UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

MESTRADO EM CIÊNCIAS DA COMPUTAÇÃO

VITÓRIA REGINA NICOLAU SILVESTRE

DOP-MS: SERVIÇO DE *OFFLOADING* DE DADOS USANDO UMA ARQUITETURA DE MICROSSERVIÇOS COM SUPORTE A ANONIMIZAÇÃO DE DADOS

FORTALEZA

2021

VITÓRIA REGINA NICOLAU SILVESTRE

DOP-MS: SERVIÇO DE *OFFLOADING* DE DADOS USANDO UMA ARQUITETURA DE MICROSSERVIÇOS COM SUPORTE A ANONIMIZAÇÃO DE DADOS

Dissertação apresentada ao Curso de Mestrado em Ciências da Computação do Departamento de Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciências da Computação. Área de Concentração: Engenharia de Software.
Orientador: Fernando Antonio Mota Trinta, DSc.
Coorientador: Lincoln Souza Rocha, DSc.

FORTALEZA

2021

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

S593d Silvestre, Vitória Regina Nicolau.
DOP-MS: Serviço de offloading de dados usando uma arquitetura de microsserviços com suporte a anonimização de dados / Vitória Regina Nicolau Silvestre. – 2021.
95 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2021.
Orientação: Prof. Dr. Fernando Antonio Mota Trinta.
Coorientação: Prof. Dr. Lincoln Souza Rocha.

1. Mobile Cloud Computing. 2. Offloading. 3. Microsserviços. I. Título.

CDD 005

VITÓRIA REGINA NICOLAU SILVESTRE

DOP-MS: SERVIÇO DE *OFFLOADING* DE DADOS USANDO UMA ARQUITETURA DE MICROSSERVIÇOS COM SUPORTE A ANONIMIZAÇÃO DE DADOS

Dissertação apresentada ao Curso de Mestrado em Ciências da Computação do Departamento de Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciências da Computação. Área de Concentração: Engenharia de Software.

Aprovada em: 25 de Fevereiro de 2021

BANCA EXAMINADORA

Fernando Antonio Mota Trinta, DSc (Orientador)
Universidade Federal do Ceará (UFC)

Lincoln Souza Rocha, DSc (Coorientador)
Universidade Federal do Ceará (UFC)

Lincoln Souza Rocha, DSc
Universidade Federal do Ceará (UFC)

Paulo Antonio Leal Rego, DSc
Universidade Federal do Ceará (UFC)

Vinícius Cardoso Garcia, DSc
Universidade Federal de Pernambuco (UFPE)

Dedico este trabalho aos meus pais.

AGRADECIMENTOS

Primeiramente a Deus por todas as bençãos ao longo da minha vida. Aos meus pais, Evandro e Neide, pelo amor, carinho e apoio incondicional. Não existem palavras para descrever tudo que vocês fizeram e fazem por mim.

A minha querida avó Terezinha (in memoriam), que sempre acreditou em meus sonhos e me colocou em todas as suas orações.

Aos meu irmãos, Vladimir e José, pelo apoio e incentivo incondicional.

Ao meu noivo e companheiro de vida Anderson Almada, por toda paciência e companheirismo durante essa jornada.

Aos meus orientadores, professor Fernando Trinta e professor Lincoln Rocha, por todo o acompanhamento durante o desenvolvimento deste trabalho. Principalmente ao professor Trinta por todas as longas horas de orientação e paciência.

Aos professores Paulo Rego e Vinícius Cardoso, que compõem a banca examinadora e certamente contribuirão para o sucesso da solução desenvolvida.

À Universidade Federal do Ceará, pela oportunidade de fazer o mestrado.

Ao Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat), pelas experiências adquiridas e pela oportunidade de desenvolver minhas habilidades profissionais trabalhando em projetos.

À todos os amigos e colegas que me incentivaram e acompanharam essa caminhada ao meu lado.

À todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

"O sonho é que leva a gente para frente. Se a gente for seguir a razão, fica aquietado, acomodado."
(Ariano Suassuna)

RESUMO

Com o aumento do uso dos dispositivos móveis, as aplicações estão cada vez mais complexas, requisitando um grande poder de processamento e armazenamento de dados, o que caracteriza um desafio quando a limitação computacional desses dispositivos é levada em consideração. A técnica de *offloading* possibilita a migração de dados e a realização do processamento em um ambiente remoto, permitindo a (i) economia de armazenamento no dispositivo móvel e (ii) compartilhamento de dados. Várias infraestruturas de software foram criadas para ajudar no desenvolvimento dessas aplicações que realizam *offloading* de dados. Porém, foram identificados aspectos importantes no cenário de *offloading* que não foram contemplados nessas infraestruturas, como modelos de política de sincronização de dados configurável, mecanismos de privacidade dos dados que são migrados e análises de escalabilidade e desempenho. Esse trabalho apresenta uma solução para auxiliar o desenvolvimento de aplicações móveis que utilizam da migração de dados, incluindo dados contextuais, dos dispositivos móveis para um ambiente remoto usando uma arquitetura de microsserviços. Em alguns cenários (e.g., aplicações médicas de monitoramento de pacientes), os dados provenientes de diferentes dispositivos são usados para medir, inferir e compreender seu ambiente de execução. A solução proposta neste trabalho é denominada DOP-MS, um serviço de *offloading* de dados usando uma arquitetura de microsserviços com suporte a anonimização de dados. Seu desenvolvimento é baseado na evolução de dois trabalhos anteriores: COP e CAOS-MS. Foram realizados dois grupos de experimentos: uma prova de conceito para validar a solução desenvolvida e testes de desempenho e escalabilidade para verificar se a utilização de uma arquitetura de microsserviços trouxe benefícios relacionados a desempenho e escalabilidade para a solução proposta. Como resultados desses testes foi percebido que a migração de dados é vantajosa tanto em termos de espaço para armazenamento, acesso a informações que estão centralizadas no *cloudlet* e também na possibilidade de inferir situações com base nos dados disponíveis. O experimento de desempenho e escalabilidade detectou que a arquitetura de microsserviços fornece um melhor suporte a escalabilidade e um desempenho melhor conforme aumentamos a instância do serviço de *offloading* de dados. Por fim, o trabalho apresenta uma analise estatística proveniente dos dados obtidos durante os teste realizados.

Palavras-chave: *mobile cloud computing*; *offloading*; microsserviços.

ABSTRACT

Due to mobile devices' growing presence in our daily routine, mobile applications are becoming increasingly complex, requiring more powerful processing capability and more extensive data storage, which characterizes a challenge when computational constraints of these devices are taken into account. The data offloading technique enables data migration into a remote environment, allowing (i) storage savings on the mobile device and (ii) sharing data among users. Several software infrastructures have been proposed to help the development of mobile applications with data offloading features. However, they lack essential features for data offloading, such as configurable data synchronization policy models, privacy mechanisms for offloaded data, and scalability and performance analyses. This work presents a solution to assist the development of mobile applications that use data migration, including contextual data, from mobile devices to a remote environment, based on a microservice architecture. In some scenarios (e.g., medical patient monitoring applications), data from different users may be used to infer new situations and understand their execution environment. The proposed solution here is called DOP-MS, a data offloading service using a microservice architecture with support for data anonymization. DOP-MS development is based on the evolution and integration of two previous works: COP and CAOS-MS. We conducted two groups of experiments: a proof of concept to validate the developed solution and performance and scalability tests to verify if a microservice architecture brought benefits related to performance and scalability for the proposed solution. As a result of these tests, we concluded that data offloading provides benefits in savings in storage mobile devices and creates new possibilities for inferring situations based on multiple users' sharing data. The performance and scalability experiments showed that the microservice architecture provides better support for scalability and better performance as long the number of DOP-MS instances is provided. Finally, the work presents a statistical analysis from the data obtained during the tests performed.

Keywords: mobile cloud computing; offloading; microservices.

LISTA DE ILUSTRAÇÕES

Figura 1 – Arquitetura da *Mobile Cloud Computing* 21
Figura 2 – Locais de Execução do *Offloading* 23
Figura 3 – Categorias de Serviço no SMAll 35
Figura 4 – Organização de Camadas de Orquestração da Plataforma *SMAll* 36
Figura 5 – Visão Geral da *SEMAR* 38
Figura 6 – Arquitetura do *PUMA* 41
Figura 7 – Arquitetura da Solução 43
Figura 8 – Estratégia de Privacidade 43
Figura 9 – Arquitetura do COP 46
Figura 10 – Arquitetura da Plataforma CAOS-MS 48
Figura 11 – Cenário Motivador 53
Figura 12 – Funcionamento da Solução 55
Figura 13 – Arquitetura do DOP-MS integrado com o CAOS-MS 56
Figura 14 – Fluxograma de decisão sobre uso ou não do *offloading* de processamento . . 58
Figura 15 – Etapas para configuração dos dados a serem enviados no processo de *offloading* 66
Figura 16 – Configuração para o recebimento de dados 66
Figura 17 – Tela de um registro de Prontuário - Medical App 72
Figura 18 – *Screenshots* da Execução do Filtro de Consulta por tratamento ao Covid19 . 74
Figura 19 – *Screenshots* da Execução do Filtro 02 74
Figura 20 – Tempo Gasto no Envio de Dados pelo DOP-MS 76
Figura 21 – Tempo de execução do *offloading* de dados com 200 tuplas 78
Figura 22 – Tempo de execução do *offloading* de dados com 4000 mil tuplas 79
Figura 23 – Tempo de Execução do *Offloading* de Dados 80
Figura 24 – Tempo Detalhado 81

LISTA DE TABELAS

Tabela 1 – Comparativo Entre os Trabalhos Relacionados 50
Tabela 2 – Funcionalidades da Aplicação Medical App 65
Tabela 3 – Especificação dos Cenários de Teste 77
Tabela 4 – Resultados dos testes ANOVA 83
Tabela 5 – Resultados dos testes T-Student 83
Tabela 6 – Comparativo Entre os Trabalhos Relacionados e a Solução DOP-MS 87

LISTA DE CÓDIGOS-FONTE

Listagem 1 – Trecho de Código-Fonte para Offloading e Anonimização de Dados . 61
Listagem 2 – Implementação da interface responsável por invocar o método `Offlodable` 67
Listagem 3 – Trecho de Código-Fonte da utilização de filtros na Aplicação Medical App 68
Listagem 4 – Configuração da classe cadastro de paciente 69
Listagem 5 – Implementação da classe onde são definidos os dados que devem ser migrados 70
Listagem 6 – Implementação simulada do sensor de localização 70
Listagem 7 – Tupla do DOP-MS - Register 75
Listagem 8 – Trecho de *script* do serviço de *start* 94
Listagem 9 – Trecho de *script* do *microservive* de Stop 94
Listagem 10 – Trecho de *script* do serviço de *destroy* 95

LISTA DE ABREVIATURAS E SIGLAS

CAOS-MS	*CAOS Microservices*
COP	*Contextual data Offloading service with Privacy support*
DOP-MS	*Data Offloading service with suPport for data anonymisation based on Micro-Service*
GPS	*Global Positioning System*
HIPAA	*Health Insurance Portability and Accountability Act*
IoT	*Internet of Things*
LoCCAM	*Loosely Coupled Context Acquisition Middleware*
MCC	*Mobile Cloud Computing*
MQTT	*Message Queue Telemetry Transport*
SOA	*Service oriented architecture*
VGI	*Volunteered Geographic Information*
WLAN	*Wireless Local Network*

SUMÁRIO

1 INTRODUÇÃO 14
1.1 Contextualização 14
1.2 Motivação 15
1.3 Objetivos e Contribuições 17
1.4 Metodologia 18
1.5 Organização da Dissertação 19
2 FUNDAMENTAÇÃO TEÓRICA 20
2.1 Computação móvel em nuvem 20
2.1.1 Offloading 22
2.1.1.1 Offloading de dados *24*
2.1.2 Privacidade dos dados 24
2.1.2.1 Anonimização de dados *25*
2.1.2.2 Técnicas de anonimização *26*
2.1.3 Crowdsourcing 27
2.2 Estilo Arquitetural de Microsserviços 28
2.3 Considerações Finais 33
3 TRABALHOS RELACIONADOS 34
3.1 Sintetização dos estudos relacionados 34
3.1.1 A service-oriented approach to crowdsensing for accessible smart mobility scenarios (SMAll) 34
3.1.2 Smart Environment Monitoring and Analytical in Real-time (SEMAR) . . 37
3.1.3 Smart Mobility and Sensing: Case Studies Based on a Bike Information Gathering Architecture (PUMA) 40
3.1.4 A Blockchain-Based Approach for Privacy Control of Patient's Medical Records in the Fog Layer 42
3.1.5 COP 45
3.1.6 CAOS-MS 47
3.1.6.1 Arquitetura *47*
3.2 Comparativo Entre os Trabalhos 49
3.3 Considerações Finais 50

4 SERVIÇO DE *OFFLOADING* DE DADOS USANDO UMA ARQUITETURA DE MICROSSERVIÇOS COM SUPORTE A ANONIMIZAÇÃO DE DADOS 52
4.1 Visão Geral 52
4.2 Arquitetura 55
4.2.1 Servidor 56
4.2.2 Cliente 57
4.3 Filtros 57
4.4 Modelo e Privacidade de Dados 58
4.5 Políticas de Sincronização 61
4.6 Tecnologias Utilizadas 62
4.7 Considerações Finais 63
5 PROVA DE CONCEITO E EXPERIMENTOS 64
5.1 Ambiente de teste 64
5.2 Prova de Conceito 65
5.2.1 Implementação da Medical App 65
5.2.2 Descrição do Experimento 71
5.2.3 Teste de validação da solução 72
5.2.4 Teste de comparação do tempo de execução do offloading de dados no COP e DOP-MS 75
5.2.5 Teste de execução dos filtros utilizando o serviço de offloading de processamento do CAOS-MS 78
5.2.6 Análise Estatística 81
5.3 Considerações Finais 84
6 CONCLUSÃO E TRABALHOS FUTUROS 86
6.1 Resultados Alcançados 86
6.2 Limitações 88
6.3 Trabalhos Futuros 88
REFERÊNCIAS 89
APÊNDICE A – *SCRIPTS* DA EXECUÇÃO DOS SERVIÇOS 94

1 INTRODUÇÃO

Esta dissertação apresenta uma solução para auxiliar o desenvolvimento de aplicações móveis que utilizam a migração de dados, incluindo informações contextuais, dos dispositivos móveis para um ambiente remoto usando uma arquitetura de microsserviços. Em alguns cenários (e.g., aplicações médicas de monitoramento de pacientes), tais aplicações utilizam de dados provenientes de diferentes dispositivos para agregá-los com a finalidade de medir, inferir e compreender seu ambiente de execução. Esta solução é apresentada neste trabalho como um serviço de *offloading* de dados usando uma arquitetura de microsserviços com suporte a anonimização de dados chamada de *Data Offloading service with suPport for data anonymisation based on MicroService* (DOP-MS), e seu desenvolvimento é baseado na evolução de trabalhos anteriores.

1.1 Contextualização

Os últimos anos apresentaram uma grande popularização e crescimento no uso de dispositivos móveis na sociedade (e.g., *smartphones*, *tablets* e *smartwatches*). Esses dispositivos tem cada vez mais capacidade de processamento e uma quantidade cada vez maior de sensores embarcados (e.g., acelerômetro, giroscópio e *Global Positioning System* (GPS), dentre outros), que possibilitam o sensoriamento de dados do ambiente para que estes possam ser interpretados e processados por diversas aplicações.

Porém, a mobilidade proporcionada pelos dispositivos móveis traz consigo limitações impostos para seu uso. Dispositivos móveis para serem portáteis precisam ser compactos e alimentados por bateria (SANTOS *et al.*, 2017b). O uso prolongado dos dispositivos e o aumento na complexidade das aplicações, associado ao uso de um número cada vez maior de sensores impõe que dispositivos sejam recarregados com frequência. Para serem leves e confortáveis ao usuário final, dispositivos móveis possuem limitações como menor capacidade de armazenamento e processamento (limitado ao *hardware* do dispositivo), quando comparados aos tradicionais computadores pessoais. Apesar da substancial melhoria das novas gerações de dispositivos móveis, a quantidade de informações e a complexidade dos procedimentos delegados a estes dispositivos ainda impõe restrições no processamento e consumo de energia. Embora existam dispositivos móveis com maior capacidade de processamento e armazenamento, estes são caros e inacessíveis a maioria da população. Então, as limitações apresentadas estão mais

relacionadas aos dispositivos móveis mais comuns e não aos *smartphones* com maior capacidade de processamento (DINH *et al.*, 2013) (SATYANARAYANAN *et al.*, 2009a).

As limitações previamente apresentadas trazem impacto ao desenvolvimento de aplicações móveis. Uma abordagem para minimizar esses problemas é a *Mobile Cloud Computing* (MCC). De acordo com (DINH *et al.*, 2013), a MCC fornece um conjunto de serviços equivalentes aos da nuvem, adaptados à capacidade de dispositivos com recursos restritos, e com isso trazendo melhorias de desempenho das aplicações nos dispositivos. Na literatura, também existem abordagens para auxiliar na descentralização do processamento de dados e operações, diminuindo o consumo energético dos dispositivos. Uma das mais conhecidas e utilizadas abordagens é o *offloading* (KUMAR; LU, 2010). Essa técnica permite que os dados ou processos sejam migrados de dispositivos com baixo poder de processamento para serem processados por dispositivos com maior poder computacional com objetivo de ganho de desempenho ou economia de recursos.

