UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ - UNIOESTE

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Colegiado de Ciência da Computação

Curso de Bacharelado em Ciência da Computação

Estendendo a capacidade de dispositivos IoT com Serverless

Trabalho de Conclusão de Curso

João Pedro Bach Dotta

Cascavel–PR

2021

UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ - UNIOESTE

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Colegiado de Ciência da Computação

Curso de Bacharelado em Ciência da Computação

João Pedro Bach Dotta

Estendendo a capacidade de dispositivos IoT com Serverless

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel.

Orientador(a): Guilherme Galante
Coorientador(a): Marcio Seiji Oyamada

Cascavel–PR

2021

JOÃO PEDRO BACH DOTTA

ESTENDENDO A CAPACIDADE DE DISPOSITIVOS IOT COM SERVERLESS

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Guilherme Galante (Orientador)
Colegiado de Ciência da Computação, UNIOESTE

Prof. Marcio Seiji Oyamada (Coorientador)
Colegiado de Ciência da Computação, UNIOESTE

Prof. Edmar Bellorini
Colegiado de Ciência da Computação, UNIOESTE

Cascavel, 25 de julho de 2022

Agradecimentos

Agradeço todas as amizades presentes na minha jornada, que eu tenha impactado, na vida de vocês, o quanto vocês impactaram a minha.

Agradeço a todos os professores pelo conhecimento compartilhado e pela competência ao exercer suas funções. Um obrigado especial a meu orientador, Guilherme Galante, e, aos professores Marcio Oyamada, Edmar Bellorini e Adair Santa Catarina.

Agradeço, principalmente, meus pais, Elvio Luiz Dotta e Carmen Bach Dotta, que sempre estiveram presentes para mim. É impossível expressar, em palavras, a quantidade de gratidão que eu sinto por ter vocês na minha vida.

Agradeço meus irmãos Luiz, Rafael e Felipe, vocês sempre foram exemplos para mim e, toda minha família, que nos mantenhamos unidos e nos apoiando, sempre.

So crucify the ego, before it's far too late
And leave behind this place so negative and blind and cynical
And you will come to find that we are all one mind
Capable of all that's imagined and all conceivable
So let the light touch you so that the words spill through
And let the past break through, bringing out our hope and reason
Reflection - Tool

Resumo

Dispositivos IoT estão em crescente desenvolvimento, com tendências para esse desenvolvimento ser ainda maior nos próximos anos, devido a sua capacidade de impactar a vida de seres humanos no mundo todo. Porém, devido a estrutura desses dispositivos e à sua heterogeneidade, sistemas IoT apresentam limitações e desafios em diversos aspectos como, por exemplo, desempenho, consumo de energia e armazenamento de dados. Uma possível solução para minimizar essas limitações é a integração de dispositivos IoT com o paradigma de computação em núvem, *Serverless*. A computação *Serverless* se diferencia de outras abordagens de computação em núvem devido a facilidade de desenvolver sistemas escaláveis, seguros e performáticos sem a necessidade de gerenciar servidores por parte do desenvolvedor. Esse trabalho tem como objetivo analisar a integração dessas duas tecnologias, afim de concluir se essa pode ser utilizada para ajudar com as limitações de dispositivos IoT notando, principalmente, consumo de energia e desempenho. Uma aplicação de detecção de faces foi adaptada para ser executada em um dispositivo IoT e no serviço de computação em núvem *Lambda* da AWS, e, através de experimentos, métricas foram coletadas para concluir o impacto que a tecnologia *Serverless* pode ter dentro do desenvolvimento de sistemas IoT. A partir dos resultados, concluimos nesse estudo que a integração da tecnologia *Serverless* é possível no desenvolvimento de sistemas IoT, e funciona para minimizar limitações como, por exemplo, quando desempenho é uma prioridade ou quando a quantidade de bateria desses dispositivos é escassa.

Palavras-chave: *Internet of Things*, Dispositivos IoT, *Serverless*, *Function as a Service*, Computação em Núvem.

Lista de figuras

Lista de códigos

Lista de abreviaturas e siglas

IoT	*Internet of Things*

Sumário

1

Introdução

Internet of things (IoT ou Internet das coisas em português) pode ser definido como uma rede de coisas, tais como sensores, atuadores, controladores ou qualquer outro dispositivo capaz de monitorar o que está acontecendo a sua volta e se conectar à Internet, que permite com que essas coisas interajam com humanos ou outras coisas e realizem qualquer tipo de tarefa com pouca ou nenhuma intervenção humana. Essas coisas gerarão dados que serão analisados e processados para chegar em seu objetivo final e podem ou não ter a capacidade de ser controlado pelo usuário por uma interface gráfica (AGUDELO-SANABRIA; JINDAL, 2021).

A infraestrutura de um sistema IoT pode ser muito heterogênea dada a grande variedade de dispositivos que podem ser usados nessa área. Geralmente, consiste da própria coisa que monitorará a sua volta como sensores, atuadores, dentre outros, a nuvem que pode processar os dados com bom desempenho, apenas armazená-los, dentre outras funções, a camada de comunicação entre a coisa e a nuvem para que os dados sejam usados e a camada de controle que tem como objetivo o manuseio e controle do sistema, mantendo-o seguro e funcionando (MILENKOVIC, 2020).

O mercado de IoT é uma tendência atualmente, já que seu uso pode impactar positivamente diversos casos do mundo real, como, por exemplo na natureza, sendo útil em diversos pontos, desde uma economia de água ou energia até a previsão de um possível desastre natural, na logística, podendo ser usado em cidades inteligentes para monitoramento de tráfego de veículos e semáforos inteligentes, no desenvolvimento de casas inteligentes podendo-se obter economia de energia, segurança, e um maior controle da casa, no mercado automotivo, na medicina, dentre outros (MARJANI et al., 2017).

Apesar de avanços em tecnologia nessa área, dispositivos IoT ainda sofrem com diversas limitações devido a sua infraestrutura e heterogeneidade. Pode ser devido ao seu tamanho, já que, em vários casos, dispositivos IoT tem uma quantidade limitada de espaço físico para serem implantados e, mais notavelmente, devido à bateria. A bateria é, atualmente, um dos

maiores problemas no desenvolvimento de dispositivos IoT, acompanhada pelo desempenho e armazenamento, já que um desempenho maior ou um armazenamento ideal de dados geralmente significa um aumento no consumo de energia e dispositivos IoT devem ser o mais eficiente possível (LYNN et al., 2020).

Devido a essas limitações, o desenvolvimento de sistemas IoT com todos os requisitos ideais como segurança, privacidade, evolução, disponibilidade, escalabilidade, desempenho, dentre outros, pode ser muito desafiador. Uma possibilidade de mitigar essas limitações é estender a capacidade dos dispositivos através do uso da computação em nuvem, mais especificamente com o uso do modelo de serviço chamado *Serverless*, também conhecido como *Function as a Service* (FaaS).

O paradigma *Serverless* é um modelo de desenvolvimento em nuvem que consiste no desenvolvedor implementar microsserviços (funções curtas e específicas) em uma aplicação dirigida por eventos que depois é implantada na plataforma. A plataforma é a responsável por escalar o desempenho e os recursos a serem utilizados dependendo da carga de trabalho necessária para executar o serviço. O modelo *Serverless* contém algumas vantagens no processamento em nuvem como possível custo reduzido da aplicação e maior facilidade em seu desenvolvimento (AGUDELO-SANABRIA; JINDAL, 2021).

O *Serverless* é um modelo atrativo para desenvolvimento de sistemas IoT pois facilita o desenvolvimento de sistemas escaláveis, seguros e com boa performance e também pode prover vantagens como reduzir custos operacionais e reduzir custos e tempo de implantação e desenvolvimento (AGUDELO-SANABRIA; JINDAL, 2021). Apesar das vantagens, devido à alta virtualização da plataforma FaaS, esse modelo pode não ser adequado para algumas classes de aplicações, tais como aquelas que exigem tempo de resposta de nanosegundos ou milissegundos.

Dentro desse contexto, considerando que o desenvolvimento de sistemas IoT eficientes ainda é um desafio, o objetivo dessa pesquisa é analisar como essas duas tecnologias podem se integrar para minimizar as limitações dos dispositivos IoT, em quais ocasiões vale a pena ter essa integração, quais os cenários mais adequados para essa tecnologia e se, de fato, a tecnologia *Function as a Service* tem potencial no desenvolvimento de sistemas IoT. Para isso, deve-se avaliar um ou mais casos de uso em que essa integração é possível e verificar os resultados provenientes dessa integração.

Nesse trabalho, uma aplicação integrando um dispositivo IoT e uma plataforma *serverless* foi desenvolvida e, a partir dessa, métricas foram coletadas e observadas para avaliar o impacto que a tecnologia *serverless* pode ter dentro do cenário de dispositivos IoT e se, realmente, essa tecnologia teve um impacto positivo nessa integração.

O restante desse trabalho é organizado da seguinte forma: O Capítulo 2 aprofunda os conceitos de IoT, *Serverless*, a integração dessas tecnologias e discute trabalhos relacionados enquanto o Capítulo 3 oferece, de forma aprofundada, as tecnologias utilizadas e os passos

realizados para satisfazer o objetivo desse estudo. O capítulo Capítulo 4 demonstra o cenário de testes desse trabalho e os resultados provenientes desses testes e, por fim, o Capítulo 5 conclui o trabalho.

2

Fundamentação Teórica

Nesse capítulo, são apresentados os principais conceitos necessários para o entedimento desse trabalho. Na seção 2.1, aprofundamos os conhecimentos sobre *Internet of Things*, enquanto na seção 2.2, sobre o conceito *Serverless*. Na seção 2.3, exploramos como essas duas tecnologias podem ser integradas e, por fim, na seção 2.4, analisamos como esse trabalho se diferencia à outros estudos sobre essa integração.

2.1 Internet of Things

2.1.1 Conceito

Pode-se afirmar que o termo *Internet of Things* (IoT) ainda não tem uma definição única e universal. A IBM (*International Business Machines Comporation*) define IoT como "o conceito de conectar qualquer dispositivo a Internet e a outros dispositivos" e cita que IoT "se refere à crescente quantidade de dispositivos conectados a Internet que capturam ou geram quantidades enormes de dados diariamente" (FIROUZI et al., 2020).

Apesar disso, pode-se definir IoT como uma rede de coisas que permite a essas coisas monitorarem o ambiente ao seu redor, interagirem com humanos e outras coisas e executar qualquer tipo de tarefa com pouca ou nenhuma intervenção humana. Essas coisas podem ser definidas como quaisquer tipo de sensores, controladores, atuadores ou qualquer outro tipo de dispositivo capaz de se conectar à Internet (AGUDELO-SANABRIA; JINDAL, 2021). Para que um dispositivo seja parte do conceito de IoT, esse dispositivo precisa ter uma unidade de processamento, uma fonte de energia, um sensor/atuador, um tipo de conexão à Internet e um endereço como, por exemplo endereço IP, para que possa ser unicamente identificado (FIROUZI et al., 2020).

Os dispositivos e objetos de uma rede IoT estão interconectados dentre uma variedade

de tipos de comunicação como Bluetooth, Wi-Fi, redes de celulares como 4G e 5G, LPWAN, dentre outros. Esses dispositivos transmitem dados e podem receber comandos remotamente através de outros dispositivos, o que permite uma integração direta com o mundo físico através de computadores e outros sistemas (MARJANI et al., 2017). Essa rede IoT consegue criar um ambiente mais inteligente e autossuficiente que depende menos na entrada de entidades físicas e mais na entrada de outras coisas, fazendo com que o usuário lide menos com tarefas rotineiras (PINTO; DIAS; FERREIRA, 2018).

Existe um vasto número de aplicações IoT que podem ser agrupadas em diversos domínios tais como saúde, tráfego, logística, agricultura, cidades inteligentes e automação de processos. Apesar dos avanços na área IoT recentemente, o desenvolvimento de sistemas IoT ainda provê diversos desafios como, por exemplo identificação de dispositivos, interoperabilidade, segurança, privacidade, eficiência de energia, desenvolvimento escalável, dentre outros também são importantes no desenvolvimento de dispositivos IoTs.

Apesar dos desafios, ao conectar bilhões de dispositivos na Internet, IoT tem provado seu potencial em transformar sociedades e atraiu muita atenção e investimentos das áreas industriais e acadêmicas (TRILLES; GONZALEZ-PEREZ; HUERTA, 2020). Os autores de (MANYIKA; CHUI; BUGHIN, 2013) preveem que o número de dispositivos conectados na Internet chegará a 2,3 trilhões no ano de 2025 e, também, os autores de (INSTITUTE, 2015) preveem um possível valor gerado de um pouco mais que 11 trilhões de dólares para o mesmo ano.

