UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

CAIO WELITON NASCIMENTO QUEIROZ

UMA COMPARAÇÃO ENTRE ARQUITETURAS CONVENCIONAL E *SERVERLESS* UTILIZANDO ATRIBUTOS DE QUALIDADE EM APLICAÇÕES *EHEALTH*

QUIXADÁ

2021

CAIO WELITON NASCIMENTO QUEIROZ

UMA COMPARAÇÃO ENTRE ARQUITETURAS CONVENCIONAL E *SERVERLESS* UTILIZANDO ATRIBUTOS DE QUALIDADE EM APLICAÇÕES *EHEALTH*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientador: Prof. Dr. Emanuel Ferreira Coutinho

QUIXADÁ

2021

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

Q43c Queiroz, Caio Weliton Nascimento.
Uma comparação entre arquiteturas convencional e serverless utilizando atributos de qualidade em aplicações eHealth / Caio Weliton Nascimento Queiroz. – 2021.
70 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2021.
Orientação: Prof. Dr. Emanuel Ferreira Coutinho.

1. eHealth. 2. Computação em nuvem. 3. Computação sem servidor. I. Título.

CDD 005.1

CAIO WELITON NASCIMENTO QUEIROZ

UMA COMPARAÇÃO ENTRE ARQUITETURAS CONVENCIONAL E *SERVERLESS* UTILIZANDO ATRIBUTOS DE QUALIDADE EM APLICAÇÕES *EHEALTH*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em: ___/___/___

BANCA EXAMINADORA

Prof. Dr. Emanuel Ferreira Coutinho (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. João Marcelo Uchôa de Alencar
Universidade Federal do Ceará (UFC)

Prof. Dr. Jefferson de Carvalho Silva
Universidade Federal do Ceará (UFC)

Prof. Me. Maurício Moreira Neto
Universidade Federal do Ceará (UFC)

A minha familia, aos professores, colegas e amigos que sempre me apoiaram.

AGRADECIMENTOS

À minha mãe, Maria, que sempre esteve comigo, nada que eu diga e faça pode retribuir o que ela fez, faz e fará por e para mim.

Ao meu pai, Carlos, que sempre me incentivou a estudar e sempre demonstrou um grande orgulho e esperança pelo que eu poderia alcançar.

Ao meu irmão Carlos e minha irmã Carol, por estarem comigo desde sempre brigando, brincando e conversando.

A toda minha família por sempre contribuir e apoiar o meu desenvolvimento pessoal, acadêmico e profissional.

Aos meus colegas de apartamento Leandro, Rener, Renan, Sabino e Herom por me acolherem e apoiarem durante minha estadia no eterno AP 204.

Aos meus colegas de curso e amigos da famosa "*Gang of four*", Samuel, Publio e Elenilson por estarem presentes durante toda a graduação e espero que por muito mais tempo. E também aos outros amigos de graduação Larissa, Klyvia, Edval, Eduarda e toda a turma que sempre se ajudou no decorrer dos anos.

Ao meu orientador, professor Emanuel pela paciência, apoio e competência ao me direcionar durante o desenvolvimento do trabalho.

À toda comunidade acadêmica da Universidade Federal do Ceará e a todos os colegas do curso de Engenharia de Software pelas vivências e aprendizados juntos.

E a todas as outras pessoas que não foram aqui citadas, mas que contribuíram de alguma forma para o meu desenvolvimento e o desenvolvimento desse trabalho.

“A inteligência é o que você usa quando não sabe o que fazer.”

(Jean Piaget)

RESUMO

O setor de *healthcare* vem se especializando cada vez mais nas demandas dos usuários nos últimos anos. Com isso, ele vem se desenvolvendo dentro de uma cadeia de saúde inteligente e sustentável. Nesse contexto surgem aplicações referentes a formas de prevenção e educação, diagnóstico, terapia e atendimento prestados por meio da tecnologia digital, independentemente do tempo e do local, chamadas de aplicações *eHealth*. Em paralelo, outra tecnologia que vem se destacando ao longo do tempo é a computação em nuvem. O presente trabalho visa realizar um estudo comparativo entre uma arquitetura convencional e uma *serverless*, ambas na nuvem da *Amazon Web Services* (AWS), afim de investigar a viabilidade de uma arquitura *serverless* em aplicações *eHealth*. Para isto foram identificados tipos de aplicações *eHealth* e requisitos foram levantados para o tipo de aplicação selecionado, com isso arquiteturas foram desenhadas e as aplicações implementadas. Testes de carga realizados nas aplicações geraram dados para que uma coleta fosse feita. Com os resultados obtidos é possível afirmar que a aplicação *serverless* é mais vantajosa em relação a convencional levando em consideração o tempo médio de resposta de requisição e escabilidade. Este trabalho se apresenta, portanto, como uma contribuição no planejamento e desenvolvimento de aplicações *eHealth* e *serverless*, campos que possuem bastante potencial, mas ainda pouco explorado.

Palavras-chave: *eHealth*. Computação em nuvem. Computação sem servidor.

ABSTRACT

The healthcare industry has been increasingly specializing in user demands in recent years. With this, it has been developing within a smart and sustainable health chain. In this context, applications referring to forms of prevention and education, diagnosis, therapy and care provided through digital technology arise, regardless of time and place, called eHealth applications. In parallel, another technology that has stood out over time is cloud computing. This paper aims to carry out a comparative study between a conventional and a serverless architecture, both in the cloud of Amazon Web Services (AWS), in order to investigate the feasibility of a serverless architecture in eHealth applications. For this, types of eHealth applications were identified and requirements were raised for the type of application selected, with that architectures were designed and the applications implemented. Load tests performed in the applications generated data for a collection to be made. With the results obtained it is possible to affirm that the serverless application is more advantageous in relation to the conventional one taking into account the average request response time and scalability. Therefore, this paper presents itself as a contribution in the planning and development of eHealth and serverless applications, fields that have a lot of potential, but still little explored.

Keywords: eHealth. Cloud computing. Serverless computing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AWS	*Amazon Web Services*
HTTP	*Hypertext Transfer Protocol*
OMS	Organização Mundial de Saúde
TIC	Tecnologias de Informação e Comunicação
NIST	*National Institute of Standards and Technology*
WEB	*World Wide Web*
SaaS	*Software as a service*
PaaS	*Platform as a service*
IaaS	*Infrastructure as a service*
FaaS	*Function as a service*
IoT	*Internet of Things*
EHR	*Eletronic Health Records*
EMR	*Eletronic Medical Records*
PHR	*Presonal Health Records*
TI	Tecnologia da Informação
GPRS	*General Packet Radio Service*
UMTS	*Universal Mobile Telecommunications System*
aaS	*as a service*
REST	*Representational State Transfer*
API	*Application Programming Interface*
RF	Requisitos Funcional
RNF	Requisitos Não Funcional
CRM	Conselho Regional de Medicina
IP	*Internet Protocol*
SPA	*Single Page Application*
npm	*Node Package Manager*
URL	*Uniform Resource Locator*
JSON	*JavaScript Object Notation*
CPU	*Central Process Unit*
KB	*Kilobyte*

SUMÁRIO

1 INTRODUÇÃO

Segundo a Organização Mundial de Saúde (OMS), uma saúde melhor é fundamental para a felicidade e o bem-estar humano. Isso também contribui de maneira importante para o progresso econômico, pois populações saudáveis vivem mais, são mais produtivas e economizam mais. Muitos fatores influenciam o estado de saúde e a capacidade de um país de fornecer serviços de saúde de qualidade para seu povo (WHO, 2016).

Nos últimos anos, o setor de *healthcare* ("cuidados de saúde" em português) vem se especializando cada vez mais nas demandas dos usuários. E para tal, ele vem se desenvolvendo dentro de uma cadeia de saúde inteligente e sustentável. Embora extremamente necessário e recorrido, este setor na maioria das regiões do mundo enfrentam desafios para gerenciar o envelhecimento da população em rápido crescimento, indivíduos com condições crônicas, mortalidade infantil, surtos frequentes de epidemias de doenças e ambientes agressivos com baixa sanidade, falta de recursos de água potável e poluição crescente (FARAHANI *et al.*, 2018).

Em resposta a esses desafios, desde os anos 90, as Tecnologias de Informação e Comunicação (TIC) - estimuladas principalmente pelo crescimento e sucessso da Internet - exerceram um papel importante na melhoria do acesso, na eficiência, na qualidade e, portanto, na eficácia de qualquer processo relacionado a *healthcare* (ACETO *et al.*, 2018).

Nesse contexto, essas tecnologias da informação e comunicação para reforçar a saúde e os cuidados de saúde são conhecidas como aplicações *eHealth*. Aplicações estas que referem-se a formas de prevenção e educação, diagnóstico, terapia e atendimento prestados por meio da tecnologia digital, independentemente do tempo e do local. Como um conceito abrangente, inclui noções associadas, como telemedicina, *mHealth*, *tele-care*, *ePublic health*, *eMental health* ou *tele-health* (OSSEBAARD; GEMERT-PIJNEN, 2016). O termo *eHealth* vem se tornando bem mais frequente em estudos recentes, da mesma forma que aplicações concretas que dão suporte aos cuidados com a saúde estão cada vez mais presentes no cotidiano das pessoas uma vez que esses cuidados estão se tornando um hábito comum na vida das pessoas.

Paralelamente a isso, um conceito que está bastante relacionado com essas aplicações é a computação em nuvem. Este que por sua vez é definido pelo *National Institute of Standards and Technology* (NIST) (MELL; GRANCE, 2011) como um modelo para permitir acesso onipresente, conveniente e de rede sob demanda a um conjunto compartilhado de recursos de computação configuráveis que podem ser rapidamente provisionados e liberados com o mínimo esforço de gerenciamento ou interação do provedor de serviços. Em vista disso, aplicações

eHealth se beneficiam das vantagens providas por essa tecnologia para solucionar problemas e melhorar seus serviços.

Além disso, os serviços de computação baseados em nuvem cresceram em popularidade nos últimos anos (WANG; WEI, 2015). Os recentes avanços em computação em nuvem têm sido amplamente utilizados em diferentes domínios e altamente empregados no *design* de soluções baseadas em nuvem na área de *healthcare*. Com as soluções baseadas na nuvem, os serviços podem ser acessados por pacientes de qualquer lugar usando seus próprios telefones celulares ou navegadores da *World Wide Web* (WEB) (GHANEM; ALKHAL, 2018).

Contractor e Patel (2017) apresentaram modelos de serviços providos pela computação em nuvem que descrevem o escopo e o controle de uma organização sobre o ambiente de computação e caracteriza o nível de abstração para seu uso. Além disso, afirmaram que dentre esses modelos de serviço, os mais conhecidos e frequentemente usados são *Software as a service* (SaaS), *Platform as a service* (PaaS) e *Infrastructure as a service* (IaaS).

Contudo, outro modelo de serviço que vem crescendo e sendo largamente usado é *Function as a service* (FaaS). Este que é também conhecido como *serverless* ("sem servidor" em português). O paradigma sem servidor tem vantagens para consumidores e provedores. Do ponto de vista do consumidor, um desenvolvedor de nuvem não precisa mais provisionar e gerenciar servidores, máquinas virtuais ou contêineres como o componente básico da computação para oferecer serviços distribuídos. Em vez disso, o foco está na lógica de negócios, definindo um conjunto de funções cuja composição permite o comportamento desejado do aplicativo (BALDINI *et al.*, 2017).

Em Lloyd *et al.* (2018), os autores afirmaram que as plataformas de computação sem servidor integram o suporte a escalabilidade, disponibilidade e recursos de tolerância a falhas diretamente como recursos da estrutura. Afirmaram também que os benefícios prometidos tornam a plataforma muito atraente para hospedar qualquer aplicativo. E apresentaram uma investigação abrangente sobre os fatores que influenciam o desempenho de microsserviços proporcionados pela abordagem *serverless*.

Logo, a utilização de uma abordagem *serverless* entra como uma solução para desafios como desempenho e confiabilidade, apontados por Rahmani *et al.* (2018), e também escalabilidade, apontada por Farahani *et al.* (2018), desafios estes causados pelo paradigma atual dos sistemas de saúde, já que estes sistemas estão caminhando para serem cada vez mais inteligentes e onipresentes.

Após estabelecer e fundamentar a compatibilidade dos desafios em aplicações *eHealth* com as vantagens de abordagens *serverless*, este trabalho visa investigar a viabilidade de abordagens *serverless* em comparação a abordagens convencionais em termos de escalabilidade e desempenho em aplicações *eHealth*. Para isso, serão coletados dados de aplicações, provedor de nuvem e de experimentos a serem realizados. Com isso, podem se beneficiar deste trabalho arquitetos e desenvolvedores de aplicações *eHealth*, interessados em aplicações *serverless* e quem mais tiver interesse nessas áreas.

1.1 Objetivos

Nesta Seção, serão apresentados os objetivos deste trabalho.

1.1.1 Objetivo geral

O objetivo geral deste trabalho é realizar uma comparação entre uma arquitetura *serverless* e uma arquitetura convencional utilizadas em aplicações *eHealth* através dos atributos de qualidade escalabilidade e desempenho.

1.1.2 Objetivos Específicos

a) Identificar tipos de aplicações *eHealth* que podem ser beneficiados por uma abordagem *serverless*.
b) Identificar requisitos de aplicações *eHealth*.
c) Desevolver uma arquitetura *serverless* e uma arquitetura convencional e implementá-las como aplicações *eHealth*.
d) Levantar métricas para avaliar os atributos de qualidade escalabilidade e desempenho em aplicações *serverless*.

1.2 Organização

Este trabalho está organizado da seguinte forma: no Capítulo 2, são retratados os trabalhos relacionados com o presente. No Capítulo 3, são apresentados os termos e conceitos necessários para o entendimento do trabalho. O Capítulo 4, contém os procedimentos metodológicos. No Capítulo 5, são apresentados a execução de alguns procedimentos. O Capítulo 6, expõe os resultados obtidos. Por fim, o Capítulo 7 apresenta as conclusões e trabalhos futuros.

2 TRABALHOS RELACIONADOS

Para um melhor embasamento deste trabalho foi realizada uma revisão bibliográfica que consistiu em buscas de trabalhos nas principais bases de bibliotecas virtuais (ACM[1], IEEE[2], ScienceDirect[3] e Springer[4]) e motores de busca como Google Acadêmico[5]. Neste Capítulo, os trabalhos resultantes dessa revisão que foram considerados relacionados com o presente trabalho são apresentados e discutidos a fim de levantar semelhanças e diferenças, bem como validar e viabilizar a proposta apresentada.

Nastic *et al.* (2017) apresentaram desafios que desenvolvedores sofrem ao realizarem análises de dados em aplicações de borda, como processo manual e propensão a erros. Nesse contexto, com o objetivo de fornecer uma plataforma *full-stack* para suportar análises de dados em tempo real em nuvem e borda de maneira uniforme, os autores propuseram uma abordagem que implementa a análise de dados em tempo real na nuvem em aplicativos de computação de borda, esta na qual o processamento acontece no local físico (ou próximo) do usuário ou da fonte de dados. Além disso, apresentaram um modelo de aplicação, discutem seus principais requisitos e desafios de design com base em cenários reais de casos de uso em *healthcare*.

A grande semelhança entre Nastic *et al.* (2017) e este trabalho está na apresentação de uma arquitetura de referência utilizando *serverless* para a melhoria de algum tipo de aplicação. Ambos utilizam aplicações *eHealth*. A diferença entre os trabalhos está no fato de que este faz comparações entre uma arquitetura *serverless* e uma convencional, o que não ocorre em Nastic *et al.* (2017). Além disso, computação de borda não entra no escopo deste trabalho.

Ghanem e Alkhal (2018) apontaram desafios enfrentados por pessoas que sofrem de Alzheimer e seus familiares como o esquecimento de rostos e nomes, vagando e se perdendo. Com o objetivo de ajudar o paciente como se o cuidador estivesse fisicamente com ele o tempo todo, apresentaram um sistema baseado em nuvem como uma solução para atender às necessidades dos pacientes com Alzheimer e de seus cuidadores. Para isso, propuseram dois aplicativos móveis para otimizar o processo de prestação de cuidados.

Os pontos em comum entre o presente trabalho e Ghanem e Alkhal (2018) estão na proposta de uma arquitetura *serverless* para a solução de problemas em aplicações *eHealth*.

[1] https://dl.acm.org
[2] https://ieeexplore.ieee.org
[3] https://www.sciencedirect.com
[4] https://link.springer.com
[5] https://scholar.google.com

Entretanto, as diferenças são notadas na delimitação do público alvo da solução, assim não demonstram funcionalidades genéricas ou comuns à outros tipos de aplicações *eHealth*. Além disso, no presente trabalho são realizadas comparações entre uma abordagem *serverless* e uma convencional, o que não acontece em Ghanem e Alkhal (2018).

Manogaran *et al.* (2018) retrataram a dificuldade para processar e analisar *big data* para encontrar informações de valor que podem ser usadas na tomada de decisões e também sobre a segurança dos dados, esta que é um requisito essencial em sistemas de *big data* de assistência médica. Nesse cenário, propuseram uma arquitetura para a implementação da *Internet of Things (IoT)* para armazenar e processar *big data*, obtidas de sensores, para aplicações *healthcare*. A arquitetura proposta consiste em duas sub-arquiteturas principais, a primeira chamada Meta Fog-Redirection (MF-R) que usa tecnologias de *big data*, como Apache Pig e Apache HBase, para coleta e armazenamento dos dados dos sensores. A segunda usa agrupamento e escolha (GC) que é usada para garantir a integração da *fog computing* com a computação em nuvem.

Manogaran *et al.* (2018) assemelha-se com o trabalho proposto por apresentarem uma solução para aplicações de *healthcare*, entretanto para uma funcionalidade específica dessas aplicações que é análise de dados. Além disso, fazem uma comparação com outros modelos de análise, embora os elementos comparados não se apliquem ao presente trabalho, se assemelham com as comparações que serão realizadas.

Azimi *et al.* (2017) apontaram que sistemas IoT centralizados em nuvem carecem de confiabilidade, pontualidade e disponibilidade, bem como terceirizar totalmente a análise de dados na borda da rede pode resultar em diminuição do nível de precisão e adaptabilidade. Abordando essas questões propuseram uma arquitetura hierárquica de computação (HiCH), para sistemas de monitoramento de saúde baseados em IoT. O HiCH se beneficia dos recursos oferecidos pela *fog computing* e a computação em nuvem e apresenta uma metodologia de gerenciamento personalizada para sistemas IoT de *healthcare*. Demonstraram a eficácia do HiCH por meio de uma abrangente avaliação de desempenho e avaliação em um estudo de caso contínuo de monitoramento remoto da saúde, com foco na detecção de arritmia em pacientes que sofrem de doenças cardiovasculares.

A interseção entre este trabalho e o de Azimi *et al.* (2017) ocorre na apresentação de uma arquitetura para solução de problemas em aplicações *eHealth*. Entretanto, utilizaram *IoT* e *fog computing*, que não entram no escopo deste trabalho. As principais diferenças são a não utilização de *serverless* e a delimitação a aplicações voltadas a pacientes com doenças

cardiovasculares.