Ao longo dos últimos anos, várias plataformas de suporte ao *offloading* têm sido propostas (ÁLVAREZ *et al.*, 2019; XIA *et al.*, 2014; COSTA *et al.*, 2015; SOMMER; KOCH, 2020; CUERVO *et al.*, 2010; GOMES *et al.*, 2017a; CÂNDIDO *et al.*, 2019b). Esta dissertação tem especial interesse em dois desses estudos: (*i*) o COP (*Contextual data Offloading service with Privacy support*) (GOMES *et al.*, 2017a), uma solução para a disseminação de dados contextuais em ambientes de MCC, e (*ii*) o *CAOS Microservices* (CAOS-MS) (CÂNDIDO *et al.*, 2019b) que permite a realização do *offloading* de processamento. Ambas soluções são propostas do Grupo de Pesquisa em Engenharia de Software, Redes de Computadores e Sistemas (GREat) da Universidade Federal do Ceará. Apesar das evoluções, como o suporte a manutenibilidade, configuração, implantação e escalabilidade do CAOS-MS, a parte relacionada ao *offloading* de dados ainda apresenta limitação em relação aos dados utilizados nas inferências contextuais, pois estas informações são limitadas apenas aos dados contextuais, inviabilizando o armazenamento de qualquer outro tipo de dado referente à lógica de negócios de uma aplicação móvel(*e.g.*, um registro médico que possui nome do paciente, idade, peso, altura, medicações utilizadas).

1.2 Motivação

Aplicações móveis utilizadas atualmente usam troca de informações (e.g., mensagens) em um modelo de rede social (NAVARRO *et al.*, 2015). Quando utiliza-se esse modelo, um enorme conjunto de dados heterogêneos (e.g., multimídia) é gerado. Esses dados podem

ser levados em consideração em uma tomada de decisão para aplicações móveis e sensíveis ao contexto. Além disso, existem aplicações que utilizam dados de mais de um usuário para melhorar a inferência sobre uma nova situação contextual (YURUR *et al.*, 2014). Em geral, essas aplicações específicas necessitam de um grande conjunto de dados para prover serviços diferenciados aos seus usuários.

Com o aumento da quantidade de dados e informações contextuais monitoradas pelas aplicações móveis modernas, o processamento das inferências e o gerenciamento de contexto tem se tornado cada vez mais complexos. Como dito anteriormente, apesar dos avanços tecnológicos dos dispositivos móveis, eles ainda são limitados se comparados aos *desktops* e também ao poder computacional oferecido na nuvem. Por tais razões, a MCC por meio do *offloading* tem um grande potencial para mitigar esses problemas (SHIRAZ *et al.*, 2013; FERNANDO *et al.*, 2013; SANAEI *et al.*, 2014).

Ao se estudar *offloading*, boa parte dos trabalhos se concentra apenas na migração de processos entre dispositivos móveis e infraestruturas de execução remota para melhoria de desempenho. Porém, o *offloading* de dados também pode contribuir para tais aplicações, abrindo ainda a possibilidade de compartilhamento de informações de usuários através de uma tecnologia de *crowdsourcing*.

Porém, ao se tratar sobre migração de dados, a privacidade destas informações torna-se um fator crucial a ser tratado. É necessário lidar com questões relacionadas à privacidade dos usuários e definir como e quais dados devem ser anonimizados, ou mesmo definir quem pode ter acesso a tais dados. Quando negligenciadas, tais questões podem acarretar na divulgação pública dos dados dos usuários, e inviabilizar o *offloading* em aplicações móveis.

Como dito anteriormente existem várias plataformas de suporte ao *offloading*, uma dessas soluções é o CAOS-MS, uma refatoração do CAOS (GOMES *et al.*, 2017b), que é um *framework* para *offloading* que foi desenvolvido inicialmente em uma arquitetura monolítica, em que os módulos dependem de diversos recursos compartilhados (e.g., memória, banco de dados, arquivos) e não são independentemente executáveis e escaláveis (DRAGONI *et al.*, 2017a). A versão com arquitetura de microsserviços proposta em (CÂNDIDO *et al.*, 2019b) é denominada CAOS-MS e melhora a manutenibilidade, configuração, implantação e escalabilidade dos módulos do CAOS. O CAOS-MS focou apenas na parte relacionada ao *offloading* de processamento do CAOS. Assim, a parte relacionada ao *offloading* de dados não foi alterada.

A arquitetura de microsserviços apresenta diversos benefícios em relação a monolítica, benefícios que serão abordados nos capítulos seguintes desse trabalho. Como (*i*) variedade na forma de gerenciamento de disponibilidade e escalabilidade de partes específicas do sistema, (*ii*) maior capacidade de utilização de diferentes tecnologias, (*iii*) redução do tempo de entrega no mercado e (*iv*) melhor compreensão da base do código (FAN; MA, 2017).

A evolução do CAOS apresentou ganhos significativos relacionados a desempenho e escalabilidade da solução. Portanto, é necessário que os módulos relacionados ao mecanismo de *offloading* de dados sejam alterados. Além disso, é necessário que a solução tenha suporte a migração e armazenamento de qualquer tipo de dado, o que abrange a possibilidade de desenvolvimento de aplicações que tanto realizam inferência contextual como precisam armazenar dados de uso geral (*e.g.*, um aplicativo de gerenciamento de prontuários médicos).

1.3 Objetivos e Contribuições

Dado que o *offloading* de dados é visto como uma estrategia interessante para economizar a capacidade de armazenamento do dispositivo móvel e também realizar o compartilhamento de dados, o objetivo desse trabalho é investigar os benefícios que o *offloading* de dados pode trazer para o desenvolvimento de aplicações móveis e o compartilhamento de dados entre os usuários. Outro objetivo identificado nesse trabalho é realizar uma investigação de como uma arquitetura de microsserviços pode auxiliar no desempenho de escalabilidade em uma solução de *offloading* de dados.

Com base nos objetivos listados anteriormente, o trabalho identifica como o *offloading* de dados pode criar novas possibilidades para o desenvolvimento de aplicações móveis através do compartilhamento de informações entre múltiplos usuários. Esse trabalho pretende oferecer uma solução para *offloading* de dados em que: (a) os dados que serão migrados possam ser selecionados pelo desenvolvedor e (b) possam ser utilizados por múltiplos usuários.

Com o intuito de mitigar os problemas apresentados, este trabalho de mestrado apresenta uma solução para suporte ao *offloading* de dados que permite que o desenvolvedor escolha o que deve ser migrado entre o dispositivo móvel e um ambiente remoto de armazenamento. Essa solução é denominada de DOP-MS, a Data Offloading service with suPport for data anonymisation based on MicroServices. Este serviço é uma evolução da solução COP (GOMES *et al.*, 2017a). Na extensão do serviço de *offloading*, foram incluídas funcionalidades que permitem lidar com:

- O armazenamento dos dados do domínio da aplicação e dados contextuais gerados pelos usuários do sistema;
- Disseminação de dados entre o dispositivo móvel e o ambiente remoto de armazenamento de forma transparente e configurável; e
- A anonimização dos dados de forma ajustável, para restringir a divulgação e difusão de dados de cada usuário do sistema.

E por fim, é realizado uma avaliação da solução desenvolvida através de uma prova de conceito utilizando uma *toy application*, e também testes de desempenho e escalabilidade.

1.4 Metodologia

A metodologia utilizada na construção desse trabalho, foi organizada seguindo as seguintes etapas:

- **Revisão da literatura:** foi realizada uma pesquisa envolvendo os conceitos de *offloading* de dados, *mobile cloud computing*, *frameworks de offloading* e também trabalhos que se relacionassem mais especificamente ao contexto de migração de arquiteturas, processos, técnicas e tecnologias utilizadas nos processos de migração de arquiteturas monolíticas para a abordagem de microsserviços;
- **Seleção e análise dos trabalhos relacionados:** Foi realizada uma seleção de trabalhos relacionados ao trabalho aqui apresentado. Estes trabalhos foram estudados, analisados e comparados em relação à proposta aqui apresentada;
- **Escolha do processo de desenvolvimento/ferramentas/tecnologias utilizadas para desenvolvimento do DOP-MS:** a partir da revisão de literatura e seleção dos trabalhos relacionados, foram selecionadas as tecnologias, práticas e abordagens que poderiam ser utilizadas na construção desse trabalho;
- **Analise do problema:** foi realizado um estudo sobre o serviço de *offloading* de dados, identificados os requisitos necessários para a solução. Posteriormente foi realizado uma priorização dos requisitos funcionais(*e.g.*, compartilhamento de dados, definição dos dados) e não funcionais (*e.g.*, escalabilidade, privacidade);
- **Projeto da solução**: após a analise do problema foi projetada a arquitetura da solução;
- **Implementação do DOP-MS:** foi realizado a implementação da solução baseada em uma arquitetura de microsserviços;
- **Experimentos:** por fim, foi desenvolvida uma aplicação chamada Medical App como

prova de conceito para o serviço proposto nesse trabalho. Também foram realizados testes de desempenho e escalabilidade na solução e em ambas arquiteturas, a fim de verificar quais contribuições a solução baseada em microsserviços poderia trazer em relação a versão monolítica.

1.5 Organização da Dissertação

Esta dissertação está organizada em seis capítulos. Este capítulo descreveu uma breve introdução ao tema, contextualizando o assunto abordado, a motivação, os objetivos, as contribuições e a metodologia deste trabalho.

O Capítulo 2 trata da fundamentação teórica, apresentando os conceitos importantes adotados na pesquisa. Dentre eles, pode-se citar: *offloading* de dados, arquitetura de microsserviços e politicas de anonimização de dados.

O Capítulo 3 apresenta os trabalhos relacionados a solução proposta. Foram analisados trabalhos sobre ferramentas de *offloading* e compartilhamento de dados.

O Capítulo 4 apresenta o serviço proposto, bem como sua arquitetura, além das políticas de sincronização e anonimização utilizadas. Também é apresentado um cenário motivador para o trabalho.

O Capítulo 5 apresenta uma prova de conceito com exemplos de código de utilização do serviço proposto. Neste capítulo também são detalhados as metodologias, procedimentos e resultados obtidos tanto na avaliação do mecanismo de recuperação dos dados como na avaliação do envio desses dados do dispositivo móvel para o ambiente remoto.

Por fim, o Capítulo 6 descreve os resultados alcançados por este trabalho, assim como as suas conclusões e as possíveis melhorias a serem consideradas em trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A proposição de um serviço de *offloading* de dados é baseada em uma série de tópicos. Este capítulo apresenta os principais conceitos que fundamentam esta dissertação de mestrado. Primeiramente, a Seção 2.1 apresenta os conceitos relacionados a computação móvel em nuvem. As Seções 2.1.1 e 2.1.2 apresentam respectivamente uma contextualização sobre *offloading* e privacidade de dados. Na Seção 2.1.3 é apresentado o que é *crowdsourcing* e na Seção 2.2 é apresentado o estilo arquitetural de microsserviços.

2.1 Computação móvel em nuvem

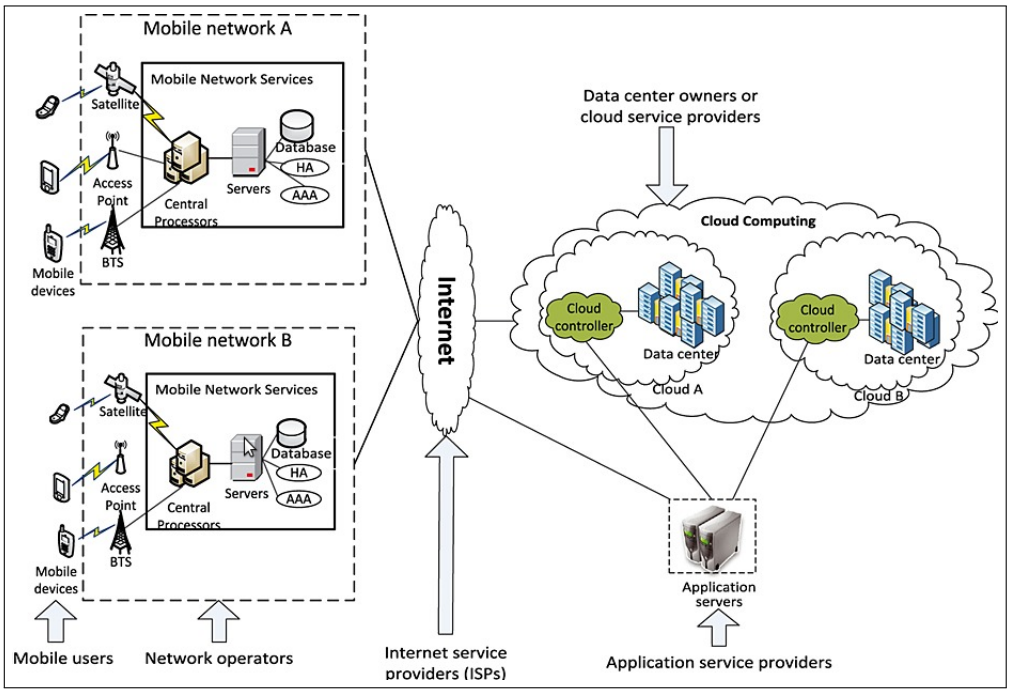
Computação móvel em nuvem (do inglês, *Mobile Cloud Computing - MCC*) é um paradigma que incorpora duas tecnologias heterogêneas: (*i*) computação móvel e (*ii*) computação em nuvem (FERNANDO *et al.*, 2013). MCC visa contornar as limitações dos dispositivos móveis em relação ao desempenho e consumo de energia, por meio do uso de recursos da nuvem para aumentar tanto a capacidade de computação quanto a capacidade de armazenamento desses dispositivos.

Segundo Dinh *et al.* (2013), a MCC busca prover um conjunto de serviços equivalentes ao da nuvem, porém adaptados à capacidade de dispositivos com recursos limitados, por meio de melhorias na infraestrutura de telecomunicação, de modo a melhorar o provisionamento destes serviços. Segundo Othman *et al.* (2013), MCC trata-se de uma integração da tecnologia de computação em nuvem com dispositivos móveis para tornar os dispositivos móveis cheios de recursos em termos de poder computacional, memória, armazenamento, energia e consciência de contexto.

A Figura 1 apresenta a arquitetura da MCC desenvolvida pelos autores do trabalho (DINH *et al.*, 2013). Nessa figura, os dispositivos móveis são conectados às redes móveis por meio de estações base (e.g., estação transceptora base, ponto de acesso ou satélite) que estabelecem e controlam as conexões (ligações aéreas) e interfaces funcionais entre as redes e dispositivos móveis. As solicitações e informações dos usuários móveis (e.g., ID e localização) são transmitidas aos processadores centrais que estão conectados aos servidores que fornecem serviços de rede móvel. As operadoras de rede móvel podem fornecer serviços a usuários móveis como autenticação, autorização e contabilidade com base no agente doméstico e nos dados dos assinantes armazenados em bancos de dados. Depois disso, as solicitações dos assinantes são

enviadas para uma nuvem por meio da Internet. Na nuvem, os controladores de nuvem processam as solicitações para fornecer aos usuários móveis os serviços de nuvem correspondentes.

Figura 1 – Arquitetura da *Mobile Cloud Computing*



Fonte: Dinh *et al.* (2013)

Na realidade, não existe um consenso sobre um conceito único para MCC. Para Shiraz *et al.* (2013), MCC é “o mais recente paradigma computacional prático, que estende a visão da computação utilitária de computação em nuvem para aumentar os recursos limitados dos dispositivos móveis". Já segundo Sanaei *et al.* (2014), MCC apresenta-se como “uma rica tecnologia de computação móvel que utiliza recursos elásticos unificados de nuvens variadas e tecnologias de rede com funcionalidade irrestrita, armazenamento e mobilidade para servir uma multidão de dispositivos móveis em qualquer lugar, a qualquer momento através do canal de *ethernet* ou internet independentemente de ambientes heterogêneos e plataformas baseadas no princípio *pay-as-you-use*”.

Uma das principais motivações para o uso da MCC é permitir que dispositivos móveis com configurações não tão poderosas de *hardware* (*e.g.*, processador, memória, etc.) possam executar tarefas complicadas ou computacionalmente custosas (DENG *et al.*, 2014). Isso se deve por existir técnicas da MCC que são usadas para solucionar/melhorar os serviços móveis. Dentre as inúmeras técnicas desenvolvidas para a MCC, destaca-se o *offloading*, técnica que está diretamente relacionada a esta dissertação de mestrado.

2.1.1 Offloading

O *offloading* é uma abordagem que visa aumentar o desempenho e reduzir o consumo de energia de dispositivos móveis por meio da migração de processamento ou dados de dispositivos móveis para outras infraestruturas, com maior poder computacional e de armazenamento (FERNANDO *et al.*, 2013). É importante destacar que *offloading* é diferente do modelo cliente-servidor tradicional, em que um dispositivo cliente sempre delega a responsabilidade de realizar o processamento ao servidor remoto. No *offloading*, quando não há conexão entre dispositivo móvel e servidor ou não há ganhos na migração de uma tarefa ou dados, o processamento ainda é realizado localmente pelo dispositivo móvel. Já quando existe a conexão que permite comunicação com uma infraestrutura remota e existem ganhos na migração de dados ou processamento, o *offloading* é realizado.

Conforme destacado em (FERNANDO *et al.*, 2013), existem dois tipos principais de *offloading* citados na literatura: (*i*) processamento e (*ii*) dados. O *offloading* de processamento é a delegação de um processamento computacional do dispositivo móvel para outro ambiente de execução (e.g., *laptop*), a fim de prolongar a vida útil da bateria de um dispositivo cliente ou então diminuir o tempo de processamento de uma tarefa. Já o *offloading* de dados tem por objetivo estender a capacidade de armazenamento do dispositivo móvel, retirando dados do armazenamento local de um dispositivo e enviando-o para armazenamento em um dispositivo ou servidor com maior capacidade de armazenamento. Estes dados podem ser processados e reenviados para o dispositivo móvel.

Segundo Silva (2019), a utilização do *offloading* é defendida pois apesar de avanços em diversos componentes dos dispositivos móveis, eles ainda possuem recursos limitados se comparados aos tradicionais computadores pessoais. Assim sendo, Yi *et al.* (2015) afirmam que o *offloading* de computação pode superar as restrições de recursos em dispositivos móveis, porque algumas tarefas de computação intensiva podem se beneficiar do *offloading* no desempenho de aplicativos, economizando armazenamento e vida útil da bateria.

