INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS
CAMPUS INHUMAS
BACHARELADO EM SISTEMA DE INFORMAÇÃO

EURIPEDES ANTERO VIEIRA
GABRIEL EDUARDO DE BESSA MACIEL

IMPLEMENTAÇÃO DE SERVERLESS AS A SERVICE NA PLATAFORMA DE MIDDLEWARE DOJOT

INHUMAS
2022

FICHA CATALOGRÁFICA ELABORADA PELA COORDENAÇÃO DE BIBLIOTECA DO SEU CÂMPUS

Dados Internacionais de Catalogação na Publicação

Vieira, Euripedes Antero

V658 Implementação de Serverless as a Service na plataforma de middleware Dojot [Manuscrito]. / Euripedes Antero Vieira, Gabriel Eduardo de Bessa Maciel – Inhumas: IFG, 2022.

43 f.

Bibliografia.

Orientador: Prof. Dr. Leandro Alexandre Freitas.

Trabalho de conclusão de curso de Bacharelado em Sistemas de Informação – IFG/Câmpus Inhumas, 2022.

1. Serveless. 2. Dojot. 3. Cidades inteligentes. 4. Maciel, Gabriel Eduardo de Bessa. 5. Freitas, Leandro Alexandre (orientador). I. Título.

CDD 003.5

Ficha catalográfica elaborada pela bibliotecária
Larissa Stefane Rodrigues de Lima - CRB/1-3424
Instituto Federal de Educação, Ciência e Tecnologia de Goiás
Câmpus Inhumas – Biblioteca Atena

TERMO DE AUTORIZAÇÃO PARA DISPONIBILIZAÇÃO

NO REPOSITÓRIO DIGITAL DO IFG - ReDi IFG

Com base no disposto na Lei Federal nº 9.610/98, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia de Goiás, a disponibilizar gratuitamente o documento no Repositório Digital (ReDi IFG), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, em formato digital para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IFG.

Identificação da Produção Técnico-Científica

[ ] Tese	[ ] Artigo Científico
[ ] Dissertação	[ ] Capítulo de Livro
[ ] Monografia – Especialização	[ ] Livro
[X] TCC – Graduação	[ ] Trabalho Apresentado em Evento

[ ] Produto Técnico e Educacional - Tipo: _____

Nome Completo do Autor:

Matrícula:

Título do Trabalho:

Autorização - Marque uma das opções

1. (X) Autorizo disponibilizar meu trabalho no Repositório Digital do IFG (acesso aberto);
2. ( ) Autorizo disponibilizar meu trabalho no Repositório Digital do IFG somente após a data ___/___/_____ (Embargo);
3. ( ) Não autorizo disponibilizar meu trabalho no Repositório Digital do IFG (acesso restrito).

Ao indicar a opção **2 ou 3**, marque a justificativa:

( ) O documento está sujeito a registro de patente.

( ) O documento pode vir a ser publicado como livro, capítulo de livro ou artigo.

( ) Outra justificativa: _____

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O/A referido/a autor/a declara que:

i. o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;

ii. obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autor/a, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia de Goiás os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;

iii. cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia de Goiás.

Inhumas, 17 / 01 / 2023.

Local Data

gov.br
Documento assinado digitalmente
GABRIEL EDUARDO DE BESSA MACIEL
Data: 17/01/2023 11:42:32-0300
Verifique em https://verificador.iti.br

Assinatura do Autor e/ou Detentor dos Direitos Autorais

INSTITUTO FEDERAL
Goiás

TERMO DE APROVAÇÃO

EURIPEDES ANTERO VIEIRA

GABRIEL EDUARDO DE BESSA MACIEL

IMPLEMENTAÇÃO DE SERVERLESS AS A SERVICE NA PLATAFORMA DE MIDDLEWARE DOJOT

Trabalho de Conclusão de Curso, no formato de artigo científico, apresentado ao curso de Bacharelado em Sistemas de Informação, do Instituto Federal de Educação, Ciência e Tecnologia de Goiás - Campus Inhumas/IFG, como requisito parcial à obtenção do título de Bacharel em Sistemas de Informação, desenvolvido na linha de pesquisa Redes de Computadores e Sistemas Distribuídos, sob orientação do Prof. Leandro Alexandre Freitas.

Aprovado em 23/09/2022.

BANCA EXAMINADORA

Documento assinado digitalmente
gov.br LEANDRO ALEXANDRE FREITAS
Data: 17/01/2023 08:21:07-0300
Verifique em https://verificador.iti.br

Prof. (Leandro Alexandre Freitas)

(Presidente - orientador)

Documento assinado digitalmente
gov.br RODRIGO CANDIDO BORGES
Data: 17/01/2023 09:58:27-0300
Verifique em https://verificador.iti.br

Prof. (Rodrigo Cândido Borges)

Prof. (Rogério Sousa e Silva)

(Membro externo)

EURIPEDES ANTERO VIEIRA
GABRIEL EDUARDO DE BESSA MACIEL

IMPLEMENTAÇÃO DE SERVERLESS AS A SERVICE NA PLATAFORMA DE MIDDLEWARE DOJOT

Trabalho de Conclusão de Curso, no formato de artigo científico, apresentado ao curso de Bacharelado em Sistemas de Informação, do Instituto Federal de Educação, Ciência e Tecnologia de Goiás - Campus Inhumas/IFG, como requisito parcial à obtenção do título de Bacharel em Sistemas de Informação

Orientador: Prof. Dr. Leandro Alexandre Freitas

INHUMAS
2022

AGRADECIMENTOS

Aos nossos familiares, que sempre estiveram ao nosso lado, pela amizade incondicional e pelo apoio demonstrado ao longo de todo o período dedicado a este trabalho. E um agradecimento especial, ao nosso professor, Leandro Alexandre Freitas, por ter sido nosso orientador e ter desempenhado tal função com muito carinho e respeito, sempre nos apoiando e buscando o nosso melhor.

LISTA DE ILUSTRAÇÕES

Figura 1 – A arquitetura em micro serviços da plataforma Dojot ---------- 17

Figura 2 - Exemplo de integração com o IoT *Agent* ---------- 18

Figura 3 - Etapas da metodologia ---------- 24

Figura 4 – Arquitetura *Serverless as a Service (SlaaS)* utilizando *Cloud* e *Edge Computing* ---------- 27

Figura 5 – Inicialização do IoT *Agent* OCPP *Serverless* ---------- 28

Figura 6 – FaaS OCPP *BootNotification* ---------- 29

Figura 7 - Versão do Serviço Implementada ---------- 31

Figura 8 - Modelo Incremental ---------- 33

Figura 9 - Modelo de reuso de componentes ---------- 33

SUMÁRIO

1 **INTRODUÇÃO ---------- 12**

1.1 MOTIVAÇÃO ---------- 12

1.2 JUSTIFICATIVA ---------- 13

1.3 OBJETIVOS ---------- 12

1.3.1 Objetivos Gerais ---------- 14

1.3.2 Objetivos Específicos ---------- 14

1.4 ORGANIZAÇÃO DA MONOGRAFIA ---------- 14

2 **REFERENCIAIS TEÓRICOS ---------- 15**

2.1 INTERNET DAS COISAS E CIDADES INTELIGENTES ---------- 15

2.2 DOJOT ---------- 16

2.2.1 IoT Agents---------- 17

2.3 MOBILIDADE ELÉTRICA ---------- 18

2.3.1 Open Charge Point Protocol ---------- 19

2.4 SERVERLESS E FUNCTIONS AS A SERVICE ---------- 20

2.4.1 Frameworks Serverless ---------- 21

2.4.1.1 Knative ---------- 21

2.4.1.2 Kubeless ---------- 22

2.4.1.3 Serverless Framework ---------- 22

2.5 TRABALHOS RELACIONADOS ---------- 22

3 **METODOLOGIA ---------- 24**

3.1 ESTUDO DA PLATAFORMA DOJOT E DO PARADIGMA SERVERLESS -- 24

3.2 PROPOSTA DA ARQUITETURA E INTEGRAÇÃO DO COMPONENTE SLAAS ---------- 25

3.3 DESENVOLVIMENTO DAS FAAS NA ARQUITETURA PROJETADA ------ 25

3.4 ANÁLISE DAS VANTAGENS E DESVANTAGENS DO COMPONENTE EM COMPARAÇÃO COM A DOJOT STANDARD ---------- 25

4 **SERVERLESS AS A SERVICE ---------- 26**

4.1 VISÃO GERAL DA ARQUITETURA ---------- 26

4.2 DETALHES DA IMPLEMENTAÇÃO ---------- 28

4.3 COMPORTAMENTO DA VERSÃO IMPLEMENTADA NO CONTEXTO DA MOBILIDADE ELÉTRICA PARA A DOJOT ---------- 29

5 **DESENVOLVIMENTO ---------- 32**

5.1 PROCESSO DE DESENVOLVIMENTO ADOTADO ---------- 32

5.2 FERRAMENTAS UTILIZADAS ---------- 34

5.2.1 Git ---------- 34

5.2.2 Visual Studio Code ---------- 34

5.2.3 AWS Lambda ---------- 34

5.2.4 Serverless Framework ---------- 35

5.2.5 NodeJS ---------- 35

5.3 FAAS DESENVOLVIDAS ---------- 35

6 **CONSIDERAÇÕES FINAIS ---------- 37**

6.1 VANTAGENS ---------- 37

6.2 DESVANTAGENS ---------- 38

7 **CONCLUSÃO ---------- 39**

8 REFERÊNCIAS BIBLIOGRÁFICAS ---------- 40

Implementação de Serverless as a Service na plataforma de middleware Dojot

Euripedes Antero Vieira
Gabriel Eduardo De Bessa Maciel

RESUMO

Este trabalho resulta de uma pesquisa sobre a aplicação do paradigma de programação serverless na plataforma para cidades inteligentes Dojot. O paradigma serverless é usado para implementar serviços em nuvem por meio de Funções como Serviço (FaaS), retratando o avanço dos paradigmas de desenvolvimento e a maturidade das soluções em nuvem, tendo uma arquitetura que, permite o desenvolvedor direcionar a responsabilidade do provisionamento, monitoramento e escalabilidade da aplicação para o provedor de nuvem. A plataforma Dojot, desenvolvida pelo Centro de Pesquisa e Desenvolvimento em Telecomunicações (CPqD), visa modernizar as plataformas para cidades inteligentes e fornecer soluções para a Internet das Coisas (IoT). Dessa forma, propomos o desenvolvimento de um componente que permita gerenciar FaaS de diferentes serviços em plataformas para cidades inteligentes distintas e para validação, desenvolvemos uma versão *serverless* para o serviço OCPP no âmbito do problema de mobilidade elétrica já implementado na Dojot, para analisar as vantagens e desvantagens da utilização do paradigma serverless na plataforma Dojot.

