de Curso – Coordenadoria do Curso de Engenharia de Software, Universidade Tecnológica Federal do Paraná. Dois Vizinhos, 2018.

Este estudo exploratório investiga dois serviços de computação em nuvem sem servidor para identificar e comparar suas características e maturidade. *Serverless* emergem como tecnologia de ponta para implementar microsserviços. Contudo, serviços *serverless* possuem diferenças significativas em relação a (i) características, (ii) preços e (iii) maturidade. Identificar estas diferenças é importante na decisão de adoção do serviço. Primeiramente, são identificados os serviços *serverless* da categoria de Funções como Serviço (FaaS) mais relevantes na atualidade e em seguida foram selecionados dois, o Aws Lambda e o Azure Functions. Em segundo lugar, foi feitu um estudo dos serviços e em seguida alguns recursos dos serviços escolhidos foram comparados. Após, foi realizado um experimento piloto para identificar a facilidade de uso dos serviços na perspectiva do desenvolvedor, sem o objetivo de generalização de resultados, foi realizado como primeira tentativa de validar uma possível hipótese que tinha o foco identificar se o esforço (com base no tempo) para criação e alterações de funções é o mesmo ou não, independente da plataforma escolhida, para isto foi realizada uma atividade de experimentação que teve baixa aderência de participantes. Por fim a experimentação é analisada e seus resultados são restritos à um relacionamento entre feedback dos participantes com relação à utilização das plataformas, seu conhecimento obtido após o treinamento e na justificativa do tempo de realização de tarefas experimentais.

Palavras-chave: Microsserviços, Cloud, FaaS

ABSTRACT

Souza de Paula, Gabriel. *SERVERLESS* SERVICES EVALUATION: A PILOT EXPERIMENTATION. 57 f. Trabalho de Conclusão de Curso – Coordenadoria do Curso de Engenharia de Software, Universidade Tecnológica Federal do Paraná. Dois Vizinhos, 2018.

This exploratory study investigates two serverless cloud computing services to identify and compare their characteristics and maturity. Serverless emerge as the leading technology for implementing microservices. However, services *Serverless* have significant differences in relation to (i) characteristics, (ii) prices and (iii) maturity. Identifying these differences is important in the decision to adopt the service. First, the serverless services of the most relevant Function-as-Service (FaaS) category are identified today and then two, the Lambda Aws and the Azure Functions, were selected. Secondly, a study of the services was made and then some resources of the services chosen are compared. Afterwards, a pilot experiment was performed to identify the ease of use of the services from the perspective of the developer, without the objective of generalization of results, was carried out as the first attempt to validate a possible hypothesis that had the focus to identify if the effort (based on time) for creation and change of functions is the same or not, regardless of the platform chosen, for this was performed an experiment activity that had low adherence of participants. Finally the experimentation is analyzed and its results are restricted to a relationship between participants' feedback regarding the use of the platforms, their knowledge obtained after the training and the justification of the time to perform experimental tasks.

Keywords: Microservices, Cloud, FaaS

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Serverless refere-se a uma nova geração de plataformas para a implantação de aplicações na qual o gerenciamento da infraestrutura é abstraído do desenvolvedor. Assim, o desenvolvedor pode se concentrar apenas na lógica de implementação da aplicação e deixar por conta da plataforma *serverless* a responsabilidade de suprir demandas relacionadas às questões operacionais, como escalabilidade e monitoramento de recursos (ADZIC; CHATLEY, 2017).

Recentemente, serviços *serverless* ganharam destaque aparecendo no radar de tendências tecnológicas da *Thoughtworks* (BADRI et al., 2017). O radar é mantido pelo conselho de tecnologia da empresa de software *Thoughtworks* [1], e visa agrupar tendências tecnológicas que impactam significantemente na indústria de software. Adicionalmente, serviços *serverless* apareceram na pesquisa com desenvolvedores do *Stack Overflow* [2] no ano de 2017 como a 2ª plataforma mais explorada pelos desenvolvedores (STACKOVERFLOW, 2017). Por fim, plataformas *serverless* têm sido consideradas como uma tecnologia conveniente para a implementação de microserviços (CASTRO et al., 2017). Microserviços é uma arquitetura para o desenvolvimento de aplicações na qual uma aplicação é constituída por um conjunto de pequenos serviços independentes. Cada um dos serviços é executado em seu próprio processo e, frequentemente, tem seu próprio banco de dados (NAMIOT; SNEPS-SNEPPE, 2014).

A Amazon Web Services[3] (AWS), plataforma da Amazon[4] para o provimento de serviços em nuvem, lançou o primeiro serviço *serverless* em nuvem no final de 2014. A Microsoft®também seguiu essa tendência e lançou seu serviço *serverless* em seguida (BALDINI et al., 2017). Contudo, a similaridade entre os serviços *serverless* fornecidos por AWS e Microsoft dificulta o processo de decisão de qual serviço adotar. Esse processo de decisão é importante pois uma vez escolhida a plataforma, os aplicativos desenvolvidos dependem tecnicamente da plataforma para sua execução, o que dificulta a migração do

[1] http://www.thoughtworks.com
[2] http://stackoverflow.com
[3] http://aws.amazon.com
[4] http://www.amazon.com

aplicativo para outras plataformas. Por isso, é fundamental identificar as principais diferenças entre as plataformas *serverless*, considerando não apenas suas características, mas também o ponto de vista do desenvolvedor.

Este estudo investigou a seguinte questão de pesquisa: *"Como os serviços serverless podem ser comparados entre si?"* Para a condução deste estudo, as seguintes hipóteses foram estabelecidas:

- Os serviços *serverless* variam significativamente em relação aos seus preços mas pouco em relação às suas características;
- O esforço para criar e alterar código FaaS varia entre as plataformas escolhidas.

O trabalho teve como foco principal a execução de um experimento piloto para identificar a facilidade de uso dos serviços na perspectiva do desenvolvedor sem objetivo de generalização de resultados, tendo em vista que a quantidade de dados obtidas no experimento foi insuficiente para uma análise que chegassem a resultados mais precisos, por isso, foi realizado uma primeira tentativa de validação das hipóteses.

1.1 OBJETIVOS

1.1.1 OBJETIVO GERAL

O objetivo geral deste trabalho foi identificar e comparar as características e maturidade dos serviços *serverless: AWS Lambda e Microsoft Azure Functions.*

1.1.2 OBJETIVOS ESPECÍFICOS

Do objetivo acima, derivaram-se os seguintes objetivos:

1. Apresentar características das plataformas *serverless* estudando suas funcionalidades;
2. Comparar qualitativamente as características dos serviços *serverless*;
3. Apresentar resultados de um experimento com desenvolvedores para identificar a facilidade de uso dos serviços *serverless* mediante o esforço aplicado e o feedback dos mesmos com relação às plataformas.

1.2 ORGANIZAÇÃO DO TRABALHO

O restante deste trabalho encontra-se organizado da seguinte forma:

O Capítulo 2 introduz conceitos fundamentais de serviços *serverless* e os dois serviços investigados neste estudo, onde inicialmente foram identificadas algumas características com base na literatura de *serverless*, em seguida foi especificado o foco em Funções como Serviço (Seção 2.2.3) e identificado suas características que foram comparadas com base em suas funcionalidades (Seção 2.3).

O Capítulo 3 lista os materiais e métodos aplicados na condução deste estudo.

Em seguida o Capítulo 4 apresenta os resultados da experimentação e os discute.

Por fim o Capítulo 5 apresenta as conclusões para este estudo.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo introduz conceitos fundamentais de *microsserviços*, *serverless*, funções como serviço (FaaS) e os dois serviços investigados neste estudo. Primeiramente, a Seção 2.1 introduz os conceitos fundamentais para o entendimento a respeito da arquitetura de microsserviços apresentando suas características em comparação às da arquitetura monolítica, após, a Seção 2.2 aborda a respeito do paradigma *serverless*, destacando suas características juntamente aos seus dois modelos *BASS (Backend como serviço)* e FaaS, a subseção 2.2.1 abordará alguns benefícios na utilização de *serverless* e a subseção 2.2.2 falará de algumas limitações, como o foco deste trabalho é em FaaS a subseção 2.2.3 abordará especificamente a respeito do mesmo. Por fim, a Seção 2.3 apresenta uma análise das funcionalidades dos serviços escolhidos para este estudo *AWS Lambda* e *Azure Functions*.

2.1 CONCEITOS DE *MICROSSERVIÇOS*

Microsserviços é uma arquitetura para o desenvolvimento de aplicações na qual uma aplicação é constituída por um conjunto de pequenos serviços independentes que são executados em seus próprios processos (NAMIOT; SNEPS-SNEPPE, 2014), comumente comunicam-se através de mecanismos leves, geralmente através de uma Interface de Programação de Aplicação (API) que use o Protocolo de Transferência de Hipertexto (HTTP). Comumente estes serviços são publicados de maneira independente através de processos de implantação automatizados (LEWIS; FOWLER, 2014).

Em contrapartida aos microsserviços tem-se as aplicações baseadas na arquitetura monolítica, que caracterizam-se por agruparem seus componentes em uma única unidade que é executada em um único processo (LEWIS; FOWLER, 2014). Sistemas de software monolíticos tornam-se difíceis de serem mantidos devido ao aumento de complexidade, atividades de desenvolvimento, teste e implantação tornam-se um fardo tendo em vista que uma única alteração acarreta na re-implantação de todo o software (FOWLER, 2017).

Na Figura 1 podem ser observados exemplos de um ambiente de aplicação monolítico

à esquerda e de um ambiente baseado em microsserviços à direita.

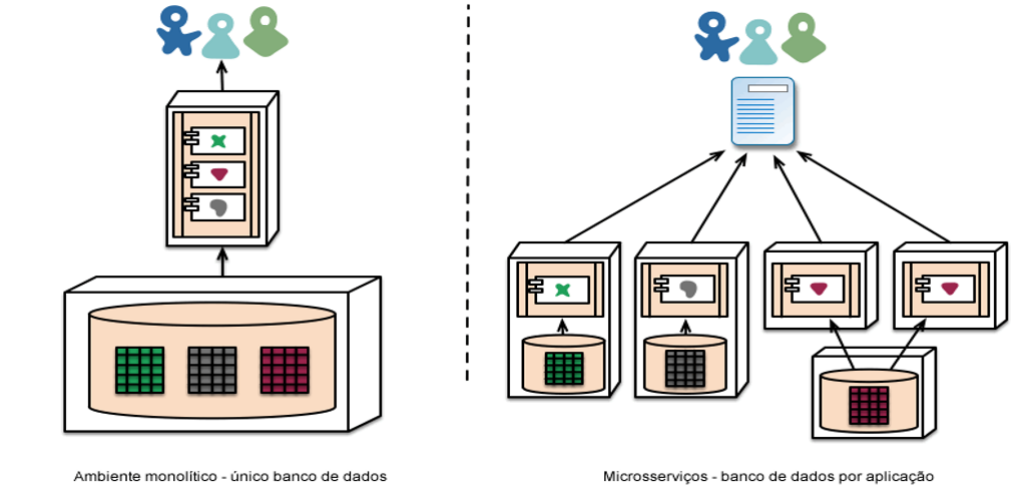

Figura 1: Ambiente monolítico e ambiente de microsserviços

Fonte: (LEWIS; FOWLER, 2014)

2.2 CONCEITOS DE *SERVERLESS*

Aplicações de arquitetura *serverless* são aquelas que fazem uso de serviços de terceiros para realizar tarefas que são tradicionalmente realizadas por servidores, quebrando assim a ideia de um cliente comunicando-se exclusivamente com um único e específico servidor que possui toda lógica de negócio, lógica de aplicação e integrações. O modelo tradicional pode ser observado no Item 1 da Figura 2. Ao quebrar um modelo arquitetural tradicional, observa-se que os controles de fluxos de aplicação são repassados para a camada cliente que integra-se via API com serviços que comumente são de terceiros (Item 2), no modelo *serverless* denominado BaaS os serviços integráveis podem fornecer lógica estrutural como a de autenticação (Item 3), lógica de negócios (Item 4), acesso à base de dados hospedadas na Web (Item 5) e outros recursos, tem como por exemplo o *Firebase*[1] serviço BaaS da Google (JANAKIRAMAN, 2016). Caso seja necessário a manutenção de lógica de negócio ou o encapsulamento da base de dados no lado do servidor, pode ser utilizado o segundo modelo, o FaaS, a utilização desse representa na arquitetura sem servidor a substituição de servidores de longa duração por serviços de poder computacional de curta duração com utilização por demanda (BADRI et al., 2017), na qual a lógica é implementada em funções que são implantadas direto em plataformas (Item 6).

Serverless possuem benefícios e limitações ambos são abordados respectivamente nas

[1] `https://firebase.google.com/?hl=pt-br`

subseções 2.2.1 e 2.2.2, além disso, como foi citado anteriormente, *serverless* dividem-se em duas áreas, *Backend* como serviço (BaaS) e Funções como serviço (FaaS), a subseção 2.2.3 tratará especificamente de FaaS devido o mesmo ser o foco deste trabalho.