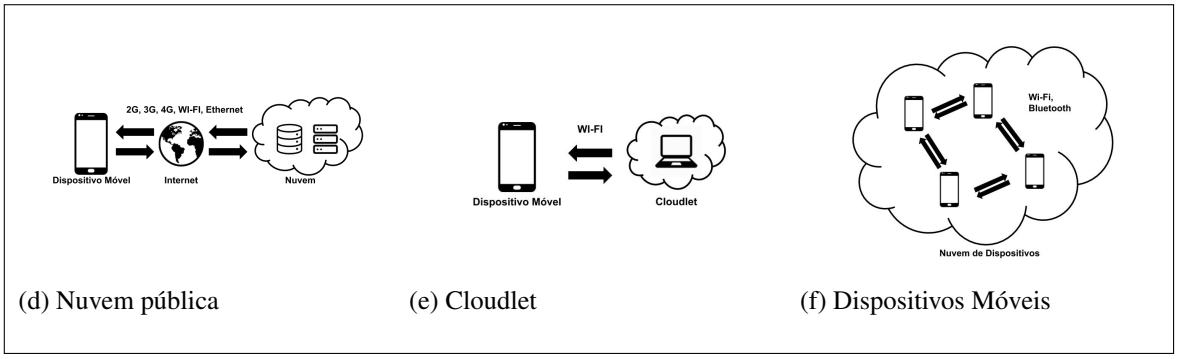
Ao se migrar uma tarefa ou dado, existem diversas questões que precisam ser respondidas em projetos de aplicações móveis com suporte ao *offloading*. Uma delas é do local para onde a tarefa ou dado deve ser migrado. Em geral, existem três possíveis alternativas para a realização de um *offloading*, quer seja de dados ou de processamento. São eles: *(i)* uma nuvem pública; *(ii)* um *cloudlet*; e *(iii)* outro dispositivo móvel, conforme representado pela Figura 2.

Na realização de procedimentos de *offloading* tendo uma nuvem pública como alvo,

os dispositivos móveis podem utilizar tanto redes celulares (*e.g*, 2G, 3G e 4G) ou conexões por redes Wi-Fi para migrar seus processos ou dados para máquinas virtuais ou servidores reais, hospedadas em provedores públicos ou privados e administrados por terceiros (GOMES *et al.*, 2016). Esta opção sofre pela latência inerente à separação geográfica entre dispositivos clientes e servidores.

Como resposta ao atraso na comunicação entre dispositivos clientes e servidores, Satyanarayanan *et al.* (2009b) definiram o conceito de *cloudlet*. Este conceito consiste em um dispositivo, computador ou *cluster* de computadores, rico em recursos e confiável, disponível para uso por dispositivos móveis que estejam próximos, geralmente na mesma rede local sem fio. Para Gomes *et al.* (2016), a vantagem da execução em *cloudlet* é que as redes Wi-Fi, em geral, possuem velocidades maiores e latência menores em relação às redes celulares. Por fim, como a figura 2 (c) mostra, o *offloading* pode ser executado em outro dispositivo móvel. Este último cenário remete a situações onde não há uma infraestrutura pré-estabelecida, e tecnologias de comunicação *ad-hoc* como (*e.g*,Wi-Fi, Bluetooth e ZigBee) podem ser utilizadas para a conexão entre os dispositivos envolvidos (SANTOS *et al.*, 2017a).

Figura 2 – Locais de Execução do *Offloading*



(d) Nuvem pública (e) Cloudlet (f) Dispositivos Móveis

Fonte: Gomes *et al.* (2016)

Segundo Silva (2019), não existe um consenso sobre os critérios a serem analisados para saber quando realizar ou não o *offloading*, pois a melhoria no desempenho desejada depende do contexto em que a aplicação está inserida. Desta forma, um critério como a condição da rede pode ser relevante para uma aplicação que necessita de uma resposta rápida e nem tanto para os demais tipos de aplicações. Entretanto, em (KUMAR *et al.*, 2013) é discutido a questão de quando decidir o *offloading* de processamento baseado no particionamento de uma aplicação móvel, na qual pode ser de duas formas: (*i*) estática ou (*ii*) dinâmica. Quando o particionamento da aplicação é estático, o programa é particionado em tempo de desenvolvimento, e a decisão já

definida no projeto da aplicação. Já no particionamento dinâmico, as decisões podem se adequar a diferentes condições de tempo de execução (*e.g*, largura de banda de rede flutuantes). Neste último caso, faz-se necessário que uma tomada de decisão seja realizada com base em critérios. De acordo com (DENG *et al.*, 2014), dentre os diversos fatores que podem afetar adversamente a eficiência das técnicas de *offloading*, destaca-se a limitação da largura de banda entre dispositivos móveis e servidores da nuvem e a quantidade de dados que devem ser trocados entre eles.

2.1.1.1 Offloading de dados

De acordo com (SANTOS *et al.*, 2017b), *offloading* de dados tem como objetivo estender a capacidade de armazenamento do dispositivo móvel. É retirado o armazenamento do dispositivo e enviam-se os dados para um ambiente com maior capacidade de armazenamento, também tendo a possibilidade de enviar dados processados de volta para o dispositivo móvel.

Com esta migração de dados, além de "aliviar" a carga dos dispositivos móveis, abrem-se novas possibilidades para aplicações pelo compartilhamento e processamento de informações de múltiplos usuários. Porém, este repasse de dados também traz uma série de desafios, sendo o principal deles relacionado à privacidade dos dados transmitidos entre os dispositivos móveis e os servidores do *offloading*. Estas informações podem variar desde dados contextuais até informações pessoais de uma aplicação específica.

2.1.2 Privacidade dos dados

O conceito de privacidade está ligado ao isolamento, intimidade, algo que é privado, que é secreto ou restrito, isto é, "Vida privada; intimidade, privatividade" (SAMPAIO; MATOS, 2019).

De acordo com (JR *et al.*, 2014), existem três elementos básicos em privacidade: *(i)* confidencialidade, *(ii)* anonimato e *(iii)* isolamento. A confidencialidade é a garantia do resguardo das informações dadas pessoalmente em confiança e proteção contra a sua revelação não autorizada. O anonimato é a qualidade ou condição do que é anônimo, isto é, sem nome ou assinatura. Com o advento das mensagens por telecomunicações e, em particular, pela Internet, designa o ato de manter uma identidade escondida de terceiros. Por fim, o isolamento determina como a integridade da transação é visível para outros usuários e sistemas (JR *et al.*, 2014).

A lei brasileira 12.965, promulgada em 23 de abril de 2014 (também conhecida como Marco Civil da Internet) não especifica exatamente como proteger a privacidade da informação, mas estabelece princípios, garantias, direitos e deveres para o uso da Internet no Brasil. Em seu art. 3o , parágrafos II e III, apresenta-se o princípio da proteção da privacidade e segurança dos dados pessoais. Em 14 de agosto de 2018, o congresso nacional do Brasil aprovou a nova lei 13.709, denominada LGPD - Lei Geral de Proteção de Dados com previsão de início de vigência em 2020 (MENDES; DONEDA, 2020).

A privacidade da informação tem importância ainda maior em determinados domínios. Por exemplo, a privacidade de dados em aplicações médicas é considerado um aspecto crítico, sendo por exemplo, formalmente regulamentada em certos países, como no caso dos Estados Unidos, por meio do *Health Insurance Portability and Accountability Act* (HIPAA) (O'HERRIN *et al.*, 2004). Este ato estabelece padrões para proteger registros médicos de indivíduos ou qualquer informação pessoal de saúde aplicável aos planos de saúde e profissionais da área que realizam transações eletrônicas, garantindo: *(i)* custódia segura (física e lógica); *(ii)* limites; *(iii)* condições; e *(iv)* autorização de uso da informação (SILVA *et al.*, 2019b).

2.1.2.1 Anonimização de dados

De acordo com (SILVA *et al.*, 2019b) uma das principais estratégias na busca da proteção da privacidade é a anonimização de dados. O processo de anonimização visa mascarar ou ofuscar os dados antes destes serem disponibilizados ou compartilhados (SILVA *et al.*, 2017). Quando aplicada corretamente, as técnicas e modelos de anonimização podem evitar a recuperação da identidade de indivíduo.

De acordo com (SILVA *et al.*, 2017), um passo importante para a anonimização é definir quais conjuntos de dados (atributos) devem ser anonimizados e quais técnicas devem ser aplicadas a cada um deles. Os atributos devem, portanto, ser classificados de acordo com a sensibilidade da informação que cada um representa.

A classificação dos atributos está dividida em: *(i)* atributos identificadores, que identificam os indivíduos (e.g, nome, CPF, RG); *(ii)* atributos semi-identificadores, que, se combinados com informações externas, expõem indivíduos ou aumentam a certeza sobre suas identidades (e.g, data de nascimento, CEP, cargo, tipo sanguíneo); e *(iii)* atributos sensíveis, que se referem a condições específicas dos indivíduos (e.g., salário, exames médicos) (CAMENISCH *et al.*, 2011).

2.1.2.2 Técnicas de anonimização

Depois de identificar os atributos de acordo com a sua sensibilidade de divulgação (identificadores, semi-identificadores e sensíveis), técnicas de anonimização podem ser aplicadas para proteger a identidade dos indivíduos (OHM, 2009) (JR *et al.*, 2014). Abaixo são listadas as principais técnicas de anonimização apresentadas no trabalho de Silva *et al.* (2019b):

- **Supressão**: é a remoção completa do atributo, isto é, a exclusão da coluna correspondente aos dados a serem anonimizados. Geralmente, essa técnica é utilizada nos dados identificadores;
- **Generalização**: a ideia de generalização é, em vez de excluir, manter apenas parte dos dados. A técnica de generalização pode ser uma boa opção quando se busca o equilíbrio entre utilidade e privacidade. Como exemplo, podemos citar o CEP, cujos números representam o escopo geográfico, ou seja, quanto mais à esquerda o número, maior o seu alcance;
- **Agregação**: trata-se da disseminação de dados estatísticos resumidos, ou seja, dados não brutos, para liberar estatísticas agregadas, protegendo os indivíduos contra a reidentificação;
- **Criptografia**: consiste em usar cifragem para ocultar os dados reais;
- **Distúrbio**: conhecido como mascaramento, substitui os valores reais por dados fictícios;
- **Substituição**: técnica na qual dados são substituídos por outros que não estão relacionados aos dados originais;
- **Embaralhamento**: os itens são embaralhados aleatoriamente por dados semelhantes, mas

da mesma tabela onde estão armazenadas as informações;

- **Anulação**: também conhecido como “truncamento”, indica que os valores originais são substituídos por dados nulos.

De acordo com os autores do trabalho (BRITO; MACHADO, 2017), as técnicas mais utilizadas na grande maioria dos casos, são técnicas de supressão ou de generalização.

2.1.3 Crowdsourcing

A massificação das redes de comunicação e a possibilidade de produção de dados por usuários finais fez com que um novo cenário se criasse para aplicações modernas, o chamado *crowdsourcing*. Neste modelo, a produção de dados para aplicações é feita de forma coletiva por usuários, geralmente de forma voluntária, em que tal esforço provê informação enriquecida que beneficia a todos que utilizam uma dada aplicação ou serviço. Um típico exemplo do uso de *crowdsourcing* é o Waze[1], aplicativo bastante popular para mobilidade urbana que indica as vias com melhor condição de circulação a partir de informação recebida de seus milhões de usuários. Quanto mais usuários participam no sistema, melhor a qualidade da informações sobre o trânsito nas ruas. De fato, *crowdsourcing* tem uma profunda dependência na base de voluntários para gerar informação de valor.

De acordo com Mateveli *et al.* (2015), chama-se *crowdsourcing* as iniciativas que visam coletar dados com a ajuda de grande quantidade de colaboradores humanos. Aplicações *Volunteered Geographic Information* (VGI) podem ser considerado um tipo especial de *crowdsourcing* em que os dados de interesse são geograficamente localizados. Quando os cidadãos utilizam sensores ambientais acoplados a dispositivos moveis, o termo é adaptado para *crowdsensing*, e os dados passam a ser medicões realizadas pelos sensores embarcados em *smartphones* ou *tablets*.

No trabalho de Mateveli *et al.* (2015), é apresentado uma classificação de *crowdsourcing* e *crowdsensing* de acordo com duas dimensões: *(i)* a forma de captura dos dados e *(ii)* o nível de colaboração do usuário. Essa classificação é apresentada a seguir:

- ***Crowdsourcing* ativo**: o usuário fornece informações conscientemente, tipicamente de forma voluntaria, sabendo como a mesma será utilizada, como no caso do *OpenStreetMap* [2].

[1] https://www.waze.com/
[2] https://www.openstreetmap.org/

- ***Crowdsourcing* passivo**: a informação é extraída a partir de material postado pelo usuário para outras finalidades, ou seja, dados úteis eventualmente contidos no material postado para outras finalidades são usados sem o conhecimento explícito do usuário. Um exemplo comum é a coleta automática de *tweets* associados a *hashtags* específicas.
- ***Crowdsensing* ativo**: o usuário fornece ativamente informações que é capturada por sensores embutidos em seu dispositivo móvel. Um exemplo é o ato de fazer *check-in* em um ponto de interesse com posição determinada por GPS.
- ***Crowdsensing* passivo**: a informação é capturada por sensores em dispositivos móveis, mas sem a necessidade de interação com o usuário. Um exemplo é a captura de outros dados através de sensores embarcados em *smartphones*.

2.2 Estilo Arquitetural de Microsserviços

Sommerville (2011) define que arquitetura de software é a organização fundamental de um sistema, incorporando em seus componentes, suas relações entre si e com o meio ambiente, e os princípios que regem seu design e evolução. A arquitetura é o elo critico entre o projeto e a engenharia de requisitos, pois identifica os principais componentes estruturais de um sistema e os relacionamentos entre eles.

Existem algumas características importantes em uma boa arquitetura de software. Dentre elas está a sua modularidade. De acordo com (RICHARDSON, 2018), modularidade significa projetar os componentes da aplicação separadamente e independentemente. Segundo (LINDVALL *et al.*, 2002), modularização é fundamental em arquiteturas de software. A proposta da modularização é dividir o sistema em pequenas partes, proporcionando um sistema mais legível e com maior manutenibilidade.

Uma arquitetura modular multiplica e descentraliza as partes principais de um software, fazendo com que tenha um acréscimo de valor em um projeto que utiliza a modularidade (BALDWIN; CLARK, 2000). De acordo com (SANT'ANNA *et al.*, 2007), a complexidade de um projeto de arquitetura de software, é em grande parte, causada pela modularização indevida dos componentes do sistemas.(e.g., tratamento de exceções e persistência).

Quando um sistema possui módulos independentes ele pode ser mantido da forma que está, sendo mais fácil alterações individuais ou até mesmo em todos os módulos do sistema. De acordo com (BALDWIN; CLARK, 2000), as decisões de alterações são descentralizadas do ponto de vista de que os projetistas encarregados pelos módulos possam tomar decisões de

alteração de forma autônoma. Portanto, um projeto modular diminui a complexidade, simplifica as eventuais mudanças e retorna uma implementação mais fácil ao incentivar o desenvolvimento simultâneo de inúmeras partes de um sistema.

Quando falamos em arquitetura de software, uma das mais tradicionais e conhecidas é a arquitetura monolítica, que é utilizada para se desenvolver software (SOMMERVILLE, 2011). Dentre suas características podemos ressaltar uma base de código única para atender a vários serviços e interfaces diferentes. Os softwares que são desenvolvidos com esse modelo de arquitetura, tem uma única base de código para atender a vários serviços e interfaces diferentes. Quando temos uma única base de código, há também uma maior facilidade o desenvolvimento, a implantação e a escalabilidade do aplicativo, desde que o tamanho da base de código seja pequena. Uma base de código monolítica é uma boa opção no início do projeto devido às qualidades mencionadas acima e porque não há distribuição de código que possa adicionar complexidade (KALSKE, 2019).

Nas arquiteturas monolíticas, o desenvolvimento do software pode ser bastante simples no início, mas à medida que o sistema cresce, sua complexidade também aumenta. Uma maneira típica de lidar com a complexidade em uma aplicação que possui uma arquitetura monolítica é dividir o aplicativo em diferentes camadas (KALSKE, 2019).

(DRAGONI *et al.*, 2017a) afirma que assim como boa parte das arquiteturas escaláveis, a escalabilidade de um software monolítico é feita adicionando novos nós do sistema. Embora isso simplifique a escalabilidade, também limita este recurso. Os componentes do software que precisam de mais recursos computacionais precisam ser escalados juntamente com os componentes que podem preencher sua carga de trabalho com menos recursos. Isso significa custos adicionais para a organização, já que os recursos necessários para instanciação do software crescem de acordo com o tamanho da aplicação.

Além disso, Dragoni *et al.* (2017b) afirmam que geralmente, no momento em que arquiteturas monolíticas são submetidas a cargas sucessivas de dados, existe uma dificuldade acentuada em identificar quais dos componentes do sistema estão sendo realmente afetados, visto que a execução do sistema é em um único processo. Isto implica diretamente na escalabilidade, pois mesmo que apenas um único componente esteja sofrendo sobrecarga, todos os módulos do software terão que ser escalados. Ainda que for sabido qual o componente está sofrendo sobrecarga, não é possível escalar de forma isolada.

A arquitetura monolítica necessita lidar com circunstâncias de utilização do sistema

em que o número de usuários supera a capacidade do servidor. Além do mais, arquiteturas monolíticas são difíceis de gerenciar e manter, em consequência da falta de ferramentas que visem a modularização (FAN; MA, 2017). Para (VILLAMIZAR *et al.*, 2015), escalar sistemas monolíticos é um desafio, tendo em vista que eles frequentemente apresentam diversos serviços, alguns deles mais utilizados que outros. Se os serviços utilizados com maior frequência necessitarem ser redimensionados devido ao fato de serem altamente exigidos, todo o conjunto de serviços também terá que ser redimensionado simultaneamente, o que acarreta que os serviços que não são utilizados com tanta frequência no sistema, passe a consumir grande quantidade de recursos do servidor, mesmo quando não estão em uso.

De acordo com Richardson (2018), à medida que o tamanho da base de código aumenta, fica mais difícil adicionar novas funcionalidades e modificar as funcionalidades antigas, porque o desenvolvedor precisa encontrar o local correto para aplicar essas alterações, resultando em ciclos de desenvolvimento mais lentos.