Lee *et al.* (2018) avaliaram ambientes de computação *serverless* chamando funções em paralelo para demonstrar o desempenho e a taxa de transferência de processamentos de dados distribuídos. Apresentaram resultados de taxa de transferência, largura de banda da rede, entrada e saída de arquivo e desempenho em relação à chamadas simultâneas de aplicações executando em ambientes *serverless*. Além disso, implementam uma série de funções para processamento de dados distribuídos para lidar com a elasticidade e fizeram uma comparação entre a computação *serverless* e máquinas virtuais para extrair suas diferenças obtendo eficiência de custo e utilização de recursos.

A principal semelhança entre Lee *et al.* (2018) e o presente trabalho está na comparação de abordagens *serverless* com abordagens convencionais, como a utilização de máquinas virtuais, analisando características e provedores diferentes. A diferença está no fato de que Lee *et al.* (2018) não aplicaram as abordagens comparadas em aplicações *eHealth*, ao contrário do presente trabalho que faz uso de funcionalidades de aplicações *eHealth* para comparar as abordagens apresentadas.

Rahman e Hasan (2019) informaram que existem muitos desafios para processar e computar *big data*, como gerenciamento de servidor, armazenamento, *clustering* e implantação de algoritmos. Nesse contexto, apresentaram uma arquitetura *serverless* para análise de *big data*, mostrando como implementar, manter e gerenciar essas aplicações na AWS. Além disso, demonstram a diferença entre a análise de dados tradicional e a análise de *big data* em um sistema com arquitetura *serverless*.

Apesar de não incluirem *eHealth* no projeto, Rahman e Hasan (2019) fizeram uso de uma arquitetura *serverless* para prover uma solução para *big data*. Além disso, fizeram uma comparação entre análise de dados tradicional com análise de dados utilizando *serverless* que também se assemelha com a comparação que será realizada no presente trabalho. As diferenças entre ambos os trabalhos é que Rahman e Hasan (2019) não focaram em aplicações *eHealth* e sim em análise de *big data*.

O Quadro 1 apresenta um comparativo entre os trabalhos relacionados com o trabalho proposto, destacando as tecnologias utilizadas em cada um, se propõe ou não arquitetura, se realizam testes para comparação de abordagens e por fim a finalidade da solução apresentada.

As principais semelhanças entre os trabalhos apresentados é que todos utilizam computação em nuvem. Porém, nenhum apresentam todos os pontos destacados, exceto o

trabalho proposto. Além disso, fica evidente que todos são diferentes quanto a finalidade da aplicação, e quando próximos se diferenciam no nível de abrangência da aplicação, sendo uns mais gerais e outros mais específicos.

Quadro 1 – Comparativo entre os trabalhos relacionados e o trabalho proposto

Trabalhos	**Usa *eHealth***	**Usa *serverless***	**Propôs arquitetura**	**Realiza testes**	**Finalidade da aplicação**
Trabalho Proposto	Sim	Sim	Sim	Sim	Melhoria de atributos de qualidade em aplicações *eHealth*.
Nastic *et al.* (2017)	Sim	Sim	Sim	Não	Dar suporte a análises de dados em tempo real em nuvem e borda de maneira uniforme.
Ghanem e Alkhal (2018)	Sim	Sim	Sim	Não	Aplicações móveis para otimizar o processo de prestação de cuidados a pacientes com Alzheimer.
Manogaran *et al.* (2018)	Sim	Não	Sim	Sim	Armazenar e processar dados escalonáveis de sensores em aplicativos de assistência médica.
Azimi *et al.* (2017)	Sim	Não	Sim	Sim	Sistemas remotos de monitoramento de pacientes com doenças cardiovasculares.
Lee *et al.* (2018)	Não	Sim	Não	Sim	Melhoria na utilização de recursos e eficiência no custo.
Rahman e Hasan (2019)	Não	Sim	Sim	Não	Melhoria no processamento e computação em análise de *big data*

Fonte: Elaborado pelo autor.

No Capítulo seguinte são apresentados os termos e conceitos mais importantes para o acompanhamento e entendimento deste trabalho.

3 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo serão apresentados os principais conceitos para o desenvolvimento deste trabalho. Na Seção 3.1 são tratados os conceitos sobre *eHealth*, tipos de aplicações, aplicações e desafios enfrentados por essa tecnologia. Na Seção 3.2 são introduzidos os conceitos referentes à computação em nuvem, características e modelos de serviço. Na Seção 3.3 são apresentados os conceitos de *serverless*, sua arquitetura e casos de uso. A última Seção 3.4 apresenta os conceitos de atributos de qualidade, focando nos atributos: desempenho e escalabilidade.

3.1 *eHealth*

Shaw *et al.* (2017) afirmaram que apesar do rápido crescimento da pesquisa em *eHealth*, permanece uma falta de consistência na definição e no uso de termos relacionados a esse tema. Definições mais amplamente citadas fornecem amplo entendimento do *eHealth*, mas carecem de clareza conceitual suficiente para operacionalizar o *eHealth* e possibilitar sua implementação na prática, pesquisa, educação e política de assistência à saúde.

Ainda Shaw *et al.* (2017) apontaram que as definições mais detalhadas são geralmente específicas do contexto ou da disciplina, limitando a facilidade de tradução dessas definições em toda a amplitude de perspectivas e situações de *eHealth*. Com isso, este trabalho irá adotar o conceito de *eHealth* definido por Eysenbach (2001) que é o mais citado e aceito pela comunidade científica até o momento encontrado na literatura.

Eysenbach (2001) define *eHealth* como:

> *eHealth* é um campo emergente na interseção de informática médica, saúde pública e negócios, referindo-se a serviços de saúde e informações fornecidas ou aprimoradas pela Internet e tecnologias relacionadas. Em um sentido mais amplo, o termo caracteriza não apenas um desenvolvimento técnico, mas também um estado de espírito, uma maneira de pensar, uma atitude e um compromisso com o pensamento global em rede, para melhorar os cuidados de saúde a nível local, regional e mundial usando a tecnologia da informação e comunicação.

3.1.1 Tipos de Aplicações

Mishra e Rasool (2019) elencaram algumas formas de aplicações *eHealth* que facilitam a abrangência, armazenamento, transferência, recuperação e processamento de dados relacionados à saúde. Afirma ainda que essas formas melhoram a interação entre profissionais de saúde e pacientes ajudando no monitoramento de vários parâmetros biológicos e fisiológicos,

além de garantir tratamentos médicos distantes e oportunos. A Tabela 1 apresenta essas formas e suas definições.

Tabela 1 – Tipos de aplicações *eHealth*

Tipo	Definição
Registro Eletrônico de Saúde (EHR)	Relatório eletronicamente abrangente da saúde do paciente em geral, disponibilizando as informações necessárias ao paciente.
Registro Médico Eletrônico (EMR)	Relatório criado digitalmente a partir de uma prática específica.
Registro Pessoal de Saúde (PHR)	Ambiente privado, seguro e confidencial para manter as informações relacionadas à saúde do paciente.
Suporte virtual à decisão e equipes de assistência médica	Corroborando, trocando e trabalhando equipes de profissionais de saúde que, com a ajuda de equipamentos digitais, coletam informações sobre o paciente com o objetivo de tomar melhores decisões com o aprimoramento do conhecimento.
Prescrição eletrônica	Com a ajuda dessa estrutura tecnológica, a instituição de saúde pode escrever e enviar diretamente a receita eletronicamente para a farmácia.
Consultas eletrônicas	O agendamento e a fixação de consultas para qualquer instituição de saúde podem ser feitos de maneira mais rápida e fácil, além de reduzir o tempo de espera dos pacientes neste serviço on-line.
mHealth (*mobile health*)	Práticas de saúde por meio de dispositivos móveis e outros dispositivos sem fio.
Telemedicina e Telessaúde	A TIC, fornecendo dados e serviços de saúde remotamente para várias pessoas doentes.

Fonte: Adaptado de Mishra e Rasool (2019).

Dado essas definições é possível inferir a importância da tecnologia no setor de saúde provendo valor, segurança, confiabilidade e a possibilidade de existência desses meios. A próxima Seção aponta as aplicações *eHealth* que compõem os tipos apresentados.

3.1.2 Aplicações

Após conhecer os tipos de aplicações *eHealth* é importante conhecer as aplicações que os constituem. Para tal, a Tabela 2 exibe algumas aplicações e uma breve definição expostas por Maksimović e Vujović (2017).

Essas aplicações foram também mencionadas por Mishra e Rasool (2019) com base em Maksimović e Vujović (2017). É possível observar a partir das informações listadas o quão

Tabela 2 – Aplicações *eHealth*

Aplicação	Definição
HEARTFAID	Permite o diagnóstico pontual e *insights* mais eficazes sobre doenças cardíacas em idosos.
ALARM-NET	Uma Rede de Monitoramento de Vida Assistida e Residencial para assistência médica onipresente e adaptável.
CAALYX	Experiência Completa de Assistência ao Ambiente com base no uso de dispositivos leves e portáteis que medem os sinais vitais de um paciente e alertam automaticamente o profissional em caso de emergência.
TeleCARE	Permite o desenvolvimento de uma infraestrutura comum configurável, útil para idosos e profissionais de saúde.
CHRONIC	Um ambiente integrado de TI para o atendimento de pacientes crônicos.
MyHeart	Com a ajuda de sistemas eletrônicos e têxteis inteligentes e serviços adequados oferece os meios para lidar com doenças cardiovasculares.
OLDES	Permite uma ampla gama de serviços para idosos usando uma tecnologia inovadora de baixo custo.
SAPHIRE	Implementa o monitoramento do pacientes usando a tecnologia do agente e sistemas inteligentes de suporte à decisão.
MobiHEALTH	Facilita o monitoramento on-line e contínuo de sinais vitais, através das tecnologias GPRS e UMTS.
SAPHE	Permite monitoramento contínuo inteligente e discreto da assistência médica via redes de *telecare* com pequenos sensores corporais sem fio e sensores integrados em residências.
DITIS	Um aplicativo móvel de *eHealth* que suporta colaboração em rede para cuidados de saúde em casa.
AXARM	Uma ferramenta extensível de assistência e monitoramento remoto para telerreabilitação de doenças neurodegenerativas.
VirtualECare	Um sistema multiagente inteligente para monitorar a saúde e interagir com idosos.

Fonte: Adaptado de Maksimović e Vujović (2017).

abrangente são essas aplicações, atuando nas mais diversas sub-áreas de *healthcare*, e com isso o quanto são importantes para a saúde e bem-estar de pacientes.

3.1.3 Desafios de aplicações eHealth

Dado a importância de aplicações *eHealth* torna-se necessário que essas aplicações consigam enfrentar incidentes de segurança e prover um serviço adequado garantindo disponibi-

lidade e tempo de resposta. Para isso, arquiteturas adequadas devem ser projetadas para essas aplicações a fim de que cumpram com esses requisitos.

Abolade (2018) apontou 6 desafios que essas aplicações enfrentam nos países em desenvolvimento que envolvem várias áreas da tecnologia. São eles:

1. Falta de *stakeholders* qualificados: Os *stakeholders* ou partes interessadas apresentam baixo nível de educação e habilidades tecnológicas pobres. Existe então o desafio de treiná-los e capacitá-los.
2. Técnico e Operacional: Aplicações *eHealth* enfrentam problemas do ponto de vista técnico e operacional como falta de uma estrutura adequada para as características da qualidade da informação e percas e desaparecimento de registros eletrônicos de saúde.
3. Social e Cultural: O desenvolvimento de aplicações *eHealth* na sociedade tem suas próprias peculiaridades, que são bastante importantes devido às limitações culturais e sociais nos países em desenvolvimento. Enfrentando problemas como falta de educação ampla e contínua para uso público dos serviços de *eHealth* e resistência a mudanças devido aos hábitos.
4. Legal: Definitivamente, um dos fatores mais importantes relacionados à alocação e desenvolvimento de *eHealth* é a provisão de aspectos legais e o equilíbrio de toda a coleção de leis e regulamentos. Enfrenta problemas como falta de leis adequadas existentes sobre direitos pessoais e manutenção do ambiente privado dos pacientes e a necessidade de desenvolver uma estrutura legal e gerenciá-la na área da saúde.
5. Financeiro: As limitações de gerenciamento no uso da força de trabalho, fontes físicas e financeiras e a necessidade dos gerentes de abordar sistematicamente e concentrar as produtividades dos processos durante a operação da maioria dos recursos sob os quais eles têm autoridade aumentam a importância dos sistemas de informação existentes e de seus mecanismos operacionais. Portanto, é importante considerar desafios como a necessidade de investimento e alocação de orçamento regular no campo de *eHealth* e uso de tecnologias relevantes na seção de saúde e a falta de estrutura para análise econômica de benefícios e resultados do controle remoto da saúde.
6. Obstáculos Culturais: O maior obstáculo a ser enfrentado é a mudança de percepção sobre o compartilhamento de dados como um todo, para promover uma verdadeira cultura de integração. Isso significa que os dados gerados, enquanto gerenciados e protegidos, ainda precisam ser compartilhados entre entidades autorizadas.

Este trabalho fará uso desses desafios para nortear e embasar a solução proposta.

3.2 Computação em Nuvem

O *National Institute of Standards and Technology* (NIST) (MELL; GRANCE, 2011) define computação em nuvem como um modelo para permitir acesso onipresente, conveniente e de rede sob demanda a um conjunto compartilhado de recursos de computação configuráveis que podem ser rapidamente provisionados e liberados com o mínimo esforço de gerenciamento ou interação do provedor de serviços. Essa é a definição mais difundida e aceita na literatura, portanto será adotada nesse trabalho.

O termo "computação em nuvem" deriva à medida que as informações acessadas são encontradas no espaço virtual, ou seja, a nuvem que permite ao usuário trabalhar remotamente e permite que o usuário obtenha acesso ao seu serviço, independentemente do local (HARKUT *et al.*, 2019). Os serviços baseados em nuvem são acessíveis em qualquer lugar do mundo, desde que a conexão à Internet esteja disponível (HARKUT *et al.*, 2019).

Dutta e Dutta (2019) realiza um estudo comparativo entre *Amazon Web Services*[1] (AWS), *Microsoft Azure*[2] e *Google Cloud Platform*[3] que aponta serem os maiores provedores de serviço em nuvem do mercado, destacando a AWS como a mais popular. Com base nisso, este trabalho irá utilizar os serviços em nuvem oferecidos pela AWS.

O modelo de nuvem apresentado por Mell e Grance (2011) é composto por cinco características essenciais, quatro modelos de implantação e três modelos de serviço.

3.2.1 Características

São cinco as principais características do serviços de computação em nuvem apresentados por Mell e Grance (2011):

- Autoatendimento sob demanda: Um consumidor pode provisionar recursos como tempo de servidor e armazenameto de maneira automática sem interagir diretamente com o provedor.
- Amplo acesso à rede: Plataformas heterogêneas como celulares e computadores podem acessar recursos disponíveis na rede através de mecanismos padrões.
- Agrupamento de recursos: Recursos como armazenamento e memória são agrupados para

[1] https://aws.amazon.com
[2] https://azure.microsoft.com
[3] https://cloud.google.com

atender vários consumidores usando um modelo de multilocatário, com diferentes recursos físicos e virtuais dinamicamente atribuídos e reatribuídos de acordo com a demanda do consumidor.

- Rápida elasticidade: Há a possibilidade de provisionar elasticamente e liberar, automaticamente em alguns casos, recursos que escalam para mais e para menos dependendo da demanda.
- Serviço medido: O uso de serviços como armazenamento e contas de usuário ativas são controlados e otimizados pelos provedores em algum nível de abstração apropriado ao tipo de serviço. Esse controle fornece transparência para o provedor e para o consumidor.

Dado essas características, a computação em nuvem é, sem dúvidas, uma ótima solução para aplicações que demandam escalabilidade e desempenho. Devido a isso, este trabalho fará uso dessa tecnologia.

3.2.2 Modelos de implantação

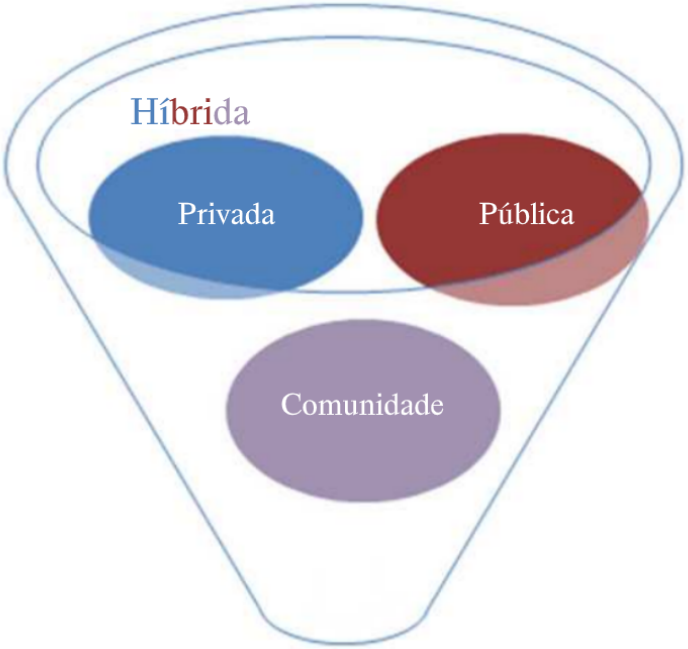
Os modelo de implantação referem-se a onde os recursos estão e quem tem acesso ou controle a esses recursos. No modelo de nuvem exposto pelo NIST (MELL; GRANCE, 2011), são definidos quatro modelos de implantação, explicados a seguir:

- Nuvem privada: Neste modelo, uma única organização composta por vários consumidores provisiona exclusivamente a infraestrutura de nuvem. Esta nuvem, pode ser de propriedade, gerenciada e operada pela organização, por terceiros ou por uma combinação deles, e pode existir dentro ou fora das instalações.
- Nuvem comunidade: Uma comunidade específica de consumidores pertencentes a alguma organização que compartilham preocupações como requisitos de segurança e política provisiona exclusivamente a infraestrutura de nuvem. A nuvem pode ser de propriedade, gerenciada e operada por uma ou mais organizações da comunidade, terceiros ou alguma combinação delas, e pode existir dentro ou fora das instalações.
- Nuvem pública: Neste modelo, a infraestrura de nuvem é aberta para acesso do público em geral. Pode ser de propriedade, gerenciado e operado por uma empresa, organização acadêmica ou governamental, ou alguma combinação deles. Existe nas instalações do provedor de nuvem.
- Nuvem híbrida: A infraestrutura de nuvem é uma composição de duas ou mais infraestruturas de nuvem distintas (privadas, comunitárias ou públicas) que permanecem entidades

únicas, mas são unidas por alguma tecnologia padronizada ou proprietária que permite a portabilidade de dados e aplicações.