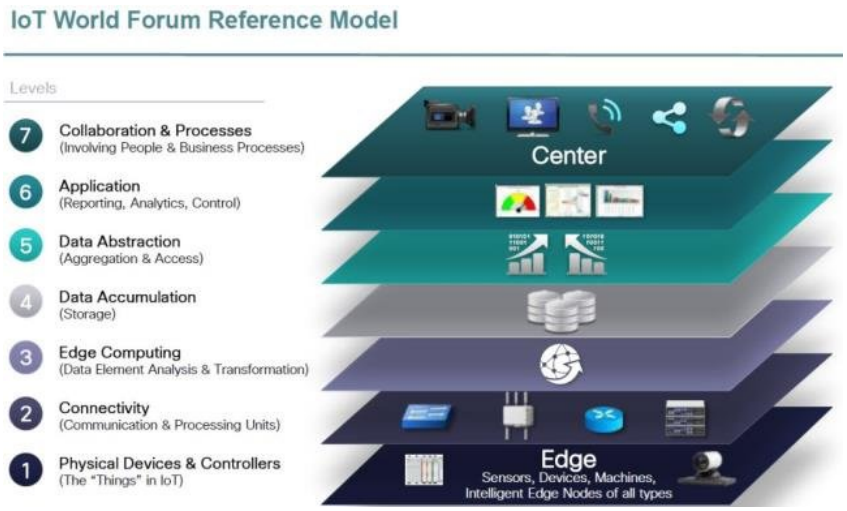
2.1.2 Arquitetura

Devido à heterogeneidade dos sistemas IoT e o fato de usar um vasto número de tecnologias diferentes, não é possível se ter uma arquitetura universal para todas as possíveis implementações de sistemas IoT, por isso, várias arquiteturas coexistirão no mundo IoT (GLOUKHOVTSEV, 2018). Apesar disso, no evento The IoT World Forum (IoTWF) de 2014, um comitê de várias empresas significativas no mercado de tecnologia como IBM, Cisco, Rockwell Automation, dentre outras, propuseram um modelo de referência para a padronização de uma arquitetura IoT (FIROUZI et al., 2020). O propósito desse modelo de referência é dar um entendimento geral de como o problema de criar e desenvolver um sistema IoT pode ser dividido. A Figura 1 ilustra o modelo dividido em sete camadas. A partir dessas sete camadas, é possível fazer uma divisão de objetivos para cada uma, assim, empresas diferentes podem contribuir juntas com diferentes camadas que interoperarão dentre si. Essas sete camadas são:

1 - Dispositivos físicos e controladores (coisas): consiste em qualquer tipo de dispositivo, sensor, atuador ou controlador que forma a rede IoT. Sua função principal é a coleta de dados mas também pode ser capaz de receber e executar instruções;

2 – Conectividade: responsável por transmitir os dados coletados pela primeira camada e mandá-los para camadas mais acima, onde os dados podem ser analisados e utilizados. O

Figura 1 – Modelo de referência IoT apresentado em 2014 no IoTWF



Fonte: Internet of things World Forum

principal objetivo ao ser desenvolvida é garantir a segurança, confiabilidade e agilidade ao transmitir os dados para outras camadas da arquitetura;

3 – Computação de borda (Análise e transformação de dados): também é conhecida como a camada *fog* pelo fato de ser onde os dados serão limpos, agregados e começarão a ser processados. Tem como responsabilidade preparar os dados para análise e armazenamento (MARJANI et al., 2017). Os dados são avaliados, possivelmente formatados ou reordenados, filtrados e checados contra erros e avisos. O objetivo da camada *edge/fog* é processar e tomar ações baseadas nos dados o mais rápido possível, melhorando o possível tráfego de rede causado por dispositivos IoT e contribuindo com ações em tempo real, o que podem melhorar o desempenho geral de um sistema;

4 – Armazenamento de dados: são preparados os dados para que possam ser armazenado no banco de dados, não importa seu formato. O objetivo é preparar o dado para que camadas acima possam acessar o dado a partir de consultas de banco de dados;

5 – Abstração de dados: aqui, o dado está consistente, completo e validado. Geralmente, na prática, em um sistema IoT, os dados podem ser armazenados em diversos bancos de dados diferentes e, o objetivo é garantir que, a partir de uma consulta, o resultado dela seja conciso, único e confiável;

6 – Aplicação: são os softwares presentes nas aplicações IoT podem executar funções específicas a partir dos dados coletados. Funções tais como monitoramento, geração de relatórios, controle de dispositivos, visualizações e análise dos dados;

7 – Colaboração e processos (envolve processos humanos e comerciais): é feito uso final da saída resultante dos processos da camada anterior. Dados e conclusões sobre esses dados são

compartilhados com outras entidades ou aplicações. O compartilhamento desses dados possibilita a formação de novas práticas de negócio bem como a melhora a eficácia de práticas já existentes. Nessa camada é onde melhor podemos enxergar os benefícios da rede IoT.

Como pode-se ver, essa arquitetura é completa em todos os seus conceitos e pode ser usada em uma grande parte de desenvolvimento de sistemas IoT, apesar de não ser apropriado para todos devido à heterogeneidade desses sistemas. Diversos autores como (GLOUKHOVTSEV, 2018); (INTERNET, 2017); (TRILLES; GONZALEZ-PEREZ; HUERTA, 2020); (XU; HE; LI, 2014) propõem uma arquitetura para sistemas IoT mais genérica e simplificada com apenas 4 camadas.

Essas 4 camadas são formadas geralmente de, primeiramente, uma camada de percepção, similar a primeira camada da arquitetura apresentada na IoTWF, composta por sensores, atuadores, controladores ou qualquer tipo de dispositivo capaz de sensorear dados ao seu redor, uma camada de conexão responsável por transmitir os dados da primeira camada para camadas superiores, uma camada de serviços, que analisa os dados e executa as tarefas requeridas pelo usuário e, finalmente, a camada de aplicação, onde os dados podem ser, de fato, utilizados para uma aplicação no mundo real.

2.1.3 Áreas de aplicação

Algumas áreas de aplicações de dispositivos IoT no mundo real são, mas não são limitadas a:

Automóveis: a evolução de dispositivos IoT permite avanços extremamente significativos no comportamento de automóveis atualmente, indo tão longe a contribuir para carros que dirijam sozinhos. Dispositivos IoT também estão presentes em diversas funções mais comuns em automóveis como multimídia e entretenimento, sensores de colisão, cruise-control adaptativo, dentre outras, contribuindo com segurança, praticidade e entretenimento no automóvel. Dispositivos IoT também estão fazendo parte no mundo automotivo sensoreando dados e os armazenando, alguns exemplos desses dados sendo sobre condições de estrada, se condutores estão dirigindo adequadamente, dentre outros, com o potencial de prever acidentes e melhorar condições de condução automotiva.

Cidades Inteligentes: o impacto de dispositivos IoT em cidades inteligentes está crescendo e, cada vez mais, cidades estudam e analisam maneiras de implementar essa tecnologia. Dispositivos IoT impactam cidades em vários aspectos incluindo manuseamento de tráfego de veículos, estacionamento inteligente, vigilância da cidade contribuindo para a segurança, monitoramento do ambiente da cidade para métricas de poluição e saúde, luzes inteligentes que podem adaptar seu nível de luminosidade baseado em clima e ambiente, dentre outras aplicações, deixando claro os possíveis benefícios dessa tecnologia na área de cidades

como um todo. Cidades como Amsterdã, Oslo, Singapura e Zurique são exemplos de cidades em desenvolvimento de sistemas inteligentes a partir de dispositivos IoT.

Ambientes hospitalares: dispositivos IoT podem ser usados por médicos para que pacientes possam ser monitorados remotamente, enviando notificações ou alertas para os médicos caso algo esteja errado e possivelmente permitindo com que um paciente fique no conforto de sua casa durante um tratamento em vez de permanecer em um hospital. Esses dispositivos podem ser genéricos capazes de monitorar métricas comuns a todos os pacientes como pressão sanguínea, batimentos cardíacos por minuto, até mais específicos como dispositivos capazes de monitorar se implantes específicos estejam de acordo com o esperado. Hospitais também estão adotando "camas inteligentes", as quais, enquanto o paciente está acomodado, podem medir os sinais vitais dele e também, se for o caso, enviar alertas para um médico ou enfermeira.

Industrias: dispositivos IoT podem ter uma grande variedade de funções na área industrial. Os dispositivos podem ser usados para monitoramento do chão de produção de uma fábrica, monitoramento de máquinas permitindo manutenção preventiva e maior confiabilidade nas máquinas, para controles automatizados de qualidade dos produtos, podendo contribuir, nesse ramo, em aspectos como segurança e produtividade.

Agricultura: a agricultura é uma parte essencial no aspecto de sociedade em que vivemos hoje em dia. Com uma população em constante crescimento, é possível dizer que áreas rurais precisarão ser cada vez mais eficientes para entregar a alimentação necessária para a população. Uma maneira de atingir isso é através de fazendas inteligentes. Dispositivos IoT podem ter uma grande atuação em fazendas coletando dados sobre umidade, velocidade do vento, qualidade do solo, infestações de pestes e, a partir desses dados, é possível que os dispositivos IoT tomem decisões inteligentes na agricultura para melhorar a qualidade e a quantidade da plantação bem como diminuir riscos e desperdícios.

As áreas citadas são áreas importantes e em crescente desenvolvimento envolvendo dispositivos IoT mas, dispositivos IoT estão longe de serem limitados apenas a elas. Outras áreas que valem a pena serem citadas incluem casas inteligentes, monitoramento ambiental, aplicações militares, geração de energia no geral, dentre outras áreas. É possível notar, então, a importância de dispositivos IoT atualmente bem como todos os possíveis benefícios que podem acarretar a sociedade.

2.2 Serverless

2.2.1 Conceito

O progresso contínuo no desenvolvimento de hardware e redes de computadores abriram caminho para a introdução de um novo paradigma, a computação em nuvem, permitindo que desempenho computacional seja visto como apenas mais uma utilidade, usada sob demanda, com capacidade virtualmente ilimitada. Para que tais tecnologias sejam usufruídas por um vasto número de usuários, abstrações são introduzidas nos formatos de *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) e *Software as a Service* (SaaS).

Na categoria SaaS, provedores oferecem diferentes tipos de serviços na forma de *software* para os usuários, sem que o usuário se preocupe com o desenvolvimento desses serviços, seu manuseamento ou sua implantação, apenas utilizando o serviço sem se preocupar com suas configurações. Google oferece vários tipos desse serviço na forma de, por exemplo, Google docs, Google sheets e alguns outros. Na categoria PaaS, provedores disponibilizam serviços como acesso à Internet, armazenamento, servidores e sistemas operacionais para que sejam comprados por desenvolvedores e usados para implantação, desenvolvimento e execução de suas aplicações. Nesse tipo de nuvem, o desenvolvedor não tem controle sobre os serviços, mas é responsável pelas configurações de seu software e precisa assegurar que sua aplicação está executando, oferecendo um nível de abstração menor que o SaaS, por exemplo. Finalmente, IaaS oferece o menor nível de abstração dos três serviços e, nesse, o usuário apenas compra e manuseia serviços como servidores, sistemas operacionais e armazenamento (HASSAN; BARAKAT; SARHAN, 2021).

Para o usuário, existem vários desafios ao manusear uma aplicação em nuvem como, por exemplo, disponibilidade, balanceamento de carga, escalonamento, segurança, monitoramento, dentre outros. Por exemplo, o usuário precisa assegurar a disponibilidade dos serviços no caso de, se uma máquina perder conexão, não afetar toda a aplicação. Outro desafio notável é balanceamento de carga, ou seja, o usuário deve assegurar que as requisições de seus serviços estão balanceados entre as máquinas para garantir que todos os recursos estejam sendo utilizados de forma eficiente (HASSAN; BARAKAT; SARHAN, 2021).

Esses desafios levaram a introdução de outro tipo de computação em nuvem chamado de *Serverless* ou *Function as a Service* (FaaS). O conceito *Serverless* é um paradigma de computação em nuvem relativamente recente em que o usuário envia pequenas funções e provê gatilhos que sinalizam quando essas funções devem ser executadas. As funções devem ser pequenas, geralmente executando em segundos, e o usuário paga apenas pelo tempo que uma função está executando. O paradigma *Serverless* ganhou muita atenção nos últimos anos devido a vários fatores tais como, redução de custos, redução de latência, melhor escalabilidade, redução de manuseio a servidores, dentre outros (HASSAN; BARAKAT; SARHAN, 2021).

O paradigma *Serverless* possui uma maior abstração quando comparado a outros serviços de computação em nuvem, abstraindo a necessidade do gerenciamento de servidores por parte

do desenvolvedor. O modelo *serverless* permite que o desenvolvedor foque na lógica de sua aplicação em vez do gerenciamento de seu servidor e de suas configurações. Por exemplo, após o desenvolvedor enviar suas funções para o provedor *Serverless*, o provedor que gerencia, escala e prove diferentes recursos para assegurar que essas funções executem corretamente (HASSAN; BARAKAT; SARHAN, 2021). Não significa que os servidores não existam, mas sim, que o desenvolvedor não precisa se preocupar com eles (CHAUDHARY; SOMANI; BUYYA, 2017).

Por exemplo, a definição do produto AWS Lambda da Amazon é: AWS Lambda permite que um código seja executado sem a necessidade do manuseio de servidores. O usuário apenas paga pelo tempo que a função está executando, sem custos quando a aplicação está em modo ocioso. Com Lambda, o usuário pode executar virtualmente qualquer tipo de código ou serviço *backend* sem qualquer tipo de administração. É necessário, apenas, que o usuário suba seu código e o serviço Lambda assegurará que a aplicação executará corretamente, com escala automática. É possível prover gatilhos para que as funções sejam executadas automaticamente ou executa-lás diretamente de qualquer navegador ou dispositivo mobile (FOX et al., 2017).

Sendo assim, *Serverless* é uma plataforma na qual desenvolvedores escrevem código sem consideração para o hardware em que esse código executará. Aplicações são divididas em funções distintas e únicas que podem ser distribuídas e atualizadas em tempo real. Funções são executadas em *containers* leves que fornecem portabilidade, escalabilidade e segurança (LEITNER et al., 2019).