Existem diversas abordagens que podem ser aplicadas na criação de componentes e serviços reutilizáveis. *Service oriented architecture* (SOA) pode ser visto como um exemplo, onde a necessidade de reutilização e robustez foi combinada com a obrigatoriedade de interoperabilidade entre os serviços, surgindo a ideia de um serviço como uma entidade de software via comunicação de mensagens, utilizando formatos e protocolos de dados padrão (e.g., XML, SOAP e HTTP) (DRAGONI *et al.*, 2017b). Outra maneira de desenvolver softwares com alto poder de manutenibilidade, reusabilidade e escalabilidade, e que recentemente tem ganhando muito espaço entre os desenvolvedores e instituições é a arquitetura baseada em microsserviços (FOWLER; LEWIS, 2014).

O estilo arquitetural de microsserviços surgiu empiricamente a partir de padrões arquiteturais utilizados no mundo real, onde sistemas são compostos por serviços que colaboram entre si para atingir seus objetivos, se comunicando a partir de mecanismos leves (e.g., APIs WEB). (LEWIS; FOWLER, 2014) definem o estilo arquitetural de microsserviços como uma abordagem para o desenvolvimento de uma única aplicação formada por um conjunto de pequenos serviços microsserviços, cada um rodando em seu próprio processo *(container)* e se comunicando geralmente por meio de uma API HTTP. Esses serviços são construídos em torno de uma parte específica do negócio e são implantados de forma completamente automatizada. Eles podem ser escritos em diferentes linguagens de programação e usar diferentes tecnologias de banco de dados. Existem também uma camada que centraliza o gerenciamento desses serviços.

Para (SOUSA, 2016), o estilo arquitetônico de microsserviços é cada vez mais utilizado na computação em nuvem. Essa abordagem promove a decomposição de aplicações monolíticas em pequenos serviços que podem ser independentemente escalados, otimizando a utilização dos recursos. (THÖNES, 2015) diz que os microsserviços representam uma pequena aplicação que possibilita o desenvolvimento, implantação, elasticidade e teste de forma independente. O autor relata ainda que um microsserviço deve possuir uma única responsabilidade executada de forma independente, possibilitando um código base simples e de fácil compreensão.

De acordo com (SUN *et al.*, 2015), a arquitetura de microsserviços desagrega softwares monolíticos complexos em um conjunto de serviços pequenos e autônomos que trabalham em conjunto. A decomposição permite que diferentes serviços sejam construídos, implantados, gerenciados e evoluídos de forma independente. Ao decorrer dessa desagregação, as chamadas das funções dos componentes são substituídas por comunicações leves entre serviços, podendo ser implementadas através de interfaces *API* bem definidas.

De acordo com Zimmermann (2017), Mehta e Heineman (2005), Fehling *et al.* (2014), existem uma série de princípios para o uso de microsserviços. Estes incluem (*i*) componentes com interfaces com granularidade fina, com unidades com responsabilidades específicas e independentes, (*ii*) Práticas de desenvolvimento orientadas ao negócio, com técnicas e princípios para identificar e conceituar serviços, (*iii*) Princípios de design baseados em computação em nuvem, incorporando distribuição, elasticidade e baixo acoplamento, (*iv*) entrega contínua descentralizada, de modo a promover um alto nível de automação e autonomia, (*v*) uso de metodologias de conteinerização de componentes leves, (*vi*) uso de técnicas e automatização de configuração, desempenho e gerenciamento de falhas, estendendo práticas ágeis para o monitoramento de serviços, em um prática popularmente conhecida como DevOps e (*vii*) suporte a Múltiplos paradigmas, o que combina ganhos de diferentes abordagens de computação e de tipos distintos de armazenamento;

Existem vários benefícios decorrentes da escolha de uso da arquitetura de microsserviços, porém neste estudo vamos usar como base um relato de experiência baseado no processo de adoção desse modelo em projetos, apresentado a seguir (LUZ *et al.*, 2018):

- **Desenvolvimento independente**: como o desenvolvimento agora está concentrado em pequenas unidades independentes de software de trabalho, as equipes de software podem trabalhar de forma independente. Por exemplo, uma equipe de software pode trabalhar na unidade responsável pelo pagamento sem interferir ou mesmo conhecer a existência

da equipe responsável por desenvolver a parte de *download*. Essa separação de serviços podem resultar em menos conflitos e problemas de comunicação. Além disso, uma determinada unidade pode ser implementada em uma linguagem de programação mais apropriada, permitindo que os desenvolvedores explorem completamente os benefícios de outras linguagens, estruturas e ferramentas de programação;

- **Implantação independente**: como os serviços são pequenos e independentes, podem ser escritos em diferentes linguagens de programação, eles podem ser implantados em diferentes *containers*. Esse benefício promove lançamentos rápidos de recursos, alterações de configuração ou correções de erros de maneira sustentável e eficiente;
- **Escalabilidade independente**: como as implantações são independentes, cada serviço pode ser dimensionado independentemente. Se um serviço estiver enfrentando mais demandas do que outro, esse serviço poderá ser ampliado, por exemplo, em uma plataforma de nuvem, completamente independente, que pode realizar o processamento por demanda, para lidar com o aumento da carga. Há também um ganho monetário nessa abordagem: como não é necessário escalar o aplicativo inteiro, pode ser necessário um número menor de instâncias de *containers* para atender à demanda.

Embora muitos outros benefícios tenham sido reivindicados e discutidos, como a reutilização e a manutenção do código, diminuição da curva de aprendizado, evitar o comprometimento de longo prazo com uma única tecnologia, existem poucos estudos avaliando se esses benefícios são realmente percebidos na prática e, em caso afirmativo, quais são os desafios que as empresas de software enfrentam para alcançá-los (LUZ *et al.*, 2018).

Apesar das vantagens obtidas com a arquitetura de microsserviços, existem diversos fatores a serem levados em consideração antes de adotar essa arquitetura. Para Balalaie *et al.* (2016), o projeto com microsserviços não é uma bala de prata. Os autores afirmam que a motivação a qual os levaram a migrar para essa arquitetura foi a alta flexibilidade obtida e auxílio das ferramentas *Spring Cloud*(SPRING, 2018) e *Netflix OSS*(NETFLIX, 2018). Contudo, ao adotar microsserviços, várias complexidades foram introduzidas no sistema, exigindo um esforço considerável para resolvê-las (e.g., serviço de registro e descoberta, padronização de protocolos de comunicação e gerenciamento de diversas base de dados).

Em (ESPOSITO *et al.*, 2016), (FAN; MA, 2017), (BALALAIE *et al.*, 2015) e (TAIBI *et al.*, 2017), é afirmado que a migração de uma aplicação monolítica para uma abordagem de microsserviços não é nada trivial. Pode-se perceber que existem diversos pontos a serem

observados em ambas arquiteturas. Por ser uma área nova, não há uma definição padrão do que pode, ou não, ser migrado para arquitetura de microsserviços. A adoção a essa abordagem, sempre vai depender do contexto e necessidades do software (DRAGONI *et al.*, 2017a).

2.3 Considerações Finais

Este capítulo apresentou alguns temas que fundamentam esta dissertação de mestrado. O serviço proposto enquadra-se no campo da *Mobile Cloud Computing* ao promover o *offloading* de dados entre dispositivos móveis e infraestruturas remotas de armazenamento e processamento, com objetivo de aliviar recursos dos dispositivos clientes.

O *offloading de dados* indiretamente pode contribuir também para suporte ao *crowsourcing*, uma vez que dados enviados por vários usuários podem ser utilizados de forma colaborativa em prol de uma comunidade de usuários que utiliza uma mesma aplicação. Porém é importante que esta colaboração garanta a privacidade de informações sensíveis dos usuários, de modo a não desencorajá-los a compartilhar dados.

Por fim, a arquitetura baseada em microsserviços tem se mostrado uma opção adequada para melhorar a escalabilidade e manutenibilidade de aplicações por meio da divisão das funções em serviços independentes. Com isso, torna-se uma escolha bem apropriada para o serviço a ser implementado, uma vez que as técnicas de *crowdsourcing* demandam um número razoável de dados e usuários.

O próximo Capítulo apresenta os trabalhos relacionados que envolvem soluções que visam o desenvolvimento de solução que dão suporte ao *offloading* de dados.

3 TRABALHOS RELACIONADOS

Neste capítulo são apresentados alguns dos estudos encontrados na literatura que se relacionam com a proposta deste trabalho. As pesquisas encontradas visaram explanar sobre o desenvolvimento de aplicações móveis que compartilham dados e fazem uso de filtros para inferir situações. A seção 3.1 apresenta esses trabalhos relacionados, enquanto a seção 3.2 apresenta um comparativo entre esses trabalhos, bem como as características consideradas relevantes durante o estudo deles.

3.1 Sintetização dos estudos relacionados

Os trabalhos aqui reportados foram encontrados a partir de buscas em bases de dados científicas por trabalhos acadêmicos relacionados ao tema de *offloading de dados*. As bases utilizadas nesta busca incluem Google Scholar[1] (como um centralizador), ACM Digital Library[2], IEEE Explorer [3] e Scopus [4]. Além destas bases, foram consultadas algumas das principais conferências nacionais e internacionais, bem como periódicos onde o tema de *offloading* de dados estivesse direta ou indiretamente relacionado aos tópicos de pesquisa destes fóruns. De forma sucinta, buscou-se por trabalhos que mencionassem explicitamente o suporte ao *offloading* de dados ou mecanismos similares, como *crowdsourcing*.

Nesta pesquisa, os termos que foram utilizados como critério de busca incluíram: *data Offloading*, *offloading* de dados, *framework* de *offloading* de dados e *crowdsourcing micro-service*. Ao todo foram encontrados oito artigos, sendo quatro extensões de um deles, fazendo com que os mesmos fossem descartados. Além destes trabalhos, esta seção também apresenta dois trabalhos que servem de ponto de partida para especificação do serviço. São eles: (*i*) o COP (GOMES *et al.*, 2017a) e o (*ii*) CAOS-MS (CÂNDIDO *et al.*, 2019a).

3.1.1 A service-oriented approach to crowdsensing for accessible smart mobility scenarios (SMAll)

No estudo conduzido em (MIRRI *et al.*, 2016), os autores apresentam uma arquitetura para ajudar a projetar e implementar aplicativos de mobilidade inteligente. De acordo com

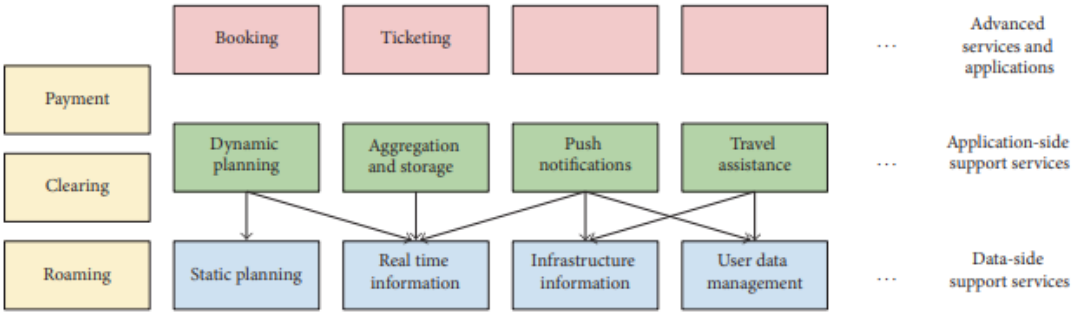
[1] https://scholar.google.com.br/
[2] https://dl.acm.org/
[3] https://ieeexplore.ieee.org/Xplore/home.jsp
[4] https://www.scopus.com/home.uri

os autores, o trabalho tem por objetivo: *(i)* coletar dados relacionados a barreiras urbanas e instalações, *(ii)* calcular caminhos personalizados para usuários com necessidades especiais e *(iii)* integração de dados abertos fornecidos pelas empresas de ônibus para identificar as características reais de acessibilidade e estimar o tempo real de chegada dos veículos nas paradas.

No trabalho é descrito uma arquitetura orientada a serviços que explora o paradigma de orquestração de microsserviços para permitir a criação de novos serviços e para tornar a gestão das várias fontes de dados mais fácil e eficaz.

A Figura 3 descreve os principais serviços organizados em camadas. Na camada inferior da arquitetura são apresentados os serviços em que são desenvolvidos para funcionalidades mais simples, por exemplo, viagens planejadas que não incluem funcionalidades em tempo real, ou serviços que processam dados básicos. O objetivo é padronizar os dados e as interfaces de software para disponibilizá-los a outros serviços. Outros serviços mais complexos, são encontrados nas camadas superiores, como as políticas muito refinadas.

Figura 3 – Categorias de Serviço no SMAll

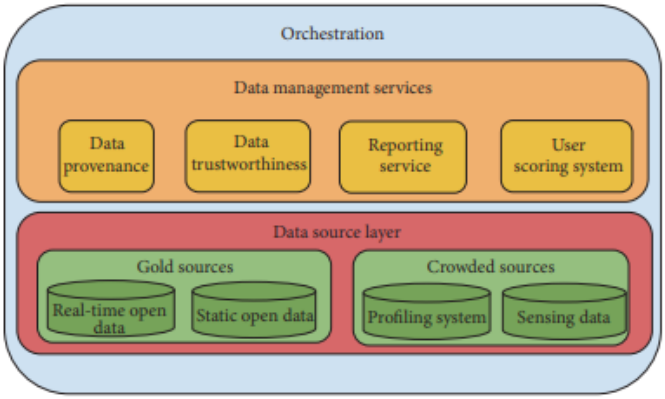


Fonte: (MIRRI *et al.*, 2016)

No trabalho é previsto a adoção de microsserviços nos componentes: *(i)* converter fonte de dados; *(ii)* serviços auxiliares (e.g., autenticação, autorização, programação, roteamento e orquestração);*(iii)* registro dos serviços; e *(iv)* a lógica de negócios que realiza a coleta, armazenamento e processamento de dados para oferecer uma visão dos dados.

Conforme detalhado pelos autores, a plataforma *SMAll* permite explorar esses serviços por meio de orquestração. Para exemplificar como ocorre tal organização, a Figura 4 mostra no nível inferior serviços básicos, como aquele que expõe dados. Na camada de gerenciamento de dados, outros microserviços podem fazer desta camada inferior e implementar mecanismos mais complexos de gerenciamento de dados.

Figura 4 – Organização de Camadas de Orquestração da Plataforma *SMAll*



Fonte: Mirri *et al.* (2016)

Vários tipos de fontes de dados alimentam o sistema. Podemos classificar categorias amplas de acordo com sua procedência (e.g., dados oficiais sobre a infraestrutura de transporte) e sua escala de tempo (e.g., em tempo real versus horários planejados e recursos estáticos). Na verdade, cada dado armazenado inclui informações específicas e tem características peculiares dependendo de sua própria fonte. Por exemplo, fontes de dados de tráfego são postados automaticamente e atualizam no sistema. A quantidade e qualidade de dados (*crowdsourcing / crowdsensing*) são fortemente influenciados por a natureza voluntária da ação e engajamento do participante. Em qualquer caso, todas as fontes de dados, independentemente de sua categoria, serão acessadas de forma homogênea, através de microsserviços apropriados.

Para fornecer serviços personalizados é necessário construir uma categoria de serviços que explore um usuário (JSON-perfil baseado), estruturado em três partes interligadas: (*i*) o Perfil Genérico *(GProfile)*, que inclui alguns dados sobre o usuário, como informações pessoais, idioma, unidade de medição, dispositivo(s) em uso, velocidade média de caminhada, dados sobre sua credibilidade e dados sobre suas vias de transporte público favoritas; (*ii*) o Perfil Urbano *(UProfile)*, que descreve as preferências dos usuários relacionadas ao ambiente urbano, expresso de acordo com suas necessidades, e preferências sobre o Ponto de Interesse urbano (POI); e (*iii*) o Perfil de Acessibilidade *(eAProfile)*, que descreve as preferências dos usuários relacionadas a acessibilidade e a interface do aplicativo.

Foi projetado e desenvolvido um protótipo específico de serviço de detecção que seria explorado pelos usuários, com o objetivo de detectar escadas e automaticamente armazenar informações sobre esse tipo de barreira urbana. O protótipo de detecção foi desenvolvido em Android e explora os sensores do acelerômetro e GPS. Para provar a eficácia da abordagem foi

apresentado dois cenários ilustrando questões de acessibilidade urbana envolvendo cadeira de rodas usuário e um usuário idoso.

A abordagem fornece: *(i)* dados em tempo real sobre meios de transporte públicos; *(ii)* dados coletados e atualizados por meio de *crowdsourcing / crowdsensing*; *(iii)* um modelo capaz de calcular a confiabilidade dos dados coletados; e *(iv)* uma definição de um perfil preciso de acordo com a preferência do usuário e necessidades.

Não foi realizado nenhuma avaliação da solução em termos eficiência, escalabilidade, robustez e usabilidade. Também é possível avaliar que não foram definidas politicas de sincronização de dados. Porém a solução apresenta criptografia como politica de privacidade de dados. Assim como o DOP-MS, o trabalho é implementado seguindo a arquitetura de microsserviços e possui uma definição dos dados.