Palavras-chave: Serverless. FaaS. Dojot. Cidades Inteligentes. OCPP.

ABSTRACT

This text results from research on the application of the serverless programming paradigm in the Dojot Smart Cities platform. The serverless paradigm is used to implement cloud services through Functions as a Service (FaaS), portraying the advancement of development paradigms and the maturity of cloud solutions, having an architecture that allows the developer to direct the responsibility for provisioning, monitoring and application scalability for the cloud provider. The Dojot platform, developed under the Telecommunications Research and Development Center (CPqD), aims to modernize platforms for smart cities and provide solutions for the Internet of Things (IoT). In this way, we propose the development of a component that allows managing the FaaS of different services on platforms for different smart cities and for validation we develop a serverless version for the IoT Agent OCPP within the scope of the electric mobility problem already implemented in dojot, to analyze the advantages and disadvantages of using the serverless paradigm within the Dojot platform.

Keywords: Serverless. FaaS. Dojot. Smart Cities. OCPP.

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Analistas preveem que o número de dispositivos para Internet das Coisas (*Internet of Things - IoT*) pode crescer para um número impressionante de cerca de 100 bilhões de dispositivos(AFOLABI, 2018, p. 1). Estes dispositivos suportam uma ampla gama de serviços que vão desde aplicações de medição baseados em sensores de baixo custo e de veículos tolerantes a atrasos até comunicações críticas, incluindo saúde, negócios e o setor automotivo (AFOLABI, 2018, p. 1). Além disso, esperava-se que, através da construção de plataformas IoT, softwares de aplicativos e ofertas relacionadas a serviços, o mercado atingiria US$1,7 trilhões em 2020 (MEHMOOD, 2017, p. 16). A IoT ganhará força com as Cidades Inteligentes, seja pela integração de dispositivos e sensores, pela convergência das tecnologias, ou ainda pelas novas possibilidades advindas das novas gerações de redes móveis de comunicação. Uma cidade inteligente é um ecossistema complexo caracterizado pelo uso intensivo de tecnologias de informação e comunicação (TIC), com o objetivo de tornar as cidades mais atrativas, sustentáveis, e um lugar único para inovação e empreendedorismo (MEHMOOD, 2017, p. 16).

Para que o conceito de cidades inteligentes sejam aplicados, diversas plataformas foram desenvolvidas para auxiliar na obtenção desses objetivos. A Dojot é uma plataforma que tem o objetivo de desenvolver e demonstrar tecnologias para as cidades inteligentes. Inicialmente com o foco nos pilares de segurança pública, mobilidade urbana e saúde, com o intuito de construir um ecossistema multidisciplinar nessas áreas (DOJOT, 2022)

Nos últimos anos, as mudanças que aconteceram no viés do desenvolvimento de aplicações, fez com que os proponentes de soluções dessem mais atenção para aplicações que minimizam ações, além das de desenvolvimento. As FaaS retratam muito bem essa política, em que o desenvolvedor precisa se preocupar somente em implementar de fato a solução. Todos os passos seguintes de *deployment* e alocação de recursos é realizado pelos Gerenciadores de Infraestrutura Virtualizada (*Virtualized Infrastructure Managers* - VIMs).

Portanto, este trabalho tem como motivação estipular, determinar e mostrar a importância do paradigma *serverless* para a construção de um componente para plataformas de cidades inteligentes. Este componente proporciona novas abordagens para estimular o desenvolvimento de aplicações, contribuindo com aplicações que beneficiam a sociedade mostrando a importância da adoção de plataformas para o desenvolvimento de cidades inteligentes.

1.2 JUSTIFICATIVA

Dojot é uma plataforma baseada em tecnologias do estado da arte e adota uma arquitetura de micro serviços, possibilitando uma implementação funcional customizada e escalável conforme a necessidade, bem como uma rápida integração com outros sistemas baseados em serviços (DOJOT DOCUMENTATION, 2022). Um dos componentes da Dojot é o *FlowBrocker*. Este componente utiliza o conceito de programação baseada em fluxos, usando uma ampla gama de nós que podem ser implementados em tempo de execução. Esses nós representam comandos, condições, objetivos e variáveis pré estabelecidas que fazem parte da construção da aplicação. Todavia, tem como desvantagem a baixa programabilidade quando se trata de aplicações com maior complexidade, pois, os nós são pré-definidos. Neste sentido, a utilização dos blocos se limita àqueles já implementados e, portanto, se houver uma demanda específica ou alguma atividade que o serviço não atenda, a complexidade da atividade aumenta (DOJOT DOCUMENTATION, 2022).

Esses nós pré-estabelecidos tornam mais enigmáticas a criação de fluxos de maior complexidade, limitando o desenvolvimento e a produtividade, afinal, só é possível utilizar os nós já implementados (DOJOT DOCUMENTATION, 2022). Diante desses cenários torna-se notório a necessidade de utilizarmos um novo paradigma para o desenvolvimento dos serviços da plataforma.

O paradigma *Serverless,* representa uma evolução dos modelos, abstrações e plataformas de programação em nuvem e é um exemplo de maturidade e ampla adoção de tecnologias de nuvem, onde o desenvolvedor não se preocupa com os aspectos operacionais de implementação e manutenção deste código e espera que seja tolerante a falhas, com escalonamento automático (BALDINI, 2017, p. 9). À frente desse cenário, o paradigma *Serverless* disponibiliza uma nova abordagem de

programação proporcionando aos desenvolvedores maior flexibilidade na criação de eventos para a plataforma. Isso ocorre pois a execução das atividades ocorre de forma mais dinâmica e escalável. Além disso, implementa recursos que estão em constante evolução no mercado e no meio científico.

Assim sendo, utilizaremos o paradigma *Serverless* para o desenvolvimento do serviço que denominados por: *Serverless as a Service* (SlaaS), utilizando padrões de projetos voltados para o desenvolvimento de FaaS. Este serviço irá implementar o protocolo *Open Charge Point Protocol* - OCPP, chamado IoT *Agent* OCPP. A partir desta implementação analisaremos se haverá uma melhoria considerável na programabilidade e usabilidade do serviço.

1.3 OBJETIVOS

1.3.1 Objetivos Gerais

O objetivo geral deste trabalho é projetar e implementar o *Serverless as a Service* na plataforma de *middleware* Dojot.

1.3.2 Objetivos Específicos

1. Projetar a arquitetura do IoT *Agent Serverless* OCPP;
2. Implementar o componente IoT *Agent Serverless* OCPP;
3. Comparar as versões IoT *Agent* OCPP da Dojot *Standard* e IoT *Agent* OCPP *Serverless*;
4. Apresentar as vantagens e desvantagens da versão IoT *Agent Serverless*.

1.4 ORGANIZAÇÃO DA MONOGRAFIA

Este trabalho está organizada da seguinte forma: no Capítulo 2 são apresentados os Referenciais Teóricos; no Capítulo 3 é apresentada a Metodologia empregada; O Capítulo 4 apresenta a arquitetura do componente; E no Capítulo 5, é apresentado os modelos de desenvolvimento empregados e as ferramentas utilizadas; O capítulo 6, apresenta as considerações finais acerca das vantagens e

desvantagens da solução; No Capítulo 7 é apresentada a Conclusão; Por fim, no Capítulo 8, as Referências Bibliográficas.

2 REFERENCIAIS TEÓRICOS

O capítulo está organizado da seguinte forma: A Seção 2.1 apresenta os referenciais teóricos de IoT e cidades inteligentes, conceituando os termos e descrevendo suas possibilidades. A Seção 2.2 descreve a plataforma dojot e seus componentes, a partir da Figura 1. A Seção 2.3 apresenta os referenciais teóricos da mobilidade elétrica, e a Seção 2.3.1 descreve o OCPP e suas principais funções. Na Seção 2.4, apresentamos as referências do paradigmas serverless, e na Seção 2.4.1 descrevemos os principais frameworks para o paradigma *serverless* que estão disponíveis. E por fim, na Seção 2.5 apresentamos os trabalhos relacionados às plataformas de cidades inteligentes e suas principais características.

2.1 INTERNET DAS COISAS E CIDADES INTELIGENTES

O termo IoT foi apresentado pela primeira vez em 1990, após o advento das técnicas baseadas na internet da época (ASHTON, 2009, p. 97). Segundo Alavi (2018, p. 2): “a IoT pode ser definida como uma infraestrutura global que permite serviços interconectando coisas físicas e virtuais usando informações interoperáveis e tecnologias de comunicação (TIC)”.