A Figura 1 apresenta os modelos de implantação explicados ilustrando que a nuvem híbrida faz uma composição de dois ou mais modelos tanto pela coloração do nome quanto pela forma tridimensional de conter todas as outras dentro de si.

Figura 1 – Comparação entre os modelos de implantação



Fonte: Adaptado de Nasr e Elbooz (2018).

Como citado anteriormente, este trabalho utilizará a AWS que é um serviço de nuvem pública oferecido pela Amazon (ZHANG *et al.*, 2015). A próxima Seção apresenta os modelos de serviço fornecidos por provedores de nuvem.

3.2.3 Modelos de Serviço

O termo *as a service* (aaS), "como um serviço" em português, é utilizado para complementar modelos de serviços distribuídos na nuvem. De acordo com a definição apresentada pelo NIST (MELL; GRANCE, 2011), existem três principais modelos de serviços, que são:

– **Software como serviço (SaaS)**: Aplicações do provedor executadas na nuvem que podem ser acessadas pelos consumidores através de vários dispositivos clientes por meio de interfaces, como navegadores WEB. Neste modelo o consumidor não gerencia ou controla qualquer infraestrutura de nuvem subjacente como rede e armazenamento (MELL; GRANCE, 2011).

Em vez de comprar, instalar e manter aplicativos de software em bases de dados ou computadores locais, uma organização precisa apenas de uma conexão de Internet para assinar e acessar aplicativos SaaS hospedados nos servidores do provedor de serviços (TADAPANENI, 2017).

– **Plataforma como serviço (PaaS)**: Neste modelo o consumidor pode implantar aplicações criadas ou adquiridas em uma infraestrutura de nuvem. O consumidor não gerencia ou controla qualquer infraestrutura de nuvem subjacente, entretanto tem controle sobre as aplicações implantadas e as definições de configuração para o ambiente de hospedagem de aplicações (MELL; GRANCE, 2011).

  Uma vantagem do PaaS é que ele reduz o tempo necessário para desenvolver e implantar softwares e, assim, ajudar as empresas a obter uma vantagem competitiva sobre seus concorrentes. A escalabilidade automática também é outra vantagem do PaaS. Isso ocorre porque o PaaS permite que as organizações escalem e se adaptem com flexibilidade à crescente demanda de largura de banda da rede ou aos requisitos adicionais de espaço de armazenamento (TADAPANENI, 2017).

– **Infraestrutura como serviço (IaaS)**: Esse modelo fornece ao consumidor processamento, rede e outros recursos fundamentais nos quais é possível implantar e executar softwares como sistemas operacionais. O consumidor não gerencia ou controla qualquer infraestrutura de nuvem subjacente, mas tem controle sobre sistemas operacionais e aplicações implantadas (MELL; GRANCE, 2011).

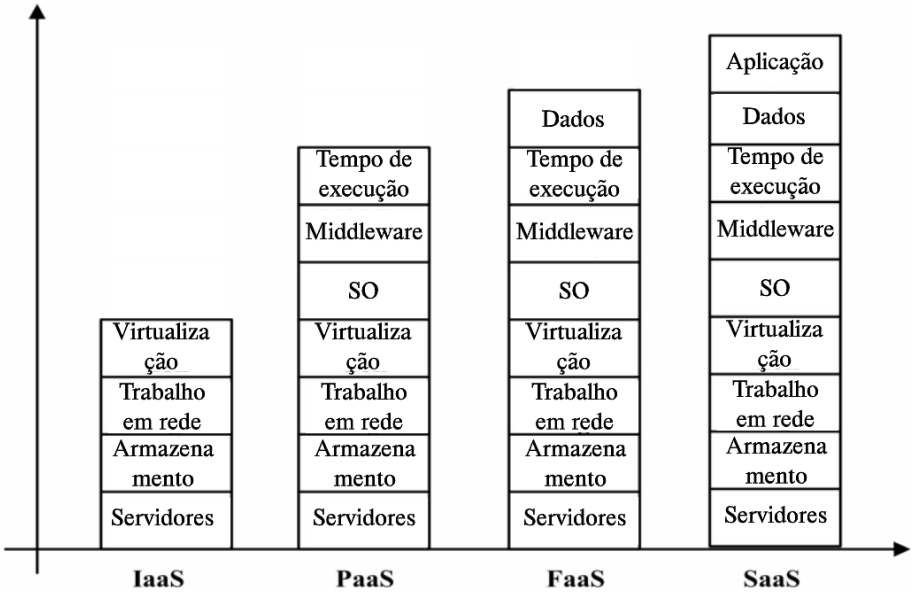
  Semelhante ao PaaS, IaaS também é uma solução econômica que é relativamente mais barata em comparação com um data center local. IaaS também é altamente confiável e disponível, garantindo assim o crescimento contínuo dos negócios. Por outro lado, o IaaS depende da velocidade de conexão com a Internet e isso pode dificultar as operações comerciais críticas se a rede do provedor de serviços for lenta. Outra desvantagem é que o IaaS é um pouco complexo de operar e, portanto, uma organização pode incorrer em um custo adicional de treinamento de sua equipe técnica (TADAPANENI, 2017).

Fica claro que cada um dos modelos apresentados por Mell e Grance (2011) tem seu próprio conjunto de benefícios que atendem a necessidades distintas de negócios variados. É importante conhecê-los pois a decisão de utilizar um modelo de serviço deve ser tomada avaliando o que eles oferecem e se isso atende aos requisitos desejados.

Novos modelos de serviço continuam surgindo como uma espécie de derivação

e evolução dos modelos principais. Baldini *et al.* (2017) aponta que essa evolução traz uma tendência em direção a níveis mais altos de abstrações de modelos de serviços, como Funções como serviço (FaaS), também conhecido como *serverless*. A Figura 2 apresenta de maneira gráfica uma comparação entre os modelos de serviço com relação aos níveis de implantação já mencionados, incluindo FaaS. FaaS é o modelo de serviço utilizado neste trabalho e que será discutido na próxima Seção.

Figura 2 – Comparação do nível de implantação entre modelos de serviço



Fonte: Adaptado de Geng *et al.* (2018).

3.3 *Serverless*

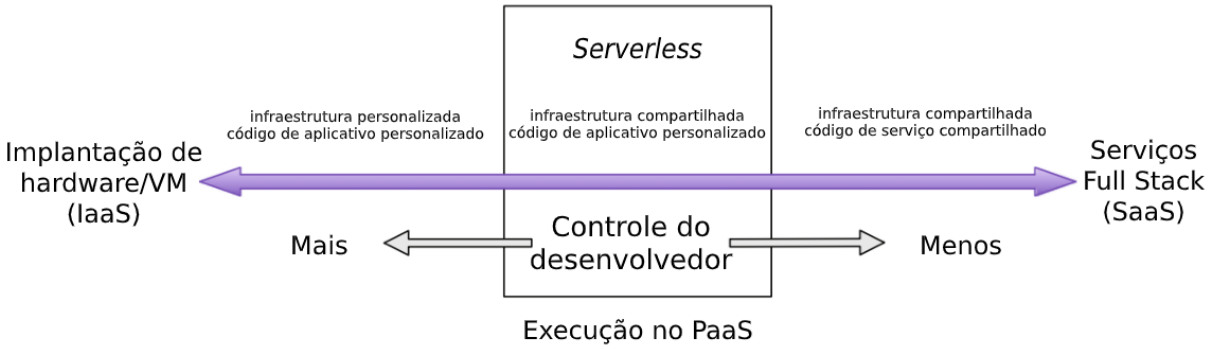
Serverless é um termo que descreve um modelo e arquitetura de programação em que pequenos trechos de código são executados na nuvem sem nenhum controle sobre os recursos nos quais o código é executado (BALDINI *et al.*, 2017). Essa é a definição de *serverless* adotada neste trabalho.

As plataformas *serverless* fornecem funções como serviço (FaaS) para os usuários finais, prometendo custos reduzidos de hospedagem, alta disponibilidade, tolerância a falhas e elasticidade dinâmica para hospedar funções individuais conhecidas como microsserviços (LLOYD *et al.*, 2018).

Definir o termo *serverless* de forma breve é difícil, pois a definição se sobrepõe a outros termos como PaaS e SaaS (BALDINI *et al.*, 2017). Baldini *et al.* (2017) aponta uma maneira de explicá-lo considerando os vários níveis de controle do desenvolvedor sobre

a infraestrutura da nuvem, conforme ilustrado na Figura 3. No IaaS o desenvolvedor tem mais controle sobre a infraestrutura operacional da nuvem ao contrário de PaaS e SaaS onde o desenvolvedor tem menos controle sobre a infraestrutura. No centro da figura, o desenvolvedor tem controle sobre o código implementado e não precisa se preocupar com aspectos operacionais.

Figura 3 – Comparação dos níveis de controle do desenvolvedor



Fonte: Adaptado de Baldini *et al.* (2017)

Ao utilizar esse modelo, este trabalho visa prover escalabilidade e desempenho para aplicações *eHealth*.

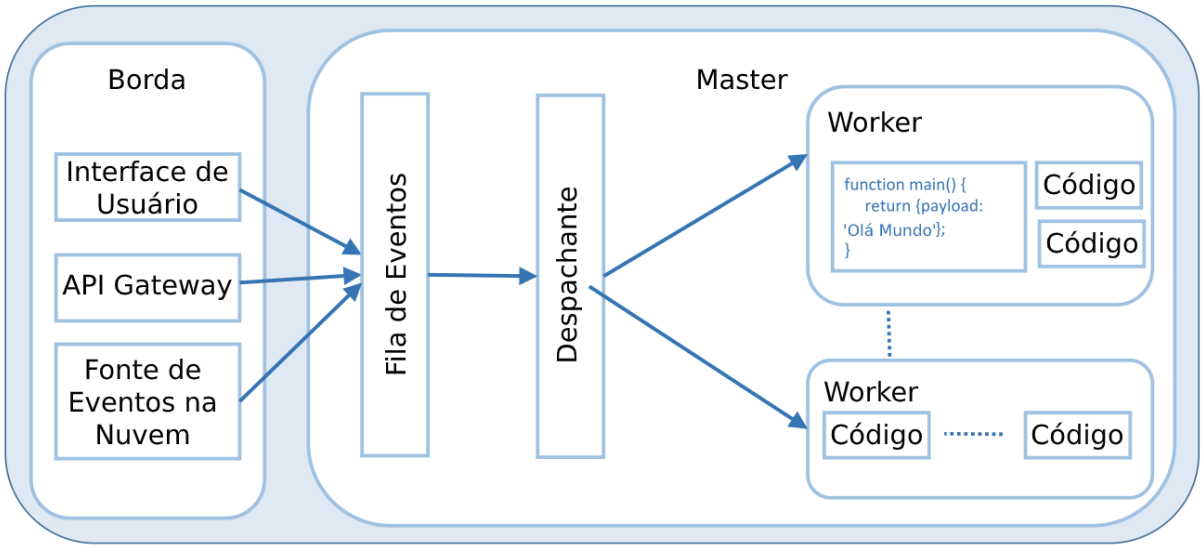
3.3.1 Arquitetura

Rajan (2018) aponta que uma arquitetura *serverless* é decomposta em "gatilhos" (eventos) e "ações" (funções), contando com a existência de uma plataforma que fornece hospedagem e ambiente de execução.

Baldini *et al.* (2017) reconhece a principal capacidade de uma plataforma *serverless* como sendo um sistema de processamento de eventos, conforme mostrado na Figura 4. O serviço deve gerenciar funções definidas pelo usuário, receber um evento enviado por HTTP ou por uma fonte de eventos, determinar para qual função despachar o evento, localizar uma instância existente da função ou criar uma nova instância, enviar o evento para a instância da função, aguardar uma resposta, reunir logs de execução, disponibilizar a resposta ao usuário e parar a função quando não for mais necessária.

Baldini *et al.* (2017) aponta ainda que um dos principais desafios é implementar essa funcionalidade, considerando métricas como custo, escalabilidade e tolerância a falhas. A plataforma deve iniciar rápida e eficientemente uma função e processar sua entrada. A plataforma também precisa enfileirar eventos, e com base no estado das filas e na taxa de chegada de eventos, agendar a execução de funções, gerenciar a finalização e desalocar recursos de instâncias ociosas.

Figura 4 – Arquitetura de uma plataforma *serverless*



Fonte: Adaptado de Baldini *et al.* (2017).

Além disso, a plataforma precisa considerar cuidadosamente como dimensionar e gerenciar falhas na nuvem.

3.3.2 Casos de Uso

Abordagens *serverless* vem sendo largamente utilizadas em várias áreas distintas. Para um melhor entendimento dos problemas que essa tecnologia tenta resolver, Rajan (2018) aponta quatro casos de uso retirados de Zhang e Zhang (2018) que são listados a seguir:

1. Computação Acionada por Eventos: Nos aplicativos de negócios de processamento multimídia, grandes volumes de arquivos são frequentemente carregados em serviços de armazenamento para processamento. Esse cenário comercial envolve uma variedade de dispositivos, como computadores de mesa ou telefones celulares, acessando diferentes tipos de arquivos de upload de conteúdo multimídia, como imagens, vídeos e arquivos de texto.
2. Transmissão de vídeo ao vivo: Em cenários de transmissão de vídeo ao vivo, o nó de síntese da transmissão recebe fluxos de áudio e vídeo dos hosts. Os dados coletados podem ser sintetizados com base na função de computação. Finalmente, o fluxo de vídeo sintetizado precisa ser enviado para a rede de fornecimento.
3. Processamento de dados da IoT: Um *framework* IoT precisa de um *design* eficiente de funções que possa receber dados de status de uma variedade de dispositivos inteligentes conectados. Além disso, ele precisa de uma arquitetura eficiente baseada em eventos para

transmitir os dados processados para outros dispositivos ou armazenar no banco de dados.

4. Sistema de Entrega Compartilhada: Um grupo global de restaurantes ou uma empresa baseada em produtos pode precisar de um sistema de notificação baseado em eventos para que o entregador possa encontrar o vendedor mais próximo para buscar o produto a ser entregue.

3.3.3 Comparação de abordagens serverless dos principais provedores de nuvem

Rajan (2018) realiza uma comparação entre modelos de serviço FaaS dos principais provedores de nuvem do mercado, mostrados no Quadro 2. Essa comparação é importante pois a partir dela é possível dimensionar o cenário atual desses serviços e quais atendem aos requisitos desejados.

Quadro 2 – Comparação entre as caracteríticas *serverless* dos maiores provedores de nuvem

Característica	**AWS Lambda**	**Google Cloud Function**	**Microsoft Azure Function**
Introdução	2015	2016	2016
Escalabilidade	Automática	Automática	Automática
Máximo de funções	Ilimitado	20 por projeto	Depende do gatilho e recursos disponíveis
Linguagens suportadas	Javascript, Java, Python, NodeJS	Javascript	C#, F#, NodeJS, Python, PHP, Bash
Execuções Concorrentes	100 execuções paralelas por conta	Ilimitadas	Baseado no serviço da aplicação
Implantação	Envios ZIP	Envios ZIP, armazenamento na nuvem	Integração com Git, *Representational State Transfer* (REST) *Application Programming Interface* (API)
Alocação de memória	Por função	Não especificado	Por serviço de aplicação
Licença	Código fechado	Código aberto	Código aberto
Modelo de precificação	Por execução de código	Por execução de código	Por execução de código
Arquitetura orientada a eventos	S3, SNS, DynamoDB, Kinesis, Cloud Watch	Cloud Pub, Cloud storage objects	Serviços do Azure e de terceiros

Fonte: Adaptado de Rajan (2018).

Este trabalho utiliza o serviço de FaaS provido pela AWS, o AWS Lambda e demais serviços que auxiliarão no desenvolvimento da solução proposta.

3.4 Atributos de Qualidade

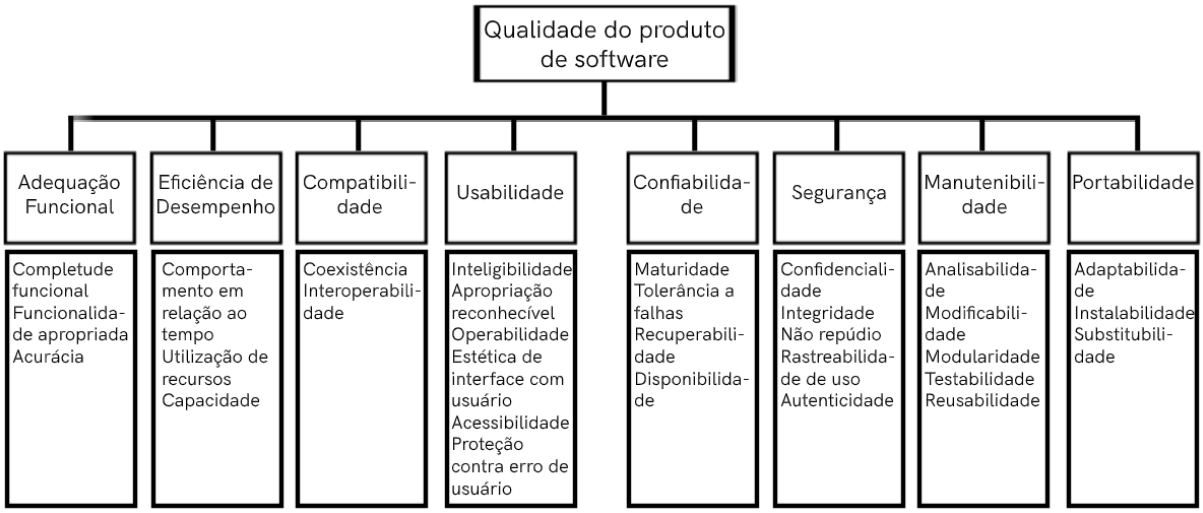
A ISO/IEC 25010 (2011) é um conjunto de padrões que propõe várias características de qualidade a serem levadas em consideração na avaliação de um produto de *software*. Esta define atributo de qualidade como a propriedade inerente ou característica de uma entidade que pode ser distinguida quantitativa ou qualitativamente por meios humanos ou automatizados. Este trabalho adota essa definição.

Ainda segundo a ISO/IEC 25010 (2011), uma das maneiras de avaliar *softwares* é utilizando medidas de *software*. Essas medidas são categorizadas em três (SANTOS *et al.*, 2016):

- **medidas de qualidade interna**: refere-se a medição de atributos de qualidade estáticos relacionado a sua arquitetura, por exemplo a quantidade de linhas de código e nível de acoplamento.
- **medidas de qualidade externa**: deriva-se da avaliação do comportamento do *software*, por exemplo a quantidade de defeitos encontrados em um teste.
- **medidas de qualidade no uso**: refere-se a quanto o *software* atende as necessidades do usuário em contextos específicos, por exemplo eficiência e satisfação.

Os modelos de qualidade interno e externo são combinados para gerar o modelo de qualidade do produto (ISO/IEC 25010, 2011) contendo 8 características e 31 subcaracterísticas como mostrado na Figura 5.