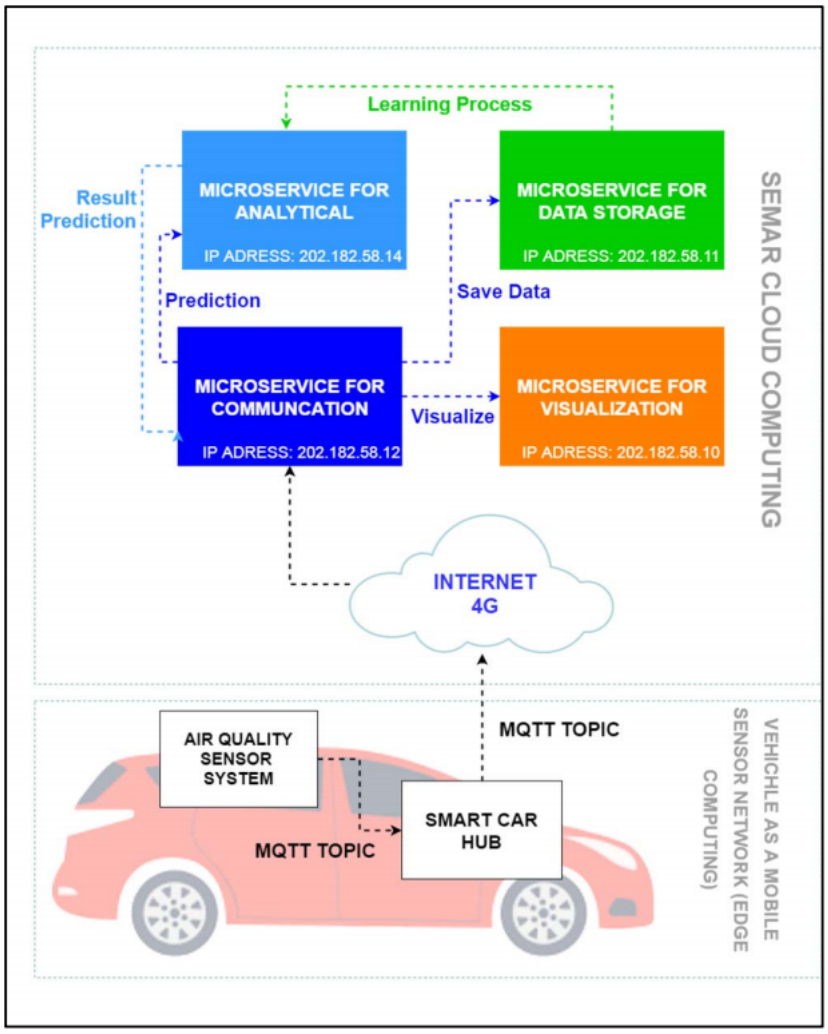
3.1.2 Smart Environment Monitoring and Analytical in Real-time (SEMAR)

Fridelin *et al.* (2018) propõem uma implementação de arquitetura de microsserviço em computação em nuvem integrada ao sistema de monitoramento da qualidade do ar baseado em sensores móveis. Este sistema consiste no Monitoramento Inteligente e Analítico de Ambientes em Tempo Real (SEMAR) como um serviço de computação em nuvem que se conecta ao Veículo como Rede de Sensores Móveis (VaaMSN) para detecção da qualidade do ar. Os serviços de computação em nuvem são divididos em vários microsserviços que consistem em comunicação, armazenamento de *big data*, análise de dados e visualização em tempo real com saídas na forma de mapas, gráficos e tabelas.

O projeto da solução inclui um sistema VaaMSN que consiste em um sistema de sensor de qualidade do ar, conexão Wifi e sensor GPS conectado a carro inteligente. VaaMSN é usado como computação de borda para distribuir dados do sensor para servidores que são divididos em 4 seções: Comunicação, Análise de Dados, Armazenamento de Dados e Visualização. Uma visão em alto nível é mostrada na Figura 5.

O Sistema do sensor da qualidade do ar é colocado no veículo que é usado para monitorar as condições da qualidade do ar. Os sensores de qualidade do ar que foram desenvolvidos são usados para detectar a qualidade do ar, onde o sensor usado consiste em medição de ozônio, nitrogênio, monóxido de carbono e para dióxido de enxofre. O processo de coleta de dados começa com a medição da qualidade do ar usando sensores de qualidade do ar baseados na rede de sensores sem fio. Este controlador coleta os dados e realiza o processamento para a unidade

Figura 5 – Visão Geral da *SEMAR*



Fonte: Fridelin *et al.* (2018)

de medição. Os dados convertidos são enviados para outro dispositivo para processamento no servidor via rede WiFi por meio de comunicação *Message Queue Telemetry Transport* (MQTT), um protocolo de transporte de mensagens baseado no padrão *publish/subscribe* para comunicação clientes/servidor. No SEMAR, dispositivos físicos e controladores são instalados no topo do veículo e recuperam dados dos sensores a cada 5 segundos e enviam para o servidor.

O *Smart Car HUB* atua como o dispositivo de controle principal que é o centro de gerenciamento e envio dos dados obtidos. O modelo de dados utilizado nesse trabalho é 'dados de qualidade do ar, latitude, longitude e hora atual'.

O microsserviço de comunicação é usado para gerenciamento do fluxo de dados na computação em nuvem, distribuindo os dados para o servidor de Big Data, visualização em tempo real e previsão. Ele é composto por *MQTT Brokers*, *MQTT Subscriber* e dois conectores.

MQTT Brokers são usados no serviço principal de comunicação MQTT para comunicação entre TCP e para comunicação com o soquete da web. Os dados do VaaMSN são distribuídos para o sistema de predição do microsserviço de Análise de Dados através do tópico MQTT. O resultado do processo de predição é enviado ao *MQTT Broker* para prosseguir para outro subsistema com o tópico MQTT e recebido pelo MQTT Subscriber. MQTT Subscriber é usado para receber dados do MQTT Broker e transferir para microsserviço para visualização e microsserviço para armazenamento de dados.

O armazenamento de dados é uma parte do microsserviço que funciona para armazenar dados enviados pela computação de borda para o banco de dados *NoSQL* como uma plataforma de *Big data*. Este subsistema consiste na plataforma de Big Data MongoDB [5] e no serviço da web REST. Os dados de qualidade do ar em formato JSON. Os dados armazenados são usados para processos analíticos, onde os dados são usados na criação de novos modelos de dados e na determinação da análise de negócios dos dados.

O microsserviço para visualização consiste em procedimentos para apresentação de dados de séries temporais, interfaces gráficas, visualização pública e paineis analíticos. Este microsserviço foi construído para visualizar a qualidade do ar enviada pelo *MQTT Broker*, e utiliza o *Grafana* [6], uma aplicação para construção de painéis analíticos de visualização, quando quando conectado a fontes de dados suportadas.

Por fim, o microsserviço para análise consiste em um processo de aprendizagem, classificação e análise de dados em tempo real. O processo de aprendizado de máquina é usado para construir o modelo de classificação dos dados de qualidade do ar que foram armazenados no MongoDB pelo algoritmo de aprendizado de máquina. O processo de classificação em tempo real é usado para prever os dados de qualidade do ar recebidos do MQTT Broker. A análise de negócios é usada para mostrar dados resumidos em determinados momentos e regiões com saída na forma de condições de qualidade do ar, este sistema usa a análise estatística dos dados obtidos.

A partir dos experimentos realizados pelos autores, é possível afirmar que a arquitetura de microsserviço utilizada consegue realizar a transmissão de dados para computação em nuvem praticamente em tempo real. Além disso, este sistema visualiza dados em tempo real com um local específico para que possa ser um método alternativo de medição da qualidade para o governo da Indonésia.

[5] https://www.mongodb.com/
[6] https://grafana.com/

O trabalho possui uma sincronia dos dados que ocorre a cada 5 segundos, o qual não é possível realizar a configuração do envio, o que gera um tráfego constante no sistema, acarretando maior gasto energético, fator considerado essencial em *Internet of Things* (IoT). Apesar dos dados trafegados serem de qualidade do ar, não é uma característica do trabalho a preocupação com a privacidade dos dados. Assim como o DOP-MS, o trabalho é implementado seguindo a arquitetura de microsserviços e possuem uma definição dos dados.

3.1.3 Smart Mobility and Sensing: Case Studies Based on a Bike Information Gathering Architecture (PUMA)

No trabalho desenvolvido pelos autores (AGUIARI *et al.*, 2017) é apresentado um serviço de mapeamento e planejamento de viagens baseado em alguns fatores que não são analisados pelos sistemas comuns. Alguns desses fatores são: personalização, sustentabilidade, segurança pessoal, bem-estar e saúde. O trabalho tem como foco o uso de bicicletas inteligentes (equipadas com sensores específicos). É apresentada uma arquitetura baseada em nuvem, personas e cenário de viagem para provar a viabilidade da abordagem.

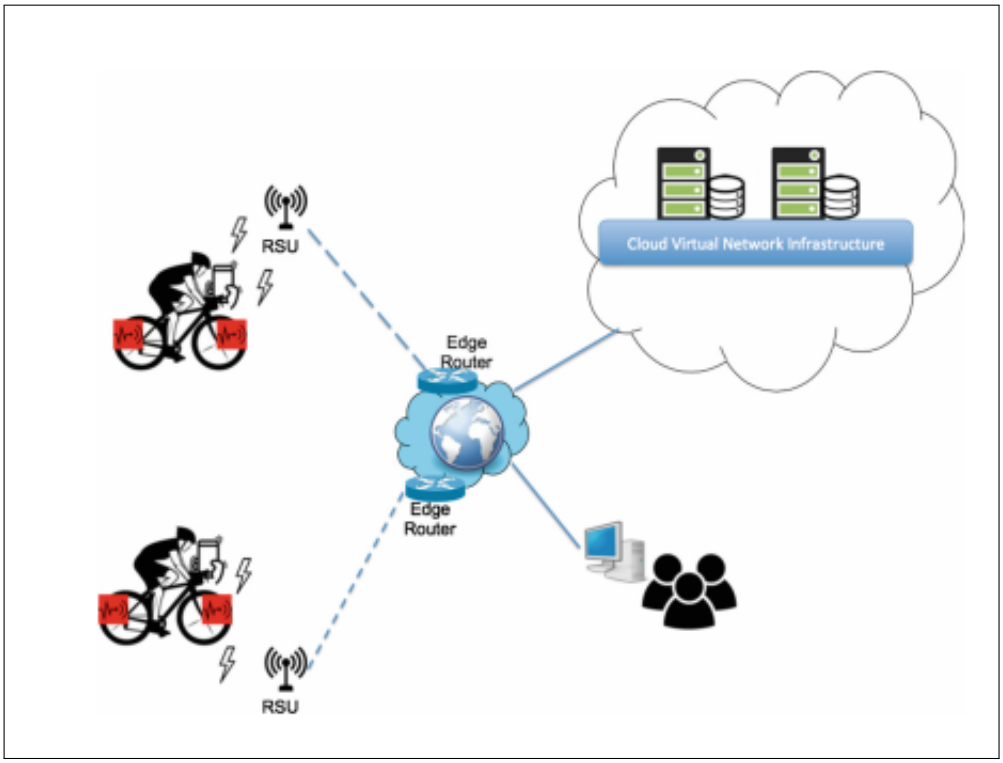
A arquitetura utilizada no trabalho é a arquitetura de microsserviços. As bicicletas são equipadas com dispositivos sensores capazes de captar diferentes tipos de informações não apenas do meio ambiente (e.g., poluição do ar), mas também da própria bicicleta (e.g., quilômetros percorridos via hodômetro). Uma vez coletadas, essas informações são enviadas a uma entidade dedicada a fornecer conectividade ou encaminhamento de dados para a nuvem via internet; esta entidade pode ser uma infraestrutura localizada ao longo da estrada, como por exemplo uma unidade de beira de estrada um *gateway* específico. A infraestrutura em nuvem é onde os dados são processados, armazenados e disponibilizados para serem consumidos por vários usuários.

A aplicação permite que os usuários possam coletar informações e personalizar um plano para um determinado caminho, dependendo de seus hábitos ou necessidades diárias. A nuvem hospeda o *software* que fornece personalização de caminhos e outras serviços, diferentes aplicativos podem exigir diferentes formas de coleta de dados. Portanto, vários usuários podem obter informações acessando, por exemplo, um aplicativo da web para planejar adequadamente com antecedência seu caminho, ou usar um dispositivo móvel em tempo real para consulta.

Na Figura 6 é apresentada a arquitetura da solução , onde pode-se observar as bicicletas acopladas dos sensores, a computação de borda fica localizada ao longo da estrada e a

nuvem é utilizada para migração e processamento de dados.

Figura 6 – Arquitetura do *PUMA*



Fonte: Aguiari *et al.* (2017)

Foram definidas três personas para execução dos testes: *(i)* Wei um acadêmico visitante americano trabalhando em uma pesquisa conjunta em Bologna por três meses; *(ii)* Sven é um estudante sueco, que vive em Bologna por 6 meses para concluir sua tese de mestrado; e *(iii)* Elena que trabalha em tempo integral na Universidade de Bologna, nasceu em Bologna e cresceu no centro da cidade. Em todos os casos, foi possível ter a melhor rota para os usuários.

Assim como o DOP-MS, o trabalho é implementado seguindo a arquitetura de microsserviços e possui uma definição dos dados. Porém não foi identificado nenhuma politica de sincronização de dados, também não foi realizado nenhuma avaliação do sistema em termos eficiência e escalabilidade.

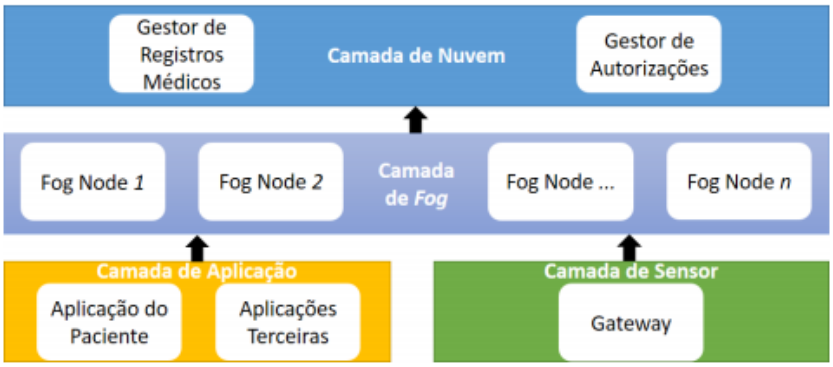
3.1.4 A Blockchain-Based Approach for Privacy Control of Patient's Medical Records in the Fog Layer

(SILVA *et al.*, 2019a) propõem uma nova abordagem que usa *blockchain* para garantir a privacidade dos dados médicos de um paciente em um ambiente de *Fog Computing*. Nessa abordagem, os pacientes são visto como proprietários dos dados, lhes sendo oferecido a capacidade de autorizar as aplicações que podem manipular as informações referentes à sua própia saúde. Além disso, também é permitido que o controle de privacidade dos dados armazenados no domínio dos *Fog Nodes* seja realizado de forma distribuída e independente da nuvem.

O trabalho desenvolvido pelos autores dá um enforque especial a um requisito não funcional de privacidade. A abordagem para controle de privacidade apresentada no trabalho pode ser observada na Figura 7, onde é apresentada a arquitetura formada por quatro camadas. A camada de aplicação é composta por dois tipos de aplicações: a aplicação do paciente que possibilita o controle das aplicações que manipulam subconjuntos de dados e permite ao paciente visualizar tais informações; e aplicações terceiras, que são encarregadas de solicitar acesso para utilização de dados do paciente e podem utilizá-los conforme autorizado. Por sua vez, a camada de sensores é composta pelo módulo *Gateway*, o qual é incumbido de receber dados gerados nos sensores que monitoram o paciente e disponibilizá-los para a camada de *fog*. Essa última é formada por *Fog Nodes*, os quais são encarregados de gerenciar um subconjunto dos registros médicos de pacientes, deixando-os mais próximos das aplicações. Esses módulos também sincronizam o subconjunto de dados e as autorizações com a nuvem.

O Gestor de registros médicos é um dos dois módulos que compõem a camada de

Figura 7 – Arquitetura da Solução

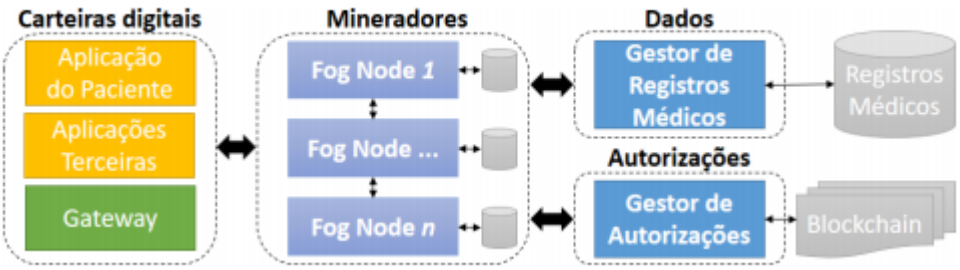


Fonte: Silva *et al.* (2019a)

nuvem. Ele é encarregado de armazenar todo o conjunto de informações dos pacientes. O outro é o Gestor de Autorizações, responsável por armazenar todas as autorizações de acesso. Este módulo também mantém o registro dos *Fog Nodes* que fazem parte da solução, de modo que apenas aqueles que estejam registrados tem autorização para interagir com o gestor de registros médicos.

Os *Fog Nodes* são entidades distribuídas. Logo, eles precisam utilizar uma estratégia de controle de privacidade que também seja distribuída, possibilitando-os a ter autonomia na gestão do acesso às informações armazenadas neles. Esse tipo de abordagem permite a eles trabalhar de forma mais otimizada, pois os *Fog Nodes* não necessitam consultar a nuvem para verificar se uma determinada aplicação tem autorização para utilizar um dado. A Figura 8 mostra a estratégia baseada em *blockchain* que foi projetada para permitir esse processo distribuído de controle de privacidade.

Figura 8 – Estratégia de Privacidade



Fonte: Silva *et al.* (2019a)

Três módulos da arquitetura funcionam como carteiras digitais do *Blockchain*. São

eles: *(i)* Aplicação do Paciente que é responsável por realizar concessões de acesso a dados por meio de transações; *(ii)* Aplicações Terceiras que através de transações realizam solicitações para manipulação de dados dos pacientes; e *(iii) Gateways* que criam transações de solicitação de permissão para enviar os dados gerados nos sensores.

O funcionamento do cenário da prova conceito pode ser descrito em cinco etapas: *(i)* O paciente registra sua aplicação para gerenciar seu conjunto de dados médicos. Neste passo, ele preenche as informações e cria sua carteira digital. Após isso, a aplicação registra-se por meio do envio de uma transação do tipo *RAP* ao *Fog Node*. *(ii)* O *Gateway* envia uma transação do tipo *SAD* ao *Fog Node* solicitando permissão para inserir dados em um determinado subconjunto de registros médicos; *(iii)* O *Fog Node* valida a transação. Em seguida, a aplicação do paciente recebe uma notificação informando que existe um componente solicitando acesso a um subconjunto de seus dados; *(iv)* o paciente abre a notificação e verifica quais são os dados e as operações que o *Gateway* está solicitando,em seguida, ele aceita conceder esse acesso. Nesse momento, a aplicação do paciente envia uma transação do tipo *CAD* ao *Fog Node* concedendo a permissão solicitada; e por fim, *(v)* o *Fog Node* valida a transação e a envia ao *Gateway*, que a partir desse momento fica autorizado a manipular os dados solicitados.