No ano de 2014, Zanella (2014, p. 22), apresentou um conjunto de desafios e possibilidades para o futuro dos dispositivos e serviços no cotidiano dos indivíduos. Dentre esses desafios e possibilidades, o artigo destaca o quão imersivo e abrangente a internet pode se tornar com o advento da IoT. Afinal, com a variedade de “coisas” como, por exemplo, lâmpadas inteligentes, sensores de monitoramento e veículos, a IoT permitirá o desenvolvimento de uma série de aplicações. Possibilitando novos serviços tanto para os cidadãos quanto para as empresas. Além dos diferentes domínios em que podemos encontrar aplicações como gerenciamento de energia e redes inteligentes, automação residencial e industrial (ZANELLA, 2014, p. 22).

Além disso, mesmo com as dificuldades adquiridas por conta da sua

novidade e complexidade, a aplicação do paradigma IoT a um contexto social é de interesse de muitos governos (ZANELLA, 2014, p. 22) . Afinal, há diversos impulsos para que sejam aplicadas as soluções de TIC na gestão de assuntos públicos, surgindo assim o termo de Cidades Inteligentes. Apesar disso, ainda havia um desconhecimento acerca de uma definição formal e amplamente aceita sobre o Cidades Inteligentes, mas também sobre os custos de implementação e as reais possibilidades de utilização.

Mais tarde, em 2018, (ALAVI, 2018, p. 2), apresentou uma relação entre a IoT e as Cidades Inteligentes, onde as cidades inteligentes são orientadas para o usuário e a IoT é orientada para tecnologia. Sendo as cidades inteligentes um dos principais impulsionadores para o desenvolvimento de aplicações IoT e de acordo com a perspectiva de cada um, as cidades inteligentes podem ser definidas de diferentes formas. Alavi (2018, p. 2) cita que uma das definições mais assertivas seria: “A infraestrutura física, a infraestrutura de tecnologia de informação, a infraestrutura social e a infraestrutura de negócios para alavancar a inteligência coletiva da cidade”.

2.2 DOJOT

Como dito na seção anterior, as cidades inteligentes não só potencializam a IoT como também auxiliam no desenvolvimento coletivo da sociedade. Isto é, a partir da necessidade da sociedade podemos desenvolver serviços que solucionam diversos problemas do coletivo.

A Dojot foi projetada com o objetivo de desenvolver e demonstrar tecnologias para as cidades inteligentes (CPQD, 2022). A plataforma foi pensada para atender inicialmente os pilares de segurança pública, mobilidade urbana e saúde, com o intuito de construir um produto multidisciplinar nessas áreas (CPQD, 2022). Com código aberto, está direcionado para ser qualificado e prover o desenvolvimento das soluções IoT (CPQD, 2022).

Para isso a plataforma Dojot foi desenvolvida com base na arquitetura de micro serviços, onde cada componente representa uma determinada função no ecossistema como um todo. Na Figura 1 é apresentada a arquitetura.

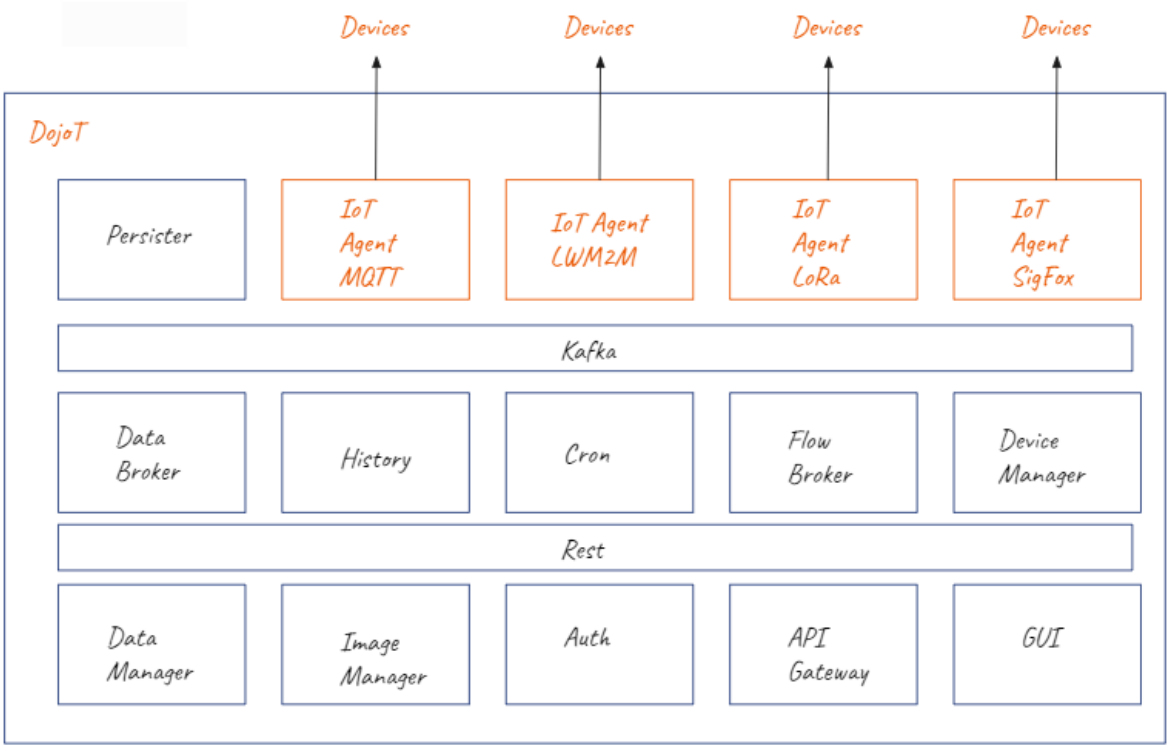

Figura 1. A arquitetura em micro serviços da plataforma Dojot

Fonte: (DOJOT DOCUMENTATION, 2022)

2.2.1 IoT *Agents*

O IoT *Agent* possui um papel muito importante dentro da plataforma, tendo como objetivo principal, publicar dados de um determinado dispositivo para outros serviços da Dojot. Esses dispositivos são aqueles que enviam dados para a Dojot, podendo ter sensores e também podem ser configuráveis. Outrossim, é necessário algum tipo de conectividade com outras aplicações para que possam enviar suas leituras para esses serviços (DOJOT DOCUMENTATION, 2022).

Ademais, caso um determinado protocolo de comunicação não esteja desenvolvido, o componente possui uma base que permite o desenvolvimento de qualquer *agent* (DOJOT DOCUMENTATION, 2022). Possibilitando assim, a integração de uma gama de dispositivos e serviços.

Na Figura 2, apresentamos um exemplo de integração do IoT *Agent* com o serviço de carregamento veicular. O Veículo Elétrico (*Electric Vehicle* - EV) através do Ponto de Carregamento requisita a recarga do veículo. O IoT *Agent,* que implementa o OCPP, obtém essa requisição e valida os dados enviados. Com a

validação, o IoT *Agent* retorna para o Ponto de Carregamento a resposta da requisição feita pelo EV.

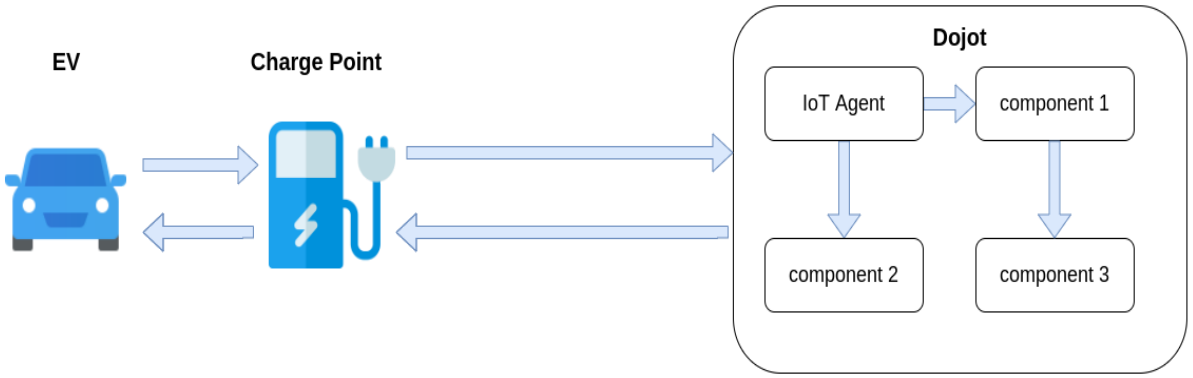

Figura 2. Exemplo de integração com o IoT *Agent*

Fonte: Próprio Autor.

2.3 MOBILIDADE ELÉTRICA

Neste tempo, é inegável a movimentação tanto governamental quanto da indústria para que os veículos no futuro sejam elétricos (NOVAIS, 2016, p. 5). As razões, nós já estamos vivenciando: emissões vertiginosas de Dióxido de Carbono (CO2), que provoca o aquecimento global e diversos problemas relacionados à poluição (NOVAIS, 2016, p. 5). As recentes crises hídricas como a seca histórica na China em 2022, e no Brasil nos últimos anos, nos apresentam os efeitos que a poluição pode causar (NOVAIS, 2016, p. 5).

Novais (2016, p. 5) cita que: “Ações drásticas, principalmente no meio do transporte precisam ocorrer para que a emissão de CO2 e a temperatura média do planeta não sofra tanto impacto nos próximos anos”. A área de transporte é responsável por 23% da emissão de todo o planeta, com tendência a chegar em 50% se nada for feito (NOVAIS, 2016, P. 5).