Usando provedores *Serverless*, desenvolvedores devem providenciar funções relativamente e, idealmente, atômicas e definir gatilhos para quando essas funções devem executar como, por exemplo, requisições HTTP ou eventos no sistema. Idealmente, desenvolvedores utilizando a tecnologia *Serverless* podem se beneficiar de instanciações simples, esforços operacionais reduzidos e apenas pagam pelo que é executado, enquanto o provedor tem a possibilidade de fornecer seus serviços a um grande número de usuários com relativamente poucos recursos (LEITNER et al., 2019).

Em um ambiente *Serverless*, funções geralmente descrevem apenas partes de uma aplicação maior. Funções devem executar dentro de um limite de tempo (alguns minutos) e, caso não aconteça, a execução será finalizada por alcançar o limite de tempo. As funções podem receber dados de entrada e podem, também, produzir dados para que possam ser analisados posteriormente.

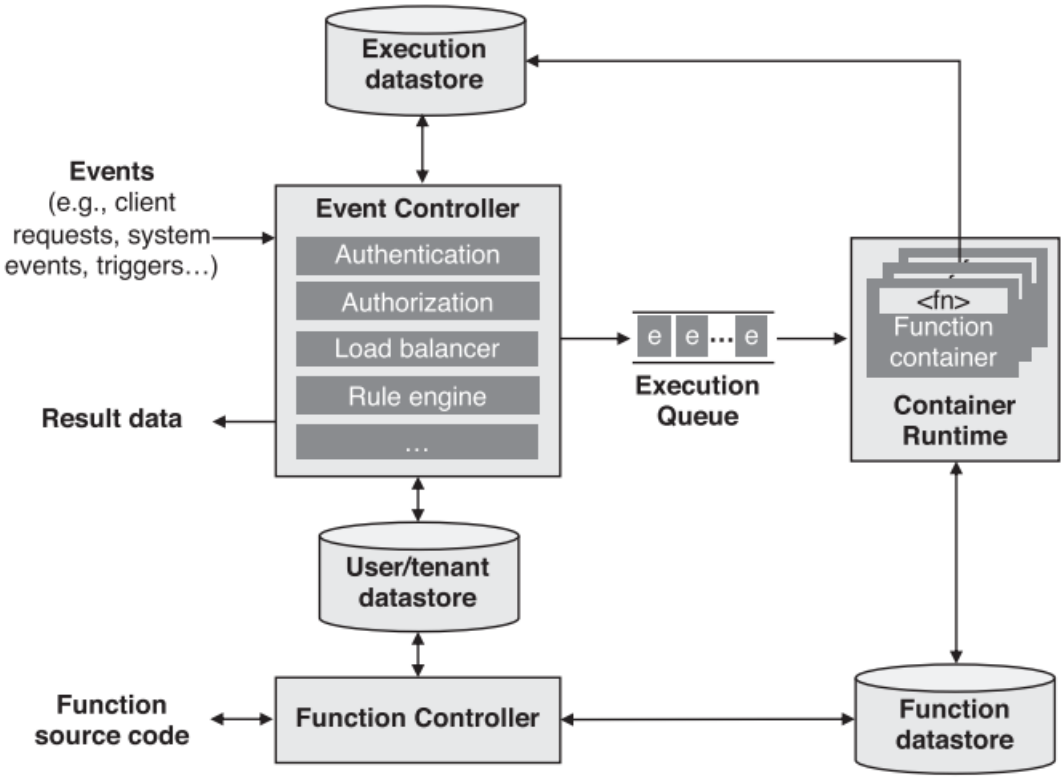
Há diversos gatilhos para uma função *serverless* como, por exemplo, requisições de clientes, eventos produzidos por algum sistema, fluxos de dados ou, até, regras complexas com uma combinação de todos os gatilhos acima pode descrever uma regra de execução de uma função. No entanto, os gatilhos são específicos a cada provedor e podem variar (LEITNER et al., 2019).

2.2.2 Arquitetura

Devido às abstrações da tecnologia *Serverless*, provedores desse paradigma possuem uma responsabilidade muito grande para garantir todos seus recursos de maneira confiável. Por exemplo, AWS possui uma confiabilidade de 99,999% em seus serviços, enquanto plataformas *Serverless* da Google e IBM possuem uma confiabilidade de 99,95% e 99,9% respectivamente (CALDERONI; MAIO; TULLINI, 2022). Devido a isso, a arquitetura de um provedor *Serverless* é um fator muito importante nessa tecnologia, a qual será explicada abaixo.

A arquitetura da Figura 2 é baseada no sistema *Serverless* Apache OpenWhisk. Ao construir o projeto, desenvolvedores interagem com o controlador de funções do provedor para criar, atualizar ou deletar funções. Junto às funções, gatilhos e, possivelmente, regras de execução, são fornecidos ditando quando uma função deve ser executada. O código das funções é persistido no banco das funções (*function datastore*) e os gatilhos e regras são persistidos no banco do usuário (*User/tenant datastore*) (LEITNER et al., 2019).

Figura 2 – Arquitetura *Serverless* baseado na plataforma Apache OpenWhisk



Fonte: (LEITNER et al., 2019)

Em tempo de execução, eventos são processados pelo controlador de eventos (*event controller*), o qual determina quais funções devem ser executadas baseados nas regras e gatilhos fornecidas pelo desenvolvedor anteriormente. O controlador de eventos também determina qual

container deve executar a função, baseado na carga atual do sistema. A execução é colocada na fila de execução (*Execution Queue*) para que o *Container Runtime* a processe. O controlador de eventos decide qual container deve executar a função ou se um novo container deve ser iniciado. A função é, então, executada no container selecionado e os resultados da execução são persistidos no banco de dados de execuções (*Execution datastore*) para que possam, depois, ser retornados para o cliente através do controlador de eventos. O *Container Runtime* pode cancelar funções que estejam demorando demais para executar e, eventualmente, irá destruir containers que estão ociosos (LEITNER et al., 2019).

2.2.3 Vantagens

Há vários benefícios relacionados ao uso de *Serverless*. Os artigos (HASSAN; BARAKAT; SARHAN, 2021) e (LEITNER et al., 2019) citam tais vantagens, mas não se limitam a:

Custo Benefício: a maneira que aplicações *Serverless* funciona relacionado ao seu custo é um grande fator quando considerado por desenvolvedores. Como aplicações *Serverless* são abstraídas do manuseio de servidores por parte do desenvolvedor, o usuário paga, apenas, pelo tempo de execução de suas funções, sem ser cobrado quando a aplicação está ociosa.

Escalabilidade: como desenvolvedores não precisam se preocupar com a escalabilidade de suas aplicações quando utilizando a tecnologia *Serverless*, isso pode ser considerado uma grande vantagem quando comparado a outros tipos de sistemas. Aplicações *Serverless* irão escalar para cima ou para baixo automaticamente de acordo com o número de requisições do usuário, fornecendo escalabilidade elástica sem qualquer tipo de configuração necessária pelo usuário.

Gerenciamento de servidores: Na computação *serverless*, desenvolvedores não precisam se preocupar com o manuseio e configuração de servidores. Provedores *serverless* tomam conta de todo o manuseio e manuntenção de *hardware* e *software* necessários para que a aplicação execute corretamente. Ainda mais, qualquer tipo de administração operacional é feita pelo provedor, permitindo a desenvolvedores focar em outros recursos como uso de CPU, memória e armazenamento.

Fácil de instanciar: aplicações *Serverless* são fáceis de instanciar. Por exemplo, para implantar uma aplicação, desenvolvedores só precisam subir suas funções para o provedor. O provedor irá cuidar de toda a infraestrutura necessária para que as funções executem corretamente.

Latência reduzida: aplicações *Serverless* não são hospedadas em um servidor específico, ou seja, a função de um usuário pode executar em qualquer servidor e, geograficamente, em "qualquer lugar". Isso significa que provedores podem executar funções o mais perto possível da localização do usuário, o que reduz a distância que a requisição do usuário precisa percorrer e, consequentemente, reduz a latência da requisição.

Os benefícios de *Serverless* também se mostram cada vez mais presentes em várias áreas das ciências e engenharias. Pesquisadores percebem que *Serverless* pode trazer diversos benefícios quando comparado a uma aplicação monolítica como, por exemplo, o uso de hardware mais eficiente para melhor processamento, o uso de hardware especializado que só está disponível em computadores remotos, mover dados para computar em outros computadores e vice-versa e, finalmente, a habilidade de execução automática de funções por gatilhos a partir de certos eventos. Por exemplo, físicos na FermiLab informam que, uma análise de dados que demora 2 segundos em uma máquina local pode ser transferida para um dispositivo FPGA nos serviços de nuvem da Amazon, onde a mesma função demora 30 milissegundos para executar. Adicionando o tempo de latência de 20 segundos para a comunicação com os servidores, o tempo total de execução é de 50 milissegundos, um aumento de 40 vezes na velocidade de execução. Esse é só um exemplo de vários que podem ser citados em domínios de aplicação específicos (CHARD et al., 2019).

2.2.4 Desvantagens

Apesar das vantagens apresentadas, nenhuma tecnologia é perfeita, e o conceito *Serverless* também possui suas desvantagens, explicadas a seguir.

Para desenvolvedores, continua um desafio adotar o *mindset* correto para utilizar da melhor forma as ferramentas da arquitetura *Serverless*. Uma forma de melhorar esse problema deve ser feito com mais documentações sobre a tecnologia e um acesso fácil a esses recursos (HASSAN; BARAKAT; SARHAN, 2021).

Devido as funções serem relativamente pequenas e, geralmente, de uma baixa complexidade, as próprias funções podem servir como unidades de testes, que podem ser executadas na máquina do desenvolvedor localmente. Porém, testar a integridade de uma aplicação que geralmente possui diversas funções e serviços externos é uma tarefa complicada, pois, simular um ambiente *Serverless* em uma máquina local é muito difícil e, as vezes, impossível (LEITNER et al., 2019).

No mesmo tópico, serviços *Serverless* são idealmente feitos para executar microsserviços, fazendo com que a composição de uma aplicação grande em diversos microsserviços bem como a decomposição de uma aplicação monolítica para a arquitetura *Serverless* ainda sejam considerados um desafio (LEITNER et al., 2019).

Além dessas dificuldades para desenvolvedores, *vendor lock-in* também é um problema. Todos os provedores *Serverless* oferecem um pacote de serviços customizados e APIs e, devido à integração do serviço *Serverless* com outros serviços em nuvem, migrar uma aplicação *Serverless* de um provedor para outro é um tanto difícil (LEITNER et al., 2019).

Também existem alguns problemas relacionados a desempenho de aplicações *Serverless* como, por exemplo, caso nenhum *container* esteja ativo, o tempo de inicialização de um pode

atrasar a execução de uma função em diversos segundos. Nos experimentos desse trabalho, por exemplo, *containers* demoraram, em média, 12 segundos para inicializar, demonstrando a desvantagem abordagem.

A heterogeneidade dos hardwares utilizados pode fazer com que tempos de execução de funções sejam difíceis de prever e, também, regras complexas de gatilhos de funções podem atrasar significativamente seu tempo de execução em algumas plataformas (SCHEUNER; LEITNER, 2020).

Os problemas de desempenho são mais aparentes, especificamente, no caso de computação científica, onde existem dependências de softwares complexos e de máquinas com maior capacidade de processamento, coisas que podem não estar disponíveis atualmente em provedores de computação em nuvem públicos. Atualmente, uma função AWS Lambda não pode executar por mais de 15 minutos, utilizar mais de 10 GB de memória RAM, usar alguns tipos de dispositivos aceleradores de hardware como placas de vídeo e usar um armazenamento maior que 10GB, o que torna o uso de funções *Serverless* na computação científica muito mais complicado (RISCO et al., 2021).

2.2.5 Provedores

Abaixo, segue uma lista dos provedores comerciais e soluções *Open-Source* mais populares na área de computação *Serverless* citadas no artigo (CHARD et al., 2019).

2.2.5.1 Plataformas comerciais

Amazon Lambda é um serviço que suporta diversas linguagens de programação bem como diversos gatilhos para execução das funções como interface Web, CLI, SDK e outros serviços AWS. Funções são cobradas baseado no quanto de memória foi alocado e para cada 100ms de tempo de execução. Amazon também possui um serviço chamado Greengrass no qual funções são implantadas em plataformas IoT.

Google Cloud Functions é um serviço *Serverless* da Google, parecido com o serviço Lambda mencionado anteriormente, o qual usa as mesmas métricas para cobrança e possui gatilhos similares. Também possui gatilhos através de *webhooks* HTTP e através de um número de sistemas de terceiros como, por exemplo, Github, Slack e Strippe.

Azure Functions permite usuários criarem funções em uma linguagem de programação nativa por interface Web ou CLI. Gatilhos podem ser executados através de um horário específico, requisições HTTP e através de alguns serviços como CosmosDB e Blob storage. O preço da Azure é similar ao preço de cobrança da Amazon para computação e armazenamento.

2.2.5.2 Plataformas Open-Source

Apache OpenWhisk é a plataforma *Serverless* open source mais popular, com sua arquitetura sendo base do serviço IBM Cloud Functions. OpenWhisk define uma programação orientada a eventos, consistindo de ações que não possuem estado, funções executaveis, gatilhos para quais tipos de eventos a OpenWhisk deve seguir e, regras que associam um gatilho com uma ação.

Fn é uma ferramenta open-source da Oracle que pode ser implantada em qualquer computador com um sistema Linux possuindo acesso de administrador para executar containers Docker. Fn tem suporte para relatórios específicos e, além disso, é uma das poucas plataformas *Serverless* open-source que podem ser implantadas no sistema operacional Windows.