A abordagem para controle de privacidade que é apresentada pelos autores do trabalho incorpora um conjunto de estratégias que ajudam a garantir a manutenção da segurança dos dados do paciente. O anonimato e privacidade são suportados pois o processo de envio de transações utiliza os endereços anônimos que representam as carteiras digitais; a autenticação, uma vez que as transações usadas na solução são assinadas digitalmente com a chave privada da carteira digital que a está gerando. Confidencialidade, já que o acesso aos dados de um paciente é realizado por meio de autorizações de acesso, as quais são concedidas pelo próprio paciente. Por fim Confiança, pois a organização de saúde que instancia a solução proposta se encarrega de gerenciar os *Fog Nodes* que terão permissão para participar da rede privada do *Blockchain*.

O trabalho apresentado tem um arquitetura monolítica e não apresenta um modelo de dados. Porém foi identificado a utilização de *Blockchain* e anonimato para garantir a privacidade dos dados dos usuários. Não foi identificado nenhuma politica de sincronização de dados, e também não foram realizados testes de desempenho.

3.1.5 COP

No trabalho de (GOMES *et al.*, 2017a) é apresentado um serviço de *offloading* de dados contextuais com suporte a privacidade denominado de *Contextual data Offloading service with Privacy support* (COP). Esse serviço amplia as possibilidades para os desenvolvedores de aplicações móveis e sensíveis ao contexto que utilizam o compartilhamento de dados contextuais dos usuários do sistema, bem como prover uma mecanismo de privacidade para que o usuário possa definir se os dados dele devem ser compartilhados com os outros usuários.

O COP baseia-se em (*i*) um modelo de contexto, (*ii*) uma política de privacidade e (*iii*) políticas de sincronização. O modelo de contexto foi estendido do *Loosely Coupled Context Acquisition Middleware* (LoCCAM) (MAIA *et al.*, 2013), o qual foi alterado para que a solução pudesse lidar com a migração de dados contextuais. Na solução proposta, os dados de diversos dispositivos do sistema são migrados para um ambiente centralizado de armazenamento e processamento, assumindo o papel de um *cloudlet* (GOMES *et al.*, 2017a).

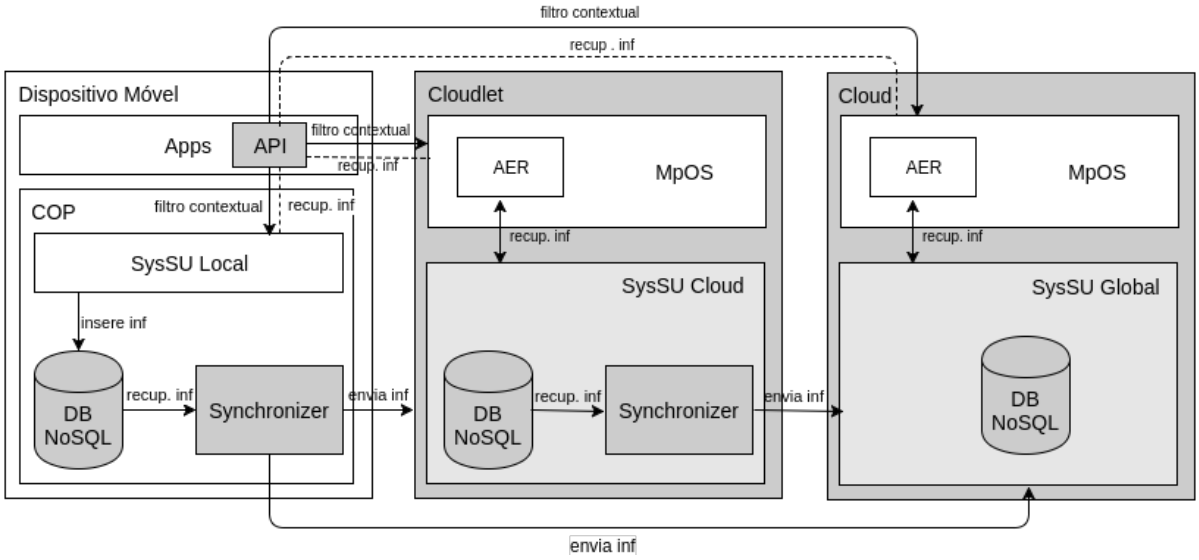
Para as políticas de privacidade foram estabelecidos dois conceitos: dados contextuais sensíveis e *cloudlet* confiável. Os dados contextuais sensíveis estão relacionados com a visibilidade desses dados (i.e., público e privado). O *cloudlet* confiável ou *gateway* é definido pelo usuário (e.g., um *desktop* da casa desse usuário). Se os dados contextuais forem definidos como privados, eles são difundidos para um *cloudlet* confiável. Caso não seja um *gateway*, esses dados não são migrados.

O COP estabelece estratégias diferentes para a migração dos dados, os quais foram denominadas de políticas de sincronização. São elas: por tempo, por quantidade de tuplas e orientado à conexão *Wi-Fi*. A primeira indica que, periodicamente (e.g., a cada 30 segundos), as tuplas são enviadas para o ambiente remoto. O período para o envio dos dados é configurável. Caso o usuário não defina explicitamente outra estratégia a ser usada, esta é a estratégia padrão utilizada. Já a segunda, implica que, se uma quantidade pré-estabelecida de tuplas for alcançada, as tuplas serão enviadas ao *cloudlet*. Por fim, a terceira estratégia define que apenas quando uma conexão *Wi-Fi* for estabelecida, as tuplas do dispositivo móvel serão enviadas à infraestrutura de nuvem. Se caso o usuário estabeleça essa política, mas utilize uma conexão 3G/4G, os dados contextuais não serão migrados.

A arquitetura do COP é composta por 3 camadas, conforme ilustrado na Figura 9. A primeira camada é representada pelo dispositivo móvel, onde os dados contextuais são capturados e armazenados localmente. A segunda camada é representada por um *cloudlet*, que recebe dados

contextuais de usuários conectados a uma mesma *Wireless Local Network* (WLAN). Por fim, o último nível representa um espaço de tuplas hospedado em um serviço de nuvem, e que pode agregar dados de diferentes *cloudlets* e usuários que compartilham seus dados contextuais. No lado do dispositivo móvel, tem banco de dados NoSQL, que atua como uma espécie de *cache* local das informações de contexto, até que o processo de *offloading* de dados ocorra.

Figura 9 – Arquitetura do COP



Fonte: Gomes *et al.* (2017a)

O componente *Synchronizer* presente no dispositivo móvel é responsável por recuperar os dados contextuais do banco de dados no dispositivo móvel e enviar para o ambiente remoto. O *Synchronizer* utiliza de estratégias de envio definidas nas políticas de sincronização. Sempre que as tuplas forem enviadas para o ambiente remoto, o banco de dados local é esvaziado para evitar o acúmulo de dados no dispositivo móvel.

Tanto no *cloudlet* como na nuvem, existem componentes responsáveis pelo gerenciamento de dados contextuais compartilhados. No caso do *cloudlet*, este componente é chamado de SysSU *Cloud*, enquanto que o serviço que agrega dados de todos os *cloudlets* e usuários é chamado de SysSU *Global*. Ambos possuem um banco de dados NoSQL, similar ao que executa no dispositivo móvel, e que tem por objetivo a persistência dos dados contextuais dos dispositivos móveis que compõem o sistema.

No COP, além das tuplas dos dispositivos móveis serem enviadas para o *cloudlet*, elas podem ser enviadas entre *cloudlets*, ou para um repositório central na nuvem, permitindo que os dados contextuais de diversos dispositivos móveis sejam aglutinados em um único local.

Diferente do lado móvel, não existe uma política de esvaziamento do banco de dados no *cloudlet*, pois se caso a nuvem seja inacessível (i.e., problemas de conexão), basta que o dispositivo móvel possua conexão *Wi-Fi* e um *cloudlet* em execução, para que o usuário acesse os dados compartilhados.

Apesar de apresentar uma solução que viabiliza o *offloading* de dados, o COP possui duas fortes limitações. A primeira está relacionada ao modelo de dados, pois só é permitida a migração de dados contextuais e atualmente a maioria das aplicações utiliza diferentes tipos de dados. A segunda é que sua arquitetura apresenta um projeto monolítico, acarretando dificuldades para escalabilidade da solução em ambientes com maior número de usuários.

3.1.6 CAOS-MS

No trabalho de (CÂNDIDO *et al.*, 2019b) é apresentado o desenvolvimento do CAOS-MS, uma plataforma com suporte a *offloading* de processamento para dispositivos móveis baseada na arquitetura de microsserviços. O CAOS-MS evolui uma solução monolítica previamente construída, porém com restrições de escalabilidade e com forte acoplamento.

O autor detalha o processo de refatoração no trabalho (CÂNDIDO *et al.*, 2019a) e concepção do CAOS-MS a partir de sua versão monolítica, além de experimentos comparativos de desempenho e escalabilidade entre a arquitetura original e a nova arquitetura proposta (GOMES *et al.*, 2017b).

3.1.6.1 Arquitetura

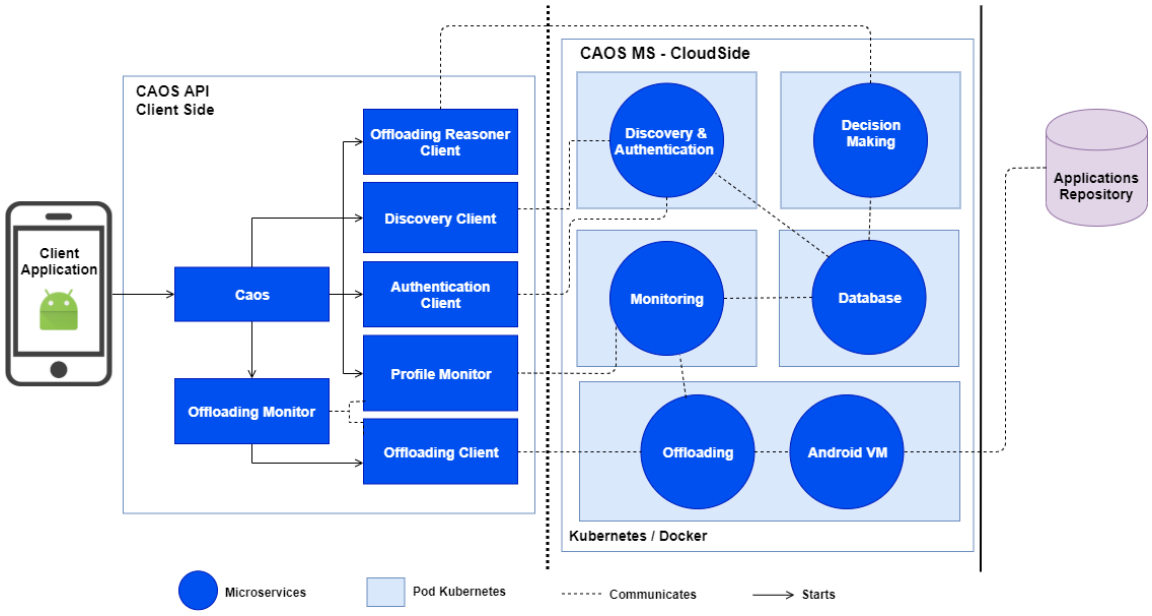
A arquitetura da plataforma CAOS-MS é ilustrada na Figura 10. Pode ser observado que toda a comunicação entre o dispositivo móvel e os microsserviços ocorre de forma individual.

Do lado servidor, temos o primeiro *microsserviço* de autenticação e descoberta. A autenticação tem uma lista de dispositivos que estão conectados à plataforma, enquanto a função do serviço de descoberta é fornecer aos dispositivos que se autenticaram no CAOS-MS, informações sobre as portas dos demais microsserviços do lado servidor.

O *microsserviço* de *Tomada de Decisão* assume a responsabilidade de criar as árvores de decisão e atualizar os dispositivos conectados para determinar a viabilidade ou não da realização de um *offloading de processamento*. Essa ação ocorre sempre que houver mudanças entre as árvores geradas no microsserviço e a existente na aplicação em execução.

O *microsserviço* de *Banco de Dados* executa uma instância com o banco de dados

Figura 10 – Arquitetura da Plataforma CAOS-MS



Fonte: Cândido *et al.* (2019b)

`Postgres`[7], fornecendo mecanismos para persistência de dados aos microsserviços de *Autenticação e Descoberta*, *Monitoramento* e *Tomada de Decisão*. Os dados neste *microsserviço* incluem as métricas de monitoramento, informações sobre os métodos que foram executados na plataforma e informações sobre o histórico de execução de *offloading* dos dispositivos conectados ao CAOS-MS.

O *microsserviço* de Monitoramento (*Monitoring*), realiza a comunicação entre o serviço de Monitoramento e a API cliente. Esse *microsserviço* também é responsável por monitorar os processos executados pelo *microsserviço Offloading*.

O *microsserviço Android VM* inicia automaticamente uma imagem com o sistema operacional Android configurada com todas as dependências necessárias para realização dos processos de *offloading*. Após a inicialização do Android ou atualização das aplicações, ocorre uma sincronização automática das aplicações contidas no repositório de aplicações.

Para obter os resultados desse estudo foram realizados testes e a solução CAOS-MS mostrou-se eficaz nos quesitos escalabilidade e desempenho. Os testes realizados comprovaram que o uso de microsserviços não levou a perda de desempenho, e ainda melhorou substancialmente a escalabilidade da solução. A escalabilidade viabilizada pela arquitetura de microsserviços tornou a execução do *offloading* de processamento consideravelmente mais rápida, enquanto possibilitava melhor utilização dos recursos computacionais, visto que somente os

[7] https://www.postgresql.org/

microsserviços que estavam recebendo muitas requisições eram escalados.

Outro resultado observado neste trabalho de dissertação é que as lições aprendidas na migração da versão monolítica do CAOS para a versão baseada em uma arquitetura microsserviços pode resultar em um catálogo de padrões e decisões de projeto na identificação, delimitação de escopo e contextos, refatorações e decisões de projeto que podem ser úteis em outros projetos de modernização de software com características similares para uma arquitetura microsserviços.

A solução CAOS-MS limitou-se somente aos componentes referentes ao *offloading* de processamento. Desta forma, o *offloading* de dados, bem como a aquisição contextual presentes na versão original, não foram migradas para a nova versão baseada em microsserviços.

3.2 Comparativo Entre os Trabalhos

Para o comparativo entre os trabalhos relacionados, foram levantadas características relevantes desses trabalhos em relação ao serviço proposto nesta dissertação. As características consideradas para a construção de uma infraestrutura de suporte que visa facilitar o desenvolvimento de aplicações móveis que utilizam o compartilhamento de dados e são desenvolvidas usando uma arquitetura de microsserviços:

- **Plataforma**: refere-se às plataformas móveis suportadas pela solução (e.g., Android, iOS, Web);
- **Arquitetura**: corresponde ao modelo de arquitetura utilizado para de desenvolver a solução (e.g., Monolítico, Microsserviços);
- **Modelo de dados**: refere-se ao modelo de dados utilizado na solução (e.g., chave-valor, ontologia);
- **Política de Sincronização**: essa característica refere-se à possibilidade de configurar como os dados serão enviados para o ambiente remoto na solução:

  Flexível: permite escolher a política mais adequada a aplicação;

  Controlada: não permite modificação na politica de envio de dados; e

  Não encontrado: não foi encontrado uma política de envio de dados.
- **Privacidade**: essa característica mostra qual mecanismo de privacidade é utilizado nos dados (e.g., anonimização, criptografia), ou se não tem informações ou não é fornecido suporte à privacidade de dados.

A Tabela 1 detalha o comparativo entre os trabalhos relacionados e as características elencadas.

Tabela 1 – Comparativo Entre os Trabalhos Relacionados

Trabalho	Plataforma	Arquitetura	Modelo de Dados	Política de Sincronização	Privacidade
COP	Android	Monolítico	Chave-Valor	Flexível	Anonimização
CAOS-MS	Android	Microsserviços	Não possui	Não possui	Não possui
SMAll	Android WEB	Microsserviços	Chave-Valor	Não encontrado	Criptografia
SEMAR	WEB	Microsserviços	Chave-Valor	Controlada	Não possui
PUMA	Android WEB	Microsserviços	Chave-Valor	Não encontrado	Criptografia
(SILVA *et al.*, 2019a)	Android	Monolítico	Não encontrado	Não encontrado	*Blockchain* Anonimização Autenticação

Fonte: Elaborado pela autora

Com as informações da Tabela 1 foi possível analisar as características encontradas nos trabalhos relacionados. Pode-se observar que a maioria dos trabalhos possuem um modelo de dados chave-valor que é utilizado para representação de dados. Também foi possível perceber a adoção da arquitetura de microsserviços por quase todos os trabalhos, com exceção do COP e do trabalho desenvolvido pelos autores (SILVA *et al.*, 2019a).

Quando observamos as politicas de privacidade, a maioria dos trabalhos possuem politicas de privacidade de dados para restringir o acesso aos dados que estão sendo migrados. Sobre as políticas de sincronização não conseguimos identificar se são usadas e quais são usadas em alguns trabalhos, isso aponta para que esses trabalhos não possuam politicas especificas para migração de dados, e quando encontradas são inflexíveis (não possibilitam ser alteradas ou modificadas pelos desenvolvedores).

3.3 Considerações Finais

Esse capítulo apresentou os trabalhos relacionados e um conjunto de características elencadas para uma infraestrutura de suporte a migração de dados desenvolvida sobre uma arquitetura de microsserviços.

De acordo com a revisão da literatura, foram encontrados seis trabalhos que mais se relacionam com o trabalho proposto nesta dissertação, e elencadas as seguintes características para a comparação entre essas soluções: Plataforma, Arquitetura, Modelo de dados, Políticas de Sincronização e Privacidade.