Ao contrário do que se pensa, os *Electric Vehicles* (EVs) não surgiram recentemente. No século XIX, em meados de 1859, foi criada a bateria de chumbo-ácido. E em 1901, o inventor da lâmpada, Thomas Edison, desenvolveu a bateria de níquel-ferro, com 40% a mais de densidade energética que as de chumbo-ácido (NOVAIS, 2016, p. 6). No entanto, para a época a evolução dos veículos a combustão se mostravam bem menos complexas e caras.

Outrossim, Novais (2016, p. 6) aponta que, mesmo com a o mercado de

veículos a combustão já consolidado e com a indústria do petróleo em seu máximo, acredita-se que nesse contexto não haverá recuos, pois a conjuntura atual é favorável, e mesmo que um produto "desapareça", não impedirá o surgimento de outro com as novas demandas para o mercado. Como a produção de baterias e motores elétricos.

2.3.1 Open Charge Point Protocol

Um dos fatores principais que afetam a popularização dos EVs é o carregamento dos veículos em uma tomada residencial padrão. No entanto, essa praticidade possui um limitador pois, o carregamento em tomadas residenciais pode levar mais de 8 horas. No entanto, em *Charge Points* (CP) específicos, esse carregamento pode ser bem mais rápido, em torno de 40 minutos. Esses CPs são sistemas que realizam o carregamento dos EVs (MINGMING, 2021, p. 783). E para que ocorra esse carregamento, o OCPP padroniza as operações que ocorreram para o carregamento elétrico. Isto é, especifica as operações entre um *CP* e uma *Central System* (CS) que tem como objetivo acomodar qualquer tipo de técnica de cobrança e reduzir a integração geral e os custos de investimento para desenvolvimento e operação dos CPs (MINGMING, 2021, p. 783).

Mingming (2021, p. 781) descreve as principais vantagens do OCPP:

> “(1) oferece um bom suporte aos proprietários de estações de carregamento para usar uma abordagem flexível para selecionar e alternar seus operadores de rede, o que melhora a utilização das estações de carregamento e evita sua ociosidade; (2) sua função básica é fornecer comunicação entre estações de carga e provedores de serviços de rede, mas também pode fornecer serviços à rede; (3) permite a disponibilização de uma interface unificada de informação nas estações de carregamento e serviços de tarifas de *roaming* consistentes, que podem servir diretamente para incentivar os utilizadores a adquirirem veículos elétricos.”

Quando ocorre uma interação entre CP e CS há diversas operações que podem ser iniciadas tanto pela estação de carregamento quanto pelo sistema central (OCPP 1.6, 2015). Sendo que cada operação representa uma determinada atividade

no contexto do carregamento dos veículos elétricos e no gerenciamento do sistema central (OCPP 1.6, 2015). A versão implementada na Dojot implementa as seguintes requisições: *Heartbeat*, *Boot Notification, Status Notification, Start Transaction* e *Stop Transaction*.

A documentação oficial (OCPP 1.6, 2015) define essas operações como: *Heartbeat* a função que informa ao Sistema Central que um ponto de carga ainda está conectado, um ponto de carga envia um sinal após um intervalo de tempo configurável (OCPP 1.6, 2015, p. 37). *Boot Notification* é a requisição enviada pelo Ponto de Carregamento para o Sistema Central informando seus dados (versão, fornecedor e etc) (OCPP 1.6, 2015, p. 34). *Status Notification* é a requisição enviada pelo Ponto de Carregamento para o Sistema Central informando mudanças e erros do Ponto de Carregamento para o Sistema Central (OCPP 1.6, 2015, p. 40). *Start Transaction* é a requisição enviada pelo Ponto de Carregamento informando ao Sistema Central que um carregamento foi iniciado (OCPP 1.6, 2015, p. 39). E, por fim, *Stop Transaction* é a requisição enviada pelo Ponto de Carregamento informando ao Sistema Central que um carregamento foi interrompido (OCPP 1.6, 2015, p. 45).

2.4 *SERVERLESS* E *FUNCTIONS AS A SERVICE*

O paradigma *Serverless* (Sem servidor) está emergindo como uma nova forma de implantar aplicações em nuvem, isso em grande parte devido a popularização das arquiteturas em containers e micro serviços (Parse, 2022). Baldini (2017, p. 2), apresenta que a popularização do termo *Serverless* dispôs de um crescimento exponencial entre os anos de 2011 e 2016.

Serverless reúne inúmeras possibilidades e desafios, ficando a sua utilização dependente de uma análise quanto aos seus benefícios para um determinado projeto (BALDINI, 2017, p. 2). De modo que, um usuário de um serviço em uma Infraestrutura como Código (IaaS), tem como ponto positivo a abstração da maioria, senão todas as necessidades operacionais enquanto desenvolve uma determinada aplicação, reduzindo custos de implementação de código e também no tempo de desenvolvimento. Baldini (2017, p. 2) cita que: “A abstração do “*server”* traz a renúncia no que diz respeito às decisões sobre o design arquitetural da aplicação, que estão relacionados, entre outros pontos, à Qualidade do Serviço (QoS)”.

Reforçando os pontos apresentados acima, o termo *serverless* foi adotado para descrever um modelo e arquitetura de programação em que pequenos trechos de código são executados na nuvem sem nenhum controle sobre os recursos nos quais o código é executado (BALDINI, 2017, p. 3). Nesse sentido, o paradigma *serverless* reduz os custos operacionais, por conta do gerenciamento eficiente dos recursos em nuvem, minimizando o esforço necessário para gerenciar uma aplicação em nuvem.

A respeito dos trechos de códigos mencionados, eles são *Functions as a Service* (FaaS). Isto é, o desenvolvedor projeta a solução em uma função simples que possui uma única funcionalidade, e o provedor que mantém a solução, a invoca sob demanda (WOLSKI, 2019, p. 236). Dessa forma, o usuário paga apenas pelos recursos utilizados no período em que a função esteja em execução, sendo mais econômico que um provedor que cobra pelos recursos utilizados mesmo que a sua aplicação não seja requisitada (WOLSKI, 2019, p. 236).

2.4.1 Frameworks Serverless

As seções seguintes trazem diferentes *frameworks* para o desenvolvimento de arquiteturas *Serverless*. Sendo que todos são *Open Source.*

2.4.1.1 Knative

O Knative define um conjunto de objetos como *Custom Resource Definition* (CRDs) do Kubernetes. Esses recursos são usados para definir e controlar como sua carga de trabalho sem servidor se comporta no cluster (KNATIVE, 2022). O Knative estende o Kubernetes com o objetivo de providenciar um conjunto de componentes *middlewares* que são essenciais para construir aplicações modernas e baseadas em contêineres (ROCHA, 2021).

2.4.1.2 Kubeless

Kubeless é uma estrutura de *serverless open source* executada em cima do

Kubernetes. Kubeless permite implantar código sem ter que se preocupar com infraestrutura. O Kubeless usa recursos do Kubernetes para fornecer dimensionamento automático, roteamento, monitoramento e solução de problemas (KUBELESS, 2022).

2.4.1.3 Serverless Framework

O *Framework Serverless* auxilia no desenvolvimento e implantação de funções do *Amazon Web Services* (AWS) Lambda, juntamente com os recursos de infraestrutura da AWS necessários. É uma CLI que oferece estrutura, automação e melhores práticas prontas para uso, permitindo que você se concentre na construção de arquiteturas sofisticadas, orientadas a eventos e sem servidor, compostas por funções e eventos (SERVERLESS, 2022).

2.5 TRABALHOS RELACIONADOS

Nesta subseção destacamos as aplicações e trabalhos que nos permitam entender como o mercado de soluções para Cidades inteligentes estão se comportando, além da integração das plataformas com o contexto da mobilidade elétrica. As plataformas para Cidades Inteligentes a serem destacadas, possuem características semelhantes, afinal, InterScity, Fiware e Dojot são de código aberto e tem como objetivo, o desenvolvimento de soluções para cidades inteligentes a partir da evolução da tecnologia.

Correlacionando as possibilidades de plataforma para Cidades Inteligentes com a Dojot, a plataforma InterSCity (2022, p. 1) é: “Um projeto de pesquisa colaborativa hospedado pelo Instituto Nacional de Ciência e Tecnologia (INCT) da Internet do Futuro para Cidades Inteligentes”. A plataforma InterSCity foi desenvolvida de forma incremental como uma plataforma baseada em microsserviços de código aberto para permitir pesquisa, desenvolvimento e experimentos colaborativos em Cidades Inteligentes (DEL ESPOSTE, et al. 2018). Tendo como objetivo, aplicar novas técnicas de Ciência da Computação e Tecnologia para enfrentar problemas sociais urbanos em bairros desprivilegiados e

populações de baixa renda, aproveitando os dados existentes e coletando e analisando novos conjuntos de dados para apoiar a formulação de políticas públicas baseadas em evidências (INTERSCITY, 2022).

Do mesmo modo, Fiware como as plataformas Dojot e InterSCity, também é *open source*, e soluções para cidades inteligentes e serviços customizáveis. A Fiware *Foundation* impulsiona o desenvolvimento de soluções e serviços interoperantes de forma mais rápida, fácil e acessível (FIWARE FOUNDATION, 2022). Propondo um ecossistema de negócios sustentável e orientado à inovação (FIWARE FOUNDATION, 2022).