Figura 5 – Modelo de qualidade do produto



Fonte: Adaptado de ISO/IEC 25010 (2011).

Este trabalho realiza uma comparação entre aplicações implementadas a partir

das arquiteturas propostas com base em medidas de qualidade para os atributos eficiência de desempenho apresentado em ISO/IEC 25010 (2011) e escalabilidade exposto em Oquendo *et al.* (2016). Os atributos citados serão explanados nas sub-seções seguintes.

3.4.1 Eficiência de Desempenho

Como mostrado na Figura 5, a eficiência de desempenho é relativo à quantidade de recursos utilizados sob condições estabelecidas (ISO/IEC 25010, 2011) e é composta pelas sub-características:

- **Comportamento em relação ao tempo**: "grau em que os tempos de resposta e processamento e as taxas de transferência de um produto ou sistema, ao executar suas funções, atendem aos requisitos." (ISO/IEC 25010, 2011)
- **Utilização de recursos**: "grau em que as quantidades e os tipos de recursos usados por um produto ou sistema, ao desempenhar suas funções, atendem aos requisitos." (ISO/IEC 25010, 2011)
- **Capacidade**: "grau em que os limites máximos de um produto ou parâmetro do sistema atendem aos requisitos." (ISO/IEC 25010, 2011)

Este trabalho faz uso de todas as sub-características de eficiência de desempenho para avaliar as arquiteturas propostas.

3.4.2 Escalabilidade

Escalabilidade não é uma característica ou sub-característica de qualidade exposta pelo modelo de qualidade da Figura 5. Para Oquendo *et al.* (2016), escalabilidade se refere a quanto um sistema pode ser efetivamente e eficientemente escalonado para aumentar o uso operacional. Oquendo *et al.* (2016) ainda aponta que em arquitetura de software, escalabilidade se refere a como a arquitetura pode ser dimensionada para lidar com um número crescente de uso de componentes que pode ter várias ordens de magnitude.

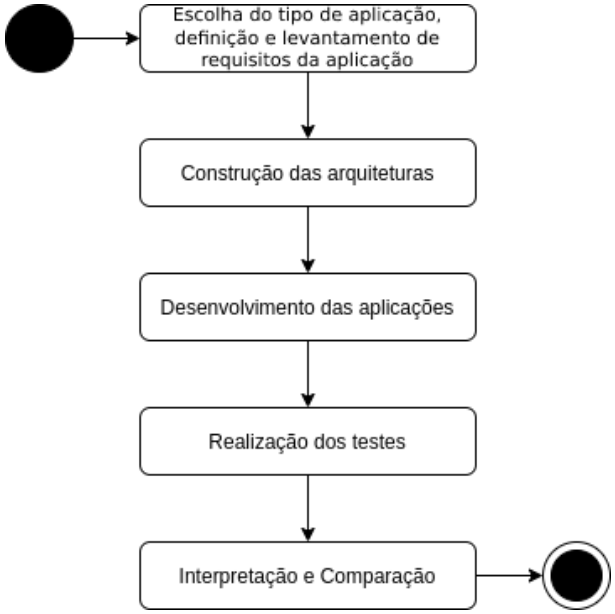
O atributo de qualidade de escalabilidade se refere à qualidade usada para quantificar os efeitos. Isso está geralmente ligado aos efeitos de oscilação de desempenho: tempo de resposta, taxa de transferência e consumo de recursos Oquendo *et al.* (2016).

Este trabalho utiliza escalabalidade para avaliar as arquiteturas propostas.

4 PROCEDIMENTOS METODOLÓGICOS

Neste Capítulo são apresentados os passos necessários para a execução deste trabalho. As cinco seções apresentadas a seguir representam as etapas de desenvolvimento deste trabalho, onde: na Seção 4.1, é explicado a escolha do tipo de aplicação, a definição e levantamento de requisitos da aplicação selecionada; na Seção 4.2 é mostrado a construção das arquiteturas; a Seção 4.3, explica como se deu o desenvolvimento das aplicações; na Seção 4.4, são realizados os testes e coleta dos dados. Por fim na Seção 4.5 é explicado a validação da proposta mediante aos dados coletados e interpretados. A seguir, na Figura 6, é mostrada as etapas da metodologia adotada.

Figura 6 – Etapas do desenvolvimento



Fonte: Elaborado pelo autor.

4.1 Escolha do tipo de aplicação, definição e levantamento de requisitos da aplicação

Com o objetivo de obter uma solução consistente, um tipo de aplicação *eHealth* será escolhido em meio aos que foram listados na Subseção 3.1.1. Ao selecionar um tipo, uma aplicação que esteja no contexto do tipo selecionado será escoolhida seguindo o critério de que seja possível ser implementada em tempo hábil. Por fim serão levantados requisitos para serem implementados.

4.2 Construção das arquiteturas

Ao se obter as principais funcionalidades de aplicações *eHealth*, serão elaboradas duas arquiteturas, uma *serverless* e uma convencional, para atender a essas funcionalidades. É importante ressaltar que as arquiteturas conterão exatamente as mesmas funcionalidades. As arquiteturas serão construídas com a ferramenta Draw.io[1] utilizando o pacote de ícones da AWS para ilustrar todos os componentes utilizados. Adicionalmente, será realizada uma busca na literatura para se obter métricas para comparação entre as arquiteturas.

4.3 Desenvolvimento das aplicações

Esse passo tem como objetivo a implementação das aplicações de acordo com o especificado nas arquiteturas. Primeiro será desenvolvida a aplicação utilizando a arquitetura convencional, esta será construída utilizando a plataforma *Node.js* com a linguagem *Javascript* e será executada na nuvem. Após isso, será implementada a aplicação *serverless* replicando as especificações da arquitetura, esta também será desenvolvida na plataforma *Node.js* com a linguagem *Javascript*. Além disso, uma interface gráfica simples será desenvolvida para a utilização das aplicações.

4.4 Realização dos testes

Esse passo visa a elaboração, realização e coleta dos testes de carga de trabalho para a avaliação das aplicações implementadas. A execução e coleta dos dados serão realizados com a ferramenta Artillery[2], uma ferramenta de código aberto, moderna, poderosa e fácil de usar para testes de carga e testes funcionais. Um Teste de carga são realizados para verificar qual o volume de transações, acessos simultâneos ou usuários que um servidor/software/sistema suporta. Também serão coletados os dados de *logs* gerados pelos serviços da AWS.

4.5 Interpretação e Comparação

Finalmente, este passo visa a interpretação dos dados coletados pela ferramenta de teste e pelos *logs* gerados pela AWS. Ao fim, será feita uma comparação entre os resultados das aplicações aplicando as métricas para validação das arquiteturas propostas.

[1] https://app.diagrams.net
[2] https://artillery.io

5 DESENVOLVIMENTO

Neste Capítulo é apresentado o processo de desenvolvimento e execução dos passos 4.1, 4.2 e 4.3 apresentados no Capítulo anterior deste trabalho e traz de forma detalhada os procedimentos para a realização das atividades relacionadas com à elaboração da proposta de solução e com o desenvolvimento das aplicações.

5.1 Ideação da proposta

A etapa inicial para a elaboração da proposta foi selecionar o tipo da aplicação *eHealth* a ser trabalhada em meio aos apresentados na Subseção 3.1.1. Foi selecionado o tipo Registro Eletrônico de Saúde (EHR) levando em consideração os critérios: tempo hábil para o desenvolvimento da proposta, facilidade de encontrar informações e referências que corroborassem com a elaboração da solução e menor complexidade técnica, respectivamente de acordo com a importância adotada pelo autor.

O passo seguinte foi a seleção da aplicação a ser desenvolvida de acordo com o tipo selecionado. Como descrito na Tabela 1 aplicações de EHR abrangem dados dos pacientes em geral, após buscas na literatura e em sites populares que tratam de *eHealth* foi percebido que a aplicação mais comum desse tipo de aplicação são prontuários eletrônicos, portanto, esta foi a solução escolhida.

Possuindo o tipo de aplicação e a aplicação definidos, a etapa seguinte foi elencar as funcionalidades que fariam parte da aplicação desenvolvida. A Subseção seguinte mostra os requisitos da aplicação.

5.1.1 Requisitos

Uma etapa fundamental que antecede a elaboração da arquitetura e escolha da tecnologia é o levantamento das funcionalidades que farão parte da solução, pois com isso poderá ser alaborado uma arquitetura com o intuito de atender as necessidades elencadas. O processo de levantamento dos requisitos funcionais teve como base buscas na literatura por trabalhos que abordassem funcionalidades de prontuários, análise de funcionalidades em aplicações de prontuário disponíveis online e em sites populares que abordavam funcionalidades de prontuários. Já os requisitos não-funcionais desejáveis estão de acordo com o que o presente trabalho se propõe a demonstrar.

Foram utilizados códigos alfanuméricos para identificar os requisitos elencados sendo os requisitos funcionais contendo o prefixo Requisitos Funcional (RF) seguido de números e requisitos não funcionais contendo o prefixo Requisitos Não Funcional (RNF) seguido de números.

5.1.1.1 Requisitos Funcionais

- **RF001 - Incluir Usuário**

  A aplicação deve permitir a inclusão de usuários dos tipos médico e pacientes, ambos informando os dados: tipo (médico ou paciente), usuário, senha, nome, cpf, data de nascimento, endereço e telefone. Médicos devem também fornecer Conselho Regional de Medicina (CRM) e especialidade.

- **RF002 - Visualizar Usuário**

  A aplicação deve permitir que um usuário possa visualizar seus dados incluídos.

- **RF003 - Alterar Usuário**

  A aplicação deve permitir que os usuários cadastrados possam ser alterados, sejam eles médicos ou pacientes.

- **RF004 - Excluir Usuário**

  A aplicação deve permitir que usuários cadastrados possam ser excluídos.

- **RF005 - *Login***

  A aplicação deve permitir que um usuário acesse o sistema através de um usuário e senha.

- **RF006 - Agendar Consulta**

  A aplicação deve permitir que pacientes possam agendar consultas.

- **RF007 - Cancelar Consulta**

  A aplicação deve permitir que pacientes possam cancelar consultas.

- **RF008 - Consultar Agendamentos**

A aplicação deve permitir que médicos possam ver consultas agendadas.

- **RF009 - Realizar Atendimento**
  A aplicação deve permitir que médicos realizem atendimentos.

- **RF010 - Incluir Diagnóstico**
  A aplicação deve permitir que médicos possam incluir um diagnóstico a uma consulta.

- **RF011 - Consultar Histórico do Paciente**
  A aplicação deve permitir que médicos possam visualizar o histórico dos pacientes. Cada paciente podem visualizar apenas seu próprio histórico.

5.1.1.2 Requisitos Não Funcionais

- **RNF001 - Eficiência de Desempenho**
  A aplicação deve responder as requisições em até dois segundos fazendo uso eficiente dos recursos disponibilizados pelo ambiente de execução.

- **RNF002 - Escalabilidade**
  A aplicação deve suportar altas cargas de trabalho, assim aumentando seus recursos quando o volume de trabalho for mais alto que o esperado.

5.2 Definição da tecnologia

Em paralelo com a pesquisa, levantamento e especificações dos requisitos foram definidas as tecnologias que seriam utilizadas na implementação das aplicações. Como serão desenvolvidas duas aplicações, o principal critério de seleção foi: tecnologias que fossem compatíveis e/ou reutilizáveis entre as duas aplicações planejadas, assim minimizando diferenças entre ambas para que testes não fosse enviesados por grandes diferenças de tecnologia ou ambiente.

Com isso, ambas as aplicações serão executadas na nuvem da AWS, que é uma subsidiária da Amazon que fornece plataformas de computação em nuvem sob demanda e APIs

para indivíduos, empresas e governos, com base no pagamento conforme o uso. Através dessa plataforma é possível se ter acesso a inúmeros serviços de computação na nuvem.

Ambas aplicações irão utilizar como armazenamento de dados o serviço *DynamoDB*, que é um banco de dados chave-valor e documentos que oferece desempenho em milissegundos de um dígito em qualquer escala. É um banco de dados totalmente gerenciado, multirregional, multiativo e durável com segurança, backup e restauração integrados e armazenamento em cache na memória para aplicações em escala de internet. Esse serviço foi escolhido pela promessa de acesso a dados com baixa latência em qualquer escala.

Para coletar *logs* e dados de execução das aplicações, bem como falhas e erros, será utilizado o serviço *CloudWatch*. Esse é um serviço de monitoramento e observação que fornece dados e insights práticos para monitorar aplicativos, responder às alterações de performance em todo o sistema, otimizar a utilização de recursos e obter uma visualização unificada da integridade operacional. Ele coleta dados de monitoramento e operações na forma de *logs*, métricas e eventos, oferecendo uma visualização unificada dos recursos, dos aplicativos e dos serviços da AWS.

A aplicação convencional irá executar no serviço *EC2* que é um serviço WEB que disponibiliza capacidade computacional segura e redimensionável na nuvem. Ele foi projetado para facilitar a computação em nuvem na escala da WEB para os desenvolvedores. Já a aplicação *serverless* irá executar no serviço *Lambda*, que é um serviço de computação sem servidor que permite executar código sem provisionar ou gerenciar servidores, criando lógica de dimensionamento de *cluster* com reconhecimento de *workloads*, mantendo integrações de eventos ou gerenciando tempos de execução. Com o Lambda é possível executar o código para praticamente qualquer tipo de aplicação ou serviço de *back-end*, tudo sem precisar de administração.

Instâncias criadas no *EC2* podem por padrão receber um *Internet Protocol* (IP) público para acesso externo, porém funções *Lambda* por serem orientadas a eventos necessitam de um gatilho para serem acionadas, aqui o gatilho utilizado será o serviço *API Gateway*, que segundo a descrição no site da própria AWS age como a “porta de entrada” para aplicativos acessarem dados, lógica de negócios ou funcionalidade de seus serviços de back-end. Usando o *API Gateway*, é possível criar APIs RESTful que habilitam aplicativos de comunicação bidirecionais em tempo real. O *API Gateway* dá suporte a cargas de trabalho conteinerizadas e sem servidor, além de aplicativos da WEB.

Com a intenção de deixar as duas aplicações ainda mais semelhantes, a linguagem *Javascript* será utilizada em ambas aplicações através do *Node.js* que é um ambiente de execução *JavaScript server-side* de código aberto e multiplataforma. Com ele é possível criar aplicações que não dependem de um *browser* para a sua execução.

A aplicação convencional utilizará *Express.js* que é um *framework* para aplicações WEB para *Node.js*, lançado como software livre e de código aberto. É feito para otimizar a construção de aplicações WEB e APIs. Além disso, é um dos mais populares *frameworks* para servidores em *Node.js*. Já para a aplicação *serverless* foi selecionado o *Serverless framework* que é um *framework* da WEB gratuito e de código aberto escrito usando *Node.js*. Além disso, é uma ferramenta de linha de comando que fornece estrutura, automação de fluxo de trabalho e melhores práticas para desenvolver e implantar sua arquitetura sem servidor.

Com o intuito de ter uma aplicação completa e ter um *feedback* visual acerca das funcionalidades implementadas, uma aplicação *front-end* simples será desenvolvida e reutilizada entre as aplicações. Essa aplicação será desenvolvida através do *framework Vue.js* que é um *framework* para construir interfaces de usuário. Ele foi projetado para ser adotado de forma incremental e pode ser facilmente escalonado entre uma biblioteca e uma estrutura, dependendo dos diferentes casos de uso. Além disso, consiste em uma biblioteca central acessível que se concentra apenas na camada de visualização e em um ecossistema de bibliotecas de suporte que ajuda a lidar com a complexidade em *Single-Page Applications*.

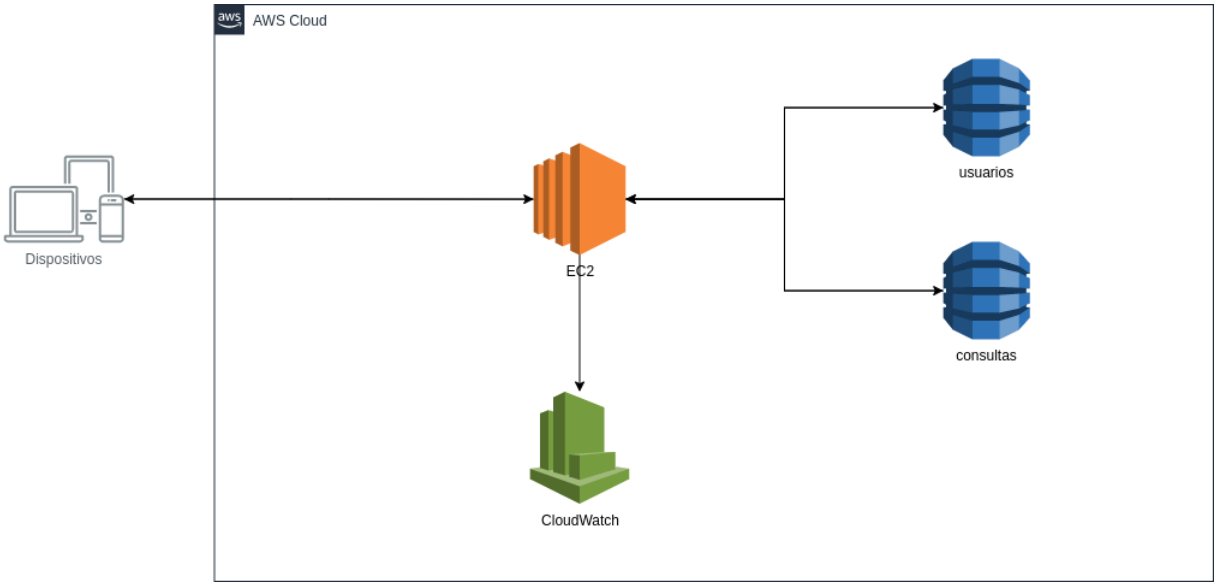
5.3 Arquitetura

Embora as aplicações tenham arquiteturas diferentes elas fornecerão as mesmas funcionalidades. Uma visualização de quais serviços selecionados para cada uma das aplicações e como eles se comunicam é apresentado nas Figuras 7 e 8.

A Figura 7 remete a aplicação convencional, esta que utiliza os serviços da AWS *EC2*, *DynamoDb* e *CloudWatch* para proporcionar as funcionalidades elencadas na Subseção 5.1.1 com a linguagem *Javascript* e com o *framework Express.js*. Como já mencionado, instâncias do *EC2* podem possuir um IP público para acesso externo, este recurso será utilizado para que seja possível se conectar a aplicação. A aplicação foi pensada de forma que qualquer requisição, seja ela para gravar ou recuperar dados, realize alguma operação no *DynamoDB*, para esta solução foram definidas duas tabelas: usuários e consultas. Cada requisição gera um conjunto de *logs* no serviço *CloudWatch* que serão utilizados para as validações apresentadas no Capítulo 6.