Depois de realizada a comparação entre as características e as soluções encontradas, foi possível verificar que nenhuma das soluções desenvolvidas em uma arquitetura de micros-

serviços consegue prover um mecanismo configurável de envio e de privacidade dos dados do dispositivo móvel para o ambiente remoto. Dessa forma, o próximo Capítulo apresenta a solução proposta e sua arquitetura.

4 SERVIÇO DE *OFFLOADING* DE DADOS USANDO UMA ARQUITETURA DE MICROSSERVIÇOS COM SUPORTE A ANONIMIZAÇÃO DE DADOS

Este capítulo apresenta o serviço proposto para *offloading* de dados em cenários de computação móvel. A seção 4.1 apresenta uma visão geral do serviço, enquanto a seção seguinte apresenta sua arquitetura. A seção 4.3 introduz o conceito de filtros, o mecanismo pelo qual dados podem ser selecionados entre informações de múltiplos usuários. As seções 4.5 e 4.4 apresentam respectivamente as políticas de sincronização e privacidade utilizadas pelo serviço para *offloading* dos dados, juntamente com o modelo de dados utilizado. Na seção 4.6 são apresentadas as tecnologias usadas na implementação, implantação e execução do serviço.

4.1 Visão Geral

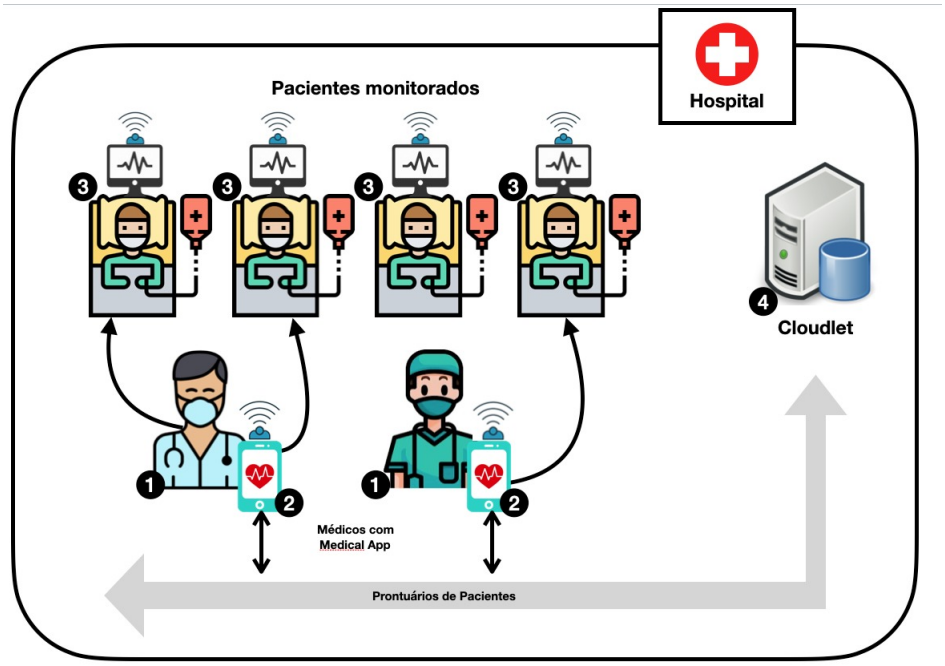
A solução proposta é uma infraestrutura de suporte para facilitar o desenvolvimento de aplicações móveis para plataforma Android, com suporte à disseminação de dados. A solução proposta também segue uma arquitetura de microsserviços. Para contextualizar melhor a relevância do trabalho proposto, apresenta-se, a seguir, um cenário motivador para sua utilização.

Em um cenário de um hospital, onde um médico realiza consultas, o mesmo utiliza uma aplicação móvel para registrar os prontuários de atendimento dos seus pacientes. Essa aplicação permite o cadastro dos pacientes, a criação de prontuários de atendimentos (que é o registro das consultas realizadas) e também a utilização de filtros para realizar buscas de dados e inferir situações. A utilização dos filtros para realizar busca e também inferir situações é restrita a base de dados local, o que nesse caso apenas as informações registradas no dispositivo móvel podem ser visualizadas.

Ao utilizar os serviços propostos pela solução, essa aplicação consegue realizar *offloading* de dados, o compartilhamento de dados de um paciente entre vários médicos e também se aproveitar dos dados que estão disponíveis no *cloudlet* para obter respostas mais ricas. Nesse caso os médicos que estão utilizando a aplicação, conseguem ter acesso a uma quantidade maior de informações, possibilitando que um médico tenha acesso ao registro de outros pacientes ou outros prontuários de atendimento do mesmo paciente. É importante ressaltar que nessa aplicação são utilizados sensores que fornecem informações sobre o paciente (e.g., batimentos cardíacos e temperatura), esses dados são utilizados para que os médicos possam monitorar o estado de saúde de um paciente. Os prontuários também são marcados com a localização e o instante (data/hora) em que foram capturadas pelo dispositivo móvel do usuário. A seguir é

apresentada uma imagem que ilustra esse cenário.

Figura 11 – Cenário Motivador



Fonte: Elaborado pela autora

A Figura 11 mostra um ambiente hospitalar em que é possível observar médicos que utilizam uma aplicação para gerenciamento de prontuários de atendimento e um *cloudlet* para armazenamento de informações. Os elementos do cenário estão numerados e são descritos a seguir:

- **01 - Médicos:** os médicos utilizam uma aplicação para gerenciar prontuários de atendimento e realizar consultas na base de dados;
- **02 - Aplicação:** essa aplicação fornece algumas funcionalidades: (a) cadastro de pacientes; (b) registro de prontuários de atendimentos e (c) utilização de filtros para realizar buscas e inferir situações;
- **03 - Pacientes monitorados:** os pacientes são monitorados, e dados de sensores são fornecidos juntamente com outras informações do paciente para compor os dados do prontuário de atendimento;
- **04 - *cloudlet*:** a infraestrutura de *cloudlet* é utilizada para execução da parte servidora do serviço proposto, o qual contém o armazenamento das informações que são migradas dos dispositivos móveis. É utilizado um *cloudlet* pois as aplicações médicas são sensíveis a latência, então conseguimos obter acesso as informações mais rapidamente.

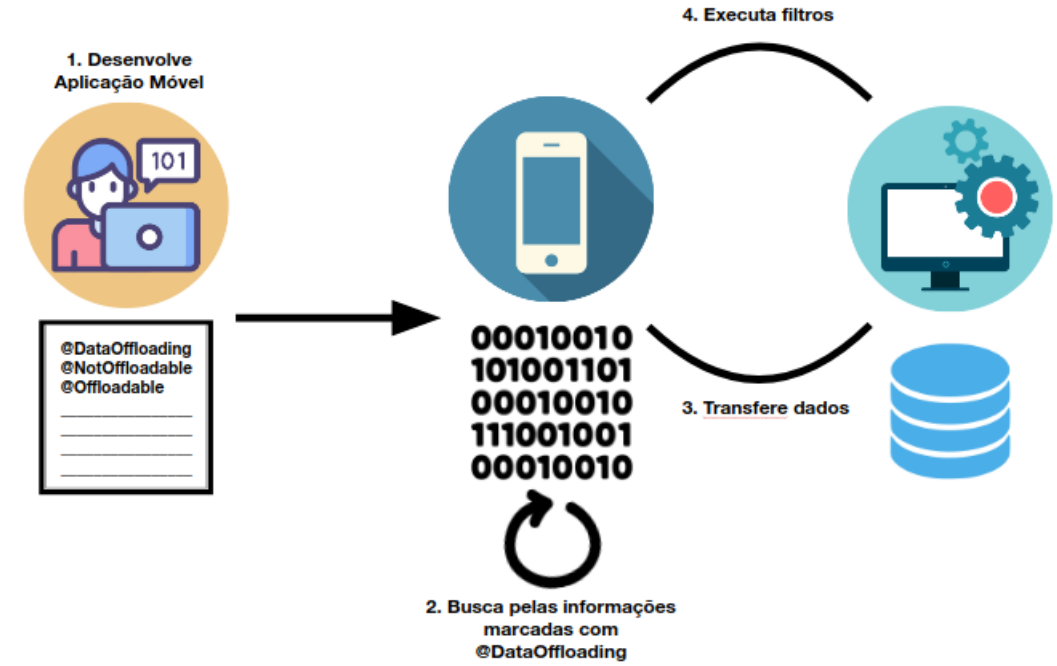
A partir da visão geral do cenário desejado, uma série de requisitos foram encontrados. Primeiro podemos listar alguns requisitos funcionais identificados: (a) cadastrar um paciente; (b) cadastrar um prontuário de atendimento; (c) acessar um prontuário; (d) permitir a migração de dados; (e) marca quais dados podem ser migrados; (f) permitir o compartilhamento dos dados e (g) retirar a carga de informações que um dispositivo móvel precisa armazenar. A partir dos requisitos elencados, pode-se destacar o de migração de dados, pois com essa funcionalidade é possível evitar o armazenamento de dados no dispositivo móvel, mitigando essa limitação, e aproveitar do compartilhamento de dados para inferir situações dos usuários do sistema.

Os requisitos não funcionais identificados são: (a) garantir um mecanismo de privacidade e (b) fornecer escalabilidade. Na solução desenvolvida, a garantia de privacidade é implementada através da técnica de anonimização de dados onde o desenvolvedor pode anonimizar os dados que não devem ser migrados ou os dados do usuário que não podem ser identificados. A escalabilidade é importante para que a solução possa continuar disponível conforme aumentar o número de requisições do serviço. Para tratar a escalabilidade, na solução proposta é utilizado a arquitetura de microsserviços.

Partindo dos requisitos listados anteriormente, foi elaborada uma estrategia para o desenvolvimento do trabalho. A ideia é propor uma abordagem baseada em outras duas ferramentas já existentes. A primeira delas é o COP que é uma solução que permite *offloading* de dados contextuais, mas que não foi implementada em microsserviços. A segunda é o CAOS-MS que permite realizar *offloading* de processamento e tem sua arquitetura baseada em microsserviços que fornece suporte à escalabilidade. As duas soluções foram desenvolvidas pelo grupo de pesquisa GREat (Grupo de Redes de Computadores, Engenharia de Software e Sistemas).

A Figura 12 apresenta o funcionamento da aplicação. Os desenvolvedores de aplicações móveis são os usuários da solução DOP-MS. Durante o processo de implementação da aplicação, o desenvolvedor deve definir quais dados devem ser migrados, anonimizados, e qual politica de sincronização de dados será utilizada.

Figura 12 – Funcionamento da Solução



Fonte: Elaborado pela autora

Os elementos descritos na imagem estão numerados e são descritos a seguir:

- **01 - Desenvolvedor:** o desenvolvedor que implementa a aplicação móvel realiza a marcação de quais dados devem ser migrados e quais dados devem ser anonimizados;
- **02 - Busca pelas marcações dos dados:** durante a realização do processo de *offloading* são buscados os dados que estão marcados com as anotações e que devem ser migrados para o *cloudlet*, assim como também é verificado quais dados devem ser anonimizados durante o processo de migração;
- **03 - Transferência de dados :** os dados passíveis de *offloading* são migrados para o ambiente remoto seguindo as politicas de sincronização de dados definidas pelo desenvolvedor da aplicação;
- **04 - Execução dos filtros:** a aplicação móvel solicita a busca por informações que estão armazenadas no *cloudlet*, através da execução dos filtros implementados pelo desenvolvedor.

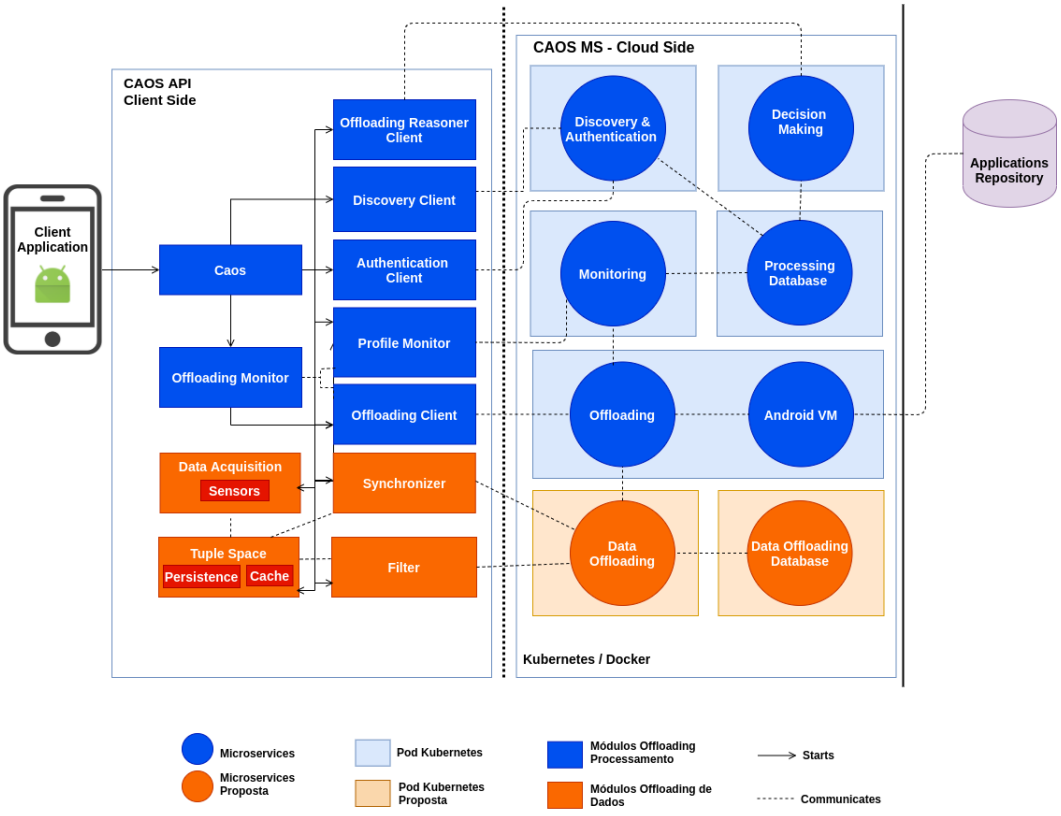
4.2 Arquitetura

Para solucionar as necessidades encontradas no cenário motivador, foi desenvolvido um serviço de *offloading* de dados usando uma arquitetura de microsserviços com suporte a

anonimização de dados chamado de DOP-MS.

Na Figura 13 é ilustrado a arquitetura do DOP-MS integrado com o CAOS-MS. Os serviços e módulos que estão destacados com a cor azul foram implementados no trabalho de (CÂNDIDO *et al.*, 2019b) e já foram explicados na seção de trabalhos relacionados. Os serviços e módulos que estão em laranja foram desenvolvidos neste trabalho. Foi definido uma estratégia de integração à arquitetura do CAOS-MS, onde foi incorporado dois microsserviços no lado servidor denominados *Data Offloading* e *Data Offloading Database*. Também foi implementado quatro módulos do lado cliente da solução: *Data Acquisition*, *Synchronizer*, *Tuple Space* e *Filter*. Essas modificações foram necessárias para fornecer suporte ao *offloading* de dados.

Figura 13 – Arquitetura do DOP-MS integrado com o CAOS-MS



Fonte: Elaborado pela autora

4.2.1 Servidor

O microsserviço *Data Offloading* é responsável por fazer o *offloading* de dados e aplicar os filtros disponíveis na solução antes de entregar os dados ao usuário que está fazendo a

requisição.

Além do microsserviço de gerenciamento de dados, foi implantado um microsserviço de banco de dados NoSQL (*Data Offloading Database*) para persistir os dados oriundos dos usuários do DOP-MS, cujo o modelo seguido é o de orientado a documentos, pois as tuplas seguem o mesmo padrão (chave-valor).

4.2.2 Cliente

O *Data Acquisition* é responsável por realizar a aquisição dos dados que serão migrados pela solução. O *Synchronizer* é responsável por realizar a sincronização de dados, que são representados utilizando o formato JSON (mais detalhes na seção 4.5). O *Tuple Space* é responsável pelo armazenamento e gerenciamento de dados local (mais detalhes na seção 4.4). Por fim, o *Filter* é responsável por fazer a aplicação dos filtros modelados (mais detalhes na seção 4.3).

O fluxo de execução da solução segue conforme descrito: os dados são obtidos através do módulo de *Data Acquisition*. Após isso os dados são direcionados para o *Tuple Space* onde são armazenados localmente e enviados conforme as politicas de sincronização definidas no módulo de *Synchronizer*. Para realizar a recuperação de dados são realizados os filtros com o módulo de *Filter*. Caso não possua rede os dados são recuperados do espaço de tupla, caso contrário os dados são recuperados do *Data Offloading* que interage com um microsserviço chamado *Data Offloading Database* que é executado no *cloudlet*.

A arquitetura do CAOS-MS atualmente está fortemente vinculada a um orquestrador de microsserviços chamado *Kubernetes* [1]. No *Kubernetes*, cada microsserviço é encapsulado em um conceito chamado *pod*. O *Kubernetes* é uma plataforma *open-source* que automatiza as operações de gerenciamento dos contêineres Linux. Ele oferece os recursos de orquestração e gerenciamento necessários para implantar contêineres com escalabilidade, facilidade e eficiência (HAT, 2019).