Kubeless é uma plataforma *Serverless* nativa da ferramenta Kubernetes. Kubeless utiliza Apache Kafka para comunicação, fornece uma CLI com funções parecidas a CLI da AWS Lambda e, suporta monitoramento de granulação fina. Usuários podem invocar funções através da CLI ou através de requisições HTTP.

2.2.6 Casos de uso

O conceito de computação *serverless*, em todo o universo de TI, vem aumento suas capacidades de processamento para atingir um grande número de domínios de aplicação. Sendo assim, a implementação de aplicações *Serverless* pode ser aplicada para diversos propósitos como, por exemplo, treinamento de redes neurais, processamento de vídeos, processamento de quantidades enormes de dados, dentre outros. Abaixo, será citado alguns casos de uso de aplicações *Serverless* na literatura, para melhor compreender seu potencial nas áreas de tecnologia.

(HASSAN; BARAKAT; SARHAN, 2021) cita alguns casos de uso que podem ser implementados usando *serverless*:

Chatbot: Uma aplicação de chatbot é desenvolvida usando computação *serverless*, a qual é integrada com o usuário através de comandos de voz ou texto. Nesse chatbot, é possível executar seis funcionalidades, as quais são: o clima, a data e o horário atual, notícias, piadas, música e alarme. Por exemplo, um usuário pode pedir a data atual via comando de voz ou texto. Essa requisição é, então, mapeada para uma ação na plataforma OpenWhisk. A ação executa a função *Serverless* de buscar a data atual e retorna a resposta ao usuário.

Busca de informação: Uma aplicação web de busca de dados é desenvolvida baseada em uma arquitetura *Serverless*. As funcionalidades da aplicação implementadas como funções AWS Lambda. A aplicação executa todos os detalhes sobre o processo da busca de informação após o usuário executá-la como tokenização, cálculo de similaridade, classificação, ordenação e remoção de palavras vazias. Após isso, envia os resultados para o usuário como documentos

armazenados no banco de dados DynamoDB, que podem ser acessados através de uma aplicação web.

Smart grid: Uma simulação no MATLAB é criada para ilustrar o uso de uma smart grid utilizando a tecnologia *Serverless* para controlar dispositivos e manusear suas cargas elétricas. Um programa MATLAB é desenvolvido para indicar o início e o fim de um modelo de grid através de um arquivo de sistema. Esse arquivo é enviado para o serviço Amazon S3 e, após isso, uma função lambda será ativada para processar a imagem e, subsequentemente, o resultado será enviado de volta para o usuário em outro arquivo de sistema. A simulação irá ler o arquivo de retorno da função e interpretar seu conteúdo para manter um balanço na corrente elétrica do grid e dos dispositivos, fazendo isso continuamente.

Computação de borda: Computação *serverless* e computação de borda foram usados para construir diferentes tipos de aplicações. Por exemplo, um AMR (*autonomous mobile robot*) foi construído utilizando essas duas tecnologias. O sistema consiste em três componentes: um módulo AMR com NVIDIA Jetson TX2 para a computação em borda, uma arquitetura *serverless* baseada na AWS e uma aplicação mobile desenvolvida usando *React Native*. A ideia principal do sistema é entregar um pacote para o usuário. Por exemplo, o usuário irá utilizar o aplicativo mobile para enviar um pacote. Quando essa requisição é feita, o IoT pode fazer requisições lambda, como adquirir a posição de entrega baseado nas coordenadas dadas. Então, o AMR começará a missão, enviando o pacote para a localização do destinatário. Nesse mesmo projeto, imagens faciais são tiradas e processadas na lambda para identificar o rosto do destinatário. Se a identidade do destinatário for confirmada na posição de entrega do pacote, a tarefa pode ser considerada completa.

IoT: A tecnologia *Serverless* tem sido usada em várias aplicações IoT. Por exemplo, uma câmera pode ser instalada para monitorar uma casa e, as fotos tiradas por essa câmera, podem ser processadas por uma função *serverless* presente na plataforma OpenWhisk. Quando a câmera detecta algo interessante como, por exemplo, um carro ou uma pessoa, a câmera envia essa imagem para a plataforma *serverless* para processamento. Essa conexão serverless pode ser usada para diferentes tipos de processamento na imagem, enviando os resultados desejados para o usuário final.

2.3 IoT + Serverless

Considerando que dispositivos IoT possuem, geralmente, capacidades de processamento e armazenamento baixas e são, muitas vezes, ligados por uma bateria, ou seja, possuem uma quantidade de energia limitada, a tecnologia *Serverless* pode oferecer uma grande vantagem devido a sua capacidade de processamento e armazenamento virtualmente ilimitada sem consumir grandes quantidades de energia do dispositivo para ser utilizada (KJORVEZIROSKI; FILIPOSKA;

TRAJKOVIK, 2021). É estimado que dispositivos IoT fazem parte de 6% do uso total da indústria *serverless* (GEORGE et al., 2020).

Outros benefícios podem ser citados como (CALDERONI; MAIO; TULLINI, 2022):

- Economiza os recursos e custos de se construir e manter uma infraestrutura de *hardware back-end*.
- Permite que o desenvolvedor foque apenas em desenvolver a estrutura *back-end* sem se preocupar com alguns aspectos como configuração de rede ou implementação de mecanismos de segurança.
- É horizontalmente e verticalmente escalável de forma transparente, sem que o desenvolvedor se preocupe com isso.
- É necessário apenas pagar quando uma função está executando, ou seja, apenas pelos recursos que são utilizados.

Apesar disso, dispositivos IoT possuem alguns desafios ao utilizar a tecnologia. Formas de transferir dados em um tempo aceitável, com pouco atraso, sempre provou ser um desafio. Enquanto a nuvem é uma escolha excelente para aplicações com tempos de percepção de humanas, ou seja, aplicações onde tempos de processamento de centenas de milissegundos ou, até, segundos, são aceitáveis, executar uma aplicação em tempos de percepção de máquina é muito mais difícil (KJORVEZIROSKI; FILIPOSKA; TRAJKOVIK, 2021).

Mesmo assim, esses desafios podem ser endereçados dependendo do tipo de função que o dispositivo IoT precisa executar. Funções com tempos de respostas críticos, ou seja, funções que precisam ser executadas e retornadas em milissegundos como, por exemplo, um carro com piloto automático detectando que há um objeto na sua frente e que precisa freiar imediatamente, não são ideias para o paradigma *Serverless*. Essas funções são melhor deixadas para serem executadas localmente, na *edge* ou *fog*, para se beneficiarem da latência baixa oferecida por essas soluções (PINTO; DIAS; FERREIRA, 2018).

Porém, qualquer tipo de função que requere um poder de processamento maior e, que, principalmente, não possui tempo de resposta crítico, pode ser executada na nuvem com os benefícios citados acima. Além disso, essas funções irão se beneficiar do alto poder de processamento que provedores *Serverless* possuem, executando em um *hardware* com uma capacidade de processamento muito superior ao do dispositivo IoT, levando menos tempo para executar mesmo ao considerar a latência para fazer a chamada da função e, a latência para receber a resposta do provedor. Para isso ser possível, o dispositivo precisa, apenas, de uma conexão com a Internet (PINTO; DIAS; FERREIRA, 2018).

2.4 Trabalhos relacionados

Vários trabalhos estudam a viabilidade de se implementar a tecnologia *serverless* relacionada a área de IoT. O artigo (KJORVEZIROSKI et al., 2021), por exemplo, estuda os principais desafios dessa integração bem como quais desafios devem ser focados para melhorar essa integração no futuro. Já os artigos (BENEDETTI et al., 2021) e (BENOMAR et al., 2021) estudaram experimentos práticos com a integração da tecnologia e seus resultados.

Porém, deve-se notar que esses artigos trabalham com a noção de se usar uma rede de dispositivos IoT os quais não computam ou processam os dados, apenas coletam dados e enviam-nos para a computação de borda. A computação de borda é, então, responsável por processar localmente os dados ou decidir se a nuvem é a melhor escolha para a tarefa.

Apesar dessa abordagem ainda possuir benefícios como possível melhora de desempenho, armazenamento praticamente infinito, auto escalabilidade, dentre outros, providos pela tecnologia *serverless*, o intuito desse trabalho é analisar dispositivos IoT que possuam a capacidade de fazer seu próprio processamento e, analisar, quando esse processamento é valido para o próprio dispositivo e quando esse processamento é melhor deixado para o servidor *serverless*, bem como vantagens/desvantagens desse *trade-of* entre computação local e computação em nuvem.

3

Estendendo a capacidade de dispositivos IoT com FaaS

Conforme apresentado no Capítulo 2, a integração entre dispositivos IoT e *Serverless* é promissora, mas também possui desafios. Nesse contexto, o objetivo geral dessa pesquisa é analisar como essas duas tecnologias podem se integrar de forma vantajosa, em quais ocasiões vale a pena ter essa integração e se, de fato, a tecnologia *Function as a Service* é eficiente no desenvolvimento de sistemas IoT. Para isso, será testado um caso de uso em que essa integração é possível e verificado os resultados provenientes dessa integração.

Para que o objetivo desse estudo seja satisfeito, uma mesma aplicação foi adaptada para ser executada tanto em um dispositivo IoT, quando na plataforma *serverless*, para que as métricas de desempenho, custo energético, tempo de latência, dentre outros, possam ser analisadas quando comparando o processamento dessa aplicação localmente, no dispositivo IoT, contra o processamento em nuvem, na tecnologia *serverless*.

3.1 Aplicação

A aplicação que atende essas especificidades proposta é uma aplicação de reconhecimento facial. O sistema implementado já deve ter uma base de treino para o reconhecimento de faces, tornando esse passo desnecessário. A função deve, apenas, receber uma imagem e, a partir da base de dados já treinada, reconhecer quantas faces e quantos olhos estão presentes nessa imagem. Esse processamento será feito no dispositivo IoT usando seu próprio *hardware* e, também, no provedor *serverless* acionado pelo próprio dispositivo. A aplicação foi desenvolvida na linguagem de programação Python, utilizando a biblioteca *OpenCV*(BRADSKI, 2000) para o processamento e tratamento das imagens com o método de *Haar-cascade* como algoritmo para o reconhecimento de olhos e faces em si.

A aplicação foi desenvolvida em Python para que o processamento local e o processamento em nuvem fo fossem equiparados. É claro que, em termos de desempenho, uma linguagem de

baixo nível como, por exemplo, C++, teria um desempenho melhor, mas, pela dificuldade de portar a biblioteca *OpenCV* na plataforma *serverless* em linguagens de baixo nível, a linguagem Python foi escolhida.

A aplicação, originalmente, foi desenvolvida para capturar, em tempo real, a *webcan* do computador em que o código está sendo executado e, a partir disso, a cada *frame*, processar e indicar, na tela, as faces e os olhos presentes na imagem. Como, para os objetivos desse estudo, esse código não é apropriado, o código foi adaptado para receber uma imagem como parâmetro, processar essa imagem e armazenar os resultados, para que a aplicação seja portável, tanto em um dispositivo IoT, quanto na plataforma *serverless*.

Um código resumido da aplicação é mostrado no Código 1 enquanto o código original está disponível no repositório (OPENCV, 2019). Códigos adaptados para IoT e *Serverless* serão mostrados em suas seções respectivas.

Código 1 – Programa face-cascade.py

```
import cv2 as cv
def detectAndDisplay(frame):
    frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    frame_gray = cv.equalizeHist(frame_gray)
    faces = face_cascade.detectMultiScale(frame_gray)
    for (x,y,w,h) in faces:
        center = (x + w//2, y + h//2)
        frame = cv.ellipse(frame, center, (w//2, h//2), 0, 0, 360, (255, 0,
    255), 4)
        faceROI = frame_gray[y:y+h,x:x+w]
        eyes = eyes_cascade.detectMultiScale(faceROI)
        for (x2,y2,w2,h2) in eyes:
            eye_center = (x + x2 + w2//2, y + y2 + h2//2)
            radius = int(round((w2 + h2)*0.25))
            frame = cv.circle(frame, eye_center, radius, (255, 0, 0 ), 4)
    cv.imshow('Capture - Face detection', frame)

face_cascade.load(cv.samples.findFile(face_cascade_name))
eyes_cascade.load(cv.samples.findFile(eyes_cascade_name))
camera_device = args.camera
cap = cv.VideoCapture(camera_device)
while True:
    ret, frame = cap.read()
    detectAndDisplay(frame)
```

3.2 Dispositivo IoT

O dispositivo utilizado para executar a aplicação e simular um dispositivo IoT foi um Raspberry PI 4 Model B, com as seguintes especificações:

- CPU: Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz;

- RAM: 2GB LPDDR4-3200 SDRAM;
- Sistema operacional: Raspberry Pi OS 64bit.

Para a aplicação ser adaptada para executar no cenário local, o código foi alterado para executar o processamento em uma imagem e armazenar os resultados em uma vetor. Uma versão resumida do código alterado para esse caso é mostrado no Código 2. O código completo está disponível no repositório (DOTTA, 2022a).

A aplicação carrega um número de imagens especificado pelo usuário, executa o processamento, armazena os resultados em arquivos e, então, envia os resultados para o banco de dados. O banco de dados utilizado, nesse caso, foi o serviço S3 da *Amazon*.