Já a aplicação mostrada na Figura 8 utiliza o serviço *Lambda* para cumprir o mesmo papel que o *EC2* na anterior. Entretanto com a premissa de computação sob demanda e com gerenciamento de recursos dinâmicos. Essa aplicação utiliza as mesmas especificações para o *DynamoDB*. Cada evento no *Lambda* também gera conjuntos de *logs* no *CloudWatch*. Um diferencial entre as aplicações é que as funções *Lambdas* são *stateless* e orientadas a eventos. Para acessá-las é necessário que algum evento as invoque, então será utilizado o serviço *API Gateway* para redirecionar e transformar requisições HTTP em eventos de chamada das *Lambdas*.

Figura 7 – Arquitetura da aplicação convencional



Fonte: Elaborado pelo autor.

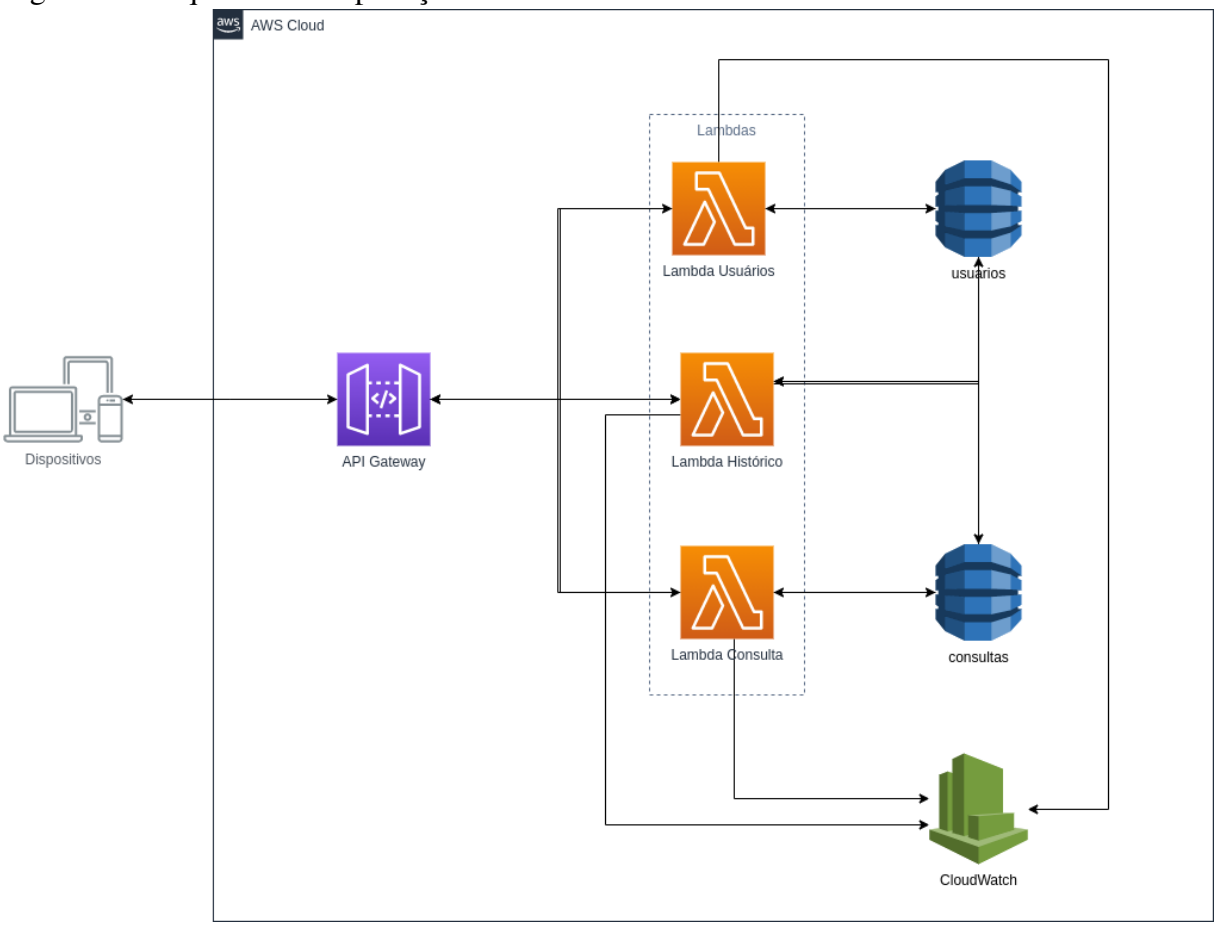
5.4 Implementação

Após as definições de tecnologias e criação das arquiteturas que seriam utilizadas nas aplicações, deu-se início a implementação das aplicações. Nesta Seção serão apresentados os módulos e a comunicação entre eles, os principais fluxos de atividades, o *deploy* das aplicações na nuvem e uma visão geral das funcionalidades. Além disso, decisões, problemas e dificuldades serão discutidas ao decorrer da Seção.

5.4.1 Módulos

Este trabalho utilizará o termo módulo para se referir a uma ou mais partes da aplicação responsável por um assunto bem definido que utilizam tarefas e componentes comuns do sistema. Ambas as aplicações possuirão três principais módulos que são usuários, consultas e

Figura 8 – Arquitetura da aplicação *serverless*



Fonte: Elaborado pelo autor.

histórico.

O módulo usuários é responsável por lidar com as operações de criação, visualização, edição, remoção e autenticação que correspondem aos requisitos RF001, RF002, RF003, RF004 e RF005. O módulo consultas é responsável pelo agendamento, cancelamento, listagem, realização e inclusão de diagnóstico de consultas que correspondem aos requisitos RF006, RF007, RF008, RF009 e RF010. Já o módulo histórico utiliza as informações pessoais de um usuário e suas respectivas consultas para formar um histórico que corresponde ao requisito RF011.

As aplicações foram construídas baseadas no princípio REST (*Representational State Transfer*, Transferencia de Estado Representacional em protuguês) que representa uma série de princípios definidos pela *World Wide Web*, visando a padronização de rotas, requisições e comunicações sem estado. Sendo assim, as aplicações implementam o protocolo HTTP, ou seja, as requisições chegam e saem através dele, os padrões são baseados em sua estrutura e seus códigos de resposta. Para tal, esse protocolo fornece nove verbos que padronizam as rotas, desses nove aqui serão utilizados *GET*, *POST*, *PUT* e *DELETE* que são definidos no Quadro 3.

Quadro 3 – Verbos HTTP e rotas utilizadas

Verbo HTTP	Definição	Rotas Utilizadas
GET	Solicita a representação de um recurso específico. Requisições utilizando o método *GET* devem retornar apenas dados.	/usuarios/:usuario /consultas /historico/:usuario
POST	É utilizado para submeter uma entidade a um recurso específico, frequentemente causando uma mudança no estado do recurso ou efeitos colaterais no servidor.	/usuarios /usuarios/entrar /consultas
PUT	Substitui todas as atuais representações do recurso de destino pela carga de dados da requisição.	/usuarios/:usuario /consultas/:id
DELETE	Remove um recurso específico.	/usuarios/:usuario

Fonte: Adaptado de (MDN, 2020)

5.4.2 Banco de dados

Como mostrado na Seção 5.2, os dados serão armazenados no serviço *DynamoDB* que é um banco de dados *NoSQL*. A Figura 9 mostra a modelagem da tabela de usuários contendo a chave primária e os atributos que comportam o que foi definido nos requisitos mostrados na Subseção 5.1.1, enquanto a Figura 10 mostra também a chave primária e os atributos afim de atender o que foi também definido nos requisitos para as consultas. Vale ressaltar que o serviço de histórico não possui uma tabela, ele utilizará as tabelas de usuários e consultas para compor os históricos dos pacientes.

Figura 9 – Modelagem da tabela usuarios

usuarios									
Chave Primária	Atributos								
usuario	tipo	senha	nome	cpf	dataNascimento	endereco	telefone	crm	especialidade

Fonte: Elaborado pelo autor.

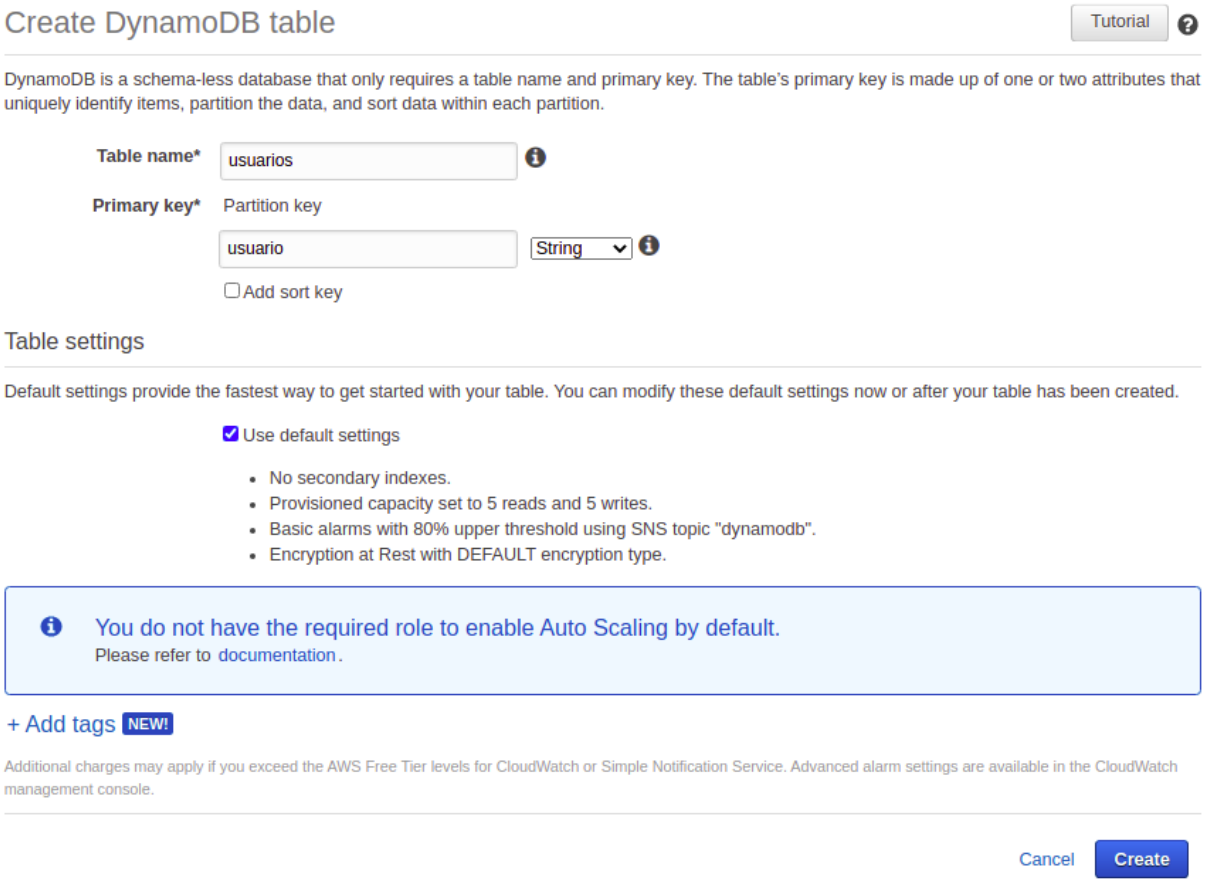
Figura 10 – Modelagem da tabela consultas

consultas							
Chave Primária	Atributos						
id	usuario	endereco	data	especialidade	status	medico	diagnostico

Fonte: Elaborado pelo autor.

As tabelas foram criadas utilizando a configuração padrão fornecida pelo serviço *DynamoDB*, como exemplo a Figura 11 mostra a tela de criação de tabela de usuários apresentando as definições padrões que são: sem índices secundários, capacidade provisionada definida para 5 leituras e 5 gravações, alarmes básicos com limite superior de 80% usando o tópico SNS "dynamodb" e criptografia em REST com tipo de criptografia DEFAULT.

Figura 11 – Criação da tabela de usuários



Fonte: Elaborado pelo autor.

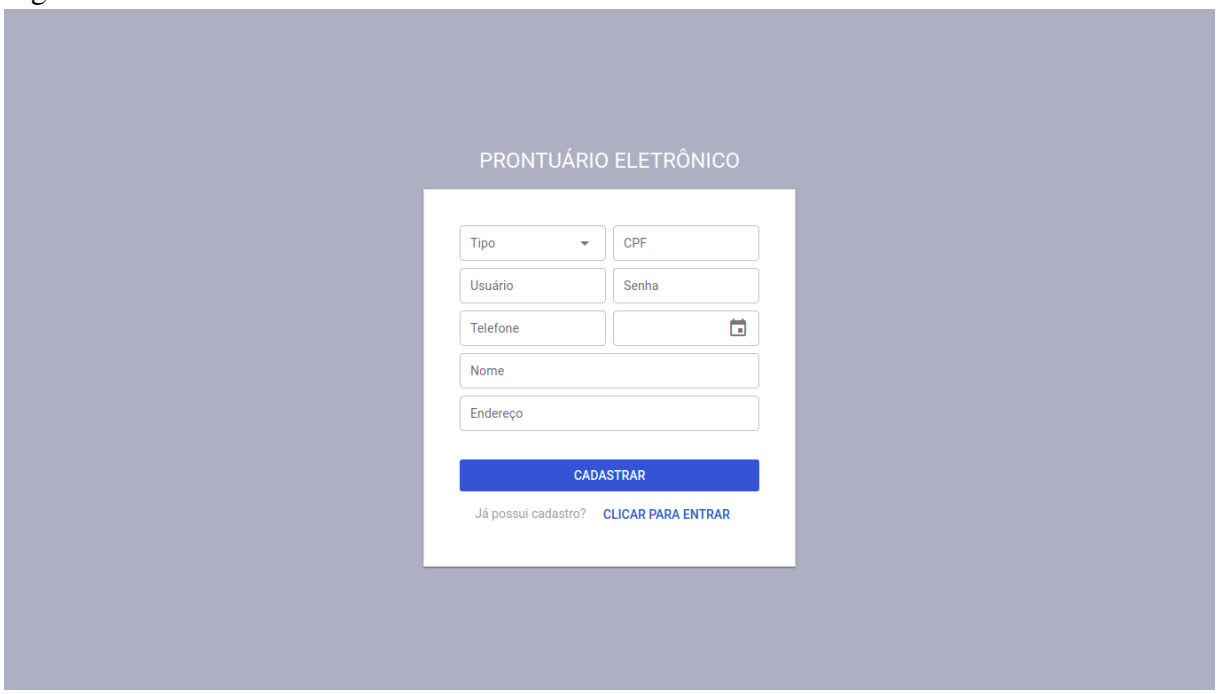
5.4.3 Aplicação front-end

Foi criado uma página WEB no formato *Single Page Application* (SPA) com os *frameworks Vue.js* e *Quasar*. Essa página foi criada apenas para se ter uma visualização da utilização dos serviços criados.

Nela os usuários podem se cadastrar como mostrado na Figura 12, vale ressaltar que usuários podem ser médicos ou pacientes, para médicos é necessário o preenchimento de dois campos adicionais: CRM e especialidade. Ao se cadastrar, usuários podem realizar o *login* no sistema informando usuário e senha como mostra a Figura 13.

Ao acessar a tela inicial mostrada 14, o usuário pode ir ao menu de consultas como mostrado na Figura 15, um paciente pode agendar e ver uma lista contendo todas as suas consultas, ainda nessa lista ele pode optar por cancelar uma consulta. Já para um médico, essa tela apresentará uma lista de consultas semelhante a anterior, porém contendo apenas as que ainda não foram atendidas e de acordo com a sua especialidade, com isso ele poderá atender a consulta e registrar um diagnóstico.

Figura 12 – Formulário de cadastro



Fonte: Elaborado pelo autor.

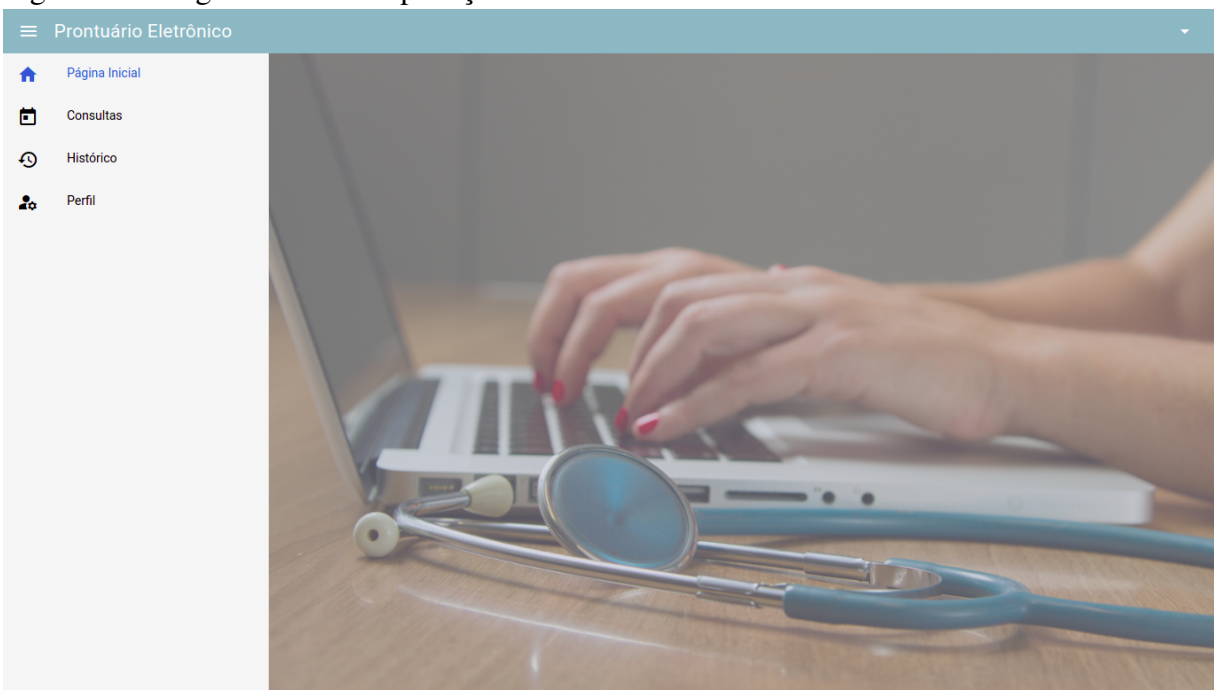
Figura 13 – Formulário de *login*

Fonte: Elaborado pelo autor.

A página WEB também permite que o paciente consulte seu histórico, este histórico possui seus dados pessoais, os dados de todas as suas consultas que já foram atendidas e seus diagnósticos, como mostrado na Figura 16. Somente pacientes possuem histórico para visualizar.