A plataforma Fiware contém diversos componentes de código aberto que podem ser usados em conjunto ou individualmente, e com componentes de terceiros para implementar projetos inteligentes. Para alentar essas informações, em sua documentação, Fiware (2022, p. 1) define sua missão como: “Construir um ecossistema aberto sustentável em torno de padrões de plataforma de software públicos, livres de royalties e orientados à implementação que facilitam o desenvolvimento de novas Aplicações Inteligentes em múltiplos setores ”.

Associando, a investigação aqui apresentada e o contexto de mobilidade elétrica. Araújo et al. (2022, p. 1) investiga como gerenciar as demandas de carregamento à medida que o mercado cresce. Por outro lado, Moraes et al. (2021, p. 1) desenvolve uma pesquisa que tem como intuito a implementação da Dojot em um cluster kubernetes.

A pesquisa aqui se relaciona com trabalhos mencionados, trazendo como contribuições a integração de diferentes plataformas para cidades inteligentes e o desenvolvimento de um componente utilizando um paradigma em ascensão de forma genérica. Além disso, deve ser destacado que a inclusão do paradigma serverless é um avanço na programabilidade e versatilidade para o desenvolvimento de aplicações nativas em nuvem.

3 METODOLOGIA

Para alcançar a proposta de uma arquitetura *Serverless* para aplicações de Cidades Inteligentes, foi preciso definir um caminho para alcançar o resultado esperado. A metodologia empregada nesta pesquisa para o desenvolvimento do serviço tem como referência o método empregado para a implementação de aplicações utilizadas no CPqD (CPqD, 2022). A Figura 3 apresenta as atividades que serão executadas para atingir o objetivo proposto no trabalho.

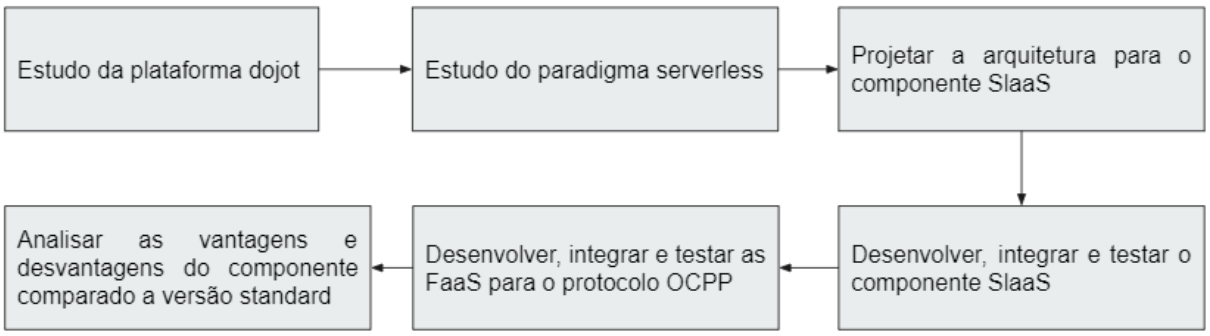

Figura 3. Etapas da metodologia

Fonte: Próprio Autor

3.1 ESTUDO DA PLATAFORMA DOJOT E DO PARADIGMA *SERVERLESS*

Na presente pesquisa, a plataforma Dojot e o paradigma *Serverless* são essenciais para a construção das FaaS. A plataforma Dojot é desenvolvida na arquitetura de micro serviços, como apresentado na Figura 1. Sendo que, cada componente da arquitetura corresponde a uma determinada função, executando suas tarefas de forma isolada e segura. Portanto, para que o componente seja projetado, analisamos o comportamento dos componentes que geram impacto na execução do IoT *Agent* OCPP, e ainda, estudamos o paradigma *serverless* e as ferramentas disponíveis que auxiliam no desenvolvimento das FaaS.

3.2 PROPOSTA DA ARQUITETURA E INTEGRAÇÃO DO COMPONENTE SLAAS

Com o estudo dos componentes internos da Dojot e a avaliação das tecnologias e ferramentas que auxiliam no desenvolvimento de arquiteturas *Serverless.* Desenvolvemos um modelo de arquitetura *Serverless* baseada na estrutura utilizada no IoT Agent OCPP da Dojot *Standard*, uma proposta que permite a integração de diversas plataformas e serviços no âmbito das Cidades Inteligentes. Para integração utilizamos o método implantado na Dojot utilizando containers e docker compose (DOJOT GIT, 2022).

3.3 DESENVOLVIMENTO DAS FAAS NA ARQUITETURA PROJETADA

Na etapa de desenvolvimento das FaaS, foi empregado o método Clean Code (LATTE, 2019, p. 969). De maneira que o desenvolvimento das FaaS seja feito com técnicas que facilitem a leitura e escrita dos códigos. Aplicando os conceitos de nomenclatura de funções e declaração de variáveis de ambiente, tornando as FaaS simples e objetivas.

Foi configurada uma base para armazenar as FaaS OCPP dentro da AWS Lambda, ou seja, as funções foram armazenadas em um provedor de nuvem para simular um caso real de um aplicação que utilize FaaS. Um exemplo de FaaS desenvolvida correlacionando a versão *standard* é apresentada na Seção 4.

3.4 ANÁLISE DAS VANTAGENS E DESVANTAGENS DO COMPONENTE EM COMPARAÇÃO COM A DOJOT *STANDARD*

Por fim, para avaliar se a arquitetura do componente SlaaS para cidades inteligentes é satisfatória, integramos as FaaS OCPP. E ao final do trabalho, descrevemos as vantagens e desvantagens do paradigma, comparamos as versões *serverless* e *standard* no contexto da mobilidade elétrica.

4 SERVERLESS AS A SERVICE

O capítulo está organizado da seguinte forma: a Seção 4.1 apresenta uma visão geral da arquitetura, ilustrando as possibilidades de integração com aplicações e plataformas. A Seção 4.2 apresenta detalhes da implementação. A Seção 4.3 descreve o comportamento da versão implementada no contexto da mobilidade elétrica para a Dojot.

4.1 VISÃO GERAL DA ARQUITETURA

A arquitetura do serviço SlaaS foi baseada na arquitetura de um dos componentes internos da Dojot, o IoT *Agent* (DOJOT, 2022). Isto implica que o conjunto de funcionalidades projetadas para o SlaaS se aproxima daqueles definidos pela Dojot, para o IoT *Agent.*

A Figura 4 ilustra a arquitetura geral do SlaaS para plataformas de Cidades Inteligentes e suas interações. Assim como o IoT *Agent* apresenta uma maneira efetiva de conectar os serviços externos aos componentes internos da plataforma (DOJOT, 2022), a SlaaS possui uma arquitetura que permite a interações entre as aplicações e as plataformas, no entanto, utilizando o paradigma *Serverless*.

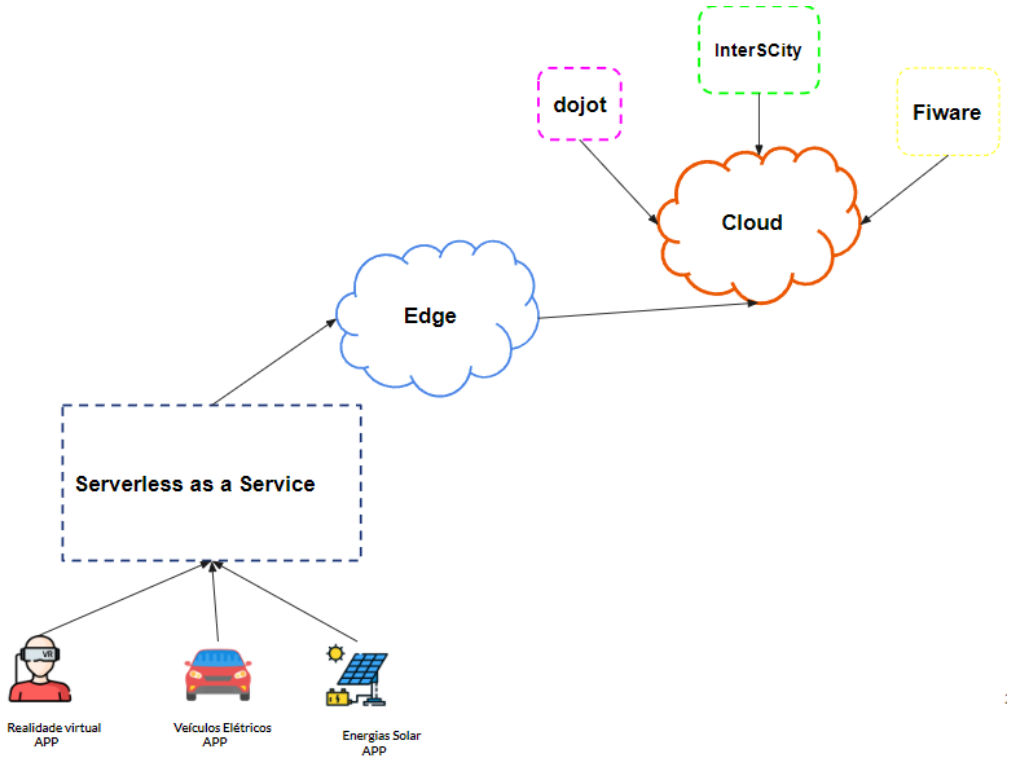

Figura 4. Arquitetura *Serverless as a Service* (SlaaS) utilizando *Cloud* e *Edge Computing*

Fonte: Próprio Autor

O fato das FaaS não dependerem de um contexto ou de uma linguagem de programação específica, motivou a utilização de uma componente que gerencia as publicações e subscrições das plataformas e aplicações (MISHRA, 2018, p. 599 - 600). Neste estilo arquitetural, podemos definir um modelo que possibilite a integração das FaaS e as plataformas de forma padronizada.