Código 2 – Programa codigoLocal.py

```
import cv2 as cv
def detectAndDisplay(image_list):
    result_list = []
    for image in image_list:
        img = cv.imread(image)
        gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        gray_img = cv.equalizeHist(gray_img)
        faces = face_cascade.detectMultiScale(gray_img)
        eyes_qnt = 0
        for (x,y,w,h) in faces:
            faceROI = gray_img[y:y+h,x:x+w]
            eyes = eyes_cascade.detectMultiScale(faceROI)
            eyes_qnt += len(eyes)
        result_list.append(faces + " " + eyes_qnt)
    return result_list

face_cascade.load(cv.samples.findFile(face_cascade_name))
eyes_cascade.load(cv.samples.findFile(eyes_cascade_name))
image_list = getTargetImages(imagePath)
results = detectAndDisplay(imageList)
sendResultsToS3(results)
```

3.3 *AWS Lambda*

A plataforma utilizada para executar a aplicação foi o serviço *Lambda* disponível na AWS, pela facilidade de utilização e configuração. A *Amazon* oferece 1 ano gratuito de todas as suas funcionalidades de nuvem, o suficiente para o desenvolvimento desse trabalho.

Devido à tecnologia *Serverless* utilizar *containers* para execução das funções, não é possivel ter controle de pacotes instalados nas máquinas, como se tem em uma máquina local. Isso apresenta um problema, devido à necessidade da biblioteca *OpenCV* para a execução da aplicação utilizada nesse trabalho.

Para resolver esse problema, é necessário utilizar o conceito *Layers* disponível nas funções da Lambda, que permite se utilizar código adicional na função na forma de um conteúdo ZIP. Dessa forma, podemos compactar o pacote *OpenCV* em uma arquitetura que espelha o container da AWS e, então, adicionamos esse pacote como uma *Layer* na função *Lambda*. Um tutorial para como fazer esse processo está disponível em (CARVALHO, 2021).

O código resumido da aplicação na plataforma *Serverless* está disponível no Código 3, enquanto a versão original está disponível no repositório (DOTTA, 2022b).

Código 3 – Programa codigoServerless.py

```
import cv2 as cv
def process_image(filename, face_cascade, eyes_cascade):
    img = cv.imread(filename)
    eyes_qnt = 0
    frame_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    frame_gray = cv.equalizeHist(frame_gray)
    faces = face_cascade.detectMultiScale(frame_gray)
    for (x,y,w,h) in faces:
        faceROI = frame_gray[y:y+h,x:x+w]
        eyes = eyes_cascade.detectMultiScale(faceROI)
        eyes_qnt += len(eyes)
    return faces, eyes_qnt

def lambda_handler(event, context):
    start_time = time.time()
    face_cascade, eyes_cascade = load_cascades()
    bucketName = event['Records'][0]['s3']['bucket']['name']
    s3Key = event['Records'][0]['s3']['object']['key']
    filename = load_image(bucketName, s3Key)
    faces, eyes = process_image(filename, face_cascade, eyes_cascade)
    end_time = time.time()
    imageName = s3Key.split('/')[len(s3Key.split('/'))-1]
    send_results_to_s3(bucketName, imageName, start_time, end_time)
```

A função *Serverless* possui um comportamento levemente diferente da função local. Ao invés de carregar imagens localmente para o processamento, a função recebe a imagem do banco de dados S3 como parâmetro para a execução. A função é ativada através de um gatilho configurado nela. O gatilho é configurado para que, quando uma imagem for enviada para uma pasta específica no banco de dados, nesse caso, images/, a função receba essa imagem como parâmetro e processe. Após o processamento, a função guarda os resultados no S3 da *Amazon*.

Para utilizar a plataforma *Serverless* dessa forma, tudo o que o dispositivo precisa fazer para processar as imagens é fazer o envio delas na pasta configurada anteriormente. O código para essa execução foi desenvolvido em *Javascript* e executado com *NodeJS* na versão 14. O código da função de envio é mostrado no Código 4.

Código 4 – Programa uploadImages.js

```
const fsPromises = require('fs').promises
const s3 = require('aws-sdk').s3()
uploadImages: async (req, res) => {
    let imageQnt = req.body.imageQnt
    let images = await fsPromises.readdir(imagesPath)
    let workingImages = images.slice(0, imageQnt)
    workingImages.forEach(async img => {
        let fileContent = fs.readFileSync(imagesPath + img)
        s3.upload({
            Bucket: process.env.AWS_BUCKET_NAME,
            Key: 'images/' + o,
            Body: fileContent
        })
    })
}
```

3.4 AWS S3

Como foi citado anteriormente, o banco de dados utilizado para essa aplicação foi o serviço S3 da Amazon. Esse serviço é um simples serviço de armazenamento de objetos que permite operações como leitura, edição e exclusão de arquivos. Esse serviço foi escolhido pela funcionalidade do gatilho presente na plataforma *Lambda*, já que ambos os serviços fazem parte do ecossistema da AWS.

O serviço foi utilizado para receber as imagens alvo a serem processadas pela plataforma *serverless*, através da função de gatilho e, também, para armazenar os resultados dos processamentos, tanto do processamento em nuvem quanto o processamento do dispositivo local. Isso foi feito para que, dessa forma, independente da abordagem utilizada ao processar as imagens, os resultados já estejam armazenados no mesmo banco de dados, nesse caso, o S3.

3.5 Coleta de custo energético

Um dos objetivos desse trabalho é analisar a quantidade de energia que um dispositivo usa ao realizar o processamento local contra utilizar a nuvem para o processamento, já que, com a nuvem, o dispositivo não precisa utilizar tantos recursos quanto se o processamento for feito localmente. Nesse caso, para executar as tarefas no *Serverless*, o dispositivo precisa, apenas, enviar as imagens para o banco.

Isso é importante, pois, como já foi explicado, geralmente, dispositivos IoT estão conectados a uma bateria, o que resulta em uma fonte de energia limitada e, ser energeticamente eficiente em sua função pode ser muito importante, dependendo do caso.

Como, nesse estudo, um dispositivo com fonte ligado diretamente na tomada foi utilizado, para medir a quantidade de energia consumida pelo dispositivo, um Arduino Nano conectado a

um módulo INA219 através de uma *protoboard* foi utilizado. O módulo INA219 é um sensor de corrente e tensão que monitora as grandezas elétricas de uma fonte.

Para estabelecer a conexão do Arduino Nano com o módulo INA219 na *protoboard*, *jumpers* foram utilizados. Conectou-se do Arduino Nano para o módulo INA219 respectivamente, os cabos, 5V ao VCC, *Ground* ao *Ground*, A4 ao SDA e A5 ao SCL. O Arduino Nano foi conectado, através de um cabo USB, a um computador e, a partir do *software* Arduino IDE, os dados foram coletados pelo *Serial Monitor*. Um tutorial mais extenso sobre esse processo está disponível em (AMARAL, 2017).

A função executada no Arduino coleta, utilizando o modo contínuo da biblioteca do módulo INA219, a corrente utilizada pelo dispositivo em formato de mA. Assumindo uma tensão constante de 5V para o dispositivo e dividindo a corrente por 1000 para obtermos a medida em A, multiplicamos a corrente pela tensão para obtermos a média de W/h consumida pelo dispositivo no momento da captura. O código dessa função foi desenvolvido utilizando Arduino IDE e é mostrado no Código 5. Vale notar que o Raspberry PI 4 utilizado nesse estudo estava em sua configuração padrão, sem otimizações de consumo de energia.

Para que tenhamos a média do consumo utilizado pelo dispositivo nos cenários de uso da aplicação desenvolvida nesse trabalho, essa função foi executada durante a execução dos cenários de testes da aplicação, ou seja, foi coletado o consumo do dispositivo processando as imagens localmente e, também, o consumo do dispositivo ao enviar as imagens para o banco de dados S3 para acionar a plataforma *serverless*, tanto na rede cabeada quanto na rede *Wireless* obtendo, assim, a média do consumo em W/h do dispositivo nesses cenários.

Código 5 – Programa collectEnergy.cpp

```
#include <Wire.h>
#include <Adafruit_INA219.h>
Adafruit_INA219 ina219;

void setup(void) {
  Serial.begin(115200);
  while (!Serial) {
      delay(1);
  }
  ina219.setCalibration_32V_2A_continuous();
}

void loop(void) {
  float current_mA = 0;
  current_mA = ina219.getCurrent_mA();
  Serial.println(current_mA);
  delay(68);
}
```

4
Resultados

Nessa seção apresenta-se o cenário de testes que foram realizados para a coleta das métricas a serem analisadas, os resultados provenientes desses testes e uma discussão sobre os resultados.

4.1 Cenário de testes

Nessa seção, descreve-se como os cenários de teste desse estudo foram estruturados e alguns detalhes de como a aplicação foi executada.

Para obter-se as métricas de tempo de execução, a aplicação foi executada 10 vezes, para quantidades de 1, 100, 500 e 999 imagens. As imagens foram retiradas de um banco de dados de faces disponível no repositório (NVLABS, 2018), e todas possuem resolução 1024x1024 com apenas 1 face presente em cada uma delas. Como o processamento da aplicação utilizada nesse trabalho é uma operação pixel a pixel, o número de faces presentes na imagem não deve impactar significativamente seu desempenho.

No dispositivo, a aplicação foi executada com envio de resultados único por imagem, ou seja, cada imagem gerou um arquivo de saída e, também, foi executado com envio de resultados em apenas um arquivo, ou seja, as imagens processadas geraram apenas um arquivo de saída. Isso foi feito pois, devido à plataforma *Serverless* executar apenas uma imagem por vez quando um gatilho é acionado, os resultados desse processamento são, necessariamente, enviados um por vez. Enquanto no dispositivo local tempos a opção de guardar todos os resultados em apenas um arquivo e enviá-lo, para termos de comparação e análise do quanto a comunicação de rede impacta no cenário dessa aplicação, é interessante que também tenhamos um cenário que se equipare a plataforma *Serverless*, ou seja, um cenário que, cada imagem processada gere um arquivo de saída.

Para a plataforma *Serverless*, as imagens foram enviadas para o banco de dados da

Amazon, o S3, e, a partir do gatilho da função, as imagens foram processadas e tiveram seus resultados enviados para o mesmo banco de dados. Como o tempo de envio de resultado não impacta o dispositivo, já que essa conexão é feita dentro dos servidores da própria Amazon, esse tempo foi desconsiderado para os objetivos desse trabalho, contando, apenas, o tempo que o container levou para executar o processamento da imagem. Nesse cenário, também foi obtido o tempo levado para o dispositivo enviar as imagens para o S3.

Para conexão da rede, o Raspberry PI estava conectado na rede Wireless "eduroam"e na rede cabeada da Universidade Estadual do Oeste do Paraná, onde esses casos foram executados. Vale notar que, como o processamento da plataforma *Serverless* é extremamente rápido e, que, possui escalabilidade horizontal e vertical, comparando com um processamento geralmente limitado de um dispositivo IoT, a maior limitante de desempenho dessa abordagem será o tempo de envio das imagens, e não, a capacidade de processamento da plataforma *Serverless*. É importante ressaltar também que, apesar da aplicação ser sequencial, a biblioteca *OpenCV* consegue fazer uso de todas as 4 threads do dispositivo utilizado nesse estudo, ou seja, todo o poder de processamento do dispositivo foi utilizado nos experimentos.

4.2 Resultados

Nessa seção, serão apresentados os resultados provenientes dos testes, os quais contam com todas as métricas propostas na metodologia do trabalho como tempo de execução da aplicação local e na plataforma *serverless*, custos energéticos do dispositivo para executar os cenários propostos e o tempo de comunicação de rede.

4.2.1 Tempo de execução

Como citado anteriormente, a aplicação desenvolvida nesse trabalho foi executada 10 vezes localmente, para obter as métricas do dispositivo, e 10 vezes acionando a plataforma *Serverless* para as redes *Wireless* e cabeada, para obter as métricas do processamento em nuvem. Todos esses casos foram executados para 1, 100, 500 e 999 imagens.

A seguir, serão apresentados os gráficos que representam o tempo de execução da aplicação para cada caso e para cada quantidade de imagem executados nesse experimento. Os gráficos possuem tempo de processamento, representado em azul, e tempo de execução total, representado pela soma da barra azul com a barra vermelha.

Ao processar no dispositivo, temos a execução com envio de resultados único, ou seja, cada imagem gerando um arquivo resultado e, com envio de resultados em lote, ou seja, as imagens processadas geram apenas um arquivo de resultado. Ao invocar a plataforma *serverless*, temos o caso com envio de imagens através da rede *Wireless* e através da rede cabeada.

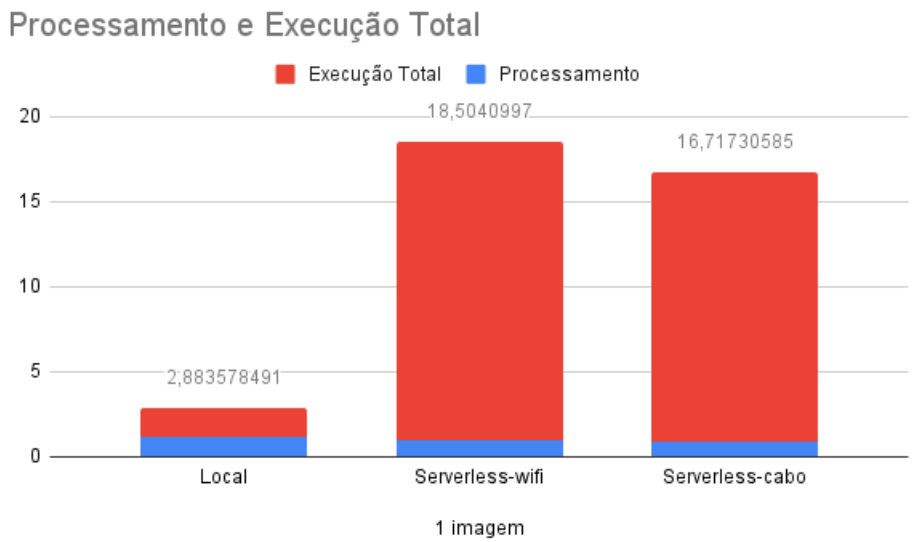
O tempo de processamento demonstra apenas o tempo que levou para as imagens serem

processadas. Isso foi coletado, no dispositivo, através da coleta do *timestamp* antes e depois de o processamento ser feito. Como, na plataforma *serverless*, não temos acesso a essa informação de forma direta devido à função ser executada através de um gatilho, cada imagem gera um arquivo de saída com o *timestamp* de início e fim do processamento dela para termos acesso a essa informação. Com os arquivos gerados, um pós-processamento é necessário para analisar o menor e o maior *timestamp* de todas as imagens de cada execução, coletando, assim, o tempo de processamento da execução.