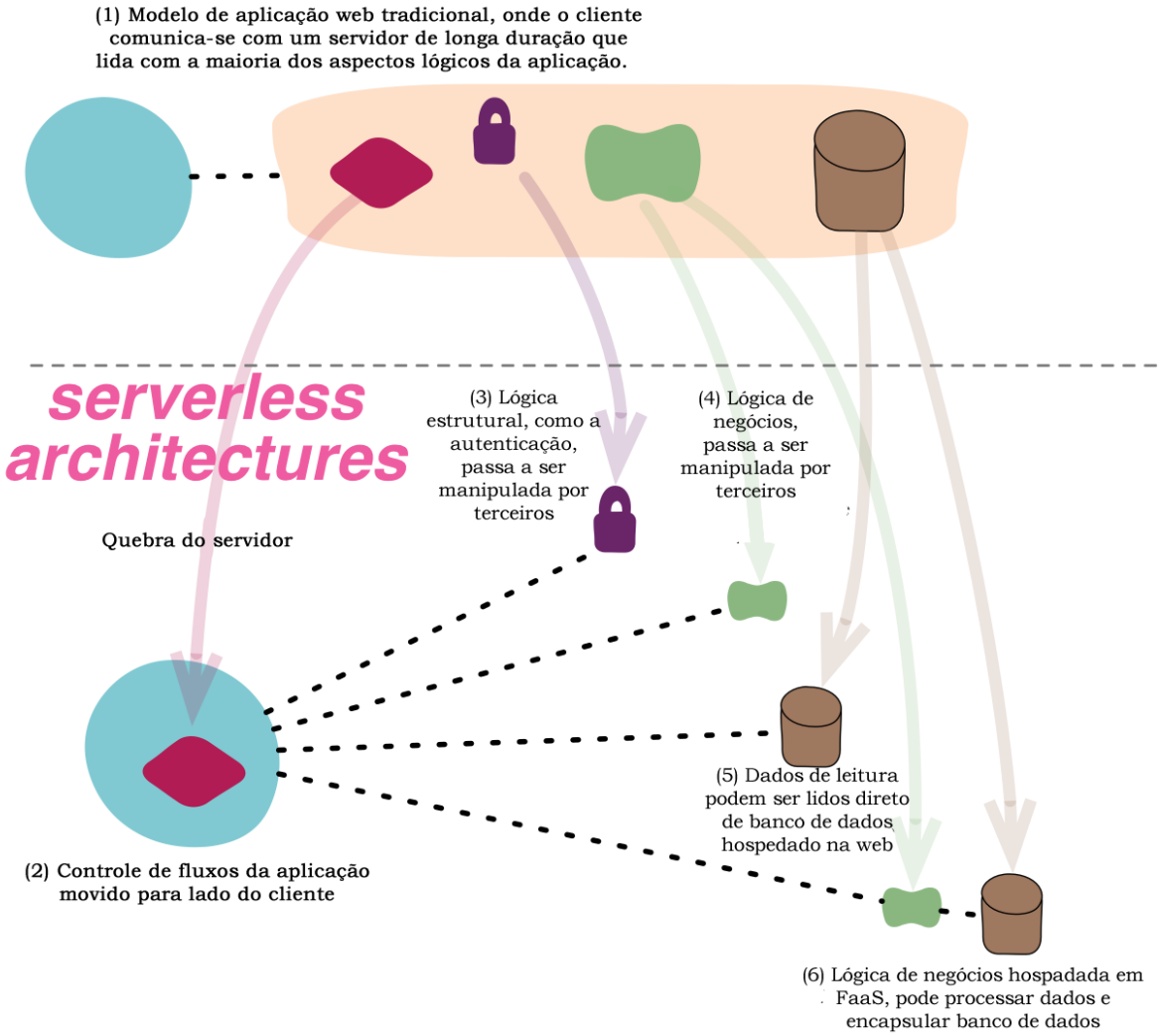

Figura 2: De arquitetura tradicional para *serverless*

Fonte: (JANAKIRAMAN, 2016)

2.2.1 BENEFÍCIOS AO UTILIZAR *SERVERLESS*

De acordo com Roberts e Chapin (2017) *Serverless* tem apresentado benefícios relacionados a:

1. Diminuição de custos de mão de obra: Esta diminuição pode ser ocasionada tanto pela redução de trabalhos operacionais como de gerenciamento de servidores e atualizações de sistemas operacionais, quanto pela redução de lógica de desenvolvimento, tendo em vista que a utilização de serviços de terceiros substitui a necessidade de implementação de serviços do zero;

2. Riscos reduzidos: Comumente as equipes que mantêm software funcionando trabalham com riscos de falhas e tempo de indisponibilidade, com serverless se reduz significativamente o número de tecnologias que se operam diariamente, onde, dependendo da plataforma contratada os riscos de indisponibilidade serão reduzidos.

3. Redução de custos de recursos: Normalmente quando se provisiona recursos como Memória RAM ou CPUs para utilização por parte das aplicações, existe um risco de excesso de provisionamento que acarreta em gastos que poderiam ser evitados, com *serverless* os recursos são adicionados sob demanda, reduzindo assim os custos.

4. Otimização impulsiona economia imediata de custos: Como o nível de cobrança passa a ser por recursos utilizados a tendência é que os serviços sejam desenvolvidos com o foco na qualidade de seu código, buscando a optimização no processamento, principalmente no que se refere à FAAS.

5. Maior flexibilidade de dimensionamento: Flexibilidade de dimensionamento de acordo com a necessidade de uso, principalmente no que se refere à escalabilidade automática.

Eivy (2017) cita que para utilizar-se do benefício de economia de custos que FaaS oferece, a implementação de serviços de forma quebrada em várias funções pequenas pode ser útil, pois cada uma desfrutará de seu nível gratuito.

2.2.2 LIMITAÇÕES DE *SERVERLESS*

Ainda de acordo com Roberts e Chapin (2017), *serverless* possui dois tipos de limitação que são as intrínsecas e as de implementação. As intrínsecas são definidas por:

1. Estado: Componentes serverless são efetivamente sem estado, ainda que serviços possam se comunicar com outros serviços para trocar informações e persistir as mesmas em alguma base de dados;

2. Latência: No que se refere à FaaS, muita comunicação entre componentes ocorre via API HTTP, que pode ser mais lenta que outros tipos de transporte de informação;

3. Testes Locais: A realização de testes locais é uma das maiores limitações da arquitetura sem servidor, desenvolvedores geralmente tem análogos locais de componentes de aplicativos (como bancos de dados) que podem ser integrados para testes da mesma forma que o aplicativo pode ser implantado em produção. Mesmo assim, é difícil obter um cenário de desenvolvimento que reflita completamente o de produção;

4. Perda de controle: Muitas limitações serverless estão relacionadas à realidade de implementação das plataformas FaaS ou BaaS que são mantidas e disponibilizadas por terceiros.

Já os limites relacionados à implementações podem ser definidos por:

1. Inicialização Fria: Em FaaS, existe um problema desempenho devido à chamada inicialização fria, como por exemplo no AWS Lambda, toda primeira inicialização de funções ocorre um processo de inicialização de *containers* que sobem toda estrutura para execução da função, uma vez inicializada a função fica em estado quente e as próximas requisições obtêm tempo de retorno menor, com um determinado tempo o *container* volta para o estado frio;

2. Limitações de Ferramentas: Devido à tecnologia serverless ser recente, não existem ferramentas bem definidas ou maduras voltadas para implantação, gerenciamento e desenvolvimento, além disso, não existem ainda especificações de padrões definidos como melhores práticas;

3. Vendor *lock-in*: O aprisionamento por parte dos fornecedores serverless também é caracterizado como limitação. Sendo esse aprisionamento determinado de acordo com a quantidade de recursos de uma mesma plataforma que sejam integrados e utilizados.

2.2.3 FUNÇÕES COMO SERVIÇO

Funções como Serviço, tratam a partir da perspectiva de execução de código de backend em plataformas de terceiros. Essas funções possuem eventos de entrada

disponibilizados pelas plataformas que executam uma implementação que pode ser escrita em linguagens também definidas pelas plataformas, em seguida essas funções executam suas rotinas estabelecidas por desenvolvedores como pode ser observado na Figura 3.

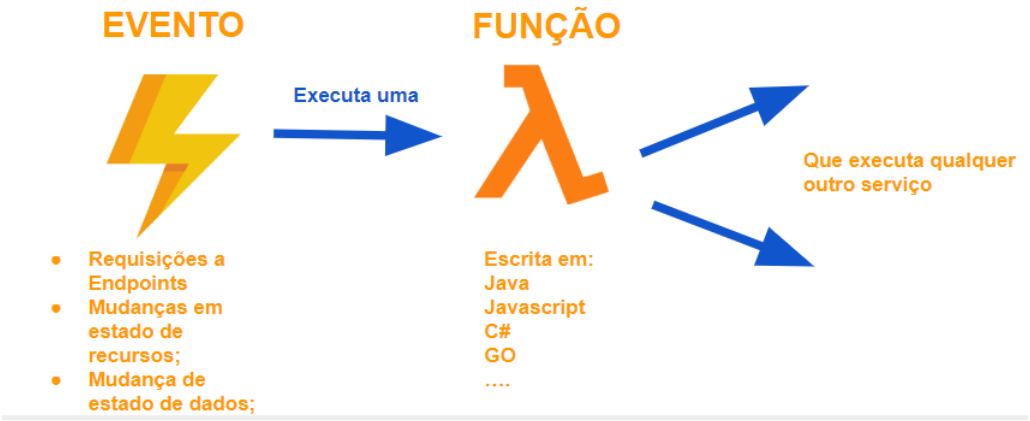

Figura 3: Processo de funcionamento de FaaS

Fonte: Autoria própria.

De acordo com Roberts (2016), FaaS possuem algumas características que as definem conceitualmente , como:

- A execução de código fonte sob demanda e sem necessidade de servidores dedicados para tal, devido a responsabilidade estar sobre a plataforma;
- A implementação não é presa a apenas uma linguagem de programação, com a opção de escolha de linguagem de acordo com as possibilidades que as plataformas oferecem;
- Escalabilidade horizontal é completamente automática, com o processamento de múltiplas execuções em paralelo sem complicações;
- FaaS são acionadas de acordo com eventos que são disponibilizados pelo provedor, como por exemplo uma função que seria executada por meio de um agendamento, ou até mesmo, disparada através de uma API que use o Protocolo HTTP; e
- Nenhum estado fica na memória para re-utilização, cada invocação de função é singular e nada dela será acessada pela invocação de uma função posterior.

2.3 COMPARAÇÃO DE RECURSOS DOS SERVIÇOS *SERVERLESS*

Nesta seção é feita uma comparação de recursos relacionados à linguagens de implementação disponíveis, limitações e integrações das plataformas Aws Lambda e Azure

Functions.

Cada serviço oferece suporte a diferentes linguagens de programação (Tabela 1), o *Azure Functions* oferece suporte a mais linguagens de programação que o AWS Lambda, totalizando sete contra cinco para a outra empresa, porém nem todas as linguagens estão disponíveis para utilização de forma confiável, de acordo com a documentação oficial a empresa Microsoft®adota três níveis de suporte (MICROSOFT, 2017b):

1. Geralmente disponível (GA): Com suporte total para uso em produção;
2. Versão prévia: Ainda não tem suporte, mas é esperado alcançar o status de GA no futuro;
3. Experimental: Não possui suporte e pode ser abandonado no futuro.

Atualmente a plataforma da Microsoft®possui duas versões, tendo C#, F# e NodeJS na primeira versão com suporte GA e PHP e *Phyton* em nível experimental, na segunda versão todas as linguagens estão em status de versão prévia com exceção F#, PHP e Phyton que não estão disponíveis, de acordo com a empresa as diferenças entre as versões são de melhoria nos recursos de integração com suporte melhorado no que se refere à desenvolvimento e testes locais. Já o AWS Lambda não informa se suas linguagens possuem níveis de suporte diferenciados, porém podem oferecer suporte à diferentes versões da mesma linguagem e também possibilitando o desenvolvimento em máquina local.

De acordo com uma pesquisa realizada no ano de 2017 pelo site StackOverflow, as linguagens Java, Javascript, Python, PHP, C# e GO aparecem como as linguagens de programação mais populares entre os desenvolvedores profissionais, sendo Javascript e Java as linguagens de maior destaque (STACKOVERFLOW, 2017). Observa-se que as linguagens destacadas pela pesquisa são as que as plataformas oferecem suporte, com exceção F#.

Tabela 1: Suporte a linguagens de programação

Linguagens	AWS Lambda	Microsoft Azure
C#	x	x
F#		x
Java	x	x
Javascript	x	x
PHP		x
Python	x	x
Go	x	

Fonte: Autoria própria.

Para cada função criada existem limitações de recursos, instâncias e invocações que os serviços impõe, como pode ser observado na Tabela 2, para ambas as plataformas foi analisado a nível pagamento pelo uso, tendo em vista que o Azure oferece outro plano de consumo baseado em assinatura com pagamento fixo mensal. No que se refere ao limite de utilização de memória RAM, cada função Aws Lambda dispõe o valor máximo de utilização de 3048 MB e com relação ao Microsoft Azure o limite é de 1536 MB, ambos os serviços possuem a configuração mínima de 128 MB de memória RAM a ser alocada para utilização e também apresentam um tempo limite de execução, sendo de 300 segundos o Lambda e 600 segundos o Azure, esta limitação é importante devido à cobrança ser pela utilização de recursos, caso sejam ultrapassadas as limitações de execução tanto de memória quanto de tempo de execução a execução é encerrada pelas plataformas.

Com relação a limitações de execução simultâneas, a AWS deixa claro que o Lambda apresenta o limite de 1000 execuções por região, na empresa os serviços são disponibilizados em diferentes regiões do planeta, por outro lado no Azure existe também a especificação de região de disponibilização porém não especifica limites de execução por região, mas sim por aplicativos de funções, que funcionam como agrupamento lógico de funções, cada aplicativo é limitado à 200 instâncias simultâneas, cada instância pode processar mais de uma solicitação por vez, por tanto não há um limite definido em número de execuções simultâneas (MICROSOFT, 2017a).

Em cada função existe um tamanho máximo que pode se ter no pacote de implantação, abrangendo a implementação e as bibliotecas adicionais, a Aws especifica 50MB e por região somando o espaço em disco utilizado de código em todas as funções o limite é de 75 GB, já o Azure não deixa claro em sua documentação limitações de pacote de implantação ou por espaço alocado em aplicativos de funções.

Tabela 2: Limitação de recursos por função

Tipos de Limitações	AWS Lambda	Microsoft Azure
Memória Ram Inicial(MB)	128	128
Máximo Memória Ram (MB)	3048	1.536
Duração max. execução (S)	300	600
Limite de Execuções Concorrentes	1000	Não Especificado
Tam. Pac. de Implantação (MB)	50	Não Especificado
Tam. Total Funções (GB)	75	Não Especificado

Fonte: Autoria própria.

As plataformas comumente oferecem uma concessão gratuita por mês no que se refere à quantidade de execuções das funções, sendo que em todas este nível gratuito que cobre um

milhão de execuções (Tabela 3).

Além disso, é oferecido uma concessão gratuita para realização de testes, a AWS se destaca por oferecer um ano grátis para testes em todas as ferramentas porém com limitação de custos à 300 dólares, o Aws Lambda possui concessão gratuita que é tida por 1 milhão de execuções ao mês. Já a plataformas da Azure para FaaS também oferece 1 milhão de execuções mensais grátis, o período de testes da plataforma é de no máximo 30 dias, com 200 dólares disponível para utilizar em qualquer serviço.

Tabela 3: Concessões gratuitas oferecidas pelas plataformas

Plataforma	Concessão Gratuita(Por mês)	Concessão Gratuita Para Teste
AWS Lambda	1 Milhão de execuções	300$
Microsoft Azure	1 Milhão de execuções	200$ para 30 dias de teste

Fonte: Autoria própria.

As plataformas oferecem também integrações com diferentes recursos, como APIs HTTP e agendadores de eventos, neste estudo foram utilizados alguns serviços das plataformas para realização de integração com os serviços FaaS. Do Amazon Lambda para integração via api HTTP foi utilizado o *Api Gateway*[2], neste serviço se criam links de acesso e determinam-se métodos HTTP que serão utilizados na API e que fazem o disparo de funções FaaS. Já na Azure o recurso de disparo via HTTP é o HTTPTrigger, quando selecionado é associado direto na função, possibilitando ajuste do link e escolha também do método HTTP.

O Evento de agendamento no Lambda é configurável direto no serviço do *CloudWatch*[3] que dispara a função de acordo com o período programado, já no Azure Functions o TimmerTrigger é configurável direto na função, em ambas as plataformas utiliza-se a programação de eventos através de uma expressão do tipo Cron para definição de frequências de execução(Amazon , 2018).

As integrações com as plataformas FaaS não se limitam à Http e eventos agendados, Eivy (2017) define que muito do poder que esta tecnologia possui vem da capacidade de acionar funções com base em outros tipos de eventos, como por exemplo a execução de eventos ao ser feito upload de arquivos em um servidor de arquivos, como é o caso do S3[4] da AWS, onde o evento dispararia uma função que processa determinada imagem e a envia para determinado repositório.