O SlaaS *Middleware* é responsável por direcionar as requisições feitas pelas plataformas para a execução das FaaS correspondentes, além de, gerenciar se as requisições são válidas. O SlaaS *Middleware* está na borda para que possa otimizar recursos hospedando aplicativos de computação intensiva, processando grandes volumes de dados antes de enviar para a nuvem (PHAM, 2020, p. 116975 - 116976).

As plataformas, têm por função demonstrar e habilitar as soluções, construindo um ecossistema que permita a integração com diversas áreas (DOJOT, 2022). Por exemplo, aplicações para Energia Solar, segurança pública, saúde, realidade virtual e mobilidade elétrica.

4.2 DETALHES DA IMPLEMENTAÇÃO

Nesta subseção apresentaremos a inicialização do IoT *Agent* OCPP *serverless* e a FaaS desenvolvida para o *Boot Notification*. Os códigos apresentados nos auxiliaram na análise do comportamento da versão do componente no contexto da mobilidade elétrica.

A Figura 5, apresenta a inicialização do IoT *Agent* OCPP *serverless, a função init* inicia a conexão HTTPS e em seguida a função cria uma instância do *Websocket* para que as mensagens possam trafegar em tempo real.

```
init() {
  logger.debug(`Initizalizing Charge Point Server`, TAG);
  try {
    var app = express();
    if(secureConnection){
      server = new https.createServer({
      cert: fs.readFileSync(CERT),
      key: fs.readFileSync(CERT_KEY)
      });
    }else{
      server = http.createServer(app)
    }

    this.socket = new WebSocket.Server({ server:server });
    this.socket.on('error',(error)=>{
      logger.error(error);
    })
    this.socket.on('connection', (ws, req) => {
      try{
        logger.debug(`Socket start connection`, TAG);
        ws.on('message', (message) => {
        this._receiveMessage(message,req, ws,this._callbackOnMessage);
        });
      }catch (error) {
        logger.error(error, TAG);
      }
    });

server.listen(SERVER_PORT, () => {
      logger.info(`Server listen on ${SERVER_PORT}`, TAG);
    });

  }catch(error){
    logger.error(error, TAG);
    }
}
```

Figura 5. Inicialização do IoT *Agent* OCPP Serverless

Fonte: Próprio Autor

A Figura 6, apresenta a FaaS referente a função OCPP *BootNotification* que envia informações do Ponto de Carregamento para o Sistema Central, como: A hora atual, o status do Ponto de carregamento e o identificador do Ponto de Carregamento.

```
constructor(){
  this.dojot = axios.create({
    baseURL: DOJOT_URL
  })
}

async bootNotification(data,deviceId){
  try {
    let url_serverless = ""
    let url = `/${deviceId}/actuate`
    let bootNotificationConf = {
      currentTime: new Date(),
      interval:14000,
      status:"Accepted"
    }

    let attrs={ bootNotificationConf}
    let res = await this.dojot.put(url,{attrs},JWT);

  } catch (error) {
    logger.info(error);
  }
}
```

Figura 6. FaaS OCPP *BootNotification*

Fonte: Próprio Autor

4.3 COMPORTAMENTO DA VERSÃO IMPLEMENTADA NO CONTEXTO DA MOBILIDADE ELÉTRICA PARA A DOJOT

A Figura 7 ilustra a arquitetura referente a versão implementada no contexto da mobilidade elétrica e suas camadas são descritas em seguida. Para a mobilidade elétrica, foi necessário a inclusão de uma *CS*, que é um componente externo que

integra o contexto da mobilidade elétrica e permite a gestão de CPs e a validação das ações entre os EVs e os CPs.

Nas bibliotecas de FaaS os serviços são armazenados e já utilizam do paradigma *serverless*, ou seja, não necessita de nenhuma configuração extra do usuário. Estas FaaS podem ser atualizadas gerando uma nova entrada para a biblioteca de funções.

O cliente, requisita uma determinada ação para o CP que interpreta a requisição e encaminha para a IoT *Agent* OCPP. O *Agent*, por sua vez, aciona a FaaS correspondente que está presente nas bibliotecas de FaaS. Por fim, o *Agent* se comunica com os componentes internos para verificação interna. Com a requisição validada, o *Agent* retorna com o status da requisição para o cliente.

O cliente, EV, a partir ***client request action***, pode iniciar uma ***transaction***, ou seja, a recarga do veículo. Internamente, o componente **IoT *Agent* OCPP *Serverless***, filtra essa requisição e valida com os componentes internos da Dojot. Logo, com a requisição válida o **IoT *Agent* OCPP *Serverless*** retorna com o status da requisição para o cliente, finalizando, para essa função, sua atividade.

Por sua vez, a **FaaS**, possui um ciclo de vida com dois estágios distintos. O estágio de *init* e *envoke*. Na fase de *init*, o provedor executa o código de inicialização da função, que ocorre durante a primeira chamada da função. E na fase de *invoke*, o provedor solicita um evento de invocação e inicializa o manipulador de funções. Depois de concluída, a FaaS retorna para o estado de “congelamento”, ou seja, não está mais consumindo recursos computacionais e consequentemente há uma economia de gastos com infraestrutura. O serviço mantém a função por algum tempo em antecipação caso haja uma nova invocação. Se a FaaS não receber nenhuma chamada por um determinado período, o provedor a encerra e a remove do ambiente (LAMBDA, 2022).

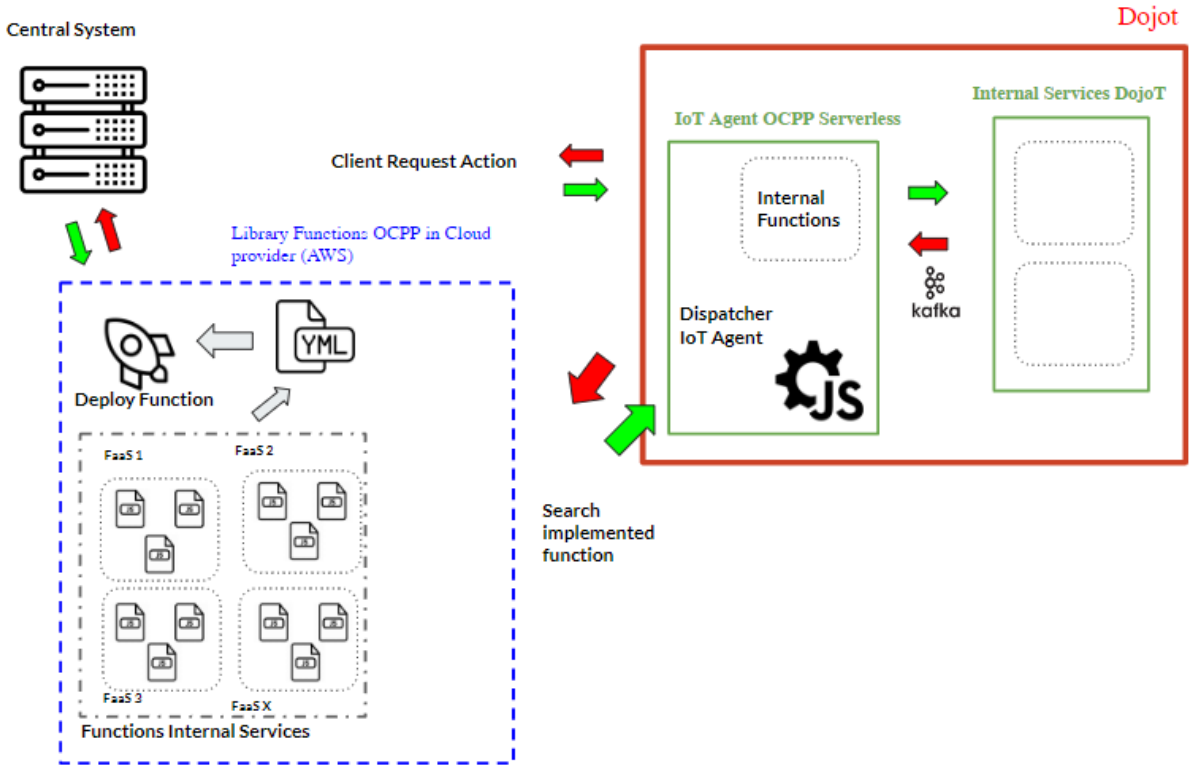

Figura 7. Versão do serviço implementada

Fonte: Próprio Autor

5 DESENVOLVIMENTO

O capítulo está organizado da seguinte forma: A Seção 5.1 apresenta o processo de desenvolvimento adotado. Na Secção 5.2 apresentamos as ferramentas utilizadas para o desenvolvimento da solução. E na Seção 5.3, descrevemos as FaaS implementadas.

5.1 PROCESSO DE DESENVOLVIMENTO ADOTADO

Um modelo de processo de software é a representação simplificada de um processo de software, representando uma perspectiva particular fornecendo informações parciais sobre ele (SOMMERVILE, 2011, p. 19). Há diversos modelos genéricos que podemos usar para explicar diferentes abordagens. No entanto, para o desenvolvimento do projeto foram utilizados dois modelos em conjunto, o modelo de Desenvolvimento incremental apresentado na Figura 8 e o modelo orientado a reuso, apresentado na Figura 9.

O desenvolvimento incremental do sistema é desenvolvido como uma série de versões. Isto é, diversas fases, intercalando entre atividades de especificações, desenvolvimento e validação (SOMMERVILLE, 2011, p. 20).