Já o tempo de execução total, ao processar no dispositivo, significa o tempo de processamento somado ao tempo que levou para que os resultados fossem enviados ao banco de dados. Esse tempo é desconsiderado na plataforma *serverless*, pela conexão ser feita de forma interna através de seus servidores e não impactar o dispositivo IoT. O tempo de execução total na plataforma *serverless* significa a diferença do *timestamp* do ínicio de envio das imagens para o processamento diminuído do *timestamp* em que a última imagem foi processada, ou seja, diminuímos o horário em que o último processamento ocorreu do horário de ínicio do envio das imagens obtendo, assim, o tempo total de execução na plataforma.

O tempo de processamento para uma imagem é mostrado na Figura 3. É possível observar nele que o processamento local para a execução de uma imagem isolada é extremamente melhor feito no dispositivo local, devido a, dessa forma, não ser possível utilizar de forma alguma a escalabilidade da plataforma *serverless* além do tempo de início de um *container* da plataforma.

Figura 3 – Tempo de processamento local e *serverless* para 1 imagem

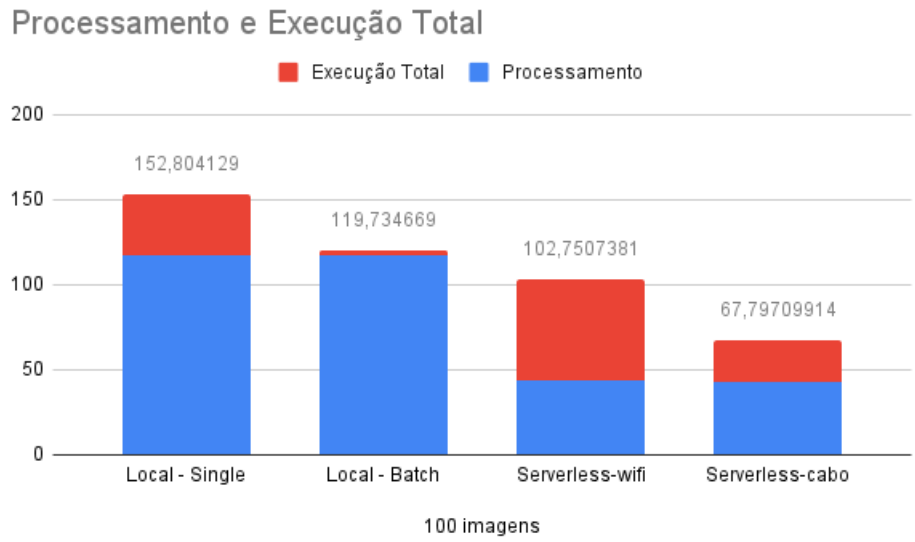


O gráfico de processamento para 100 imagens é mostrado na Figura 4. Nesse caso, podemos ver que o desempenho do processamento local é parecido com o desempenho do processamento *serverless* ao utilizar a rede *Wireless*. Percebemos também que, o envio de resultados em arquivos

únicos impacta significativamente o tempo total de execução do processamento local, quando comparado ao envio de resultados em um único arquivo.

Ainda sobre o processamento de 100 imagens, é possível observar que o processamento em nuvem para a rede cabeada leva muito menos tempo para ser executado. Isso se deve a estabilidade e velocidade de transmissão superior da rede cabeada disponível na universidade em que os experimentos foram executados e, também, devido ao fato do processamento *serverless* ser limitado, nesse caso, a velocidade com que as imagens são enviadas para o banco de dados S3. Esse resultado, porém, é esperado, já que a rede cabeada é consideravelmente mais estável e capaz em termos de transmissão de dados quando comparado a uma rede *Wireless*.

Figura 4 – Tempo de processamento local e *serverless* para 100 imagens



Na Figura 5 e Figura 6 temos os gráficos do processamento de 500 e 999 imagens, respectivamente. Podemos ver comportamentos similares nesses 2 gráficos quando comparamos os cenários de testes executados nesse experimento. Em ambos os gráficos, o processamento *serverless* na rede cabeada foi o melhor, chegando a ser mais que 3 vezes mais rápido que o processamento *serverless* na rede *Wireless* e quase 6 vezes mais rápido que o processamento local com envio de resultados em lote.

Nesses gráficos, é possível notar que, nos casos de 500 e 999 imagens, o processamento em nuvem é extremamente superior ao processamento local em termos de tempo de processamento e tempo de execução total, tanto na rede *Wireless* quando na rede cabeada, porém, principalmente, na rede cabeada. Isso se deve pela escalabilidade da plataforma *serverless*. Como citado anteriormente, nesse estudo, o maior limitante do processamento *serverless* é a velocidade em que as imagens são enviadas para o banco de dados S3 e, devido a rede cabeada ser mais potente que a rede *Wireless*, essa permite maior escalabilidade da plataforma e, portanto, um tempo total de execução e processamento consideravelmente menor.

Figura 5 – Tempo de processamento local e *serverless* para 500 imagens

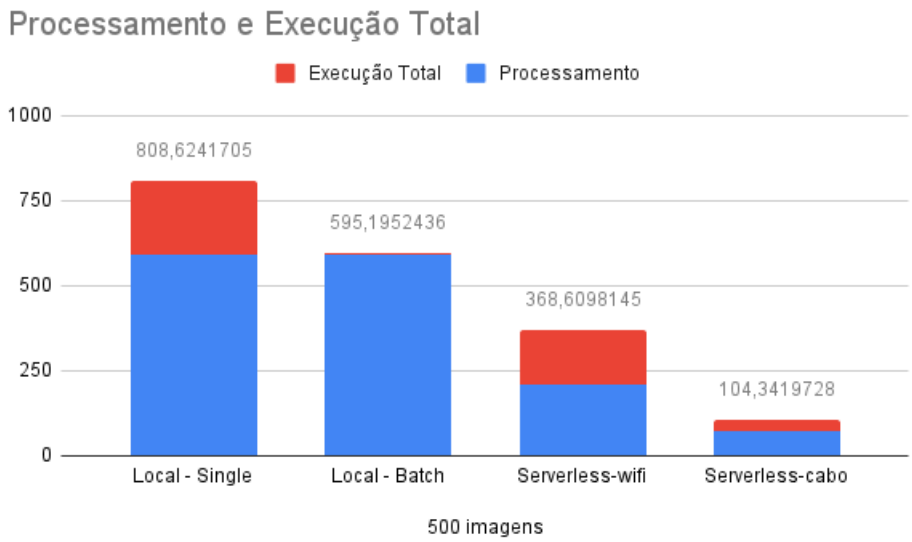
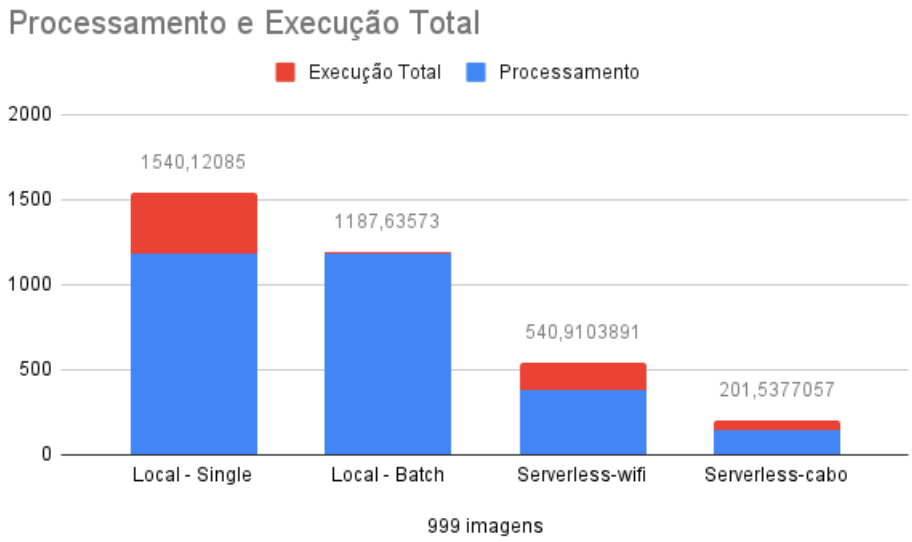


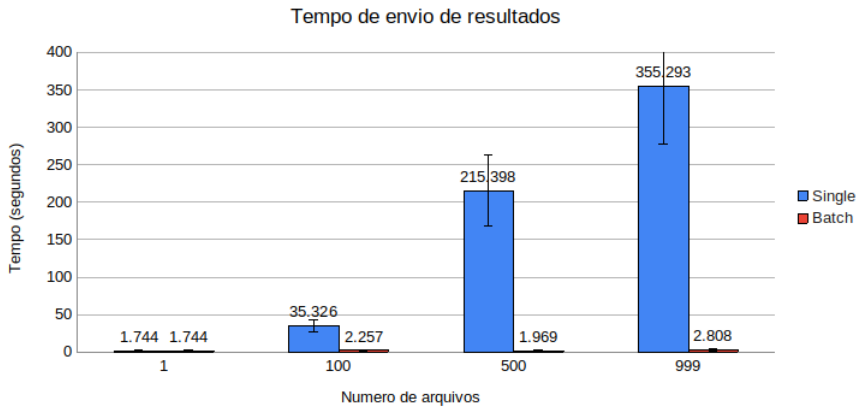
Figura 6 – Tempo de processamento local e *serverless* para 999 imagens



Para verificar melhor como os tempos de envio de resultados impactam no tempo total de execução da aplicação, um gráfico é mostrado na Figura 7. Percebe-se que, o envio de resultados em apenas, um arquivo, mostrado pela barra "Batch", se mantém similar, independente do número de imagens que foi processado, significando que, o tamanho do arquivo de resultados não altera, significativamente, o tempo de envio dele.

Já para resultados em arquivos separados, demonstrado pela barra "Single", o tempo aumenta juntamente com o número de imagens processada, ou seja, o número de arquivos, mesmo que sejam arquivos de tamanho pequeno, aumenta consideravelmente o tempo de envio

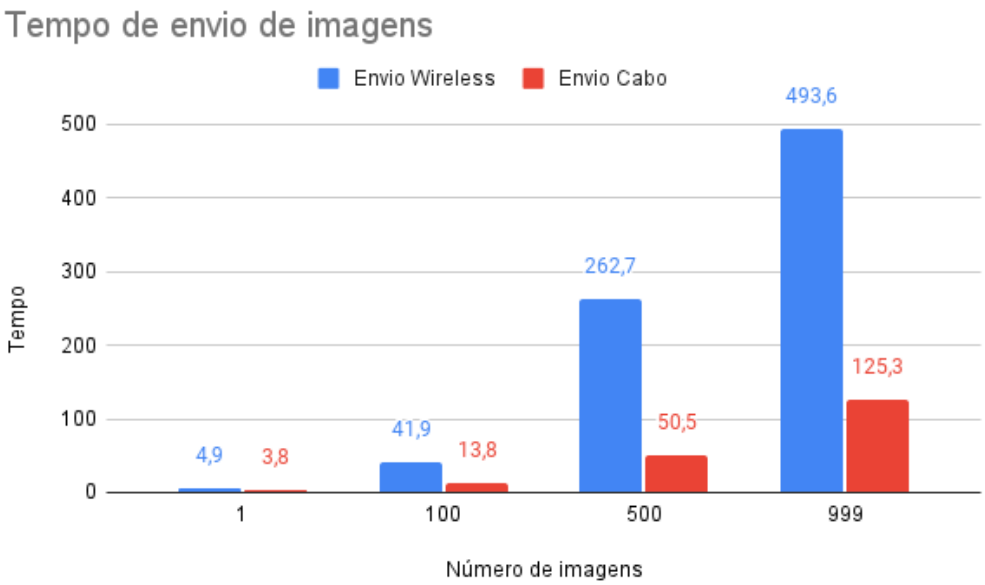
Figura 7 – Tempo de envio dos resultados para o S3



dos mesmos. Portanto, deve-se utilizar quando possível o envio agrupado dos dados.

Para a plataforma *serverless*, o tempo de envio das imagens altera consideravelmente o tempo de execução da aplicação, pois, como já foi citado, considerando que o processamento da tecnologia *serverless* é, virtualmente, ilimitado, o maior limitante dessa abordagem, nessa aplicação, é o tempo que o dispositivo leva para enviar as imagens para o banco de dados S3. Um gráfico com o tempo de envio das imagens é mostrado na Figura 8.