Com relação à visualização de logs de execução no Lambda é necessário acessar o

[2] `https://aws.amazon.com/pt/api-gateway/`
[3] `https://aws.amazon.com/pt/cloudwatch/`
[4] `https://aws.amazon.com/pt/s3/`

serviço do CloudWatch Logs, onde por cada função serão agrupados os logs das execuções, já na Azure o recurso é atrelado direto na função, sem necessidade de configuração ou acesso à um serviço terceiro.

Foi criado um tutorial que detalha tecnicamente o desenvolvimento de funções nas plataformas AWS Lambda e Azure Functions, o mesmo encontra-se no apêndice A.

3 MÉTODOS

A Seção 3.1 refere-se à escolha das plataformas para realização dessa avaliação, já a 3.2 aborda a respeito de uma análise realizada para comparação de recursos dos serviços utilizados neste trabalho e por fim a Seção 3.3 trata a respeito de uma investigação empírica por meio da aplicação de uma experimentação.

3.1 ESCOLHA E ESTUDO DE PLATAFORMAS

Para escolher quais plataformas FaaS seriam utilizadas neste estudo, primeiramente foi feito uma seleção de ferramentas com base em (BALDINI et al., 2017), que cita quatro plataformas, sendo a Cloud Functions, Azure Functions, Aws Lambda e IBM Openwhisk, após, foi utilizado o serviço Google Trends para realizar uma comparação e identificação dos serviços mais buscados no google, as que se destacaram foram (Figura 4): AWS Lambda em primeiro lugar, Azure Functions em segundo e Cloud Functions em terceiro lugar. Por fim, foram selecionadas as duas ferramentas que apresentaram melhores resultados.

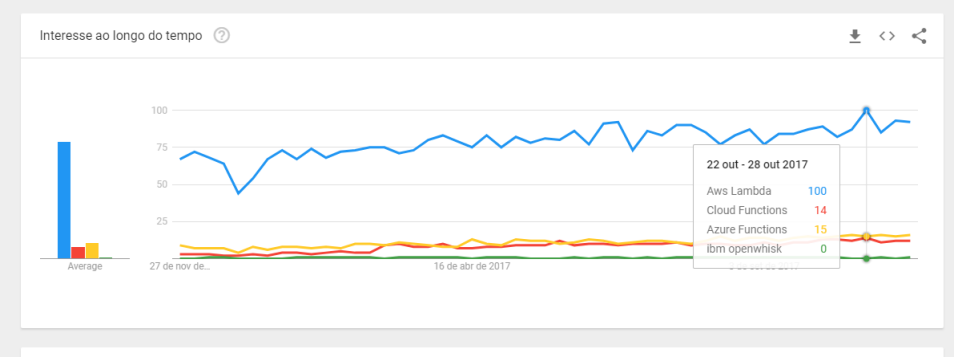

Figura 4: Gráfico com plataformas ao longo do tempo.
Fonte: (GOOGLE, 2017)

Com a escolha das ferramentas foi realizado um estudo das mesmas com base nas suas documentações oficiais, após, foi montado uma apostila que encontra-se no Apêndice A, esta

apostila serviu de material de apoio para realização de treinamento do experimento detalhado na Seção 3.3.

3.2 CRIAÇÃO DE COMPARAÇÃO DE RECURSOS DAS PLATAFORMAS

Com a análise da documentação e das próprias ferramentas, foi iniciado o processo de comparação de recursos dos serviços, comparando as linguagens que eles dão suporte, limitações de recursos e concessões gratuitas de acesso. Comparação esta que está presente na Subseção 2.3.

3.3 INVESTIGAÇÃO EMPÍRICA

Um experimento controlado é um método empírico para determinar a existência de um relacionamento de causa-efeito (WOHLIN et al., 2012) um experimento investiga uma hipótese, essa define uma relação causal entre variáveis dependentes e independentes, neste estudo piloto essa hipótese está definida na Seção 3.3.2.1. Foi usado o *framework* proposto por Wohlin et al. (2012), esse *framework* cobriu a preparação, execução e análise de resultados do experimento no nível esperado por esta pesquisa. Este experimento foi definido como um piloto, realizado com o intuito de ser uma primeira tentativa com relação à validação de suas hipóteses.

3.3.1 ESCOPO E CONTEXTO

A execução do experimento teve como objetivo a análise de duas plataformas FaaS avaliando o esforço de criação e manutenção de serviços para cada plataforma a partir do ponto de vista de desenvolvedores de software. Essa avaliação é realizada no contexto de desenvolvedores de software que estiveram criando, modificando funções e as disponibilizando nas plataformas Azure Functions e AWS Lambda.

3.3.2 PLANEJAMENTO

A subseção 3.3.2.1 aborda a respeito das variáveis e definição de hipótese, já na 3.3.2.2 é abordado a respeito da seleção dos participantes do experimento, a 3.3.2.3 cita a respeito dos instrumentos utilizados para preparar e realizar a experimentação, a subseção 3.3.2.4 aborda a respeito das tarefas desenvolvidas, já a subseção 3.3.2.5 trata sobre a realização de análise dos dados obtidos no experimento e por fim a subseção 3.3.2.6 trata de como foi realizada a execução do experimento.

3.3.2.1 DEFINIÇÃO DE HIPÓTESES E SELEÇÃO DE VARIÁVEIS

De acordo com Wohlin et al. (2012), a base para a análise estatística em um experimento é o teste de hipóteses. Para este trabalho foram adotadas as seguintes hipóteses:

- A hipótese nula: Define que o esforço para criar e alterar código serverless em diferentes plataformas é o mesmo, independente da plataforma.
- A hipótese alternativa: Afirma que o esforço para criar e alterar códigos serverless entre diferentes plataformas é diferente, independente da plataforma.

Neste experimento, foi escolhido como variável independente as plataformas, pelo fato de existir a possibilidade de mudança das mesmas e de seus recursos a serem utilizados, já variável dependente é o tempo medido em minutos que determinará o esforço para realizar alterações e criações de projetos FaaS.

3.3.2.2 SELEÇÃO DE PARTICIPANTES E DESIGN

Foi feita uma parceria com uma empresa de software de grande porte, esta localizada no Sudoeste do Paraná, possuindo mais de 400 funcionários, que forneceu uma sala para realização das atividades tanto do treinamento quanto do experimento, da empresa foram convidados alguns desenvolvedores para realização do experimento . O perfil de cada componente foi detalhado por meio de um formulário de análise de perfil, para assim identificar níveis de conhecimento de cada participante, conhecimento relacionado à programação e aos serviços que foram experimentados, o formulário encontra-se na Seção 4.1.1 juntamente com a resposta por parte dos participantes. O experimento demandou conhecimentos básicos em Javascript. O uso de desenvolvedores profissionais foi importante para aumentar o realismo (SJOBERG et al., 2002 apud SILVA, 2016).

Foi realizado o convite para realização do experimento, dez pessoas participaram da fase de treinamento que é especificada na subseção 3.3.2.6 porém na fase de realização das atividades experimentais seis renunciaram, tarefas essas especificadas na subseção 3.3.2.4. Para aplicação das atividades experimentais foi utilizado o *within-subject design*, onde cada participante realizou a mesma atividade em cada plataforma (SILVA, 2016).

3.3.2.3 INSTRUMENTAÇÃO

Projetos FAAS, plataformas *cloud* e materiais de experimentação são os instrumentos utilizados neste experimento. As plataformas utilizadas nesse experimento foram Amazon Web Services e Microsoft Azure.

Da AWS, alguns serviços foram utilizados para preparação de ambiente e experimentação, como o Gerenciador de Usuários e Acessos IAM para criação de usuários atrelados à uma conta principal, Amazon CloudWatchLogs que é o serviço de logs para acesso ao log e informações das funções criadas, Amazon CloudWatch Events para integração e disparo de funções via agendamento e também foi utilizado o Api Gateway que é o serviço de que possibilita integração e disparo de funções via API Http.

O Amazon Lambda, serviço de computação sem servidor da AWS, no qual é executado código sobre demanda possui um painel gráfico, na qual podem ser obtidas informações gerais a respeito das funções criadas e a partir do mesmo é possível criar novas e gerenciar as já existentes. No experimento foi utilizado o painel gráfico para criação e manutenção de funções e realização de integrações, o que não demandou de configurações de máquinas locais.

O Microsoft Azure Functions também possui um painel gráfico para administração de projetos de funções, que também possibilitou a preparação e a realização do treinamento e experimento no mesmo, facilitando e diminuindo riscos relacionados a configurações locais.

Nesse experimento três projetos de função para cada plataforma foram especificados para entrega das atividades, sendo um voltado para treinamento, outro para realização de alteração em funções nas plataformas e outro era voltado para desenvolvimento. Para a realização das atividades os participantes foram direcionados de forma aleatória para cada plataforma sendo no fim um total de seis projetos entregues.

3.3.2.4 TAREFAS

Como citado na Seção 3.3.2.3, foram estabelecidos três entregáveis para cada plataforma, com finalidade de ser avaliado o esforço de criação e alteração de serviços FaaS. Em um total de seis atividades, as duas primeiras foram aplicadas de forma padrão para todos os desenvolvedores como pré-teste e as outras quatro foram utilizadas para obtenção de resultados oficiais, resultados estes que são explanados na seção 4.1.2.

A primeira tarefa que foi utilizada como pré-teste consistiu no desenvolvimento de uma função em Javascript, que simulava um cadastro de um cliente, na qual deveria ser passados por

parâmetro os dados: nome, email e telefone (Sem integração com base de dados), visando apenas o recebimento da informação na função através da execução de um evento HTTP. Função esta que deveria ser disparada através de uma requisição HTTP, os dados deveriam ser impressos no log e deveria ser estabelecido como retorno a frase 'Usuario NOME cadastrado com sucesso'.

A segunda tarefa consistiu no desenvolvimento de uma função em JavaScript, que funcionava como uma calculadora que passando por parâmetro operações básicas (soma, subtração, adição e multiplicação) e determinados valores (2) a mesma retornava o resultado do cálculo através de uma requisição HTTP. Foram estabelecidos alguns para cumprimento da tarefa:

1. Escolha do projeto modelo necessário para resolução do problema;
2. Configuração de nomes e recursos necessários para a execução;
3. Implementar o algoritmo de calculadora na IDE do console WEB;
4. Realizar integração com api HTTP; e
5. Teste.

A última tarefa envolvia a manutenção de aplicações FaaS, consistia em acessar a função pré determinada e alterar o tempo de execução do agendamento para 2 minutos. Após isso, foi necessário a realização da análise o código e o log de execução, identificando e corrigindo o cálculos caso existisse algum erro de cálculo de notas de alunos, 'importadas' de uma base de dados.

A ordem foi estabelecida apenas para as duas primeiras atividades, que eram as de teste, já as quatro ultimas a ordem foi sorteada, como se evidencia por meio da Tabela 4.

Tabela 4: Ordem das atividades aplicadas

Participante	Ordem
1	1, 2, 3, 4, 5, 6
2	1, 2, 6, 3, 5, 4
3	1, 2, 6, 5, 4, 3
4	1, 2, 5, 4, 3, 6

Fonte: Autoria própria.

3.3.2.5 ANÁLISE DE DADOS

Devido ao tamanho da amostra a análise foi baseada na captura do feedback de utilização e do tempo de realização das atividades dos participantes do experimento.

3.3.2.6 EXECUÇÃO DO EXPERIMENTO

O experimento foi realizado em cinco etapas:

1. Seleção de participantes: O processo de seleção de participantes foi previamente descrito na seção 3.3.2.2. Complementando a descrição, os participantes do treinamento receberam uma identificação única para ocultação de suas identidades, em seguida, foram convidados a responderem um questionário de análise de perfil, as questões foram baseadas em Feigenspan J. et al. (apud SILVA, 2016), recolhendo informações como as de experiência profissional, os dados obtidos neste questionário encontram-se na seção 4.1.1.

2. Treinamento: O treinamento foi realizado para que os participantes adquirissem o conhecimento base para criação e manutenção de projetos nas plataformas FaaS escolhidas, para cada plataforma foi realizado uma sessão de treinamento de uma hora e meia.

3. Pré-teste: O pré teste consistiu em 1 atividade aplicada antes das atividades de experimentação, esta, foi aplicada sobre as duas plataformas, visando a familiarização do participante com relação às atividades experimentais, os resultados desta etapa foram descartados e a tarefa foi executada sem os participantes terem conhecimento do descarte.

4. Experimentação: O experimento foi realizado em uma sessão de duas horas, o experimentador não interferiu na realização das tarefas das atividades. Os dados resultantes da experimentação e sua análise se encontram na seção 4.1.2.

5. Feedback: Quando o experimento terminou, os participantes foram convidados a preencher formulário de feedback, avaliando sua experiência neste experimento. Os dados resultantes do feedback e sua análise se encontram na seção 4.1.

3.3.3 OPERAÇÃO

3.3.3.1 PREPARAÇÃO

Nesta seção, é definido como foi efetuada a preparação dos ambientes cloud para realização do experimento.

A plataforma AWS possui a possibilidade de criação de usuários a partir de uma conta raiz, o que facilitou a configuração de ambientes e permissões de uso, já na Azure foi necessário a criação de contas externas que foram adicionadas a um grupo de trabalho com permissões específicas de acesso ao ambiente cloud, o acesso era apenas para os serviços FaaS e suas ferramentas adjuntas (serviços de logs e gatilhos), cada participante recebeu um usuário devidamente configurado com acesso aos recursos necessários.

Após a criação dos usuários e concessão de acesso aos recursos dos mesmos nas plataformas foram criadas as funções que representassem as atividades de modificação e atribuídas a seus respectivos usuários.

3.4 MATERIAIS

Os recursos utilizados no projeto foram:

1. Acesso aos dois serviços serverless: como apontado na tabela 2 as plataformas oferecem concessões gratuitas de acesso, o trabalho tranquilamente conseguiu ser desenvolvido dentro dos limites apontados na tabela;

4 RESULTADOS

Neste capítulo são abordados os resultados de um experimento piloto (Seção 4.1), sem o objetivo de generalização de resultados, expressando assim uma primeira tentativa de validar uma possível hipótese, explanada na Seção 3.3.2.1, isto devido ao número de pessoas que participaram do experimento ter sido pequeno. Nesta análise de resultados a Seção 4.1.1 traz à tona o perfil dos participantes e suas avaliações com relação às atividades e as plataformas, já na Seção 4.1.2 é discorrido a respeito do tempo gasto na realização das atividades experimentais por cada participante.

4.1 EXPERIMENTO

4.1.1 CARACTERIZAÇÃO E FEEDBACK DOS PARTICIPANTES

Para realização do experimento foram convidados desenvolvedores de software profissionais, a tabela 5 sumariza questões referentes à experiência destas pessoas na área de desenvolvimento de software. De maneira geral, foram aplicadas questões voltadas para auto-avaliação que possuem uma variação de 0 (nível baixo) à 5 (nível alto), questões para obtenção de períodos e confirmação.

Com relação ao grupo de questões pessoais, foi identificado que os participantes possuem 23, 24, 27 e 30 anos de idade, sendo três homens e uma mulher. Todos possuem pós-graduação.