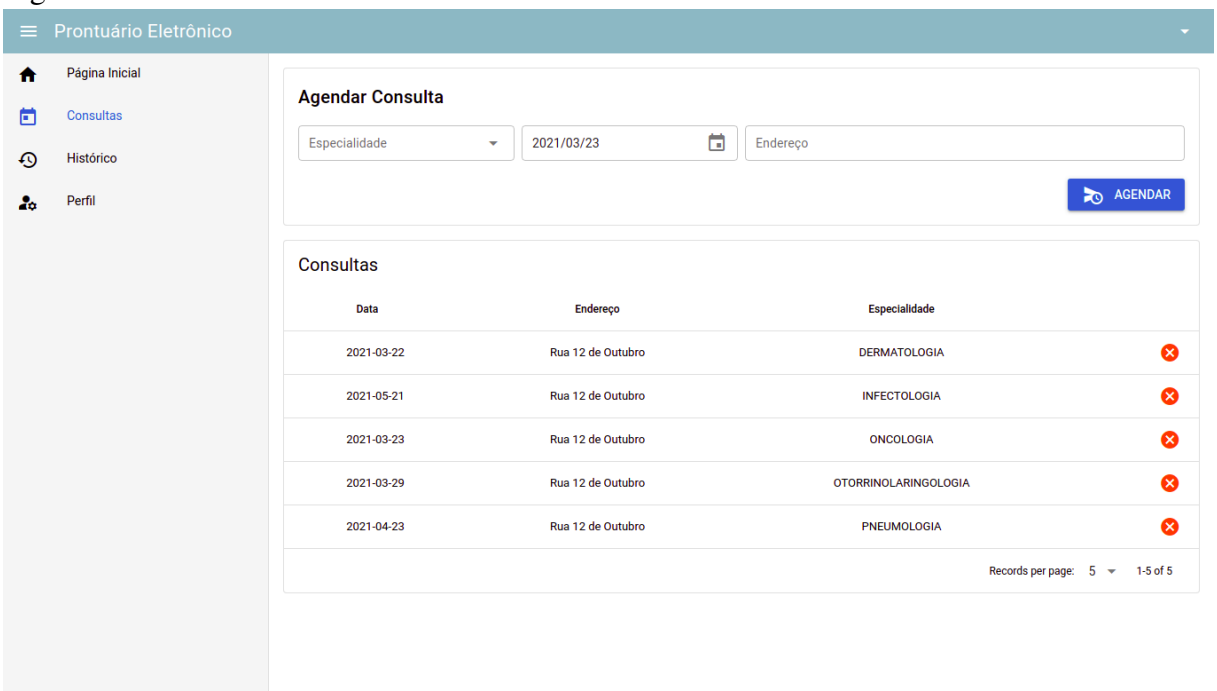
E por fim, os usuários podem alterar seus próprios dados e sair do sistema como mostra a Figura 17.

Figura 14 – Página Inicial da aplicação



Fonte: Elaborado pelo autor.

Figura 15 – Tela de consultas



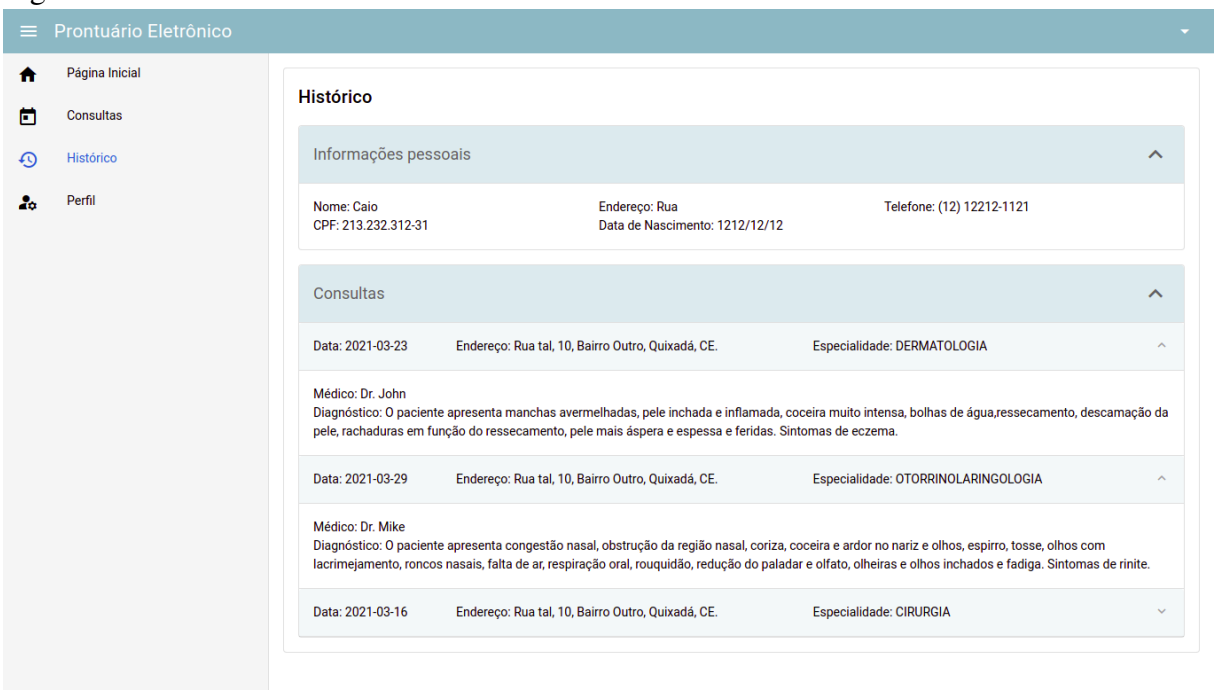
Fonte: Elaborado pelo autor.

5.4.4 Implantação

Ao término da escrita do código as aplicações foram implantadas nos serviços de nuvem mencionados na Seção 5.2.

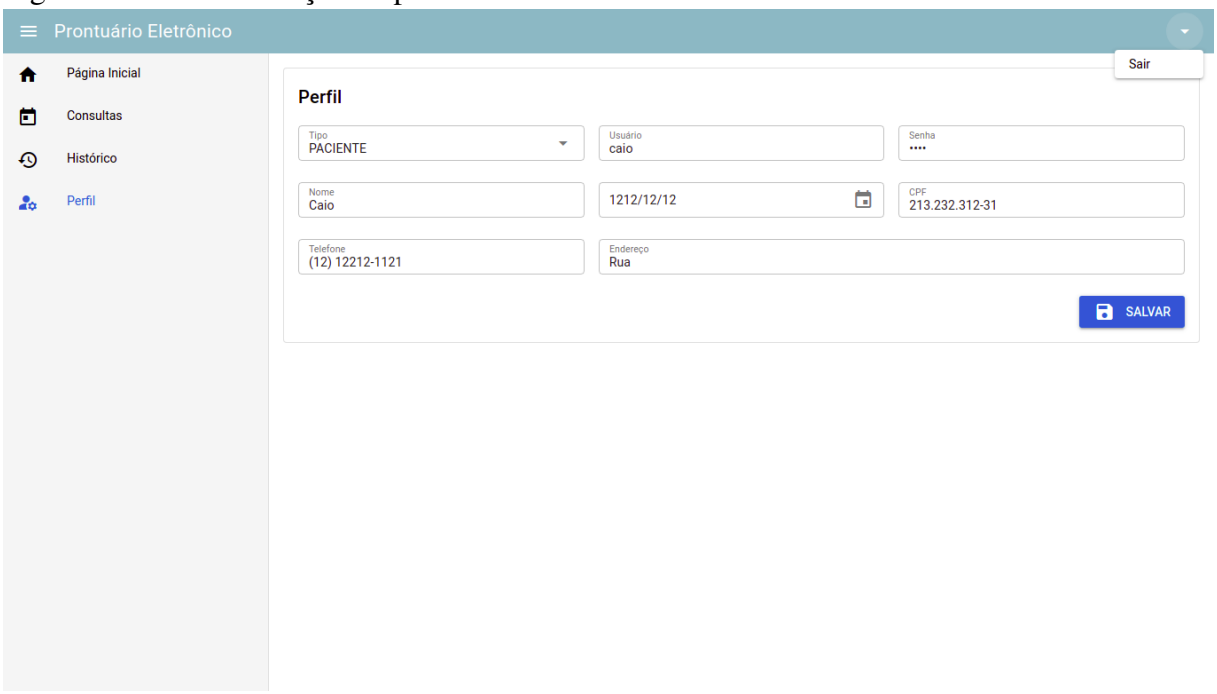
Para a aplicação convencional foi provisionado uma máquina virtual no serviço *EC2*

Figura 16 – Tela de histórico



Fonte: Elaborado pelo autor.

Figura 17 – Tela de edição de perfil



Fonte: Elaborado pelo autor.

com a imagem Ubuntu Server 20.04, do tipo t2.micro, contendo 1 CPU, 1GB de memória, 8GB de armazenamento e com as configurações adicionais de anexar um IP público automaticamente para que seja possível o acesso externo e de permitir acesso ao *DynamoDB* de dentro da instância. Com a instância criada foi instalado o *Node.js*, o *git*, feito o clone do projeto da aplicação convencional que estava em um repositório do *GitHub* e instalado as dependências. Após isso,

foi instalado o pacote pm2, este que é um gerenciador de processos para *Node.js*, com ele o servidor continua executando mesmo depois de responder uma requisição e é responsável por mantê-lo executando quando algum erro ou falha acontecer. Com o pm2 instalado, foi executado o comando para iniciar o servidor.

Para a aplicação *serverless* foi utilizado o *Serverless Framework* como mencionado na Seção 5.2. A base de uma aplicação utilizando este *framework* é o arquivo *serverless.yaml* que pode ser visto no Apêndice A, este arquivo é responsável por configurar as funções *Lambda*, aplicar regras e políticas de acessos a outros serviços, neste caso o *DynamoDB*. Além disso, este arquivo define o ponto de acesso de cada função através do *API Gateway* e os métodos suportados por cada uma delas. Com esse arquivo configurado e a aplicação desenvolvida basta executar o comando de implantação que os serviços definidos e configurados estarão disponíveis em alguns minutos.

6 RESULTADOS

Este Capítulo compreende a execução dos passos 4.4 e 4.5 dos procedimentos metodológicos que correspondem a elaboraboção e execução dos teste, bem como a coleta e interpretação dos resultados. Ressaltando que a aplicação *front-end* não será levada em consideração para o teste.

6.1 Elaboração do teste

Como as aplicações consistem em duas APIs REST, o método de testes utilizado será um teste de carga realizando requisições HTTP para as rotas implementadas através da ferramenta *Artillery*. Com o tipo de teste e a ferramenta definida, foi elaborado um *script* único de teste para ambas aplicações.

Inicialmente foi necessário instalar o pacote dessa ferramenta através do *Node Package Manager* (npm) com o seguinte comando:

```
1  $ npm install -g artillery@1.6
```

Esse comando instala globalmente a versão 1.6 da ferramenta. Após isso foi criado o arquivo com a extensão *yaml* mostrado no Apêndice B. Esse é o arquivo utilizado para configurar e que contém os testes. Na primeira propriedade deste arquivo são definidas as configurações para o teste, neste caso foram criados dois ambientes, um para a aplicação *serverless* e outro para a aplicação convencional. Dentro de cada ambiente são definidos os alvos do teste que neste caso são as *Uniform Resource Locator*s (URLs) de cada aplicação. Ainda para cada ambiente são definidas as fases de teste.

Foram criadas três fases idênticas para os ambientes. Cada fase conta com uma duração em segundos e uma taxa de chegada, esta taxa de chegada representa o número de usuários virtuais criados por segundo que executarão os cenários criados. A primeira fase foi chamada de aquecimento, esta tem duração de 10 segundos e taxa de chegada de 5 (que são os menores entre as fases), isso significa que durante 10 segundos serão criados 5 usuários por segundo, totalizando 50 usuários virtuais. A segunda fase tem como nome e objetivo aumentar a carga, isso quer dizer que a taxa de chegada aumentará gradativamente de 5 para 10 durante toda a duração, que nesta fase é 20 segundos, o total de usuários virtuais dessa fase varia a cada execução. A última fase tem como nome e objetivo manter a carga máxima, isso significa que

durante os 30 segundos definidos de duração serão criados 10 usuários por segundo, totalizando 300 usuários virtuais. O total de usuários virtuais criados poderá variar de um teste para outro.

A segunda propriedade mais externa do arquivo diz respeito aos cenários de testes, estes cenários são casos de uso das aplicações que possuem fluxos de ações que um usuário comum poderia executar através das rotas disponibilizadas. Neste trabalho foi definido apenas um cenário, este que foi criado com o intuito de simular um uso básico da aplicação e que usaria o máximo de rotas possíveis, totalizando 5 rotas. Neste cenário o usuário faz registro na aplicação, realiza o *login*, agenda uma consulta, essa consulta é atendida, finalizada e por fim o usuário visualiza seu histórico.

Sendo assim, cada usuário virtual criado irá executar o cenário de teste definido. Cada teste terá a duração de 60 segundos, salientando que essa é a duração de criação de usuários virtuais. Outros fatores como tempo de execução de cada requisição poderá implicar no tempo total do teste. Esses valores são suficientes para avaliar os atributos de qualidade definidos para este trabalho.

Cada requisição pertencente ao fluxo possui uma URL, esta é concatenada com a URL definida na configuração do ambiente para formar a rota completa que levará ao serviço requisitado. As rotas do tipo *POST* e *PUT* utilizadas possuem a propriedade *json* contendo as informações a serem persistidas, *JavaScript Object Notation* (JSON) é o formato que foi adotado neste trabalho como a forma padrão de troca de dados nas aplicações.

A ferramenta *Artillery* possibilita que o retorno de uma requisição sirva de insumo para as outras. Esse recurso foi utilizado para capturar o usuário que é retornado ao fazer o cadastro para ser utilizado no *login*. Foi capturado também o *token* de autenticação que é retornado ao realizar o *login*, este que será inserido no cabeçalho das requisições subsequentes para autorizar o acesso.

A próxima Seção apresentará os detalhes da execução do teste definido e os dados obtidos.

6.2 Execução e coleta

Com as aplicações implantadas e possuindo o arquivo configurado e com o cenário definido foram então executados os testes. Apesar de terem sido definidos dois ambientes no mesmo arquivo de teste, eles foram executados separadamente como mostrado nos comandos a seguir:

```
$ artillery run -e monolith my-test.yaml
```

```
$ artillery run -e serverless my-test.yaml
```

Para a aplicação convencional foram obtidos da ferramenta *Artillery* os dados mostrados na Figura 18. Esta Figura nos mostra que todos os usuários virtuais finalizaram os cenários de teste, o total de cenários criados e executados, o total de cenários que foram completados, a média de respostas por segundo, os tempos de respostas mínimo, máximo, média, percentis 95 e 99, além disso conta com os códigos HTTP retornados pelas requisições. A Figura 19 mostra as mesmas informações que a anterior, mas para a aplicação *serverless*, com isso será possível fazer comparações entre as duas aplicações.

Figura 18 – Resultado do teste da aplicação convencional

```
All virtual users finished
Summary report @ 11:23:07(-0300) 2021-03-28
  Scenarios launched:  498
  Scenarios completed: 498
  Requests completed:  2490
  Mean response/sec: 38.7
  Response time (msec):
    min: 106.6
    max: 10718.9
    median: 388.1
    p95: 1494.9
    p99: 2046.5
  Scenario counts:
    Usuario se registra e acessa o sistema para criar uma consulta e visualiza-la no seu historico apos ser atendido: 498 (100%)
  Codes:
    200: 996
    201: 1494
```

Fonte: Elaborado pelo autor.

Figura 19 – Resultado do teste da aplicação serverless

```
All virtual users finished
Summary report @ 12:43:04(-0300) 2021-03-28
  Scenarios launched:  510
  Scenarios completed: 510
  Requests completed:  2550
  Mean response/sec: 41.32
  Response time (msec):
    min: 195.2
    max: 6624.9
    median: 310.6
    p95: 3417.6
    p99: 5268.6
  Scenario counts:
    Usuario se registra e acessa o sistema para criar uma consulta e visualiza-la no seu historico apos ser atendido: 510 (100%)
  Codes:
    200: 1020
    201: 1530
```

Fonte: Elaborado pelo autor.

Além dos dados obtidos como resultado da execução da ferramenta *Artillery*, também foram coletados métricas fornecidas pelos próprios serviços utilizados na AWS e também de *logs* no serviço *Cloudwatch*. Os dados fornecidos pelos serviços diferentes entre as aplicações (*EC2* e *Lambda*) fornecem métricas diferentes, portanto serão visualizados e discutidos porém não comparadas.

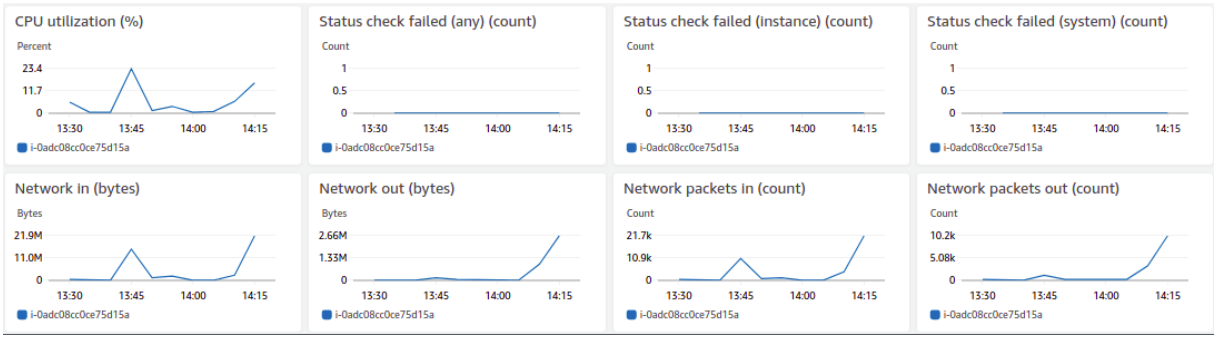
O serviço *EC2* disponibiliza um gráfico de portentagem de utilização de *Central Process Unit* (CPU), com ele será possível acompanhar o quanto os testes irão consumir. Além dessa, o serviço também apresenta gráficos de tráfego de rede, tanto de entrada e saída em *bytes* quanto de entrada e saída de pacotes, possibilitando o monitoramento do volume de dados que entra e sai do servidor.

O serviço *Lambda* fornece um gráfico de quantidade de invocações, assim é possível acompanhar o quanto cada função está sendo utilizada. Fornece também um gráfico de duração, este mostra as durações mínima, máxima e a média das invocações. Com este será feito uma pequena análise entre as próprias *Lambdas*, pois a ferramenta de teste informará os tempos de resposta de todas as requisições. Outro gráfico cedido por este serviço é o de execuções concorrentes, com ele será possível observar a quantidade de execuções de cada *Lambda* ao mesmo tempo.

Já o serviço *DynamoDB* dispõe de gráficos de capacidade de leitura e de escrita que nos permitirá observar a capacidade provisionada e o que foi realmente consumido. Outra métrica obtida para este serviço porém através do serviço *CloudWatch* foi a quantidade total de unidades de capacidade de leitura dentro de 5 minutos.