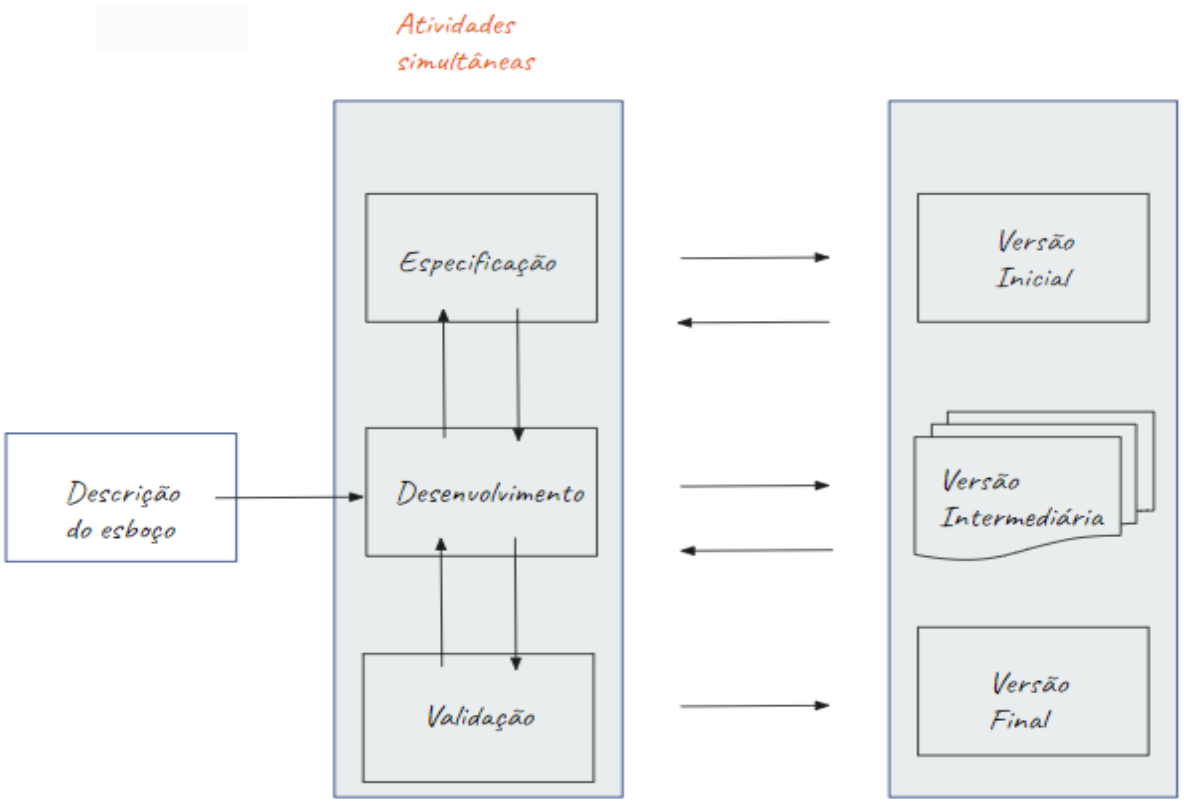

Figura 8. Modelo Incremental

Fonte: (SOMMERVILLE, 2011, p. 20)

Já a Engenharia de Software orientada a reuso é uma abordagem baseada no uso de componentes já existentes. Concentrando-se na integração desses componentes em sistemas já existentes (SOMMERVILLE, 2011, p. 20).

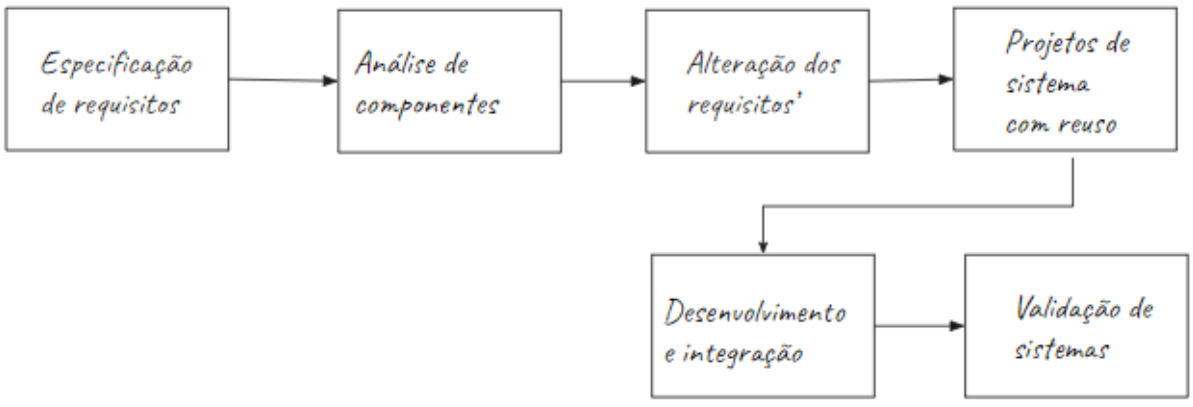

Figura 9. Modelo de reuso de componentes

Fonte: (SOMMERVILLE, 2011, p. 20)

Para o contexto do componente SlaaS o modelo incremental apresentado na Figura 8, foi utilizado para o desenvolvimento da FaaS OCPP, ou seja, a descrição do esboço, atividades simultâneas e a implantação das versões da FaaS. E o modelo de reuso de componentes, foi utilizado para a criação do IoT *Agent* OCPP *serverless* baseado no IoT *Agent* OCPP da Dojot *standard,* seguindo todos os processos apresentados na Figura 9, a especificação dos requisitos do IoT *Agent* OCPP para *serverless*, a análise do componente já implementado, a alteração dos requisitos e o processo de desenvolvimento e a validação do sistema com as alterações implementadas.

5.2 FERRAMENTAS UTILIZADAS

5.2.1 Git

Git é um sistema de controle de versão distribuído gratuito e *open source* projetado para lidar com tudo, desde projetos pequenos a muito grandes com velocidade e eficiência (GIT, 2022). Utilizado para manter o código em um repositório seguro e com um ferramental que auxilia na correção e no trabalho em equipe.

5.2.2 Visual Studio Code

O *Visual Studio Code* é um editor de código fonte *Open Source* (CODE, 2022). Utilizado por ser simples e versátil, sendo compatível com as extensões para *JavaScript*, Git e outras ferramentas que utilizamos para o desenvolvimento do serviço.

5.2.3 AWS Lambda

O AWS Lambda é um serviço de *serverless* e orientado a eventos que permite executar código para praticamente qualquer tipo de aplicativo ou serviço de *back-end* sem provisionar ou gerenciar servidores. Podemos acionar o Lambda a

partir de mais de 200 serviços da AWS e aplicativos de *Software as a Service* (SaaS) e pagar apenas pelo que usar (lambda, 2022). Utilizamos para armazenar e executar as FaaS.

5.2.4 Serverless Framework

Framework para desenvolver, implantar, solucionar problemas e proteger os aplicativos *serverless* com custos e sobrecarga radicalmente menores usando o *Serverless Framework* (SERVERLESS, 2022). Tendo em vista a integração com a AWS Lambda, o *framework serverless* foi utilizado para o desenvolvimento das FaaS OCPP.

5.2.5 NodeJS

Como um tempo de execução JavaScript assíncrono orientado a eventos, o Node.js foi projetado para criar aplicativos de rede escaláveis (NODE, 2022). Node foi utilizada para a codificação das FaaS que são utilizadas pela aplicação. Além de ser uma linguagem simples e eficiente para ser executada em FaaS.

5.3 FAAS DESENVOLVIDAS

As FaaS, possuem como características a execução de uma única tarefa, o que se encaixa perfeitamente nas especificações do protocolo OCPP, afinal, cada responsabilidade descrita no protocolo, uma FaaS pode ser responsável. De forma mais específica, essas foram as funções implementadas:

- ***Boot Notification:*** Após a inicialização, um *Charge Point* deve enviar um pedido ao *Central System* com informações sobre a sua configuração (por exemplo, versão, fornecedor, etc.). O *Central System* deverá responder para indicar se aceitará o *Charge Point*.

- ***Status Notification:*** Um *Charge Point* envia uma notificação ao *Central System* para informar o *Central System* sobre uma mudança de status ou um erro no *Charge Point.*
- ***HeartBeat:*** Para informar ao *Central Syste,* que um *Charge Point* ainda está conectado, um *Charge Point* envia uma pulsação após um intervalo de tempo configurável. O *Charge Point* deve enviar uma requisição para garantir que o *Central System* saiba que um *Charge Point* ainda está ativo.
- ***Start transaction:*** O *Charge Point* deve enviar uma requisição ao *Central System* para informar sobre uma transação que foi iniciada.
- ***Stop transaction:*** Quando uma transação é interrompida, o *Charge Point* deve enviar uma requisição, notificando ao *Central System* que a transação foi interrompida.

6 CONSIDERAÇÕES FINAIS

Com a conclusão do fluxo de execução das funções e a utilização do paradigma serverless, conseguimos obter pontos que são extremamente benéficos para a plataforma Dojot e outros que podem gerar problemas para o projeto. E com isso, podemos descrever pontos que podem beneficiar diversos projetos que desejam utilizar o paradigma serverles e não conseguem definir os impactos que sua utilização gera. Para isso, as Seções 6.1 e 6.2, descrevem esses pontos.

6.1 VANTAGENS

Desempenho: Utilizando os paradigmas de programação e as arquiteturas convencionais, os serviços são inicializados e mesmo que, o mesmo não seja “estressado” ele continuará ocupando uma parcela dos recursos computacionais. Já em *Serverless*, as funções ficam em estado de congelamento, até que sejam invocadas e a partir disso, utilizam os recursos necessários para sua execução e ao fim de sua execução, a função retorna para o estado inicial e deixa de ocupar os recursos de que foram alocados, tendo assim, uma eficiência em sua execução.