Já com relação a conhecimento técnico e avaliação profissional, é apontado na questão 1 da tabela 5 que todos os participantes já programam de 5 à 10 anos, profissionalmente (questão 2) um programa a menos de 1 ano, outro de 2 à 5 anos e os dois últimos de 5 à 10 anos, metade dos participantes possuem experiência com mais de uma linguagem de programação. Os participantes possuem conhecimento de nível baixo para médio em Javascript (questão 4), esta questão pode impactar negativamente o experimento, tendo em vista as atividades foram feitas utilizando esta linguagem para implementação de funções diretamente no console web das

Tabela 5: Mapeamento do perfil profissional e conhecimentos em desenvolvimento

#	Questão	Participante 1	Participante 2	Participante 3	Participante 4
1	Há quanto tempo você programa?	5 - 10 anos	5 - 10 anos	5 - 10 anos	5 - 10 anos
2	Quanto tempo você desenvolve software profissionalmente?	< 1 ano	5 - 10 anos	5 - 10 anos	2 - 5 anos
3	Quantas linguagens de programação você tem experiência?	3	2	> 5	> 5
4	Como você avaliaria suas habilidades de desenvolvimento atuais em Javascript?	1	1	3	2
5	Como você avaliaria suas habilidades de desenvolvimento atuais em comparação com pessoas mais experientes? em relação a algum desenvolvedor com quem você trabalhou?	1	3	4	4
6	Como você avaliaria suas habilidades de desenvolvimento atuais em comparação com seus colegas?	2	3	4	4
7	Você já modificou software de terceiros?	Sim	Sim	Não	Sim

Fonte: Autoria própria.

plataformas, porém, era exigido ao menos conhecimento básico como estruturas condicionais e de loop. Com relação à auto-avaliação de habilidades comparado a desenvolvedores experientes revela-se que a maioria se avalia entre um nível médio com tendências para alto (questão 5), em relação ao nível de habilidades comparados a colegas de trabalho a avaliação foi em nível mediano. Com relação à experiencia de modificação em software de terceiros 75% afirma que já realizou modificações.

Já a tabela 6 possui questões com a finalidade de mapear conhecimentos referentes à área do experimento, que abrange microsserviços e FaaS. A tabela aponta que metade dos participantes já trabalharam em projetos que envolviam a arquitetura de microsserviços (Questão 8) porém ninguém nunca havia utilizado nenhuma plataforma Serverless e nem possuía conhecimento referente às plataformas FaaS AWS Lambda e Azure functions (Questões de 9 à 11 respectivamente).

Tabela 6: Mapeamento de conhecimentos na área do experimento

#	Questão	Participante 1	Participante 2	Participante 3	Participante 4
8	Você já trabalhou em projetos que envolviam a arquitetura de Microsserviços?	Não	Não	Sim	Sim
9	Você já utilizou ou utiliza alguma plataforma Serverless? (Tendo em vista que Serverless abrange BAAS (Backend As A Service) e FAAS (Function As A Service).	Não	Não	Não	Sim
10	Como você avaliaria suas habilidades atuais com relação a plataforma FAAS AWS (Amazon Web Services) Lambda?	0	0	0	0
11	Como você avaliaria suas habilidades atuais com relação a plataforma FAAS Azure Functions?	0	0	0	0

Fonte: Autoria própria.

Após a realização do experimento foi realizada uma nova avaliação, esta, visando capturar o feedback com relação ao treinamento, identificando se com ele os participantes obtiveram habilidades necessárias para realização das atividades experimentais. A tabela 7 traz à tona o retorno dos participantes com relação ao treinamento, na questão 13 foi solicitado o quanto os participantes concordavam com a declaração de que as sessões de treinamento forneceram o que precisava para realização do experimento, as notas podiam variar de 0 (eu não concordo) até 5 (eu concordo totalmente), os resultados variaram de 2 à 5, tirando uma média se obtém o resultado 3.5, o que pode classificar que o treinamento refletiu parcialmente a transmissão de conhecimento com tendências positivas. De acordo com a questão 14, o participantes classificaram entre 3 e 4 seu nível de conhecimento em FaaS após o treinamento o que pode ser tido como médio com tendências para bom, em seguida classificaram o nível de dificuldade das atividades exigidas no experimento como médio (Questão 15).

Tabela 7: Feedback dos participantes em relação ao treinamento e atividades

#	Questão	Participante 1	Participante 2	Participante 3	Participante 4
13	Quanto você concorda com a seguinte declaração: "As sessões de treinamento me forneceram o que eu precisava para realização do experimento". Zero (0) significa “Eu discordo completamente”, Cinco (5) significa “eu estou totalmente de acordo"	2	5	4	3
14	Quanto você concorda que a atividade descrita neste experimento é realista? Zero (0) significa “Eu discordo completamente”, Cinco (5) significa “Eu concordo completamente”.	3	4	4	4
15	Na sua perspectiva, quão fácil foi realizar as atividades exigidas por esse experimento? Zero(0) significa “Não foi nada fácil”, Cinco (5) significa “Foi muito fácil”.	2	3	5	4

Fonte: Autoria própria.

Em seguida, foi obtido um feedback em relação à utilização das plataformas e de suas integrações (Tabela 8), após o experimento os participantes classificaram como médio seu nível de habilidades adquiridos para desenvolvimento FaaS utilizando AWS Lambda e Azure Functions (Questões 16 e 17). A experiência de utilizar o console web do Aws Lambda ficou entre 2 e 3, os agradando parcialmente com tendências negativas. Já a ferramenta do Azure foi classificada entre 3 e 4, também agradando parcialmente porém com tendências positivas. Na perspectiva dos desenvolvedores a plataforma em que foi mais fácil criar e disponibilizar functions foi a da Azure. A respeito do nível de facilidade de utilização das integrações disponíveis para com as ferramentas FaaS, foi avaliado que no Lambda o nível de facilidade foi menor do que a utilização de integrações por parte do Azure, sendo elas HTTP e *Scheduled Events*.

Tabela 8: Feedback dos participantes em relação às plataformas após o experimento

#	Questão	Participante 1	Participante 2	Participante 3	Participante 4
16	Como você avaliaria seu conhecimento em FAAS - Funções como Serviço após o experimento?	3	4	3	3
17	Como você avaliaria suas habilidades em desenvolvimento FAAS utilizando a plataforma AWS Lambda após o experimento?	2	3	3	2
18	Como você avaliaria suas habilidades em desenvolvimento FAAS utilizando a plataforma Azure Functions após o experimento?	2	3	3	2
19	Como você avalia a experiência de utilizar o console web da plataforma AWS Lambda?	2	3	3	2
20	Como você avalia a experiência de utilizar o console web da plataforma Azure Functions?	3	4	3	4
21	Na sua perspectiva qual plataforma foi mais fácil para criar e disponibilizar uma function?	Azure Functions	Azure Functions	Azure Functions	Azure Functions
22	Na sua perspectiva, quão fácil foi utilizar as integrações que a Amazon oferece para utilizar com o Lambda? (Ex: Http e Scheduled)	2	3	4	2
23	Na sua perspectiva, quão fácil foi utilizar as integrações que a Azure oferece para utilizar com o Functions? (Ex: Http e Scheduled)	3	4	4	5

Fonte: Autoria própria.

4.1.2 ANÁLISE POR PARTICIPANTE DE REALIZAÇÃO DE TAREFAS DO EXPERIMENTO

Durante o experimento em cada atividade cada participante tinha que realizar a marcação do tempo de realização de suas tarefas, a tabela 9 descreve o tempo de realização de atividades de cada participante para cada plataforma e observa-se que a criação de funções no Azure Functions demandou mais tempo que no Aws Lambda, mesmo os participantes tendo gasto maior tempo os mesmos avaliaram que tiveram uma facilidade maior de se trabalhar com

o Azure, facilidade esta que pode ser devido à possibilidade de utilizar integrações com outros serviços de maneira mais fácil, os mesmos avaliaram positivamente a utilização de integrações no Azure, resultado este citado na Seção 4.1.1 e especificado na atividade 23 da tabela 8.

Tabela 9: Tempo gasto nas atividades experimentais por participante

Participante	Tempo gasto em minutos Aws Lambda	Tempo gasto em minutos Azure Functions	Total de Minutos
1	00:10:08	00:10:04	00:20:12
2	00:15:40	00:27:52	00:43:32
3	00:28:25	00:12:38	00:41:03
4	00:16:15	00:28:38	00:44:53
Total	**01:10:28**	**01:19:12**	**02:29:40**

Fonte: Autoria própria.

O participante 1 foi o que menos tempo gastou nas atividades experimentais de ambas as plataformas levando um tempo total de 20 minutos, que divergiu da média exata dos outros participantes que foi de 42 minutos, o mesmo classificou-se no formulário de caracterização como nível de conhecimento baixo em Javascript, não conhecia nada referente às plataformas, discordou com a afirmação de que o treinamento forneceu o que precisava para o experimento, e classificou como difícil as atividades. Analisando o cumprimento das atividades no que foi solicitado percebeu-se que o resultado das atividades dele não foi compatível com as respostas por ele apresentadas no formulário de feedback, considerando que o seu resultado foi de acerto em todas as atividades.

Com relação aos participantes 2 e 4, ambos gastaram menos tempo na execução das atividades do Lambda do que na do Azure apesar de terem avaliado melhor a experiência de utilização da segunda plataforma e terem avaliado como fácil a realização de atividades de integrações com a mesma. Com relação ao seu perfil profissional ambos classificaram o conhecimento em Javascript como básico, comparando o tempo de desenvolvimento de software de forma profissional o segundo é menos experiente. No que se refere ao treinamento ambos concordaram de parcialmente para totalmente com a afirmação de que o treinamento forneceu o conhecimento necessário para o experimento, classificaram as atividades como realistas e como nível médio de dificuldade.

Já o participante 3 gastou mais tempo para completar todas as atividades no Lambda do que no Azure, em sua avaliação classificou como médio a experiência de utilização do console de ambas as plataformas, para ele a utilização de integrações de ambas as plataformas foi em nível fácil. O participante avaliou que o nível de dificuldade das atividades foi baixo e concordou que as atividades foram realistas e também concordou que o treinamento ofereceu o

que precisava para realização das atividades. Entre todos os participantes o 3 foi o que melhor avaliou suas habilidades em Javascript.

4.2 AMEAÇAS À VALIDADE

4.2.1 VALIDADE INTERNA

São pontuados algumas ameaças e como foram tratadas em relação à validade interna do experimento:

1. Seleção: Os participantes foram convidados para o experimento independentemente do nível de conhecimento, tendo como restrições experiência profissional em desenvolvimento e conhecimentos relacionados à linguagem Javascript;
2. Mortalidade seletiva: Foi uma questão de difícil tratamento, tendo em vista que os participantes tem a opção de não participar do experimento, para mitigar, como envolvia treinamento e foi realizada uma parceria com uma empresa, a universidade corporativa concedeu certificado de participação para cada participante, o que contou na empresa;
3. Contaminação: Cada participante foi instruído à realizar as atividades de maneira individual, sem consultas e auxílios de outro.

4.2.2 VALIDADE EXTERNA

As ameaças relacionadas à validade externa do experimento são tidas e contidas por:

1. Configuração do experimento: A realização das atividades experimentais foi direto no ambiente das plataformas, a única ameça que tínhamos era as plataformas suspenderem acesso a ele ou algo relacionado.
2. Tempo: Os participantes tinha o tempo de duas horas, o suficiente para realização das atividades, realizando a soma entre o tempo de atividades de pré-teste e as experimentais o tempo máximo utilizado foi de 1 hora e 20 minutos entre um dos participantes.

4.2.3 VALIDADE DE CONCLUSÃO

Nesta Seção é tratado a respeito da validade de conclusão do experimento:

1. Análise e interpretação estatística: Como o conjunto de dados é pequeno ficou inviável a análise descritiva e inferencial dos mesmos, porém não impossibilitou de uma análise prévia do feedback em relação ao tempo de realização das atividades e o perfil profissional do participante.

4.3 QUESTÕES DE PESQUISA

O experimento visava obter um feedback dos desenvolvedores com relação à utilização de plataformas *serverless* e aceitar ou não a hipótese de que o esforço para criar e alterar código serverless em diferentes plataformas é o mesmo. Com relação ao feedback, os participantes conseguiram apontar qual plataforma foi mais fácil de utilizar e qual oferecia melhores integrações, que no caso de acordo com a Seção 4.1.1 foi apontado a Azure. Porém, no que se refere à aceitação ou não da hipótese, neste cenário de participantes, considerando o tempo de realização de atividades entre as duas ferramentas por parte dos participantes pode-se apontar que o esforço para criar e alterar código serverless entre as plataformas Aws Lambda e Azure Functions foi o mesmo, porém se faz necessário uma nova aplicação do experimento devido à insuficiência de dados, apontando com uma melhor análise a aceitação ou não da hipótese, revendo as tarefas a serem realizadas e ampliando a escala através de replicação.

5 CONCLUSÃO

O objetivo deste trabalho foi a identificação e comparação de características e maturidade de serviços serverless, onde inicialmente foram identificadas algumas características com base na literatura de serverless (seção 2), em seguida foi especificado o foco em Funções como Serviço (seção 2.2.3) e escolhidos dois serviços, sendo o Aws Lambda e o Azure Functions, após, identificado suas características que foram comparadas com base em suas funcionalidades (Seção 2.3), após, foi realizado um experimento piloto para identificar a facilidade de uso dos serviços na perspectiva do desenvolvedor (Seção 3.3) sem objetivo de generalização de resultados, foi realizado como primeira tentativa de validação de uma possível hipótese que tinha o foco identificar se o esforço (medido pelo tempo) para criação e alterações de funções é o mesmo ou não, independente da plataforma escolhida, para isto foi realizada uma atividade de treinamento e experimentação que teve baixa aderência de participantes na fase de realização de atividades experimentais.

Com os resultados obtidos foi possível identificar por meio de um feedback uma plataforma que os participantes apontaram que tiveram uma maior facilidade nos processos de criação e disponibilização de funções. Já com relação ao tempo utilizado por cada participante na experimentação, devido a quantidade de amostras ser pequena, foi feita uma análise do total de minutos gastos em relação ao perfil do participante e seus conhecimentos obtidos por meio do treinamento.

Devido este ser um experimento piloto, verifica-se a necessidade em trabalhos futuros realização de melhoria do protocolo de experimentação, principalmente no que se refere à forma em que foi realizado o treinamento, buscando o aperfeiçoamento do material de experimentação, além de mais estudo na área de especificação e análise. E em seguida com a melhoria do protocolo, realizar uma nova aplicação do experimento tentando abranger mais participantes, após, realizar análises estatísticas e por fim criar o guia de referência para facilitar o processo de decisão de utilização.

REFERÊNCIAS

Amazon . Programe expressões usando Rate ou Cron. 2018. Disponível em: <https://docs.aws.amazon.com/pt$_br/lambda/latest/dg/tutorial - scheduled - events - schedule - expressions.html¿.Acessoem : 07jun.2018.

Feigenspan J. et al. . Measuring Programming Experience. n. September, 2012. Disponível em: <https://www.cs.cmu.edu/ ckaestne/pdf/icpc12.pdf>. Acesso em: 07 jun. 2018.