A Figura 20 mostra as métricas obtidas do serviço *EC2* e dela é possível observar que após o pico inicial, quando a instância foi criada, houve uma crescente na utilização de CPU e no tráfego de rede em decorrência da execução do teste. Um ponto interessante é que mesmo após o grande número de requisições o uso de CPU não chegou a 25%. Em parte isso se deve ao fato de que os serviços fornecidos pela aplicação terem um processamento simples, já que estes eram mais voltados a inserção e recuperação de dados do banco de dados.

Figura 20 – Métricas fornecidas pelo *EC2*

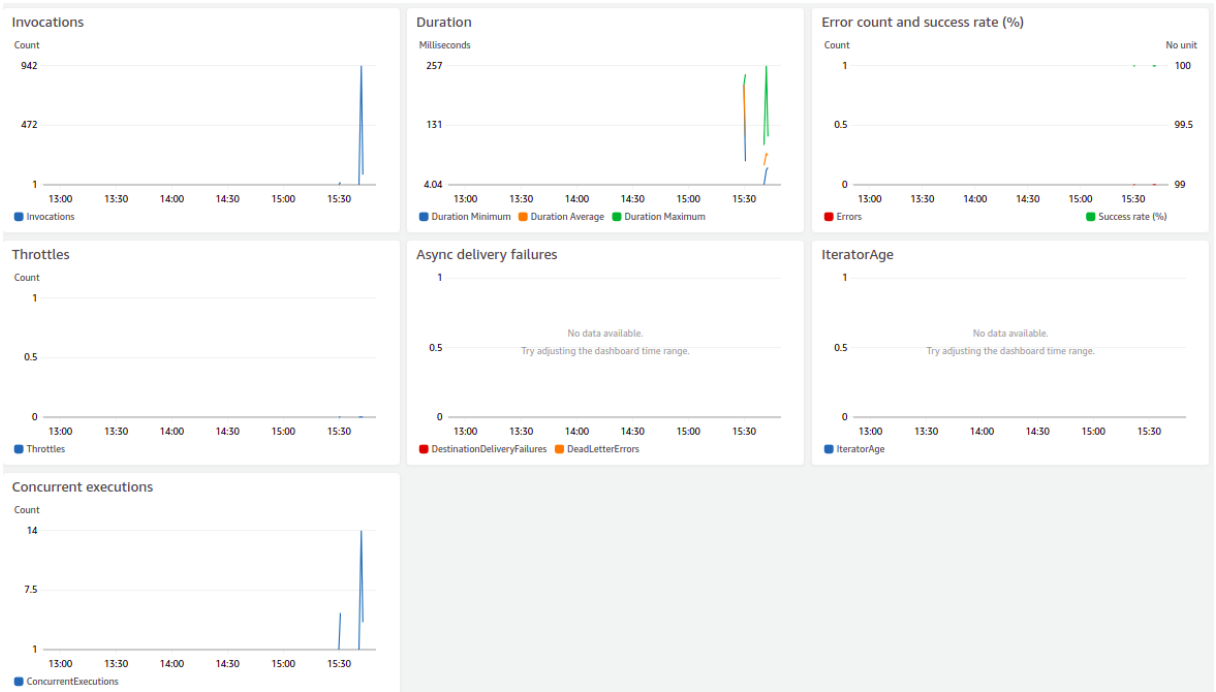


Fonte: Elaborado pelo autor.

Com é possível observar nas Figuras 21, 22 e 23, foi constatado que a *Lambda* de usuários possuiu o maior pico de invocações, enquanto a menor foi a de histórico, o que era

de se esperar já que cada usuário virtual fazia apenas uma chamada pra esse serviço. Todas as invocações retornaram com sucesso. A *Lambda* com o maior número de execuções concorrentes foi também a de usuários, chegando a atingir 14 execuções concorrentes. Entretanto a *Lambda* com o maior valor máximo de tempo de execução foi a de consultas, em parte isso se deve a alta taxa de utilização da tabela de consultas do *DynamoDB* que será mostrado a seguir.

Figura 21 – Métricas fornecidas pela *Lambda* usuários

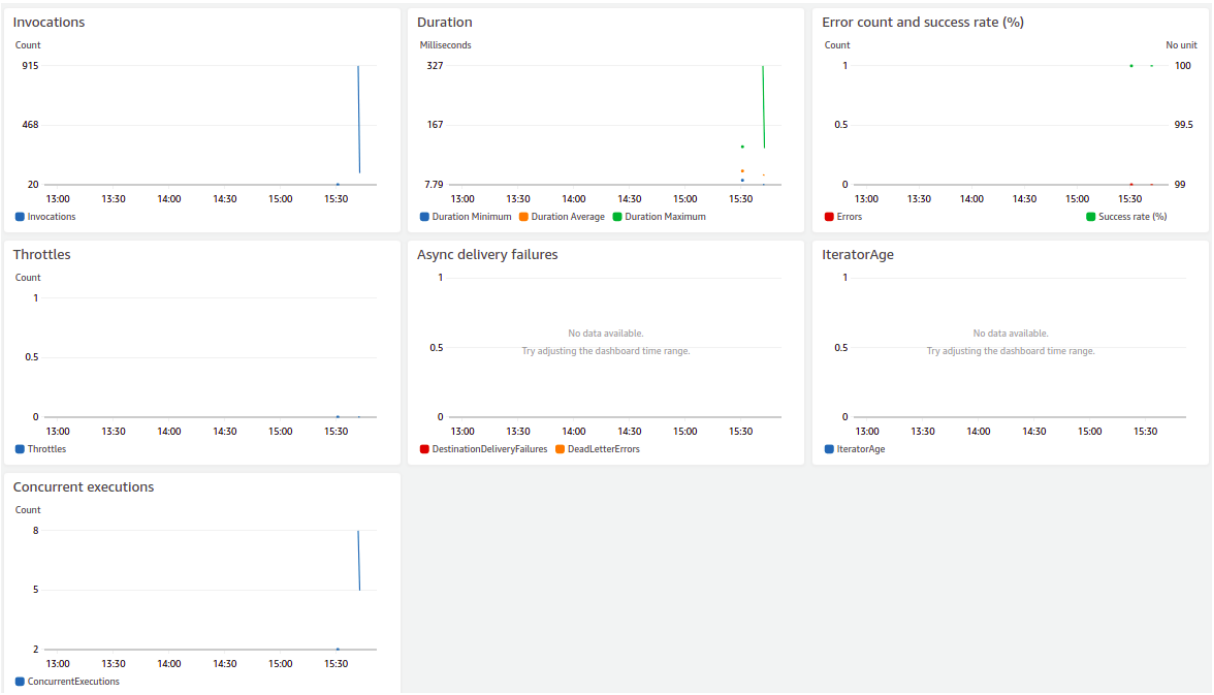


Fonte: Elaborado pelo autor.

Como já citado ambas as aplicações fizeram uso do serviço de banco de dados *DynamoDB* com o propósito de minimizar as diferenças entre os ambientes. Esse serviço fornece suas próprias métricas para o acompanhamento e *insights* como mostrados nas Figuras 24 e 25. Essas Figuras retratam a capacidade de leitura e escrita na tabela consultas após a execução dos testes nas duas aplicações, esta que foi a tabela com a maior taxa de utilização. Como é possível visualizar, a capacidade de leitura na tabela consultas na aplicação convencional foi bem maior que na aplicação *serverless*, já a capacidade de escrita foi um pouco maior na *serverless* comparada a convencional.

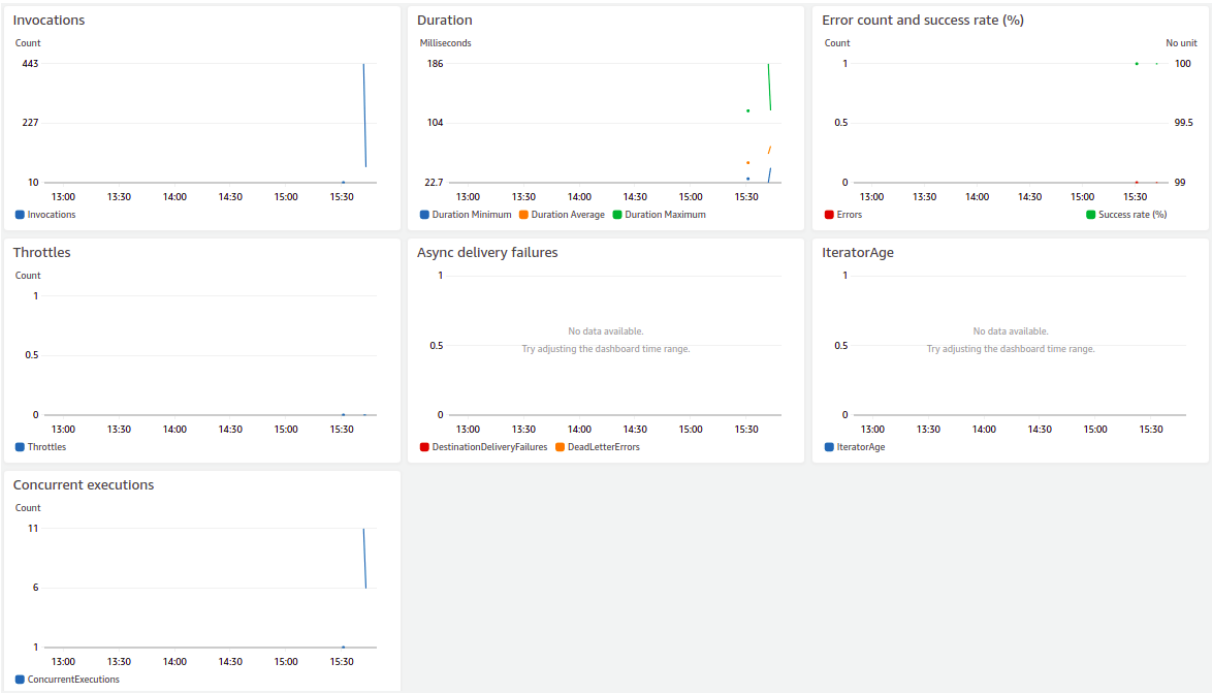
Um ponto importante a citar é que ambas as aplicações dispararam alarmes no *Cloudwatch* pois as unidades de capacidade de leitura e escrita ultrapassaram a que foi definida por padrão. Uma unidade de capacidade de leitura representa uma leitura fortemente consistente por segundo, ou duas leituras eventualmente consistentes por segundo, para um item de até 4 KB. Os pedidos de leitura transacional requerem duas unidades de capacidade de leitura para realizar

Figura 22 – Métricas fornecidas pela *Lambda* consultas



Fonte: Elaborado pelo autor.

Figura 23 – Métricas fornecidas pela *Lambda* histórico

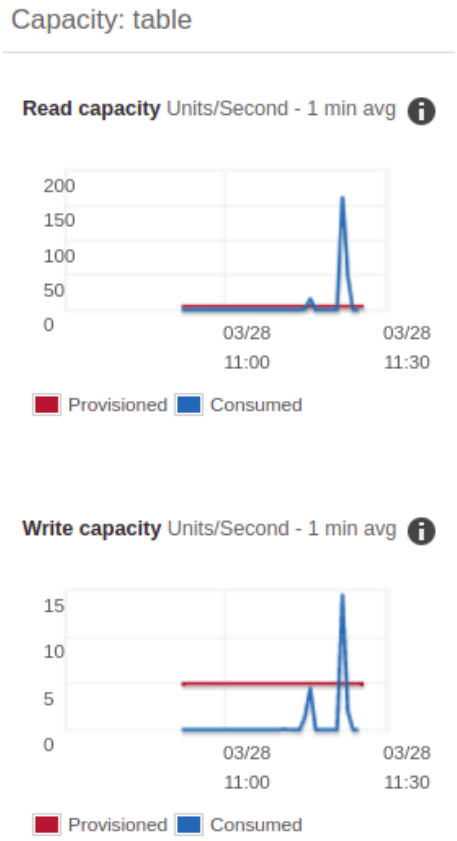


Fonte: Elaborado pelo autor.

uma leitura por segundo para itens de até 4 KB. Se for necessário ler um item maior do que 4 KB, o *DynamoDB* deve consumir unidades de capacidade de leitura adicionais. O número total de unidades de capacidade de leitura necessárias depende do tamanho do item e se é necessário uma leitura eventualmente consistente ou fortemente consistente (AWS, 2021).

A Figura 26 mostra um gráfico gerado através do alarme disparado no *Cloudwatch*,

Figura 24 – Métricas fornecidas pelo *DynamoDB* da tabela consultas na aplicação convencional



Fonte: Elaborado pelo autor.

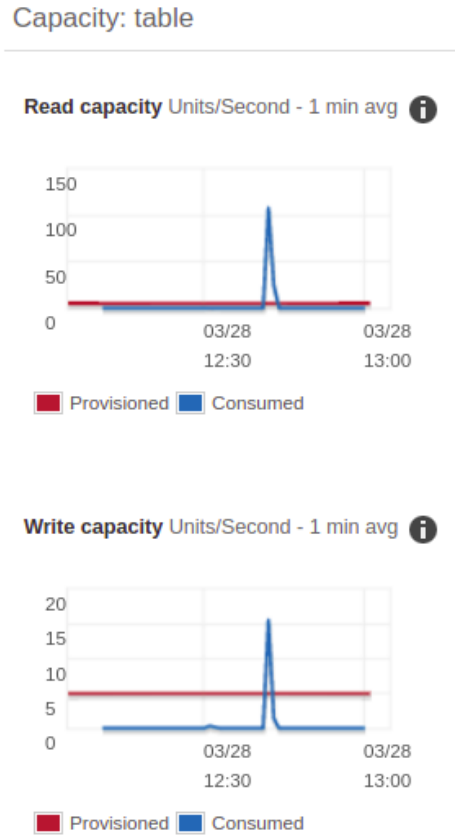
nesse gráfico é possível observar dois picos de consumo de unidades de capacidade de leitura, sendo o primeiro atingido devido a execução dos testes na aplicação convencional e o segundo atingido na execução dos testes na aplicação *serverless*. É importante observar que o primeiro pico foi bem maior que o segundo apesar de usarem os mesmos dados e por todos os registros possuírem o mesmo tamanho em *Kilobyte* (KB) em ambas aplicações, isso sugere que a aplicação convencional demandou mais unidades de capacidade de leitura devido ao tempo de resposta maior mostrado nas Figuras 24 e 25 que pode ter gerado um acúmulo de requisições ao banco de dados.

6.3 Análise

De posse das informações coletadas ao utilizar a ferramenta *Artillery* apresentados nas Figuras 18 e 19 foi possível fazer algumas considerações e análises.

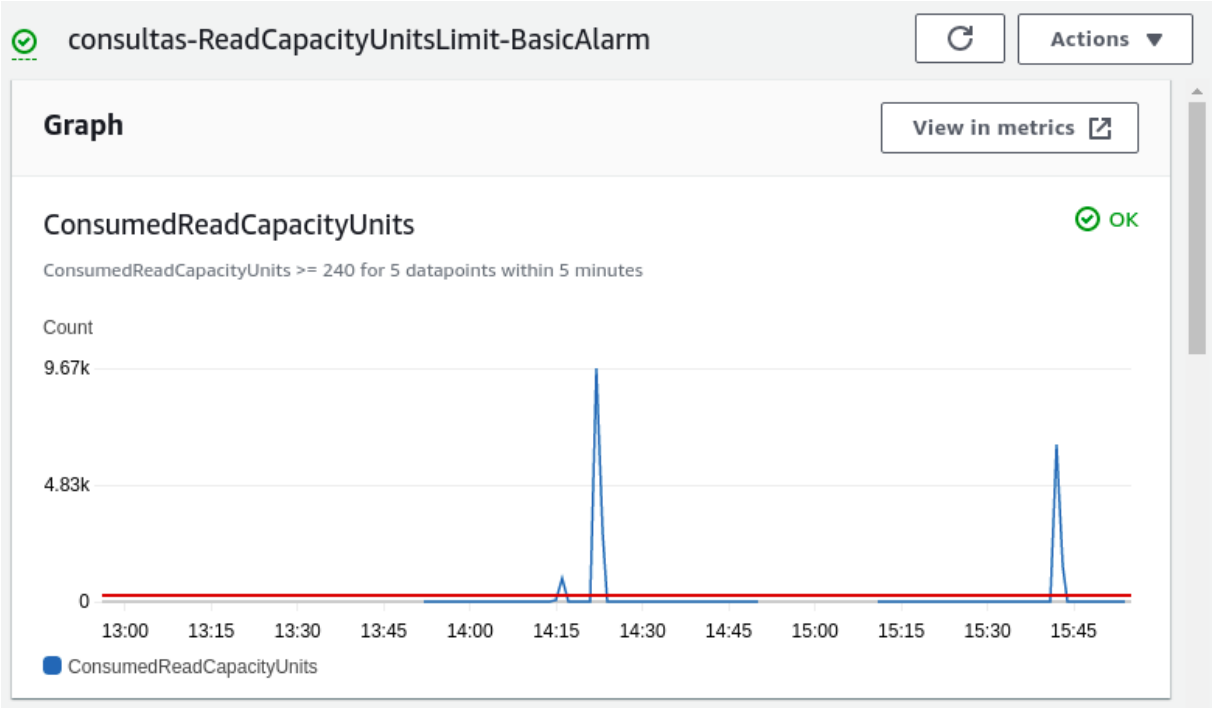
A primeira informação a ser analisada é o número de cenários criados que foram diferentes entre as duas execuções apesar de valores idênticos, sendo o teste da aplicação convencional executando 498 cenários e o teste da aplicação *serverless* executando 510 cenários.

Figura 25 – Métricas fornecidas pelo *DynamoDB* da tabela consultas na aplicação *serverless*



Fonte: Elaborado pelo autor.

Figura 26 – Unidades de capacidade de leitura consumidas na tabela consultas



Fonte: Elaborado pelo autor.

É provável que isso tenha acontecido por conta da configuração de aumentar de 5 para 10 a taxa de chegada de usuários virtuais durante 20 segundos pode ter gerado uma quantidade diferente entre os testes.

Tanto os testes da aplicação convencional quanto os da aplicação *serverless* completaram todos os cenários. Ambos os testes completaram todas as requisições obtendo retorno de sucesso que podem ser identificados pelos códigos HTTP, atentando-se ao fato da diferença da quantidade de cenários que gerou 2490 requisições no teste da aplicação convencional e 2550 no teste da aplicação *serverless*. O resultado do teste da aplicação convencional mostrou uma média de 38.7 respostas por segundo, ou seja, a cada segundo as aproximadamente 39 requisições disparadas nos cenários foram respondidas, enquanto o resultado do teste da aplicação *serverless* obteve uma média de 41.32 respostas por segundo. Essa média pode ter sido impactada tanto pelo tempo de resposta de cada requisição individualmente quanto pela quantidade de cenários maior no teste da aplicação *serverless*.