Programabilidade: O serviço é baseado em porções de códigos, FaaS, que executam uma única tarefa. Portanto, podemos criar diversas porções de código para cada necessidade de um projeto. E não obstante, podemos desenvolver essas funções em qualquer linguagem de programação, não sendo assim, um limitado para a utilização do paradigma.

Simplicidade e praticidade: Um dos principais motivos para utilizar *serverless*, o desenvolvedor foca somente no desenvolvimento sem ter que cuidar da infraestrutura, tendo assim, um foco maior naquilo que é importante para a sua função, o desenvolvimento das aplicações.

Escalabilidade: Como o paradigma é nativo em nuvem, a praticidade para escalar as funções, o torna uma boa opção para serviços que necessitam desse modelo.

Menos gastos com infraestrutura: As FaaS só consomem os recursos quando invocadas. Isto é, caso a FaaS não esteja em execução, não há gastos computacionais para mantê-la. Sendo assim, com *serverless* as aplicações não ficam ociosas, minimizando os custos.

6.2 DESVANTAGENS

Quantidade de funções: O paradigma *serverless* tem como padrão que, cada FaaS seja responsável por uma única requisição do sistema e isso faz com que a quantidade de funções sejam proporcionais à complexidade do serviço. Em outras palavras, quanto maior o serviço maior a quantidade de funções necessárias para atender seus requisitos.

Quantidade necessária de pessoas na equipe: Este problema está diretamente relacionado a complexidade do serviço e a quantidade de investimento disponível para implementação do paradigma. Pois, relacionando aos padrões utilizados para compor uma equipe no CPqD, normalmente as equipes são enxutas. Então, em um cenário que o serviço seja complexo, como é o caso da mobilidade elétrica, a quantidade de funções, como já mencionado, aumenta bastante e para manter todo o ecossistema, um único desenvolvedor ou que seja um pequeno grupo, se torna insustentável. Afinal, se um único desenvolvedor dominar os comportamentos das diversas funções, gera uma dependência do material humano, demora para entrega de novas demandas, problemas para manter o que foi planejado.

7 CONCLUSÃO

Neste trabalho foi apresentada uma abordagem para utilizar o paradigma *Serverless* em uma plataforma para cidades inteligentes, a arquitetura da plataforma foi utilizada como base para nos direcionar no desenvolvimento do componente. A arquitetura base foi um direcionar para que não precisássemos mudar uma estrutura já consolidada.

Os resultados aqui apresentados estão alinhados com as possibilidades e os desafios descritos por (BALDINI, 2017) e (WOLSKI, 2019). No que se refere 1) à simplicidade em se executar uma aplicação; 2) à abstração de configurações da infraestrutura; 3) à economia gerada pelo fato dos serviços serem executados somente quando invocados; 4) à economia de gastos com infraestrutura; E por outro lado, 5) à necessidade de se adaptar com uma arquitetura rígida, com poucas possibilidades de customização; E por fim 6) à necessidade de um controle rigoroso quando se trata da manutenibilidade de milhares de funções.

O escopo da pesquisa aqui apresentada está limitado ao contexto da mobilidade elétrica, das FaaS executadas dentro desse espectro. Apesar disso, deseja-se que a metodologia empregada na realização da pesquisa para implantação do serviço, possa ser generalizada para ser aplicada a qualquer contexto das *Smart Cities.* Espera-se que a abordagem de utilização do *Serverless as a* ***Service*** apresentada no trabalho possa ser amplamente utilizada, permitindo que as instituições e os projetos que possam se beneficiar dessa proposta.

8 REFERÊNCIAS BIBLIOGRÁFICAS

AFOLABI, Ibrahim, et al. "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions." ***IEEE Communications Surveys & Tutorials*** 20.3 (2018): 2429-2453.

MEHMOOD, Yasir, et al. "Internet-of-things-based smart cities: Recent advances and challenges." ***IEEE Communications Magazine*** 55.9 (2017): 16-24.

DOJOT. ***DojoT Soluções para IoT :*** **plataforma de desenvolvimento para IoT**. Disponível em: https://www.dojot.com.br. Acesso em: 07 set. 2022.

DOJOT DOCUMENTATION. ***DojoT Documentation***. Disponível em: https://dojotdocs.readthedocs.io/en/latest. Acesso e m: 30 set. 2022.

BALDINI, Ioana et al. Serverless computing: Current trends and open problems. In: **Research Advances in Cloud Computing**. Springer, Singapore, 2017. p. 1-20.

Ashton, Kevin. "That ‘internet of things’ thing." ***RFID journal*** 22.7 (2009): 97-114.

BERNSTEIN, David. Containers and cloud: From lxc to docker to kubernetes. **IEEE Cloud Computing**, v. 1, n. 3, p. 81-84, 2014.

ALAVIR Amir H., et al. "**Internet of Things-enabled smart cities: State-of-the-art and future trends.**" *Measurement* 129 (2018): 589-606.

ZANELLA, Andrea, et al. "Internet of things for smart cities." ***IEEE Internet of Things journal*** 1.1 (2014): 22-32.

NOVAIS, Celso Ribeiro Barbosa de. "**Mobilidade elétrica: desafios e oportunidades**." (2016).

Du, Mingming, et al. "**Research and Implementation of Communication between OCPP Client and Charging Station Management System**." *CONVERTER* 2021.7 (2021): 781-792.

Parse Cloud Code. **Cloud Code Guide**. Disponível em: https://docs.parseplatform.org/cloudcode/guide/. Acesso em: 08 set, 2022.

WOLSKI, Rich, et al. "Cspot: Portable, multi-scale functions-as-a-service for iot." ***Proceedings of the 4th ACM/IEEE Symposium on Edge Computing***. 2019.

Rocha, Pedro Lucas Moreira. ***Desenvolvimento Serverless: Soluções, Impacto e Futuro***. Diss. 2021.

FIWARE FOUNDATION. **The Open Source Platform for Our Smart Digital Future**. Disponível em: https://www.fiware.org/about-us/. Acesso em: 08 set. 2022.

Serverless. *Do more less.* ***Serverless***. Disponível em: https://www.serverless.com. Acesso em: 08 set. 2022.

Knative. ***Knative is an Open-Source Enterprise-level solution to build Serverless and Event Driven Applications***. Disponível em: https://knative.dev/docs/. Acesso em: 08 set. 2022.

Kubeless. ***Kubernetes-native serverless framework***. Disponível em: https://github.com/vmware-archive/kubeless. Acesso em: 08 set. 2022.

Araújo, Felipe, et al. "Plataforma para Coleta de Dados de Eletropostos para Veículos Elétricos: Prototipação em Bancada de Testes Real." *Anais do XIV Simpósio Brasileiro de Computação Ubíqua e Pervasiva*. **SBC**, 2022.

Moraes, Jean, et al. "Implementação de um cluster Kubernetes com a plataforma Dojot para Aplicações de Internet das Coisas." *Anais do XLVIII Seminário Integrado de Software e Hardware*. **SBC**, 2021.

DOJOT GIT. ***DojoT Deploy - Docker compose***. Disponível em: https://github.com/dojot/docker-compose . Acesso em: 1 out. 2022.

Latte, Björn, Sören Henning, and Maik Wojcieszak. "**Clean code: On the use of practices and tools to produce maintainable code for long-living**." (2019): 96-99.

Mishra, Biswajeeban. "Performance evaluation of MQTT broker servers." ***International Conference on Computational Science and Its Applications***. Springer, Cham, 2018.

PHAM, Quoc-Viet et al. A survey of multi-access edge computing in 5G and beyond: Fundamentals, technology integration, and state-of-the-art. **IEEE Access**, v. 8, p. 116974-117017, 2020.

SOMERVILLE, Ian. "**Engenharia de software, 9a**." *São Palo, SP, Brasil* (2011), p. 63-69.

Git. ***Open Source distributed Version Control***. Disponível em: https://git-scm.com/. Acesso em: 20 set. 2022.

Code. ***Visual Studio Code***. Disponível em: https://github.com/vmware-archive/kubeless. Acesso em: 08 set. 2022.

Node. ***Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine***. Disponível em: https://nodejs.org/en/about. Acesso em: 21 set. 2022.

Lambda. **AWS Lambda**. Disponível em: https://aws.amazon.com/lambda/. Acesso em: 08 set. 2022.

BURNS, Brendan; BEDA, Joe; HIGHTOWER, Kelsey. **Kubernetes**. Dpunkt, 2018.

SAYFAN, Gigi. **Mastering kubernetes**. Packt Publishing Ltd, 2017.

FRANCO, Enrico. **FaaS on Kubernetes Workload migration and optimization with Kubeless and Knative**. 2020. Tese de Doutorado. Politecnico di Torino.

MOHANTY, Sunil Kumar et al. An Evaluation of Open Source Serverless Computing Frameworks. In: **CloudCom**. 2018. p. 115-120.

HOANG, Trong Thang; TAO, Minh Thong; AU, Phu Hiep. **Research and implementation of monitoring systems Prometheus and Grafana**. 2020. Tese de Doutorado. FPTU HCM.

Collina, Matteo, Giovanni Emanuele Corazza, and Alessandro Vanelli-Coralli. "Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST." *2012* ***IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications-(PIMRC)***. IEEE, 2012.

Fontana, Neri Burato Bez, and Alexandre Davi Zanelatto. "**Arquitetura serverless baseada em eventos para aplicações WEB utilizando a AWS**." *Sistemas de Informação-Florianópolis* (2019).