ADZIC, G.; CHATLEY, R. Serverless computing: economic and architectural impact. **Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017**, p. 884–889, 2017. Disponível em: <http://dl.acm.org/citation.cfm?doid=3106237.3117767>. Acesso em: 24 nov. 2017.

AMAZON. **AWS Lambda**. 2017. 1–9 p. Disponível em: <https://aws.amazon.com/lambda/>. Acesso em: 24 nov. 2017.

BADRI, . et al. TECHNOLOGY RADAR VOL.16. v. 16, 2017.

BALDINI, I. et al. Serverless Computing: Current Trends and Open Problems. p. 1–20, 2017. Disponível em: <http://arxiv.org/abs/1706.03178>. Acesso em: 24 nov. 2017.

CASTRO, P. et al. Serverless Programming (Function as a Service). **Proceedings - International Conference on Distributed Computing Systems**, p. 2658–2659, 2017. ISSN 1063-6927.

EIVY, A. Be wary of the economics of x0022;serverless x0022; cloud computing. **IEEE Cloud Computing**, v. 4, n. 2, p. 6–12, March 2017. ISSN 2325-6095.

FOWLER, S. J. **Microsserviços prontos para produção**. 1ª. ed. São Paulo, SP: Novatec Editora, Ltda, 2017.

GOOGLE. **Google Trends**. 2017. Disponível em: <https://trends.google.com.br/trends/explore?q=Aws Lambda,Cloud Functions,Azure Functions,ibm openwhisk>. Acesso em: 24 nov. 2017.

JANAKIRAMAN, B. **Serverless**. 2016. Disponível em: <https://martinfowler.com/bliki/Serverless.html>. Acesso em: 02 jun. 2018.

LEWIS, J.; FOWLER, M. **Microservices**. 2014. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 28 mai. 2018.

MICROSOFT. **Azure Functions**. 2017. Disponível em: <http://azure.microsoft.com/en-us/services/functions/>. Acesso em: 24 nov. 2017.

MICROSOFT. **Escala e hospedagem no Azure Functions**. 2017. Disponível em: <https://docs.microsoft.com/pt-br/azure/azure-functions/functions-scale>. Acesso em: 14 jun. 2018.

MICROSOFT. **Linguagens com suporte no Azure Functions**. 2017. Disponível em: <https://docs.microsoft.com/pt-br/azure/azure-functions/supported-languages>. Acesso em: 13 jun. 2018.

NAMIOT, D.; SNEPS-SNEPPE, M. On Micro-services Architecture. v. 2, n. 9, p. 24–27, 2014.

ROBERTS, M. **Serverless Architectures**. 2016. Disponível em: <https://martinfowler.com/articles/serverless.html>. Acesso em: 24 nov. 2017.

ROBERTS, M.; CHAPIN, J. **What is Serverless?** 1ª. ed. 1005 Gravenstein Highway North, Sebastopol, CA: O'Reilly Media, Inc, 2017.

SILVA, G. C. **Factors that Impact the Cloud Portability of Legacy Web Applications**. Tese (Doutorado) — University of York, York, UK, 9 2016.

SJOBERG, D. I. et al. Conducting realistic experiments in software engineering. In: IEEE. **Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n**. [S.l.], 2002. p. 17–26.

STACKOVERFLOW. **StackOverflow Developer Survey**. 2017. Disponível em: <https://insights.stackoverflow.com/survey/2017>. Acesso em: 24 nov. 2017.

WOHLIN, C. et al. **Experimentation in Software Engineering**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. 249 p. ISBN 978-3-642-29043-5.

APÊNDICE A – TUTORIAL DE UTILIZAÇÃO DE PLATAFORMAS FAAS

As Seções A.2 e A.3 apresentam uma visão prática do funcionamento dos serviços *serverless* AWS Lambda e Azure Functions, respectivamente. Para cada serviço são apresentados os conceitos fundamentais do serviço, cenários de uso, interface gráfica de gerenciamento do serviço, e o processo de desenvolvimento do algoritmo básico apresentado na Seção A.1.

A.1 ALGORITMO PARA UTILIZAÇÃO EM FUNÇÕES

Para mostrar o funcionamento dos três serviços *serverless* investigados neste estudo, foi criado um algoritmo para realização de operações aritméticas simples. O algoritmo apresentado na Figura 5 é adaptado nas seções posteriores de acordo com as regras de cada serviço *serverless*. Foi escolhida a linguagem javascript com o framework NodeJs para esse exemplo porque as três plataformas investigadas neste estudo suportam essa linguagem. Além disso, foi adicionada a biblioteca *express* para dar apoio no desenvolvimento do serviço. O exemplo segue uma implementação simples para realização de operações aritméticas passadas em forma de texto por meio de uma requisição através do protocolo HTTP sob método POST.

```
1   var express = require('express');
2   var app = express();
3   app.use(express.json());
4
5   app.post('/calculo', function (req, res) {
6       if(req.body.calculo){
7           res.status(200).send({"resultado":eval(req.body.calculo)});
8       } else {
9           res.status(500).send('Defina alguma expressão a ser calculada!');
10      }
11  });
12
13  var server = app.listen(8080);
14
```

Figura 5: Algoritmo implementação serviço de calculadora em NodeJS.

Fonte: Autoria própria.

Na linha 1 do código da Figura 5 ocorre a importação do módulo do *framework Express*, e na segunda linha o mesmo é iniciado. Já a terceira linha é utilizada para ser habilitada a utilização de *JSON (Javascript Object Notation)*. A partir da quinta linha contém a implementação com o método post para o endpoint /calculo, passando uma função por parâmetro que possui também em seus parâmetros os objetos de requisição (req) que possui os dados de quem realiza a mesma e o de resposta (res) que é o objeto de retorno. É realizado uma validação se possui o atributo calculo no corpo da request, caso não possua, é retornado um status 500 com uma mensagem de erro, caso exista alguma expressão a ser calculada, é retornado status 200 junto com o resultado calculado, que no caso é feito por meio da função eval, sendo essa, própria do JavaScript.

A.2 AWS LAMBDA

O *Amazon Lambda* é o serviço de computação sem servidor da AWS, no qual é executado código sobre demanda. Esta sessão foi feita com base na documentação disponível no site do serviço (AMAZON, 2017). Sendo cobrado apenas o recurso utilizado em tempo de execução. De acordo com a empresa com o Lambda, o usuário pode executar o código para praticamente qualquer tipo de aplicativo ou serviço de back-end, tudo sem precisar de administração. Basta carregar o código e a plataforma toma conta de tudo o que for necessário para executar e escalar o seu código com alta disponibilidade. Na Figura 6, pode ser observado o processo básico do funcionamento da ferramenta.

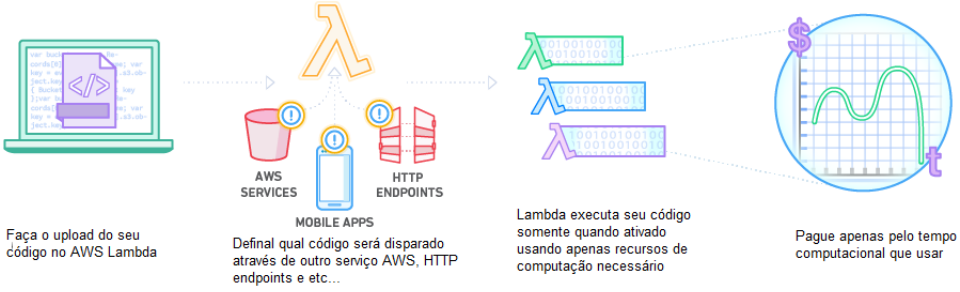

Figura 6: Processo de como funciona o AWS Lambda.

Fonte: (AMAZON, 2017)

Na Figura 6, observa-se que primeiramente é necessário realizar o upload do código na ferramenta, após, é necessário escolher um tipo de evento que irá disparar a função, como por exemplo a utilização de endpoints por meio da API-Gatway que é o serviço de construção de APIs HTTP ou disparando através de alterações em arquivos no serviço de arquivos S3, integra-

se com vários serviços da plataforma. A função é executada com apenas os recursos que ela necessita, sendo assim, é cobrado apenas o que for utilizado.

A.2.1 CENÁRIOS DE USO

A plataforma aponta alguns Cenários de Uso como por exemplo a criação de back-ends usando o AWS Lambda e o Amazon API Gateway para autenticar e processar solicitações de API. Na Figura 7, é apresentado um processo exemplificado de um fluxo de notificações de um aplicativo de meios sociais.

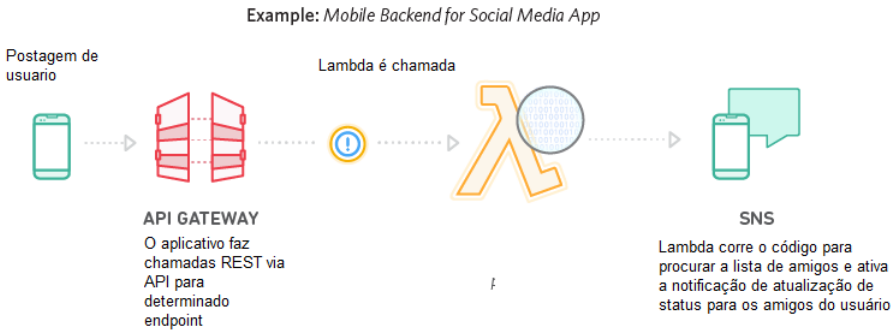

Figura 7: Exemplo de mobile backend com AWS Lambda.
Fonte: (AMAZON, 2017)

Nesse caso de notificações de postagens onde um usuario realiza um post em uma rede social, é observado que o mesmo por meio de um front-end mobile aciona a Api Gatway, que consequentemente executa a função lambda que busca os usuarios destino e aciona um serviço que envia as notificações para os usuarios amigos.

A.2.2 INTERFACE GRÁFICA DE GERENCIAMENTO DO SERVIÇO

O AWS Lambda possui um painel gráfico, onde podem ser obtidas informações gerais a respeito das funções criadas. A enumeração a seguir corresponde ao detalhamento dos itens da Figura 4.

1. No menu é possível escolher a navegação entre o dashboard ou a página de funções;
2. Pode ser visto um card com recursos disponíveis para a região em que as funções estão, por default na plataforma existe um limite tanto de espaçamento de utilização de códigos fontes que é de 75GB quanto de concorrência simultânea de funções que é até 1000 execuções, valor este que pode ser expandido segundo a mesma;

3.Podem ser obtidas informações de novidades relacionadas ao Lambda.;

4.São apontados recursos referentes a plugins e ferramentas que dão apoio à criação e gerenciamento às funções, como por exemplo o plugin do Jenkins que atua no auxílio de processos relacionados à devops;

5.Dispõe-se de métricas de todas as funções à nível de região;

6.Links de tutoriais, documentação, release notes e suporte à ferramenta;

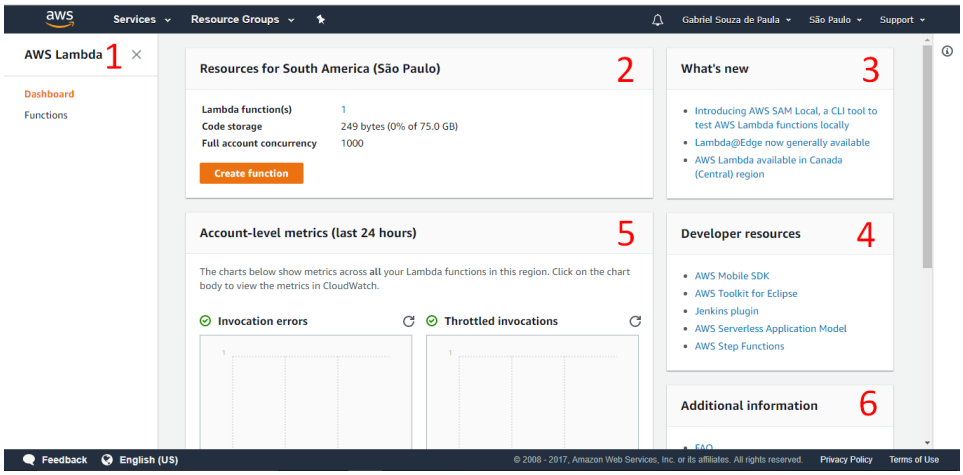

Figura 8: Painel principal AWS Lambda.

Fonte: Autoria própria.

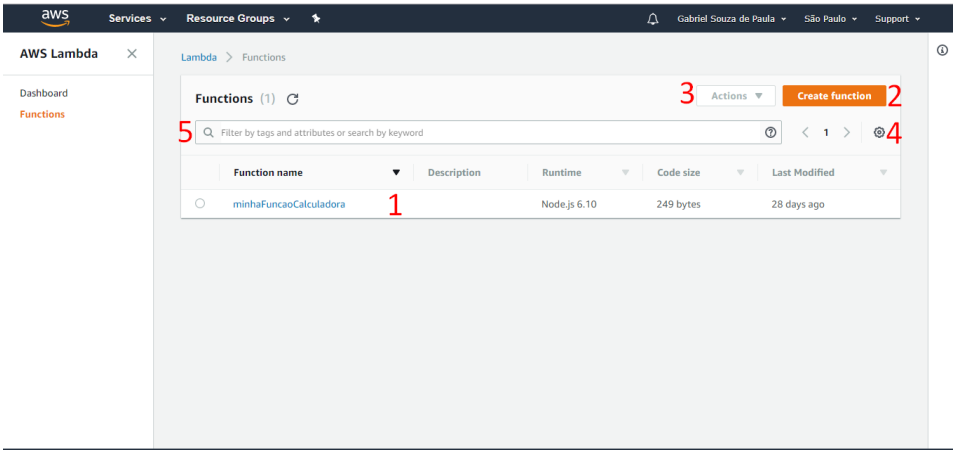

Figura 9: Painel de listagem de funções AWS Lambda.

Fonte: Autoria própria.