Figura 8 – Tempo de envio das imagens para o S3



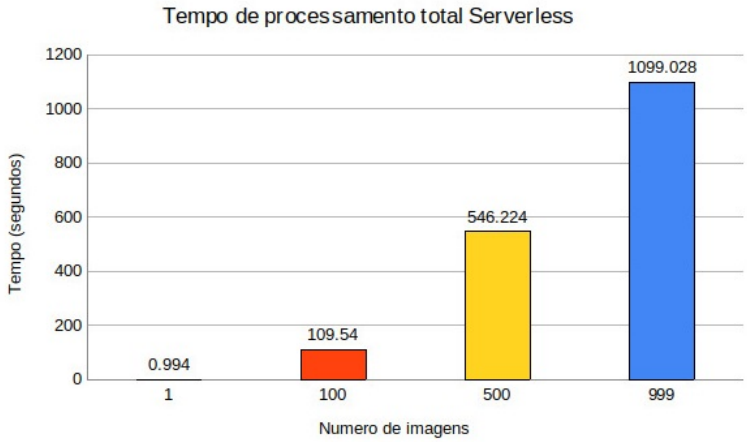
Nesse gráfico, vemos que, ao enviar uma imagem para o banco de dados, o tempo das redes não é muito diferente, sendo maior que o tempo de processamento de uma imagem no dispositivo local, tanto na rede cabeada quanto na rede Wireless, ou seja, para uma imagem

isolada, o processamento em nuvem não vale a pena. A partir de 100 imagens, esse cenário muda e, em vez de, na rede *Wireless*, cada imagem isolada demorar, em média, 4,9 segundos para ser enviada, as imagens passam a demorar 0,45 segundos para serem enviadas. Isso também se repete na rede cabeada onde, para imagens isoladas, tomava-se em média 3,8 segundos, a partir de 100 imagens, cada imagem leva 0,12 segundos para ser enviada. Também é possível ver, nesse gráfico, a superioridade da rede cabeada quando comparado a rede *Wireless* no local onde os experimentos desse estudo foram executados.

É interessante observar que, o tempo de envio das imagens somado ao tempo de processamento *serverless* é menor que o tempo total de execução da abordagem *serverless*. Isso se deve porque não é necessário, para a plataforma *serverless*, que todas as imagens sejam enviadas para começar o processamento. O gatilho funciona de forma tal que, para cada imagem enviada, a função já é acionada para processamento na mesma, ou seja, a partir da primeira imagem enviada, o processamento já é inicializado, o que reduz bastante o tempo total de execução.

A partir dos experimentos, também foi possível coletar o tempo total de execução que a plataforma *serverless* levou para realizar os processamentos. O tempo total de execução da plataforma é uma somatória dos tempos de processamento das imagens e esse tempo representa o quanto demoraria para processar as imagens caso a tecnologia *serverless* não fosse escalável, mas sim, executasse as funções em apenas um container, ou seja, de forma sequencial, similar ao processamento local. Esses tempos são mostrados na Figura 9.

Figura 9 – Tempo total de processamento *serverless*



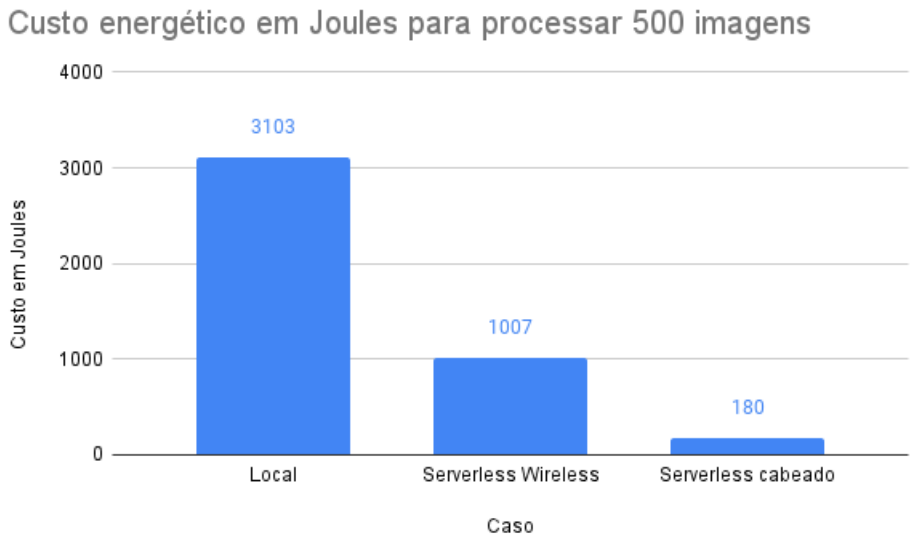
4.2.2 Custos de energia

Para calcularmos a quantidade de energia que o dispositivo gastou nos cenários de processamento e de envio de imagens, uma média da potência do quanto o dispositivo gasta em cada caso, sendo eles, processamento local, envio de imagens através da rede *Wireless* e envio de imagens através da rede cabeada foram calculados. Ao processar localmente, o dispositivo tem

um custo de, em média, 5,234W/h, na rede *Wireless*, um custo de 3,835 W/h e, na rede cabeada, um custo de 3,565W/h.

A partir das médias de potência, esses valoes foram multiplicados pelos tempos, em segundos, do processamento que 500 imagens levaram, ou seja, o custo local foi multiplicado pelo tempo de processamento local, o custo na rede *Wireless* foi multiplicado pelo tempo do envio de 500 imagens pela rede *Wireless* e o custo da rede cabeada foi multiplicado pelo tempo de envio de 500 imagens pela rede cabeada, resultando no valor energético gasto pelo dispositivo em cada caso, em Joules. Esses valores são apresentados na Figura 10.

Figura 10 – Consumo de energia para 500 imagens



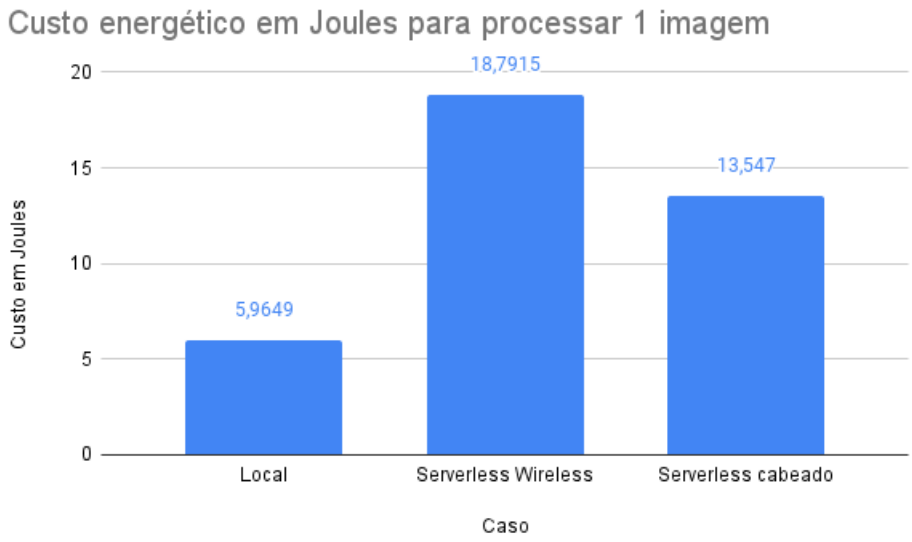
É possível ver pelo gráfico o aumento significativo de energia que o dispositivo utiliza ao realizar o processamento local, devido a ser a estratégia que utiliza o maior número de recursos do dispositivo e, a estratégia que demora mais tempo. Valendo ressaltar que, no processamento em nuvem, a única responsabilidade do dispositivo é o envio das imagens a serem processadas então, por exemplo, para 500 imagens, enquanto o dispositivo precisa processar localmente por quase 600 segundos para executar a tarefa, no caso da rede cabeada, por exemplo, o dispositivo necessita apenas de enviar as imagens, o que leva, em média, 50 segundos para 500 imagens.

Através dos valores, pode-se concluir pelo gráfico que, se o dispositivo IoT estiver conectado a uma bateria e consumo de energia for um problema, a tecnologia *serverless* é uma grande alternativa tanto utilizando uma rede *Wireless* quanto utilizando uma rede cabeada e, nesse caso, principalmente a rede cabeada quando o processamento de várias imagens de uma vez é uma opção.

É importante verificar, também, o consumo de energia o processamento de uma imagem isolada, já que esse foi o pior caso da tecnologia *serverless* quando comparado ao processamento

local. Os valores para a execução de uma imagem são apresentados na Figura 11 e foram adquiridos da mesma maneira que o consumo de 500 imagens, em Joules.

Figura 11 – Consumo de energia para 1 imagem



Podemos ver através desse gráfico que, para processar uma imagem, o envio dela na rede *Wireless* custou quase 3 vezes mais do que o processamento local e, duas vezes mais que o processamento local na rede cabeada, fazendo o processamento no dispositivo, para esse caso, a melhor escolha energeticamente.

4.3 Discussão

Nessa seção, os resultados serão discutidos e analisados mais profundamente para que, a partir deles, se possa responder à questão inicial da pesquisa. A partir dos tempos de processamento demonstrados na Figura 3, é possível ver que, para o caso de uma imagem, o tempo de processamento da tecnologia *serverless* é semelhante ao do dispositivo IoT, porém, o tempo de execução total é muito superior, com um tempo local de 2,854 segundos contra os tempos de 18.504 e 16.717 segundos da tecnologia *serverless* nas redes *Wireless* e cabeada respectivamente, ou seja, um tempo de 5 a 6 vezes maior para ser executado na computação em nuvem. Isso se deve, principalmente, pelo *cold start-up* característico da tecnologia *serverless*.

Ao executar apenas uma imagem, foi possível extrair o tempo em que a plataforma demorou para inicializar um *container*. Esse tempo foi de, em média, 12,58 segundos. Isso é interessante, pois, no caso de uma única imagem, o tempo de inicialização da plataforma *serverless* é consideravelmente maior do que o tempo que o dispositivo leva para fazer o processamento e o envio do resultado, ou seja, caso o interesse for desempenho e as imagens forem executadas em ocasiões separadas, uma a uma, o uso da tecnologia não faria sentido.

Para 100 imagens, mostrado na Figura 4, o desempenho do processamento local, principalmente com envio de apenas um arquivo de resultado, foi competitivo ao tempo de processamento da tecnologia *serverless* ao utilizar a rede *Wireless*, sendo 119 segundos localmente e 102 segundos na computação em nuvem. Porém, podemos ver, nesse caso, que a tecnologia *serverless* utilizando uma rede cabeada levou quase 2 vezes menos tempo que o processamento local. Podemos ver que o tempo de processamento da tecnologia *serverless* nas abordagens foi parecido, porém, o tempo total de execução da rede cabeada foi significativamente menor, tomando 102 segundos contra os 67 segundos da rede *Wireless*, devido à rede cabeada tomar menos tempo para o envio das imagens.

Já para um número grande de imagens, mais especificamente, em 500 e 999 imagens, mostrados na Figura 5 e na Figura 6 o processamento na plataforma *serverless* foi significativamente superior ao processamento do dispositivo IoT, tanto em tempos de processamento somente, quanto no tempo de execução total, em ambas as formas de transmissão.

Em 999 imagens, mesmo no melhor caso do dispositivo IoT, ou seja, enviando os resultados compilados em um só arquivo, o tempo total de execução do *serverless* na rede *Wireless* é de 572 segundos contra os 1187 segundos do dispositivo IoT, levando quase metade do tempo para executar a mesma função. No caso de envio dos resultados em arquivos separados, o tempo total de execução local é de 1540 segundos, um tempo muito maior contra o processamento em nuvem em qualquer uma das redes.

Nos casos de 500 e 999 imagens, também é possível ver que o desempenho da plataforma *serverless* em uma rede com taxa de transmissão de dados alta é extremamente capaz, levando, para processar 500 imagens, apenas 104 segundos contra os 368 segundos do processamento *serverless* na rede *Wireless* e sendo, ainda, quase 6 vezes mais rápido que o melhor caso do processamento local.

Considerando que, para uma imagem, o tempo de processamento local e *serverless* são parecidos, é esperado que o desempenho de um *container* da AWS seja parecido com o desempenho do dispositivo utilizado nesse trabalho. Isso é complementado com o gráfico da Figura 9, que apresenta o tempo total de processamento na plataforma *serverless*, ou seja, é a somatória de quanto tempo cada imagem levou para ser executada. Ao comparar-se os tempos respectivos para cada caso de imagem aos tempos de processamento do dispositivo IoT dos gráficos de processamento, podemos perceber tempos parecidos de execução em todas as quantidades de imagem. O tempo de processamento total da tecnologia *serverless* é levemente melhor levando, em 999 imagens, 1099 segundos para executar, contra os 1184 segundos no dispositivo local, por exemplo.

A partir dessa informação, pode-se perceber a capacidade de escalabilidade da computação *serverless*. Enquanto a somatória de processamento de, por exemplo, 500 imagens, é de 546 segundos, o tempo de processamento corrido para 500 imagens na plataforma foi de menos de 250 segundos para a rede *Wireless*, e menos de 100 segundos para a rede cabeada, ou seja, na

rede cabeada, por exemplo, pode-se afirmar que pelo menos 5 *containers* estavam ativos na execução da tarefa.

Esse fato pode ser mais ou menos significativo, dependendo da velocidade em que o dispositivo envia as imagens para o banco de dados o que depende da velocidade de transmissão de rede disponível, ou seja, o desempenho da plataforma *serverless*, nesse estudo, está diretamente ligado a capacidade de transmissão de dados da rede conectada no dispositivo. Isso significa que, caso uma rede menos capaz fosse utilizada, o processamento local poderia, em teoria, ser mais rápido que o processamento em nuvem, em termos de desempenho.

Vale notar que, apesar do dispositivo adotado nesse trabalho ser um dispositivo relativamente limitado, para comparações no meio de dispositivos IoT, o Raspberry PI utilizado é muito capaz em termos de processamento e, ainda assim, a tecnologia *serverless* superou o dispositivo na maioria dos casos de execução, concluindo a superioridade, em termos de desempenho, desse paradigma de computação em nuvem.