Como citado anteriormente ambos os resultados mostraram informações sobre o tempo de resposta das requisições que pode ser melhor observado no Quadro 4. É possível constatar que o tempo de resposta mínimo da aplicação convencional foi menor que o da *serverless*, entretanto o tempo de resposta máximo da aplicação convencional foi maior, o que fez com que a média também fosse maior. Como era de se esperar, apesar da linguagem utilizada ser altamente performática e o banco de dados também, o processador da aplicação convencional é um só, o que pode ter gerado concorrência entre processos e uma demora maior para responder as requisições diferentemente das funções *Lambdas* que são isoladas e independentes, o processamento de uma não interfere na outra e assim não gera concorrência entre processos de requisições diferentes. Em contrapartida, os percentis 95 e 99 da aplicação *serverless* foram maiores que o dobro dos percentis da aplicação convencional.

Quadro 4 – Tempo de resposta de requisições obtidos nos testes em milisegundos

	Convencional	***Serverless***
Tempo de resposta mínimo	106.6	195.2
Tempo de resposta máximo	10718.9	6624.9
Tempo de resposta médio	388.1	310.6
Percentil 95	1494.9	3417.6
Percentil 99	2046.5	5268.6

Fonte: Elaborado pelo autor.

7 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho tem como objetivo realizar uma comparação entre arquiteturas, uma *serveless* e uma convencional, aplicadas no contexto de *eHealth*, analisadas mediante os atributos de qualidade escalabilidade e desempenho.

Para tal foi necessário identificar tipos de aplicações *eHealth* que seriam beneficiados por arquiteturas *serveless* através de buscas por trabalhos na literatura. Também foi necessário identificar requisitos dos tipos de aplicações que seriam desenvolvidas consultando aplicações disponíveis online e em trabalhos na literatura. Outra etapa a ser cumprida foi desenhar arquiteturas que seriam utilizadas para a implementação de aplicações *eHealth*. Por fim, foram realizados testes de carga nas aplicações para que uma análise dos dados obtidos dos testes fosse feita.

7.1 Considerações finais

Com base nos resultados obtidos e análises realizadas no Capítulo 6 é possível afirmar que, em relação ao tempo médio de resposta, a arquitetura *serverless* é mais vantajosa. Parte dessa vantagem se dá pela alta escalabilidade proporcionada pelas funções *Lambda*. O serviço *EC2*, que foi utilizado para a aplicação convencional, fornece uma maneira de proporcionar escalabilidade através do método *Auto Scaling* que consiste em ajustar dinamicamente a quantidade de recursos computacionais em um conjunto de máquinas virtuais, entretanto seria necessário sempre ter ao menos um servidor executando, sem contar que isso requer um pouco mais de configuração na implantação. Este último é outro ponto onde a arquitetura *serverless* se sobressai pois sua utilização é totalmente sob demanda.

Um ponto observado é que foi utilizado assertivamente em ambas aplicações o serviço de banco de dados *DynamoDB* que proporciona alta escalabilidade e alta performance de leitura e escrita, com isso obtivemos menos riscos de falhas em requisições, pois poderia ocorrer gargalos de conexões com o serviço se fossem utilizados bancos de dados comuns que não fornecem meios para suportar altos volumes de conexões sem a necessidade de configurações mais avançadas e custosas.

Claramente a utilização desses serviços em geral demandam custos financeiros e que em aplicações que demandam muitas requisições ou que necessitem de um funcionamento disponível a todo instante esse custo é maior. O provedor de nuvem utilizado fornece ótimos serviços a preços justos e que se usados com inteligência e segurança agregarão positivamente

em qualquer aplicação que opte por utilizá-lo. Todos os serviços e mecanismos utilizados nesse trabalho ficaram dentro do limite gratuito fornecido pelo provedor, porntanto não houve comparações a cerca do quanto foi gasto por cada aplicação.

Durante a maturação da ideia do trabalho e escolha do tipo de aplicações *eHealth* a serem trabalhadas ocorreu uma indecisão a cerca de qual seria o melhor tipo de aplicação para que fossem realizadas comparações entre as arquiteturas, chegando a decisão de que aplicações que demandassem muitas operações no banco de dados traria uma visão mais assertiva sobre o que se prentendia alcaçar no decorrer do trabalho.

Outra dificuldade foi definir uma carga aceitável para que fossem obtidos resultados conclusivos, pois não foram encontrados trabalhos que justificassem que uma certa quantidade seria suficiente, ficando a cargo do autor definir uma carga que explorasse ao menos um pouco o poder da ferramenta de teste de carga utilizada e o poder computacional dos serviços de nuvem em que estavam implantadas as aplicações.

As análises e conclusões obtidas podem ser utilizadas por arquitetos e desenvolvedores de aplicações *eHealth* como uma fonte de dados comparativa entre arquiteturas convencionais e *serverless*, auxiliando assim na solução de dúvidas e na tomada de decisão sobre qual arquitetura utilizar.

7.2 Trabalhos futuros

Ao realizar os testes foi observado que o grande potencial das funções *Lambda* não foi amplamente explorado sendo necessário uma carga maior para se obter ainda mais resultados para se ter uma maior autoridade conclusiva em aspectos que favoreceriam seu potencial. Como trabalho futuro pretende-se realizar testes com cargas ainda maiores para que sejam obtidos resultados ainda mais poderosos.

Tendo em vista que é possível disponiblizar meios que permitam com qua as aplicações fiquem ainda mais semelhantes, outro trabalho futuro seria habilitar e configurar a habilidade de *Auto Scaling* do serviço *EC2* para que ambas aplicações fossem capazes de proporcionar escalabilidade e assim possuírem meios de lidar com cargas maiores.

Este trabalho se limitou a um tipo simples de aplicação *eHealth*, que foi o tipo EHR, para que fosse possível ser desenhadas arquiteturas e implementadas aplicações em tempo hábil. Seria interessante que trabalhos futuros tivessem uma abrangência ainda maior, explorando outros tipos de aplicações *eHealth*.

REFERÊNCIAS

ABOLADE, T. O. In: **The benefits and challenges of e-health applications in developing nations**: a review. [*S. l.*: *s. n.*], 2018.

ACETO, G.; PERSICO, V.; PESCAPÉ, A. The role of information and communication technologies in healthcare: taxonomies, perspectives, and challenges. **Journal of Network and Computer Applications**, Elsevier, v. 107, p. 125–154, 2018.

AWS. **AWS Documentation**. 2021. Disponível em: https://docs.aws.amazon.com/index.html. Acesso em: 2 abr. 2021.

AZIMI, I.; ANZANPOUR, A.; RAHMANI, A. M.; PAHIKKALA, T.; LEVORATO, M.; LILJEBERG, P.; DUTT, N. Hich: Hierarchical fog-assisted computing architecture for healthcare iot. **ACM Transactions on Embedded Computing Systems (TECS)**, ACM New York, NY, USA, v. 16, n. 5s, p. 1–20, 2017.

BALDINI, I.; CASTRO, P.; CHANG, K.; CHENG, P.; FINK, S.; ISHAKIAN, V.; MITCHELL, N.; MUTHUSAMY, V.; RABBAH, R.; SLOMINSKI, A.; SUTER, P. Serverless computing: Current trends and open problems. In: ____. **Research Advances in Cloud Computing**. Singapore: Springer Singapore, 2017. p. 1–20. ISBN 978-981-10-5026-8. Disponível em: https://doi.org/10.1007/978-981-10-5026-8_1. Acesso em: 02 abr. 2020.

CONTRACTOR, D.; PATEL, D. R. Accountability in cloud computing by means of chain of trust. **IJ Network Security**, v. 19, n. 2, p. 251–259, 2017.

DUTTA, P.; DUTTA, P. In: **Comparative study of cloud services offered by amazon, microsoft & google**. [*S. l.*: *s. n.*], 2019.

EYSENBACH, G. What is e-health? **J Med Internet Res**, v. 3, n. 2, p. e20, Jun 2001. ISSN 1438-8871. Disponível em: http://www.jmir.org/2001/2/e20/.

FARAHANI, B.; FIROUZI, F.; CHANG, V.; BADAROGLU, M.; CONSTANT, N.; MANKODIYA, K. Towards fog-driven iot ehealth: Promises and challenges of iot in medicine and healthcare. **Future Generation Computer Systems**, Elsevier, v. 78, p. 659–676, 2018.

GENG, X.; MA, O.; PEI, Y.; XU, Z.; ZENG, W.; ZOU, J. Research on early warning system of power network overloading under serverless architecture. In: IEEE. **2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2)**. [*S. l.*], 2018. p. 1–6.

GHANEM, A. S.; ALKHAL, H. A. A mobile cloud-based system for alzheimer's disease. In: IEEE. **2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)**. [*S. l.*], 2018. p. 1–5.

HARKUT, D. G.; KASAT, K.; SHAH, S. **Cloud Computing**: Technology and practices. [*S. l.*]: BoD–Books on Demand, 2019.

ISO/IEC 25010. ISO/IEC 25010:2011, systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models. **International Organization for Standardization**, v. 34, p. 2910, 2011.

LEE, H.; SATYAM, K.; FOX, G. Evaluation of production serverless computing environments. In: IEEE. **2018 IEEE 11th International Conference on Cloud Computing (CLOUD)**. [*S. l.*], 2018. p. 442–450.

LLOYD, W.; RAMESH, S.; CHINTHALAPATI, S.; LY, L.; PALLICKARA, S. Serverless computing: An investigation of factors influencing microservice performance. In: IEEE. **2018 IEEE International Conference on Cloud Engineering (IC2E)**. [*S. l.*], 2018. p. 159–169.

MAKSIMOVIĆ, M.; VUJOVIĆ, V. Internet of things based e-health systems: ideas, expectations and concerns. In: **Handbook of Large-Scale Distributed Computing in Smart Healthcare**. [*S. l.*]: Springer, 2017. p. 241–280.

MANOGARAN, G.; VARATHARAJAN, R.; LOPEZ, D.; KUMAR, P. M.; SUNDARASEKAR, R.; THOTA, C. A new architecture of internet of things and big data ecosystem for secured smart healthcare monitoring and alerting system. **Future Generation Computer Systems**, Elsevier, v. 82, p. 375–387, 2018.

MDN. **Web technology for developers**. 2020. Disponível em: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods. Acesso em: 2 abr. 2021.

MELL, P. M.; GRANCE, T. **SP 800-145. The NIST Definition of Cloud Computing**. Gaithersburg, MD, USA, 2011.

MISHRA, S. S.; RASOOL, A. Iot health care monitoring and tracking: A survey. In: IEEE. **2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)**. [*S. l.*], 2019. p. 1052–1057.

NASR, A.; ELBOOZ, S. A. Scheduling strategies in cloud computing: Methods and implementations. **American Journal of Engineering and Applied Sciences**, v. 11, n. 2, p. 426–432, 2018.

NASTIC, S.; RAUSCH, T.; SCEKIC, O.; DUSTDAR, S.; GUSEV, M.; KOTESKA, B.; KOSTOSKA, M.; JAKIMOVSKI, B.; RISTOV, S.; PRODAN, R. A serverless real-time data analytics platform for edge computing. **IEEE Internet Computing**, IEEE, v. 21, n. 4, p. 64–71, 2017.

OQUENDO, F.; LEITE, J.; BATISTA, T. Designing scalability in software architectures. In: **Software Architecture in Action**. [*S. l.*]: Springer, 2016. p. 143–154.

OSSEBAARD, H. C.; GEMERT-PIJNEN, L. V. ehealth and quality in health care: implementation time. **International journal for quality in health care**, Oxford University Press, v. 28, n. 3, p. 415–419, 2016.

RAHMAN, M. M.; HASAN, M. H. Serverless architecture for big data analytics. In: IEEE. **2019 Global Conference for Advancement in Technology (GCAT)**. [*S. l.*], 2019. p. 1–5.

RAHMANI, A. M.; GIA, T. N.; NEGASH, B.; ANZANPOUR, A.; AZIMI, I.; JIANG, M.; LILJEBERG, P. Exploiting smart e-health gateways at the edge of healthcare internet-of-things: A fog computing approach. **Future Generation Computer Systems**, Elsevier, v. 78, p. 641–658, 2018.

RAJAN, R. A. P. Serverless architecture-a revolution in cloud computing. In: IEEE. **2018 Tenth International Conference on Advanced Computing (ICoAC)**. [*S. l.*], 2018. p. 88–93.

SANTOS, D.; RESENDE, A.; JUNIOR, P. A.; COSTA, H. Attributes and metrics of internal quality that impact the external quality of object-oriented software: A systematic literature review. In: IEEE. **2016 XLII Latin American Computing Conference (CLEI)**. [*S. l.*], 2016. p. 1–12.

SHAW, T.; MCGREGOR, D.; BRUNNER, M.; KEEP, M.; JANSSEN, A.; BARNET, S. What is ehealth (6)? development of a conceptual model for ehealth: qualitative study with key informants. **Journal of medical Internet research**, JMIR Publications Inc., Toronto, Canada, v. 19, n. 10, p. e324, 2017.

TADAPANENI, N. R. Different types of cloud service models. **Available at SSRN 3614630**, 2017.

WANG, Y.; WEI, J. Toward protecting control flow confidentiality in cloud-based computation. **Computers Security**, v. 52, p. 106 – 127, 2015. ISSN 0167-4048. Disponível em: http://www.sciencedirect.com/science/article/pii/S0167404815000553.

WHO. **Health and development**. World Health Organization, 2016. Disponível em: https://www.who.int/hdp/. Acesso em: 30 mar. 2020.

ZHANG, L.; ZHANG, L. **4 Use Cases of Serverless Architecture - DZone Cloud**. 2018. Disponível em: https://dzone.com/articles/4-use-cases-of-serverless-architecture.

ZHANG, Y.; PATWA, F.; SANDHU, R. Community-based secure information and resource sharing in aws public cloud. In: IEEE. **2015 IEEE Conference on Collaboration and Internet computing (CIC)**. [*S. l.*], 2015. p. 46–53.

APÊNDICE A – ARQUIVO YML UTILIZADO PARA CRIAÇÃO E IMPLANTAÇÃO DA APLICAÇÃO *SERVERLESS*

Código-fonte 1 – Arquivo de configuração *Serverless Framework*

```
service: medical-records-with-serverless

provider:
  name: aws
  runtime: nodejs12.x
  region: us-east-1
  environment:
    usuarioTableName: ${self:custom.usuarioTableName}
    consultaTableName: ${self:custom.consultaTableName}
  iamRoleStatements:
    - Effect: Allow
      Action:
        - dynamodb:*
      Resource: '*'

plugins:
  - serverless-webpack
  - serverless-dynamodb-local
  - serverless-middleware
  - serverless-offline

package:
  individually: true

custom:
  usuarioTableName: usuarios
  consultaTableName: consultas
  serverless-offline:
```

```
29      useChildProcesses: true
30    dynamodb:
31      stages:
32        - dev
33      start:
34        port: 8000
35        inMemory: true
36        migrate: true
37      migration:
38        dir: offline/migrations
39    middleware:
40      pre:
41        - src/middlewares/authorizer.authorize
42
43  functions:
44    usuarios:
45      handler: src/handlers/usuarios.handler
46      timeout: 30
47      events:
48        - http:
49            path: usuarios/{usuario}
50            method: GET
51            cors: true
52        - http:
53            path: usuarios
54            method: POST
55            cors: true
56        - http:
57            path: usuarios/{usuario}
58            method: PUT
59            cors: true
60        - http:
```

```
61          path: usuarios/{usuario}
62          method: DELETE
63          cors: true
64      - http:
65          path: usuarios/entrar
66          method: POST
67          cors: true
68  consultas:
69    handler: src/handlers/consultas.handler
70    timeout: 30
71    events:
72      - http:
73          path: consultas
74          method: GET
75          cors: true
76      - http:
77          path: consultas
78          method: POST
79          cors: true
80      - http:
81          path: consultas/{id}
82          method: PUT
83          cors: true
84  historico:
85    handler: src/handlers/historico.handler
86    timeout: 30
87    events:
88      - http:
89          path: historico/{usuario}
90          method: GET
91          cors: true
```

APÊNDICE B – ARQUIVO YAML UTILIZADO PARA REALIZAÇÃO DO TESTE

Código-fonte 2 – Arquivo de teste

```
config:
  environments:
    serverless:
      target: "https://iktn5kwor3.execute-api.us-east-1.
         amazonaws.com/dev"
      phases:
        - duration: 10
          arrivalRate: 5
          name: Aquecimento
        - duration: 20
          arrivalRate: 5
          rampTo: 10
          name: Aumentar a carga
        - duration: 30
          arrivalRate: 10
          name: Manter a carga
    monolith:
      target: "http://52.91.154.226:3000"
      phases:
        - duration: 10
          arrivalRate: 5
          name: Aquecimento
        - duration: 20
          arrivalRate: 5
          rampTo: 10
          name: Aumentar a carga
        - duration: 30
          arrivalRate: 10
          name: Manter a carga
```

```
29   variables:
30     senha:
31       - "teste"
32     endereco:
33       - "Rua do meio 707, Centro"
34 scenarios:
35   - name: "Usuario se registra e acessa o sistema para
        criar uma consulta e visualiza-la no seu historico
        apos ser atendido"
36     flow:
37       - post:
38           url: "/usuarios"
39           json:
40             usuario: "{{ $randomString() }}"
41             senha: "teste"
42             tipo: "PACIENTE"
43             nome: "Testador da Silva"
44             cpf: "14434778005"
45             dataNascimento: "2000-01-01"
46             endereco: "{{ endereco }}"
47             telefone: "88999999999"
48             crm: null
49             especialidade: null
50           capture:
51             json: "$.usuario"
52             as: "usuario"
53       - post:
54           url: "/usuarios/entrar"
55           json:
56             usuario: "{{ usuario }}"
57             senha: "{{ senha }}"
58           capture:
```

```
59            json: "$.accessToken"
60            as: "token"
61      - post:
62          url: "/consultas"
63          headers:
64            Authorization: "{{ token }}"
65          json:
66            usuario: "{{ usuario }}"
67            endereco: "{{ endereco }}"
68            data: "2021-04-29"
69            especialidade: "OTORRINOLARINGOLOGIA"
70          capture:
71            json: "$.id"
72            as: "consultaId"
73      - put:
74          url: "/consultas/{{ consultaId }}"
75          headers:
76            Authorization: "{{ token }}"
77          json:
78            usuario: "{{ usuario }}"
79            endereco: "{{ endereco }}"
80            data: "2021-04-29"
81            especialidade: "OTORRINOLARINGOLOGIA"
82            status: "FINISHED"
83            medico: "Dr. John"
84            diagnostico: "O paciente apresenta congestao
                nasal, obstrucao da regiao nasal, coriza,
                coceira e ardor no nariz e olhos, espirro,
                tosse, olhos com lacrimejamento, roncos
                nasais, falta de ar, respiracao oral,
                rouquidao, reducao do paladar e olfato,
                olheiras e olhos inchados e fadiga. Sintomas
```

```
                de rinite."
85      - get:
86          url: "/historico/{{ usuario }}"
87          headers:
88            Authorization: "{{ token }}"
```