Na Figura 5, é apresentada a tela de listagem de funções, segundo item do menu. A seguir são apresentados os itens detalhando a tela:

1.No item primeiro da mesma tela, é possível ver informações de cada função, ao habilitar a flag é desbloqueado o componente do item 3;

2.Neste item é possível criar uma função nova;

3.Ao habilitar a flag do item 1 serão desbloqueadas algumas ações no selectbox: testar, deletar e ver detalhes;

4.São apresentadas configurações para definir informações a serem apresentadas na listagem;

5.No quinto item da tela, é possível realizar uma busca pelas já existentes;

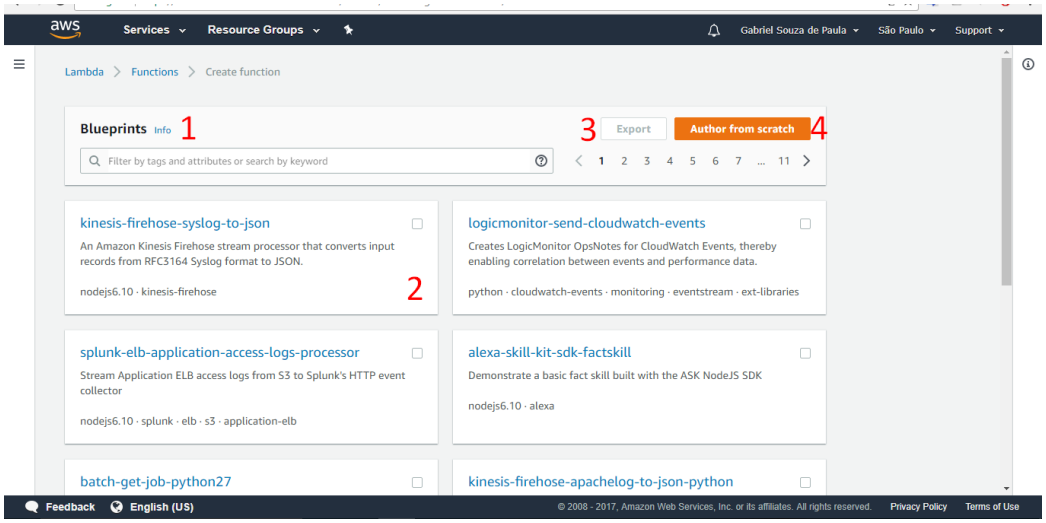

Figura 10: Tela blueprints AWS Lambda.

Fonte: Autoria própria.

Na Figura 6 é apresentada a tela de criação de funções, o lambda sugere os chamados Blueprints, que são aplicativos de amostra pré-configurados com código fonte e evento de disparo. Os itens da Figura são:

1.É possível realizar uma filtragem para encontrar o blueprint desejado;

2.Card com o blueprint, possível selecionar o desejado e exportar no item 3, caso seja selecionado no nome do que quer utilizar, é redirecionado para a tela de configuração do modelo;

3.É possível realizar a criação de função sem modelo default, ao clicar é feito o redirecionamento para criação de uma função zerada;

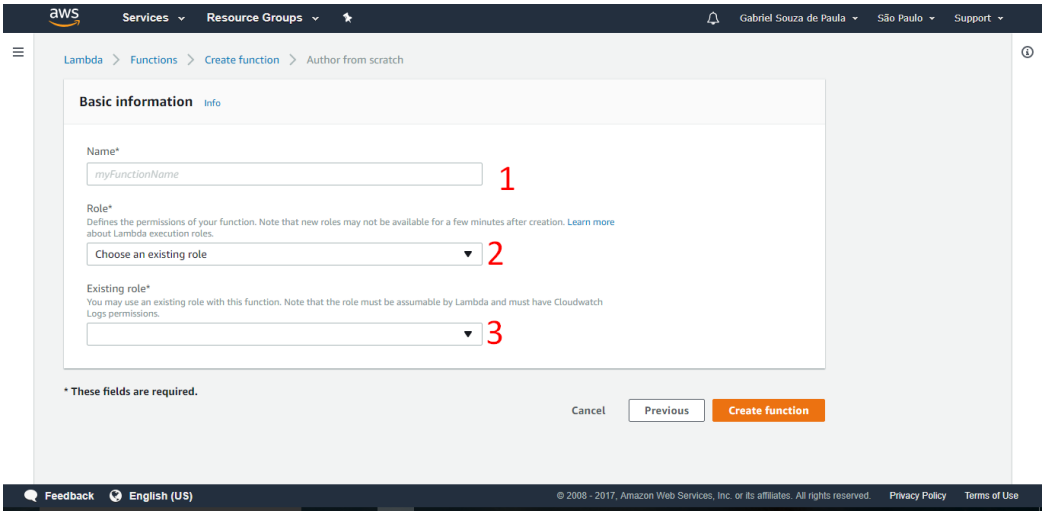

Figura 11: Tela de criação de função.

Fonte: Autoria própria.

Na Figura 7 é apresentada a tela de definição de informações básicas na criação das funções:

1.É possível definir um nome para a função;

2.Opção da definição de papéis que terão acesso á função, estes, podem ser atribuídos ou não à determinados usuários. Caso já possua um papel, é possível seleciona-la no item 3;

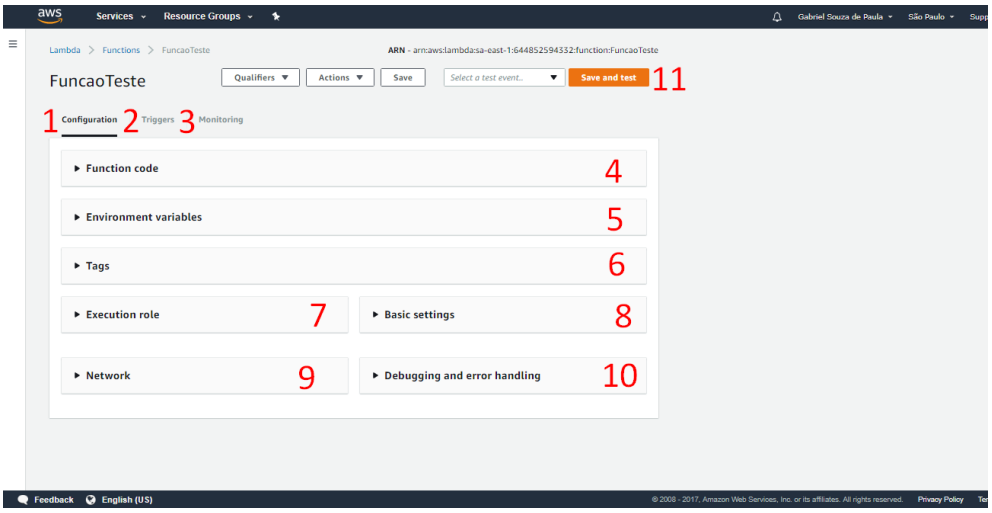

Figura 12: Tela de configuração de função.

Fonte: Autoria Própria

Já na Figura 8 é apresentada a tela de configurações da função, onde é constituído pelos itens:

1.Aba de opções de configuração da função;

2.Aba de seleção de trigger em que será disparado a função;

3.É apresentada a aba da tela de monitoramento da função;

4.Ao expandir (Figura 9) é possível escolher a linguagem da implementação da função, dependendo da mesma deverá ser feito upload de um pacote, isto no java e no c#, já com o nodejs e phyton é possível editar no próprio console. Após escolhida linguagem, deve ser apontado o handler, que é o método ou função implementada, que será executada no disparar do serviço;

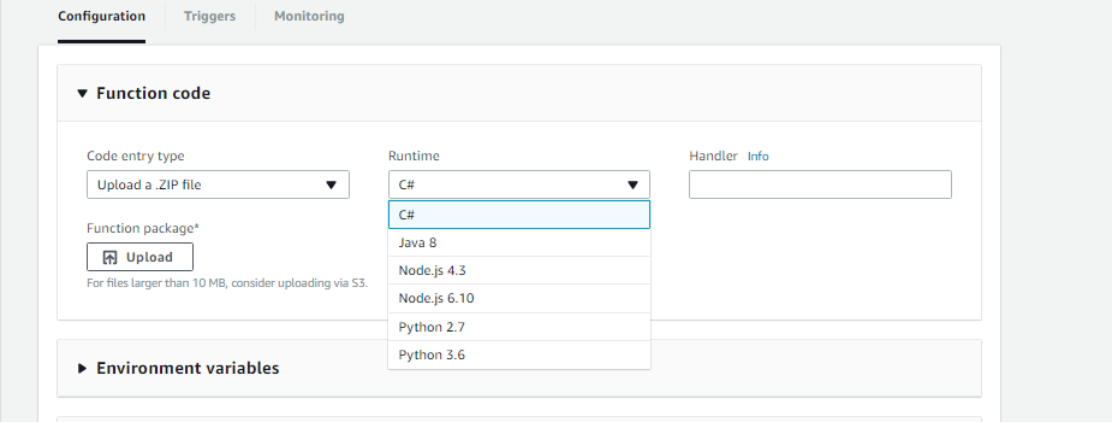

Figura 13: Tela de configuração opção de escolha de linguagem.

Fonte: Autoria Própria.

5.É possível definir variáveis de ambientes que serão acessadas em tempo de execução;

6.Possibilidade de configurar também tags para filtragem da função na tela de listagem(Figura 10).

7.Configuração de papéis de acesso à função;

8.Disponibilização de recursos de hardware com alocação de memória e definição de timeout que no caso é o tempo máximo em que a função será executada;

9.Após, é possível definir a rede em que a mesma estará disponível;

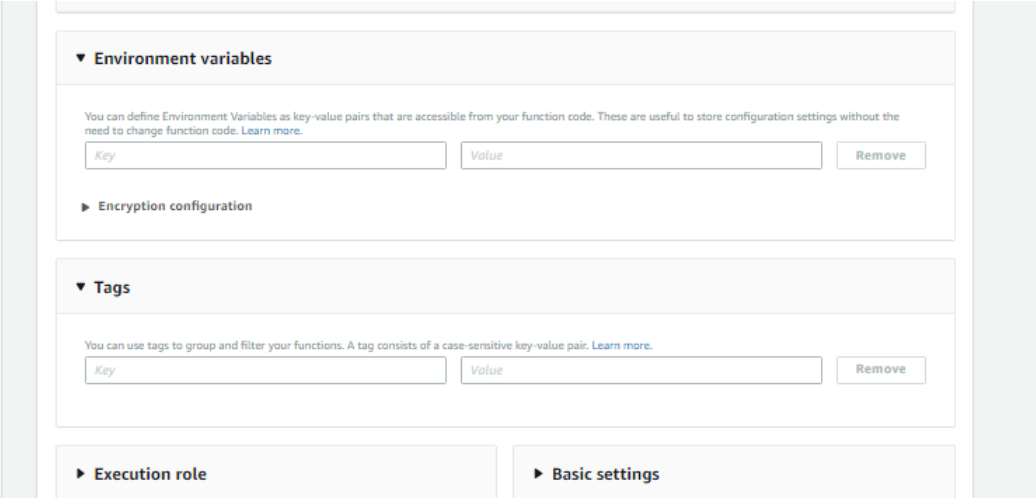

Figura 14: Tela de configuração, definição de variáveis de ambiente e tags.

Fonte: Autoria própria

10. São apresentados também recursos que auxiliam no debug com o rastreamento e monitoramento da execução (Figura 11);

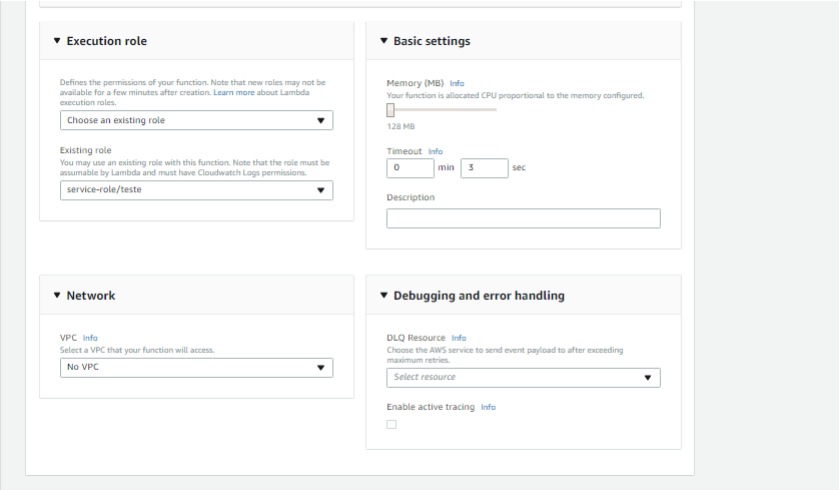

Figura 15: Tela de papeis, recursos, rede e configuração de debug.

Fonte: Autoria própria.

11. Neste item são apresentadas opções de salvamento e teste da função, deleção, exportação e alteração de versão na opção qualifiers. É possível configurar eventos de teste, na opção select test event, no item 1 da Figura 12 é possível estabelecer o template do evento a ser simulado, e no item 2 é apresentado o json com um modelo de dados a ser enviado para a função(Figura 12).

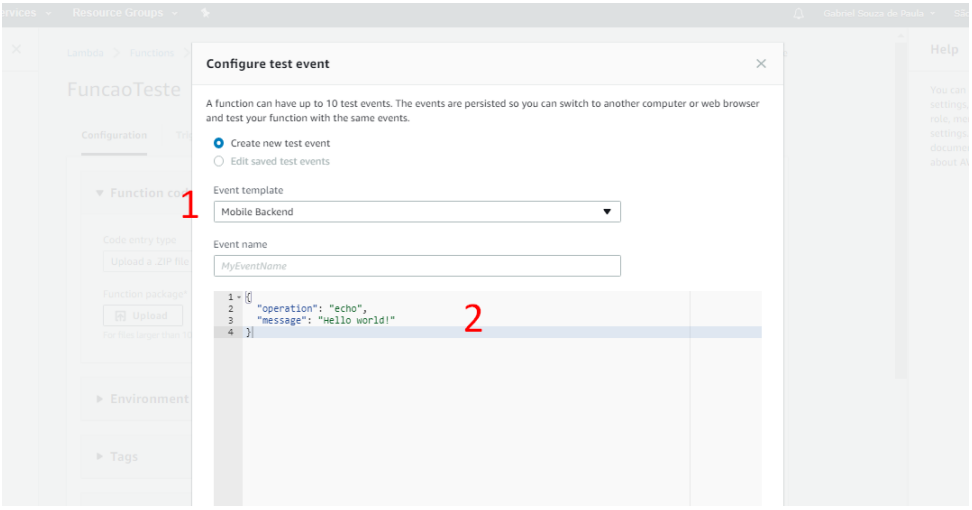

Figura 16: Tela de realização de testes.

Fonte: Autoria própria.

Na Figura 13, aba triggers da Figura 8, após selecionar add trigger, é possível selecionar algum serviço do AWS para integrar-se com a função e dispara-la, após a seleção de algum, é feito um redirecionamento para configurações de particularidades para cada serviço

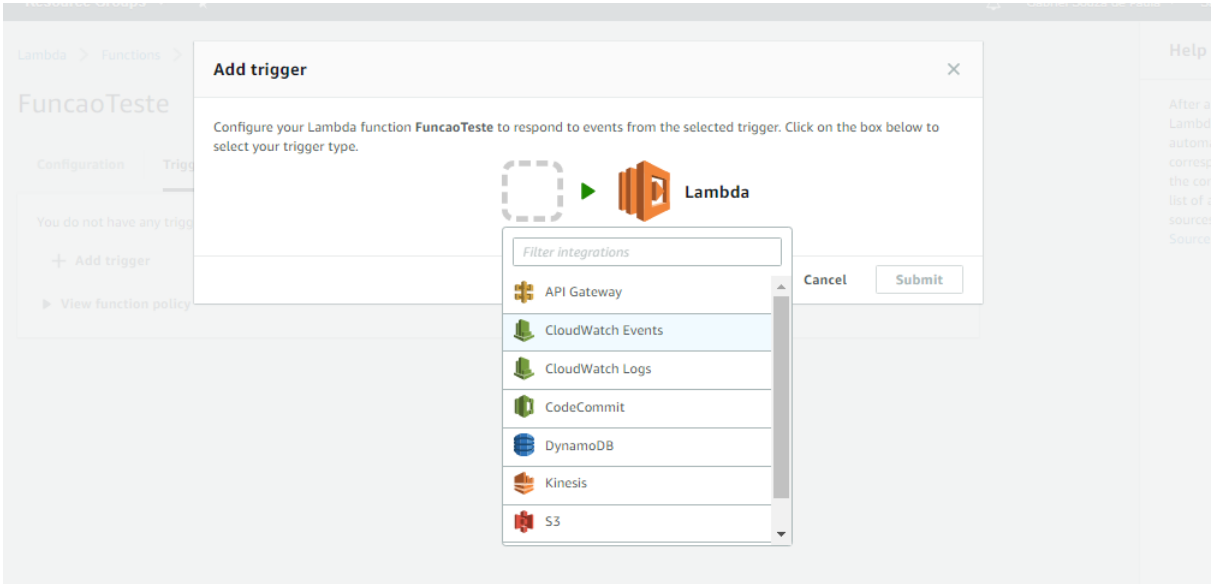

Figura 17: Tela seleção de integração com Lambda.