Apesar das possíveis vantagens do processamento em nuvem, um argumento contrário é a necessidade de pagamento da abordagem ao processamento. É possível calcular o custo dessa aplicação ao levar em conta o número de execuções necessárias e o tempo necessário para cada imagem, em média, ser executada. Como, em 999 imagens, o tempo total de processamento do *serverless* foi de 1099 segundos e, notando que a função estava configurada para utilizar 3008 MB de memória, o custo seria de $0,05 dólares. Ao processar uma imagem, o tempo de processamento da plataforma, em média, foi de 0,993 segundos, o que gerou um custo de $0,00005 dólares por imagem. Dependendo da quantidade de imagens a ser processada, pode ser um custo significativo ou um custo desconsiderável, em casos de muitas ou poucas imagens a serem processadas.

Sobre consumo de energia, para uma imagem, o gráfico mostrado na Figura 11 demonstra que o processamento de apenas uma imagem é energeticamente melhor quando a imagem é processada localmente. Isso se deve pelo fato do tempo de processamento de uma imagem isolada ser significativamente menor que o tempo de envio de uma imagem, tanto na rede *Wireless* quanto na rede cabeada.

Porém, esse cenário só ocorre para o caso de uma imagem isolada. A partir de 100 imagens, o tempo de envio dessas para o banco de dados é sempre menor que seu tempo de processamento no dispositivo em ambas as formas de transmissão e, como o dispositivo consume mais energia ao processar do que ao enviar imagens, a plataforma *serverless* será mais eficiente em termos de consumo de energia para todos os outros cenários realizados nesse trabalho. O gráfico do consumo ao realizar a tarefa com 500 imagens é mostrado na Figura 10 e os valores presentes no gráfico deixam isso extremamente claro, pois, para processar 500 imagens localmente, o dispositivo consumiu 3103 Joules e, para a computação em nuvem, 1007 e 180 Joules para as redes *Wireless* e cabeada respectivamente.

A partir disso, percebe-se que, caso o dispositivo sendo utilizado esteja conectado a uma bateria e consumo de energia seja um problema, o processamento *serverless* pode ser uma ótima alternativa em cenários onde as imagens podem ser processadas em lote. Já para o processamento de uma imagem isolada, energeticamente falando, o uso da tecnologia não valeria a pena.

5
Conclusão

O desenvolvimento e a utilização de dispositivos IoT está crescendo consideravelmente nos últimos anos devido ao seu possível impacto na vida de seres humanos em qualquer lugar do mundo, e em diversas áreas como saúde, agricultura, logística, dentre outros.

Junto com a utilização massiva desses dispositivos, desafios em seu desenvolvimento são posados e, com os desafios, soluções devem ser analisadas para minimizar ou, até mesmo, solucionar esses desafios. Esse trabalho teve o objetivo de analisar uma dessas possíveis soluções, a tecnologia *serverless*.

A tecnologia *serverless* é um paradigma de computação em nuvem relativamente novo, porém, muito promissor, que oferece facilidade de utilização e configuração por parte do usuário, computação virtualmente infinita com capacidade de containerização imensa e, com custos reduzidos ao usuário, pela falta de necessidade de infraestrutura ao utilizar a abordagem.

Nesse contexto, esse trabalho teve como objetivo avaliar a possível integração da tecnologia *serverless* em dispositivos IoT, estudando um caso de uso para que métricas fossem analisadas bem como a disponibilidade e, possível, vantagem dessa integração.

A partir dos experimentos realizados nesse trabalho, foi possível ver que a integração dessas duas tecnologias é possível e funciona para ajudar com as limitações que dispositivos IoT possuem. Para casos de desempenho, a tecnologia *serverless*, sem surpresas, foi superior ao dispositivo IoT utilizado. Porém, no caso desse trabalho, o desempenho da tecnologia estava diretamente ligado com a qualidade da conexão de rede em que o dispositivo estava conectado. Isso pode posar um problema, caso conexão de rede disponível no local do dispositivo IoT seja limitada ou, posar uma vantagem, caso a conexão de rede seja estável e capaz ao lidar com envio de arquivos.

Visto que a tecnologia *serverless* é atrelada a conexão de rede do dispositivo e, considerando, também, o tempo de inicialização do processamento da tecnologia desdo acionamento da

função, aplicações sensitivas a tempo, ou seja, que precisam executar em tempos de reação de máquina, provavelmente não são apropriadas para essa abordagem, fazendo o processamento local a melhor escolha.

A tecnologia *serverless* pode ser, também, muito útil em casos onde a energia é um problema dentro de um dispositivo IoT visto que esses, na maioria das vezes, são conectados por uma bateria e tem essa fonte de energia limitada. Foi observado através dos experimentos o quão custoso, em termos de energia, processar uma aplicação localmente pode ser para o dispositivo, fazendo o paradigma *serverless* uma alternativa extremamente eficaz para reduzir o uso de energia de um dispositivo.

Essa integração pode ser, também, situacional, visto que a tecnologia *serverless* não tem manuseio de servidores e pode ser acionada a qualquer hora, ou seja, o dispositivo pode escolher usar a tecnologia em casos específicos onde velocidade de processamento pode ser um problema e, também, quando a energia disponível no dispositivo está baixa, ou seja, dispositivos IoT podem usar a tecnologia apenas como uma alternativa para quando o processamento local não seja a escolha ideal.

Para trabalhos futuros, outras aplicações com diferentes características poderiam ser analisadas nesse contexto de integração bem como outros provedores da computação em nuvem, podendo, assim, obter resultados ainda mais conclusivos sobre a integração de dispositivos IoT com plataformas *serverless*. Poderia ser avaliado, também, testes com dispositivos IoT menos capazes ao qual foi utilizado nesse estudo, verificando o impacto de desempenho e consumo de bateria que isso geraria e, até, testes com outros tipos de conexões de rede como, por exemplo, Lora, e verificar o quanto isso impactaria na integração.

Concluindo, a partir desse trabalho, foi visto que a integração de dispositivos IoT com o paradigma *serverless* é promissor e pode ser uma alternativa dependendo do caso em que o dispositivo está atuando, tanto em casos de desempenho quanto em cenários de uso de energia.

Referências

AGUDELO-SANABRIA, S. D.; JINDAL, A. The ifs and buts of the development approaches for iot applications. 2021. Disponível em: <http://arxiv.org/abs/2101.09796>. Citado 3 vezes nas páginas 11, 12 e 14.

AMARAL, H. *Medindo corrente e tensão com o módulo INA219 - FIlipeFlop*. 2017. Disponível em: <https://www.filipeflop.com/blog/medindo-corrente-e-tensao-modulo-ina219/>. Citado na página 34.

BENEDETTI, P. et al. Experimental analysis of the application of serverless computing to iot platforms. *Sensors (Switzerland)*, MDPI AG, v. 21, p. 1–20, 2 2021. ISSN 14248220. Citado na página 28.

BENOMAR, Z. et al. Deviceless: A serverless approach for the internet of things. In: . [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2021. ISBN 9789261338817. Citado na página 28.

BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. Citado na página 29.

CALDERONI, L.; MAIO, D.; TULLINI, L. Benchmarking cloud providers on serverless iot back-end infrastructures. *IEEE Internet of Things Journal*, p. 1–1, 2022. ISSN 2327-4662. Disponível em: <https://ieeexplore.ieee.org/document/9698101/>. Citado 2 vezes nas páginas 21 e 27.

CARVALHO, P. *Creating a Python OpenCV Layer for AWS Lambda*. 2021. Disponível em: <https://betterprogramming.pub/creating-a-python-opencv-layer-for-aws-lambda-f2d5266d3e5d>. Citado na página 32.

CHARD, R. et al. Serverless supercomputing: High performance function as a service for science. *arXiv*, 2019. ISSN 23318422. Citado 2 vezes nas páginas 23 e 24.

CHAUDHARY, S.; SOMANI, G.; BUYYA, R. Research advances in cloud computing. *Research Advances in Cloud Computing*, p. 1–465, 2017. Citado na página 20.

DOTTA, J. *Face Recognition with HaarCascade*. 2022. Disponível em: <https://github.com/pistydotta/face_recognition_python/blob/main/face_cascade.py>. Citado na página 31.

DOTTA, J. *Serverless Face Recognition with HaarCascade*. 2022. Disponível em: <https://github.com/pistydotta/face_recognition_python/blob/main/face-cascade-serverless.py>. Citado na página 32.

FIROUZI, F. et al. *Intelligent Internet of Things*. [S.l.: s.n.], 2020. ISBN 9783030303662. Citado 2 vezes nas páginas 14 e 15.

FOX, G. C. et al. First international workshop on serverless report from workshop and panel on the status of serverless computing and function-as-a-service. p. 1–22, 2017. Citado na página 20.

GEORGE, G. et al. Nanolambda: Implementing functions as a service at all resource scales for the internet of things. *ACM Symposium on Edge Computing (SEC)*, 2020. Citado na página 27.

GLOUKHOVTSEV, M. Iot security: Challenges, solutions and future prospects. p. 1–44, 2018. Disponível em: <https://education.dellemc.com/content/dam/dell-emc/documents/en-us/2018KS_Gloukhovtsev-IoT_Security_Challenges_Solutions_and_Future_Prospects.pdf>. Citado 2 vezes nas páginas 15 e 17.

HASSAN, H. B.; BARAKAT, S. A.; SARHAN, Q. I. Survey on serverless computing. *Journal of Cloud Computing*, Springer Science and Business Media Deutschland GmbH, v. 10, 12 2021. ISSN 2192113X. Citado 5 vezes nas páginas 19, 20, 22, 23 e 25.

INSTITUTE, M. G. The internet of things: Mapping the value beyond the hype executive summary. *McKinsey and Company*, p. 1–18, 2015. Disponível em: <www.mckinsey.com/mgi.> Citado na página 15.

INTERNET, S. Chapter 1. the internet of things (iot) and libraries. *Library Technology Reports*, v. 53, p. 5–8, 2017. ISSN 0024-2586. Citado na página 17.

KJORVEZIROSKI, V. et al. Iot serverless computing at the edge: Open issues and research direction. *Transactions on Networks and Communications*, Scholar Publishing, v. 9, p. 1–33, 12 2021. Citado na página 28.

KJORVEZIROSKI, V.; FILIPOSKA, S.; TRAJKOVIK, V. Iot serverless computing at the edge: A systematic mapping review. *Computers*, MDPI, v. 10, 10 2021. ISSN 2073431X. Citado na página 27.

LEITNER, P. et al. A mixed-method empirical study of function-as-a-service software development in industrial practice. *Journal of Systems and Software*, Elsevier Inc., v. 149, p. 340–359, 2019. ISSN 01641212. Disponível em: <https://doi.org/10.1016/j.jss.2018.12.013>. Citado 4 vezes nas páginas 20, 21, 22 e 23.

LYNN, T. et al. *The Cloud-to-Thing Continuum Opportunities and Challenges in Cloud, Fog and Edge Computing*. [s.n.], 2020. ISBN 9783030411091. Disponível em: <http://www.palgrave.com/gp/series/16004>. Citado na página 12.

MANYIKA, J.; CHUI, M.; BUGHIN, J. Disruptive technologies: Advances that will transform life, business, and the global economy. *McKinsey Global . . .* , p. 163, 2013. Disponível em: <http://www.mckinsey.com/insights/business_technology/disruptive_technologies%5Cnhttp://www.chrysalixevc.com/pdfs/mckinsey_may2013.pdf>. Citado na página 15.

MARJANI, M. et al. Big iot data analytics: Architecture, opportunities, and open research challenges. *IEEE Access*, v. 5, p. 5247–5261, 2017. ISSN 21693536. Citado 3 vezes nas páginas 11, 15 e 16.

MILENKOVIC, M. *Internet of Things: Concepts and System Design*. [S.l.]: Springer International Publishing, 2020. Citado na página 11.

NVLABS. *NVlabs/ffhq-dataset: Flickr-Faces-HQ Dataset(FFHQ)*. 2018. Disponível em: <https://github.com/NVlabs/ffhq-dataset>. Citado na página 35.

OPENCV. *opencv/objectDetection.cpp*. 2019. Disponível em: <https://github.com/opencv/opencv/blob/3.4/samples/cpp/tutorial_code/objectDetection/objectDetection.cpp>. Citado na página 30.

PINTO, D.; DIAS, J. P.; FERREIRA, H. S. Dynamic allocation of serverless functions in iot environments. *Proceedings - 16th International Conference on Embedded and Ubiquitous Computing, EUC 2018*, p. 1–8, 2018. Citado 2 vezes nas páginas 15 e 27.

RISCO, S. et al. Serverless workflows for containerised applications in the cloud continuum. *Journal of Grid Computing*, Springer Science and Business Media B.V., v. 19, 9 2021. ISSN 15729184. Citado na página 24.

SCHEUNER, J.; LEITNER, P. Function-as-a-service performance evaluation: A multivocal literature review. *Journal of Systems and Software*, v. 170, 2020. ISSN 01641212. Citado na página 24.

TRILLES, S.; GONZALEZ-PEREZ, A.; HUERTA, J. An iot platform based on microservices and serverless paradigms for smart farming purposes. *Sensors (Switzerland)*, v. 20, 2020. ISSN 14248220. Citado 2 vezes nas páginas 15 e 17.

XU, L. D.; HE, W.; LI, S. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, v. 10, p. 2233–2243, 2014. ISSN 15513203. Citado na página 17.