Fonte: Autoria Própria.

Já na Figura 14 - aba de monitoramento, é possível acompanhar em tempo real a:

1.Quantidade de execuções da função;

2.Duração das invocações;

3.Quantidade de erros que ocorrem durante a invocação de função.

Todos os gráficos são integrados com o serviço *CloudWatch* que apresenta detalhadamente as métricas e os logs de execução.

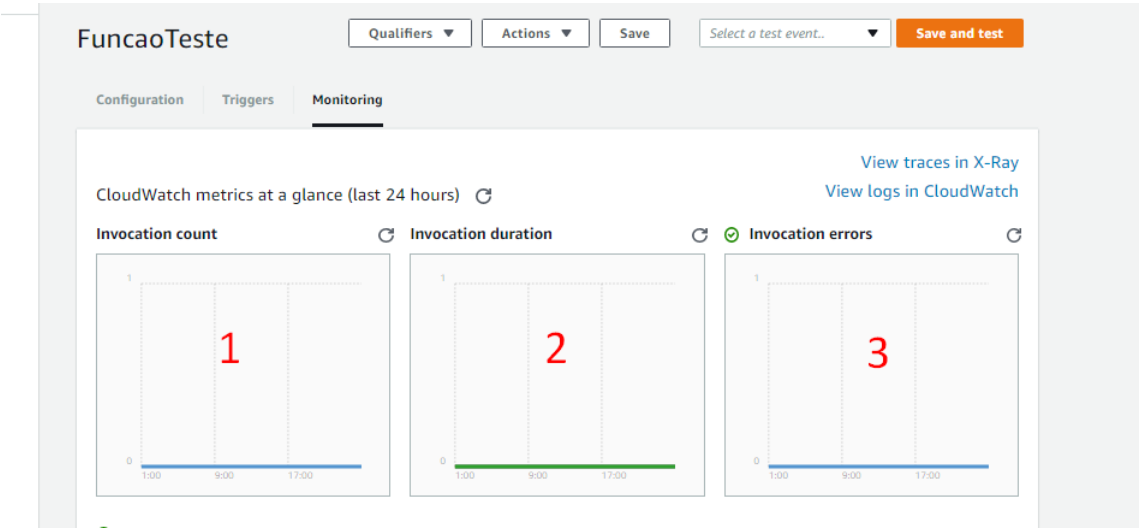

Figura 18: Tela de monitoramento AWS Lambda.

Fonte: Autoria própria.

A.2.3 CRIAÇÃO DE FUNÇÃO

Nesta seção é apresentado um exemplo de criação de função para o AWS Lambda com base no algorítmo da calculadora apontado na seção 2.1.1, utilizando NodeJs.

Deve ser seguido os seguintes itens:

1.Para isto, basta filtrar por HTTP na tela da Figura 6 e selecionar a opção httprequest com a tag com nodejs;

2.Ao abrir a tela da Figura 7 deve ser escolhido um nome para a função e um papel. Na parte inferior irá aparecer um campo com um código default, ignorar e prosseguir.

3.Após, na tela de configuração (Figura 8), com o node 6.10 selecionado, deverá aparecer um campo para edição do código (Figura 15).

4.Deverá ser trocado o código da tela pelo da Figura 15, observe que é necessário definir um nome para função a ser executada por meio da variavel exports na linha 1, que é o nome da função a ser utilizada na invocação do serviço ficando assim exports.nomeDaFunção, ao ser definido o nome, deve ser setado o mesmo no campo handler do console.

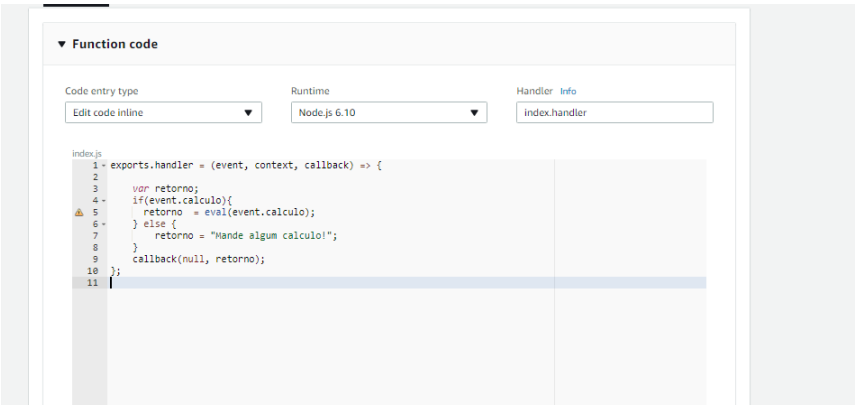

Figura 19: Tela com textarea para edição de códigos node.

Fonte: Autoria própria.

5.Em relação aos parâmetros da função, o event de acordo com a documentação para desenvolvedores, o AWS Lambda usa-o para passar nos dados do evento para o manipulador, já no context o AWS utiliza para passar informações em tempo de execução para a função. Já o parametro callback pode ser utilizado opcionalmente para retornar informações.

6.Para testar o algorítmo é necessário configurar casos de teste como pode ser visto na Figura 12, no caso o arquivo json terá apenas a seguinte descrição: {”calculo”: ”1+2+3”}.

7.No fim, basta aplicar um salvar e testar que o resultado será apresentado em uma tela de output como na Figura 16.

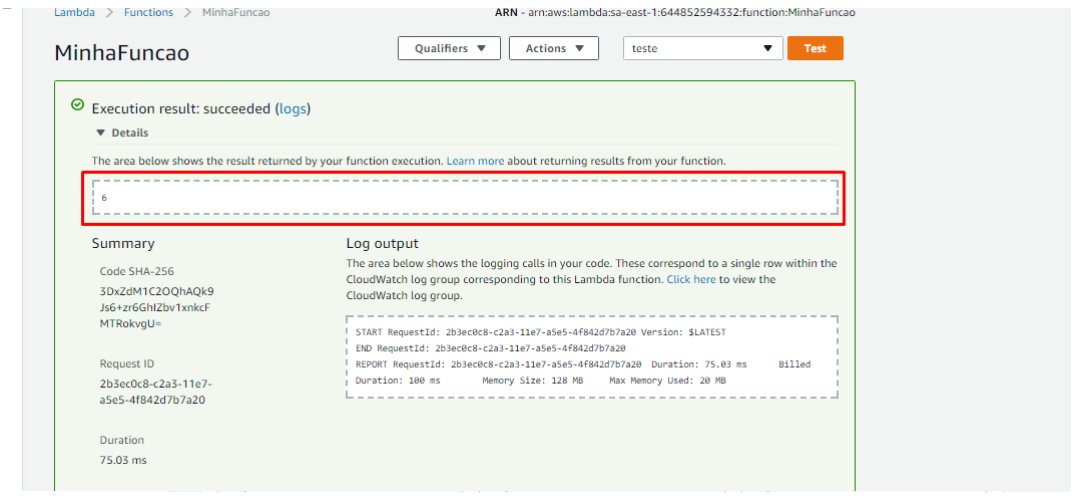

Figura 20: Tela com resultado da execução da Função.

Fonte: Autoria Própria

A.3 MICROSOFT AZURE FUNCTIONS

O Azure Functions, é um serviço de computação sem servidor da Microsoft, onde assim como as ferramentas do AWS Lambda e Google Cloud Functions é executado código sobre demanda, esta sessão baseia-se na documentação oficial do serviço da Azure (MICROSOFT, 2017). Sendo cobrado apenas o recurso utilizado em tempo de execução. As funções que podem ser disparadas por meio de variados eventos, como HTTP, Cronômetro, GitHub WebHook, Azure Cosmos DB, Azure Blob e Azure Queue.

A.3.1 CENÁRIOS DE USO

Assim como as outras plataformas o Azure Functions também possui vários cenários de uso, neste tópico, foram selecionados alguns para exemplificar aplicações práticas dos serverless.

Processamento baseado em temporizador. De acordo com a documentação da plataforma o Azure Functions dá suporte a um evento com base em um temporizador usando a sintaxe de trabalho Cron. Por exemplo, a criação de um código que é executado a cada 15 minutos e que limpa uma tabela de banco de dados com base em lógica de negócios personalizada. Figura 25.

Figura 21: Processo de cenário de uso baseado em temporizador.
Fonte: (MICROSOFT, 2017)

Outro exemplo é o de arquiteturas de aplicativo Web sem servidor, onde o Azure Functions consegue ativar um aplicativo de uma única página. O aplicativo chama funções usando a URL WebHook, salva dados do usuário e decide quais dados devem ser exibidos. Ou, é possível fazer personalizações simples, como alterar o direcionamento de um anúncio ao chamar uma função e passar suas informações de perfil do usuário.

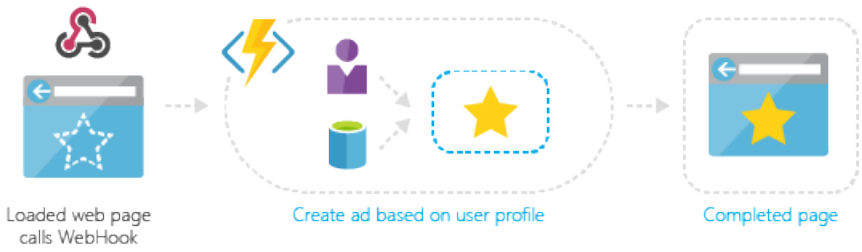

Figura 22: Processo de cenário de uso em arquiteturas de aplicativo Web.
Fonte: (MICROSOFT, 2017)

A.3.2 INTERFACE GRÁFICA DE GERENCIAMENTO DO SERVIÇO

O Azure Plataform também possui um painel para gerenciamento de seus serviços em nuvem. Como pode ser visto na Figura 27, o painel possui um menu de navegação lateral (Item 1) com acesso aos recursos e serviços utilizáveis na plataforma. Assim como o Google Cloud o Azure também possui um painel dashboard personalizável (Item 2), com informações a respeito à integridade dos serviços utilizados e recursos alocados.

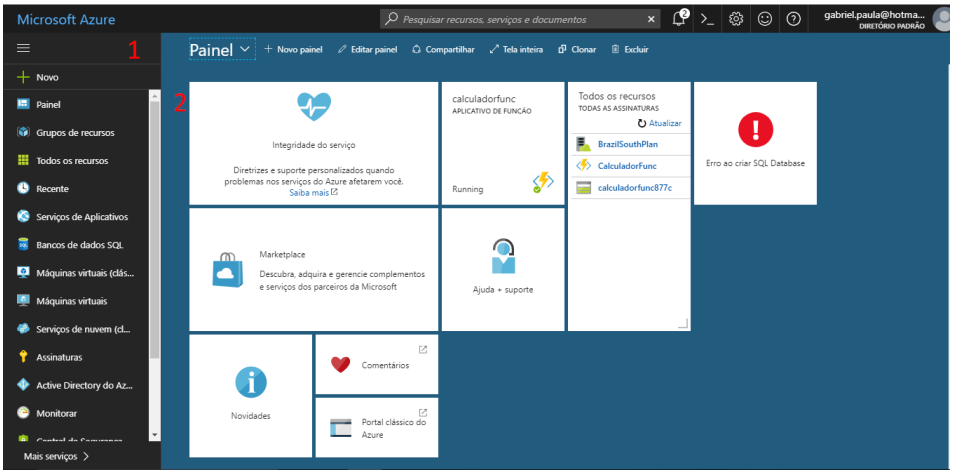

Figura 23: Painel principal Azure.
Fonte: Autoria própria

Para adicionar um novo recurso, como o de serverless, basta ir em novo e filtrar por aplicativo de funções, como pode ser visualizado na Figura 28. Nesta tela, especificamente na área do Azure Marketplace, podem ser escolhidos os recursos a serem utilizados, dês de máquinas virtuais, banco de dados e até serviços de bot.

Em seguida, na próxima tela, são apresentados links úteis e disponibilizado a opção de criação de um novo Aplicativo de Funções, que na Azure, é tida como um agrupamento de funções. Deve ser clicado em Criar.

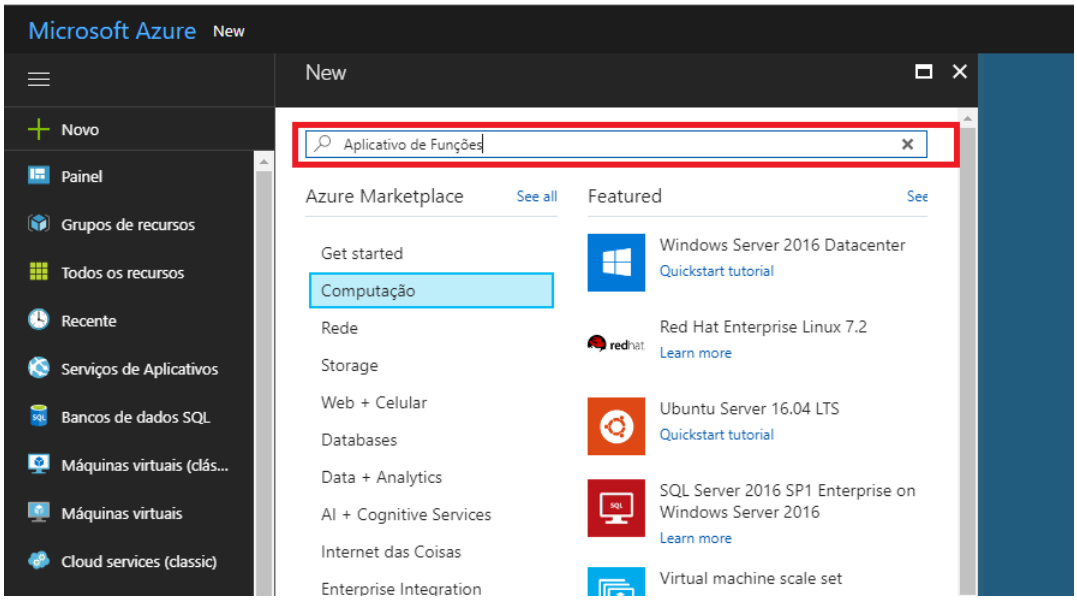

Figura 24: Tela de recursos Azure Marketplace.

Fonte: Autoria própria

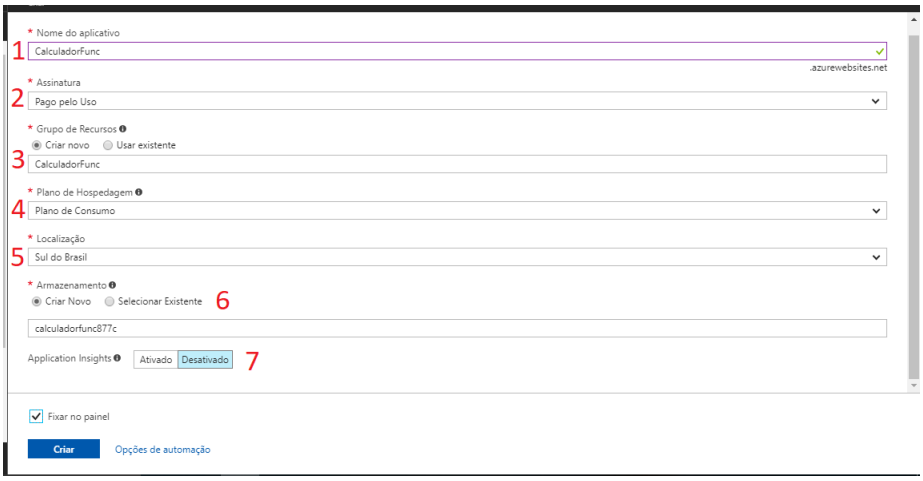

Figura 25: Tela de criação de aplicativo de função.

Fonte: Autoria própria

Após na Figura 29, será apresentado um formulário para cadastro do aplicativo de função:

1.Referente ao nome dado para a aplicação;

2.Já o segundo item é definido como o tipo de assinatura, que no caso atua na atribuição da conta a ser cobrada pelos recursos utilizados;

3.Refere-se ao grupo de recurso – que é um agrupamento de recursos utilizados pela

empresa, como por exemplo, um grupo uma aplicação com os servidores de aplicação e bancos de dados;

4.É definido um plano de Hospedagem, que divide-se em dois: um que é o plano de consumo, neste caso quando sua função é executada, o Azure fornece todos os recursos computacionais necessários. Não é necessário se preocupar com o gerenciamento de recursos e paga apenas pelo tempo de execução do código. Já no plano de serviços é apontado como um plano em que os recursos estão em todo o momento sendo utilizados em uma VM mesmo ela não estando em uso, pagando assim pela disponibilidade total e no caso o custeio irá direto para este serviço;

5.É definida a localização em que a aplicação de funções estará escalada, na América Latina existe apenas um servidor que fica no Sul do Brasil;

6.Configuração de armazenamento;

7.Refere-se à um recurso que auxilia na detecção de problemas de qualidade nas aplicações;

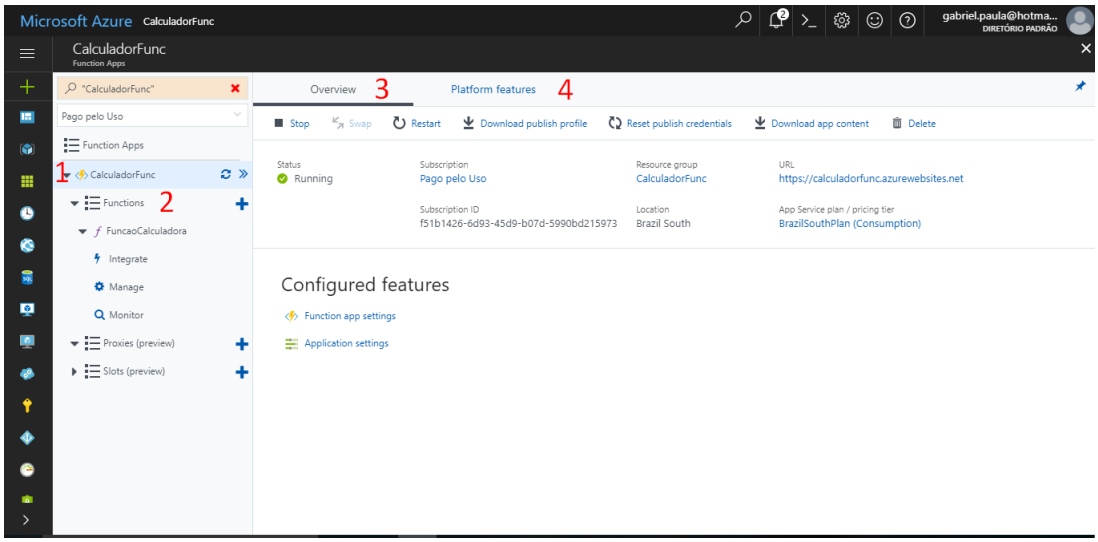

Figura 26: Tela gerenciamento de functions.

Fonte: Autoria própria

Cada aplicação de função possui um painel para detalhamentos, onde são criadas as funções (Figura 30).

1.No primeiro item deste painel é apresentado onde são listados os aplicativos de funções, dentro de cada um, existem as funções propriamente ditas;

2.Cada função (Item 2) possui uma tela de monitoramento para acompanhar gráficos de execução e status, gerenciamento para configurações a nível de estado da função com definição de variáveis de ambiente e passividade de habilitar e inabilitar a mesma, possui também uma tela de integração, com as configurações de triggers que irão dispara-la, telas estas apontadas respectivamente nas figuras 31, 32 e 33. Ao clicar em cima da função pode ser visto o código fonte;

3.Aba com informações gerais;

4.Aba de configurações da plataforma;

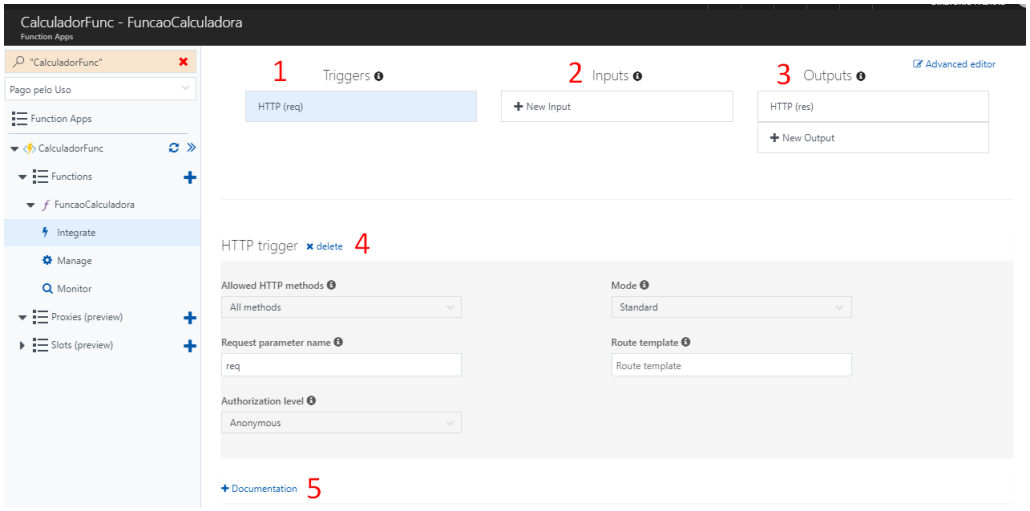

Figura 27: Tela integrações da função.

Fonte: Autoria própria

Na Figura 31 é apontado a tela de integrações da função, estas são tidas como triggers, que são o que inicia as funções.

1.No item 1 é possível adicionar uma trigger que executará esta função, já nó próximo é possível também determinar um input, que no caso são dados adicionais fornecidos à função enquanto a mesma é acionada, como por exemplo dados de um banco de dados, arquivo externo e etc;

2.Configuração de inputs dados à função;

3.No item 3 é configurada as informações de output da função;

4.É apresentado um espaço para configuração do disparador da função;

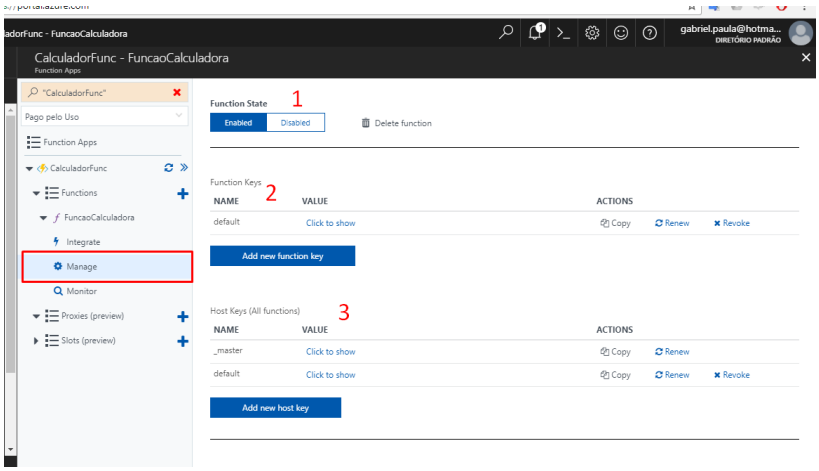

Figura 28: Tela configurações da Função.

Fonte: Autoria própria

5.Possibilidade de adicionar documentação a respeito da função criada;

Já na tela de configuração da função (Figura 32), é possível observar no item 1 as opções de desabilitar e deletar a função. Já no segundo e terceiro item são apresentados chaves das funções, que quando utiliza-se requisição http, elas restringem o acesso, tanto para função atual quanto para todas as outras.

Na Figura 33 é apresentado a tela de monitoramento, nesta, é possível observar informações a respeito da quantidade de invocações à função realizadas com sucesso(item 1), falhas(item 2), logs de invocação e detalhes a respeito das mesmas (item 3).

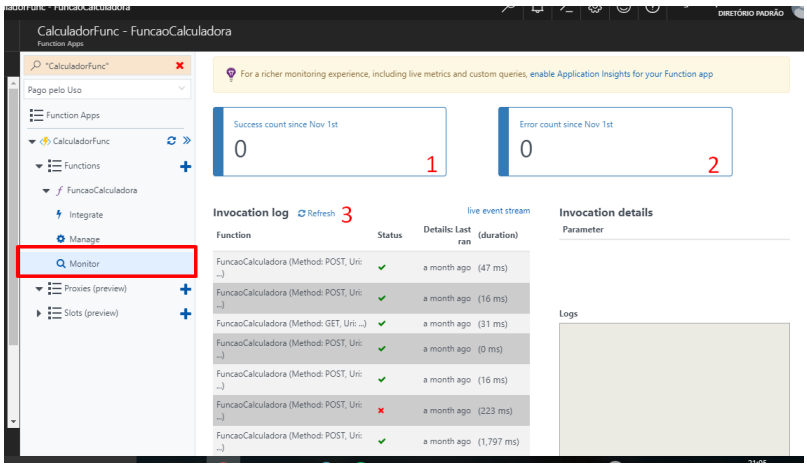

Figura 29: Tela monitoramento função.

Fonte: Autoria própria

Para editar o código fonte da função, basta clicar no nome da mesma, no console da Figura 34 é possível programar (Item 1), testar (Item 4) e editar os arquivos que fazem parte da mesma (Item 3). Ao criar a função, é possível também obter um link para testar a função ou realizar a requisição de um cliente(Item 2).

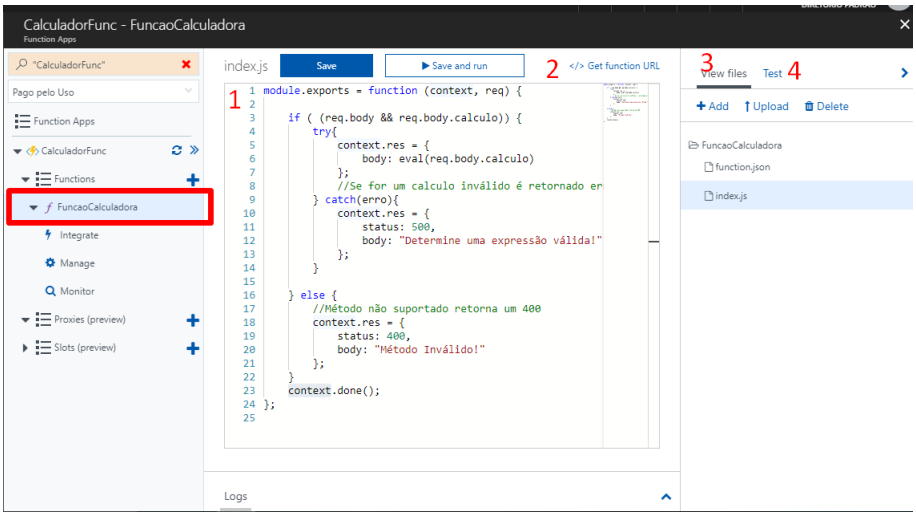

Figura 30: Tela de programação da função.

Fonte: Autoria própria

A.3.3 CRIAÇÃO DE FUNÇÃO

Para criar a função no Microsoft Azure Functions devem ser seguidos os seguintes passos:

1. Para isto, é necessário ir até a tela da Figura 29 e inicialmente criar um aplicativo de funções, definindo um nome, assinatura no modo pago pelo uso, criar um grupo de recursos com um nome FuncaoCalc, plano de hospedagem definido como plano de consumo;

2. Em seguida deve ser definida a localização da aplicação na região do Sul do Brasil, após isso deve ser definido uma região de arquivo e clicado em criar;

3. Após, deve ser aberto o console da Figura 34 para edição/criação de código fonte;

4. A implementação deve seguir como está na Figura 35, de acordo com a documentação Oficial, Todas as funções JavaScript devem exportar uma única function via a expressão:module.exports para o tempo de execução localizar a função e executá-la. Esta função sempre deve incluir um objeto context que contém dados de requisição e

de resposta, ja o req possui apenas os dados de requisição, como o corpo da mesma. Deve ser chamar context.done (linha 21), ou o tempo de execução nunca saberá que sua função terminou e a execução atingirá o tempo limite;

5. para testar basta adicionar o json {”calculo”:”1+1”} na tela de teste e executar no método post. O resultado aparecerá em um output logo à baixo;

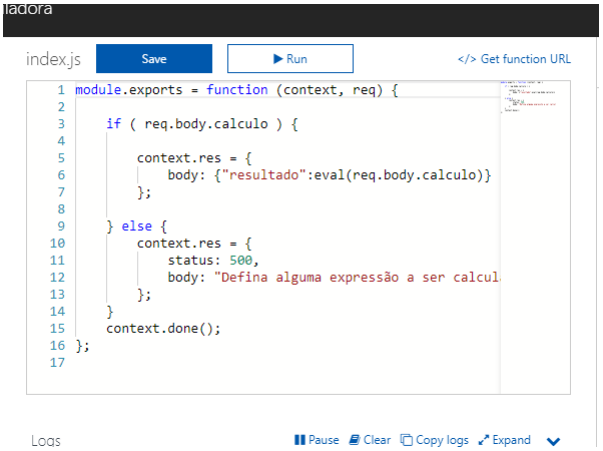

Figura 31: Tela com implementação de função.