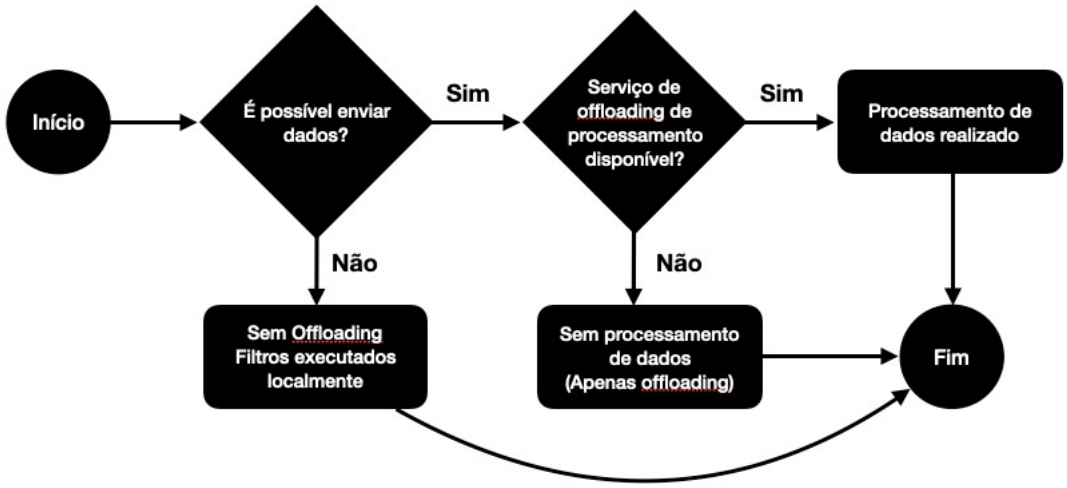
4.3 Filtros

Assim como no COP (versão monolítica), a solução proposta acessa os dados por meio de filtros de forma transparente, podendo ser no dispositivo móvel e no *cloudlet*. A partir da utilização da solução, a API realiza a descoberta do local de execução da aplicação e direciona a

[1] https://kubernetes.io/

execução dos filtros. Na Figura 14 é explicado como funciona a decisão de realizar o *offloading* de dados. Na primeira decisão tomada, é verificado se existe rede, pois é necessário a rede para realizar o *offloading*. Caso não tenha rede, não será feito o *offloading* de dados e os filtros serão executados localmente no espaço de tuplas. Caso tenha rede, será verificado se o serviço de *offloading* de processamento está disponível para que seja realizado o *offloading* e o processamento dos dados. Por fim, se o serviço de *offloading* de processamento não estiver disponível será realizado apenas o *offloading* de dados e o processamento será realizado no dispositivo do usuário.

Figura 14 – Fluxograma de decisão sobre uso ou não do *offloading* de processamento



Fonte: Elaborado pela autora

Para a execução dos filtros tanto no dispositivo móvel como em um ambiente remoto, é necessário que o desenvolvedor da aplicação implemente tais filtros. Esses filtros podem ser diferentes, pois os algoritmos remotos podem ser mais complexos que os algoritmos locais, uma vez que são executados em locais com maior capacidade de processamento e podem ter que lidar com maior quantidade de dados.

4.4 Modelo e Privacidade de Dados

Uma importante contribuição do DOP-MS é permitir que os desenvolvedores das aplicações sejam capazes de definir quais informações de suas aplicações poderão ser migradas

para um ambiente remoto de armazenamento, como por exemplo, um *cloudlet*. Porém, ao definir tais informações, também deve ser possível indicar dados sensíveis que demandem de um tratamento diferenciado para não comprometer a privacidade de usuários. No DOP-MS, a solução passa por uma extensão ao modelo proposto originalmente pelo COP.

No COP, apenas dados contextuais obtidos de sensores físicos ou virtuais podem ser migrados para um ambiente remoto. Os dados contextuais de múltiplos usuários são armazenadas em um espaço de tuplas compartilhado. Com isso são necessárias informações para individualizar os dados. A mesma ideia foi utilizada no DOP-MS, acrescentando os dados não apenas de contexto, mas de qualquer objeto do domínio da aplicação que o desenvolvedor queira migrar entre dispositivo móvel e *cloudlet*, ou mesmo nuvem. A partir desta estratégia, uma tupla para migração de dados é definida como segue:

Definição 1 (Tupla no DOP-MS) *Uma tupla no DOP-MS é representada por seis elementos. São eles:* $t = (\texttt{IdAPP}, \text{“?”}), (\texttt{IdClass}, \text{“?”}), (\texttt{IdDevice}, \text{“?”}), (\texttt{Timestamp}, \text{“?”}), (\texttt{ObjectData}, \text{“?”}), (\texttt{Sensor}, \text{“?”})\rangle$*, onde* `IdDevice` *representa a identificação do dispositivo móvel;* `IdApp` *representa a identificação da aplicação que está utilizando a solução;* `IdClass` *é utilizado para identificar a classe que representa as informações a serem migradas e* `Timestamp` *representa o instante, em milissegundos, em que o dado é inserido no ambiente remoto. Uma tupla pode ainda conter informações dos sensores, e que representam informações adicionais relacionadas ao contexto. Para cada sensor são armazenadas três informações: (i) o tipo do sensor, (ii) o valor daquele sensor e (iii) o instante em que foi inserida a informação.*

A classe Sensor é composta de três atributos: *type, value, timestamp*. O `type` serve para identificar qual é o sensor que está coletando os dados de contexto (e.g., `device.temperature`, identifica um sensor de temperatura do dispositivo móvel). O `value` armazena o dado bruto do sensor (e.g., `35`, indica que é o valor da temperatura). O *timestamp* indica o instante em que o dado foi capturado.

A estrutura apresentada na definição (4.1) caracteriza uma leitura do dado ‘Informações do cliente’ no formato de tupla para o DOP-MS. Nessa estrutura, o dispositivo móvel tem identificação “745744441” e utiliza uma aplicação com identificação “br.ufc.Registro”.

$$
\begin{aligned}
t = \langle & (\texttt{IdApp}, \text{“}\texttt{br.ufc.Registro}\text{”}) \\
& (\texttt{IdClass}, \text{“}\texttt{br.ufc.Registro.usuario}\text{”}) \\
& (\texttt{IdDevice}, \text{“}\texttt{745744441}\text{”}) \\
& (\texttt{Timestamp}, \texttt{1610551098}) \\
& (\texttt{ObjectData}, \\
& \quad \texttt{cpf}, \text{“}\texttt{123.456.789} - \texttt{10}\text{”} \\
& \quad \texttt{nome}, \text{“}\texttt{Maria de Oliveira}\text{”}) \\
& (\texttt{Sensor}, \\
& \quad \texttt{type}, \text{“}\texttt{sensor.location}\text{”} \\
& \quad \texttt{value}, \text{“}-\texttt{6.397260}, -\texttt{38.859591}\text{”} \\
& \quad \texttt{timestamp}, \texttt{1610531098}) \rangle
\end{aligned}
\tag{4.1}
$$

A partir da definição de uma tupla para o DOP-MS, cabe então ao programador definir explicitamente quais informações do domínio da aplicação que serão migradas. Na implementação do DOP-MS, a indicação destas informações acontece por meio de anotações, uma construção comum a linguagens de programação que permitem decorar classes, métodos, campos, parâmetros, variáveis, dentre outros elementos. No DOP-MS, a informação a ser migrada deve ser representada em uma classe marcada com a anotação `@DataOffloading`, como ilustrado no exemplo do trecho de código 1, onde a classe `Paciente` encapsula dados de um paciente, incluindo seu CPF, nome e endereço.

Listagem 1 – Trecho de Código-Fonte para Offloading e Anonimização de Dados

```
1 @DataOffloading
2 public class Paciente {
3
4   @NotOffloadable
5   public String cpf;
6   public String nome;
7   public Endereco endereco;
8 }
```

Com as classes marcadas, o DOP-MS vai então conseguir determinar que instâncias de objetos destas classes serão automaticamente migradas para um ambiente remoto, segundo o modelo de dados descrito anteriormente.

Já em relação à politica de privacidade, o DOP-MS adota a anonimização de dados, mais especificamente a supressão de dados (seção 2.1.2.1), fazendo com que certas informações sejam removidas quando migradas do dispositivo móvel para o ambiente remoto de armazenamento. Essa técnica resulta em dados anonimizados, que não podem ser associados a nenhum indivíduo específico. A supressão de dados no DOP-MS também utiliza a abordagem de anotação, permitindo que certos atributos de classes anotadas com `@DataOffloading`, possam ser marcadas com a anotação `NotOffloadable`. No código 1, o atributo `cpf` é um exemplo de um dado sensível que não deve migrado para o ambiente remoto pois representaria um risco à privacidade do paciente. Nesse caso, esse atributo não será migrado para o ambiente remoto e ficará apenas no dispositivo móvel do usuário para futuras realizações de filtros.

4.5 Políticas de Sincronização

Outro ponto importante é definir em que momentos os dados devem ser migrados do dispositivo para um ambiente remoto. Neste trabalho foram reaproveitadas as politicas de sincronização estabelecidas no trabalho do Gomes *et al.* (2017b), que são diferentes estratégias de envio dos dados do dispositivo móvel para o ambiente remoto, as quais são denominadas de políticas de sincronização.

Três estratégias de envio são especificadas no trabalho, a saber: (*i*) por tempo, (*ii*) por quantidade de tuplas e (*iii*) orientada à conexão *Wi-Fi*. A estratégia de envio por tempo significa que, periodicamente (e.g., a cada 30 segundos), as tuplas são enviadas para o ambiente remoto. O período para o envio dos dados é configurável. Caso o usuário não defina explicitamente outra

estratégia a ser usada, esta é a estratégia padrão utilizada. A estratégia de envio por quantidade de tuplas implica que, se uma quantidade pré-estabelecida de tuplas for alcançada, as tuplas serão enviadas para o *cloudlet*. Essa quantidade também é configurável. Por fim, a estratégia de envio orientada à conexão *Wi-Fi* define que apenas quando uma conexão *Wi-Fi* for estabelecida, as tuplas do dispositivo móvel serão enviadas à infraestrutura de nuvem. Se caso o usuário estabeleça essa política, mas utilize uma conexão 3G/4G, os dados não serão descarregados.

Essas estratégias podem ser combinadas quando escolhidas simultaneamente pelo desenvolvedor. Neste caso, a que ocorrer primeiro será executada. Por exemplo, se um desenvolvedor define que deseja utilizar as estratégias por tempo e por quantidade de tuplas, quando a quantidade de tuplas pré-estabelecidas for atingida antes do tempo definido, essas tuplas serão enviadas. O mesmo acontece se a estratégia por tempo acontecer antes. Na prática, o desenvolvedor realiza a definição das estratégias durante a implementação da aplicação, selecionando quais estratégias ele gostaria de utilizar e definindo um limiar para o serviço.

4.6 Tecnologias Utilizadas

A solução DOP-MS foi implementada na linguagem java, assim como o COP e o CAOS-MS. Também foi reutilizado o repositório de aplicação construído no trabalho de (CÂNDIDO *et al.*, 2019b).

Para o desenvolvimento utilizando o estilo arquitetural de microsserviços foram selecionadas duas ferramentas que já são utilizadas no CAOS-MS visando também a integração entre as soluções. São elas: o `Docker`[2] para conteinerização dos *microsserviços* e `Kubernetes`[3] para gerenciamento dos contêineres.

A tecnologia Docker usa o kernel do Linux e recursos do kernel como Cgroups e namespaces para segregar processos, trabalhando com a ideia de conteinerização. Assim, possibilita que os serviços sejam executados e inicializados de maneira independente e com maior velocidade (HAT, 2019). O Kubernetes é uma plataforma *open source* que automatiza as operações de gerenciamento dos contêineres Linux. Essa plataforma elimina grande parte dos processos manuais necessários para implantar e escalar as aplicações em contêineres. O Kubernetes oferece os recursos de orquestração e gerenciamento necessários para implantar contêineres em escala para essas cargas de trabalho, com facilidade e eficiência(HAT, 2019).

[2]URL: <https://www.docker.com>
[3]URL: <https://kubernetes.io>

Além do Kubernetes [4] e do Docker 19.03.8 [5], algumas das principais tecnologias utilizadas para a realização da prova de conceito e do próprio DOP-MS incluem: (i) Spring Boot 2.4.2 [6], (ii) Java 11 [7], (iii) Android 29 [8], (iv) MongoDB 4.2 e (v) Retrofit 2.0 [9].

Já em relação às ferramentas de desenvolvimento utilizadas para a construção da solução da solução, destacam-se (i) Eclipse IDE [10], (ii) GitHub [11] para controle de versão, (iii) Android Studio [12] para desenvolvimento do cliente móvel e (iv) Gradle [13] para controle de dependências.

4.7 Considerações Finais

Neste capítulo foi apresentado a proposta de um serviço de *offloading* de dados usando uma arquitetura de microsserviços com suporte a anonimização de dados. Essa solução busca oferecer suporte ao *offloading* de dados em aplicações Android.

O desenvolvedor ao utilizar a solução define quais dados serão migrados, quais dados serão anonimizados, e as politicas de sincronização utilizadas para a migração dos dados. Também é definido o modelo de dados da solução que permite a migração de dados do domínio da aplicação e dados contextuais.

O próximo capítulo apresenta exemplos de códigos de utilização do serviço proposto através da implementação de uma aplicação como prova de conceito, que baseia-se no cenário motivador.

[4] https://kubernetes.io/
[5] https://www.docker.com/
[6] https://spring.io/projects/spring-boot
[7] https://jdk.java.net/11/
[8] https://developer.android.com/
[9] https://square.github.io/retrofit/
[10] https://www.eclipse.org/
[11] https://github.com/
[12] https://developer.android.com/
[13] httpss://gradle.org/

5 PROVA DE CONCEITO E EXPERIMENTOS

Como forma de ilustrar os ganhos da migração de dados dos dispositivos móveis que utilizam a solução proposta foram realizados dois experimentos: o primeiro é uma prova de conceito com o objetivo de ilustrar os ganhos obtidos com o *offloading* de dados e depois outro experimento para verificar escalabilidade e desempenho do serviço desenvolvido. Para a realização desses testes foram utilizadas duas aplicações. A primeira aplicação chamada de Medical App baseia-se no cenário motivador apresentado previamente no capítulo 4. Esta aplicação ilustra como é possível utilizar a solução para o desenvolvimento de aplicações móveis que compartilham os dados dos usuários, bem como a utilização dos filtros disponíveis na solução. Posteriormente foi utilizada outra aplicação chamada *Integers* sobre a qual ocorreu a realização de testes de desempenho e escalabilidade, essa aplicação possibilita um maior controle sobre os dados do experimento de escalabilidade. Ao final do capítulo são apresentados resultados obtidos com a realização de uma analise estatística.

5.1 Ambiente de teste

Antes de iniciar a implementação da prova de conceito, foram necessárias algumas configurações no ambiente onde foram executados os microsserviços. Foi criada uma imagem *docker* para ser executada no kubernetes, e utilizado *scripts* para automação de tarefas no kubernetes relacionadas à inicialização dos serviços. São eles: *start*, *destroy* e *stop*. O *script start* realiza três ações: *(i)* inicializa um serviço de descoberta de *cloudlet*, *(ii)* o serviço de banco de dados e *(iii)* os serviços de *offloading* de dados e processamento. O *script stop* é utilizado para pausar todos os serviços que estão em execução. Ressalta-se que os serviços não são destruídos e as informações não são apagadas do banco de dados quando este ação é realizada, sendo feita apenas uma pausa na execução dos serviços. Por fim, o *script destroy* foi ser utilizado para destruir todos serviços que estão em execução, incluindo o banco de dados. Nesse caso, todas as informações que estão armazenadas serão descartadas. Esses comandos são descritos no Apêndice A.

A integração entre o CAOS-MS e o DOP-MS para utilização dos módulos e micro-serviços foi realizada de forma direta, i.e, uma vez que as duas soluções seguem uma arquitetura de microsserviços, a integração foi feita de forma simplificada. Foi modificado o direcionamento do *endpoint* para realizar o tratamento de duas situações: *(i)* se o serviço de processamento

do CAOS-MS não estivesse ativo, o processamento dos filtros seria realizado de forma local (o *endpoint* é configurado na aplicação por meio de uma anotação chamada *@CaosConfig*) e *(ii)* caso contrário, o serviço de processamento deveria ser utilizado, bem como o DNS do Kubernetes para descoberta do endereço do serviço.

5.2 Prova de Conceito

A aplicação Medical App tem como objetivo a gestão de prontuários de pacientes por médicos. Essa aplicação permite que prontuários médicos sejam enriquecidos com informações contextuais, como localização e tempo (data/hora) em que dados biométricos são coletados por sensores que monitoram os pacientes. Por meio do *offloading* de dados, prontuários podem ser compartilhados por médicos de um mesmo hospital para troca de informações sobre diagnósticos e prescrições. A Tabela 2 contém a lista de requisitos funcionais implementadas na prova de conceito em questão.

Tabela 2 – Funcionalidades da Aplicação Medical App

Funcionalidade	**Descrição**
Cadastro de paciente	O médico pode cadastrar um paciente
Cadastrar de prontuário	O médico registra uma consulta realizando o cadastro de um prontuário
Realizar busca pelo CPF	O médico pode pesquisar um paciente pelo CPF na hora de registrar um prontuário
Filtro 01	Média aritmética dos batimentos cardíacos de um paciente
Filtro 02	Quantidade de pessoas diagnosticas com covid
Filtro 03	Quantidade de pessoas diagnosticas com câncer
Filtro 04	Quantidade de pessoas que tomaram analgésico
Filtro 05	Quantidade de pessoas com febre, dor no corpo e dor de cabeça
Filtro 06	Tratamentos utilizados em pessoas diagnosticadas com covid
Filtro 07	Com qual doença foi diagnosticada as pessoas que tinham febre e dor no corpo
Filtro 08	Com qual doença foi diagnosticada as pessoas que tinham febre e dor no corpo

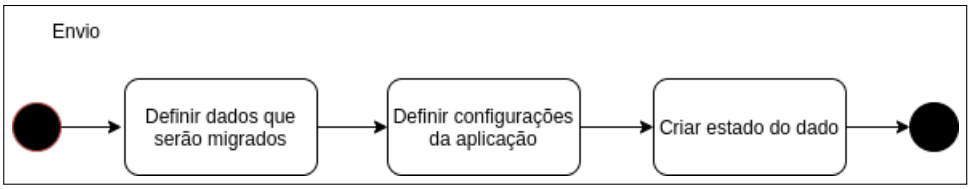
Fonte: Elaborado pela autora

5.2.1 Implementação da Medical App

A aplicação Medical App foi desenvolvida do zero. As Figuras 15 e 16 apresentam dois diagramas que contém os passos necessários para a implementação da prova de conceito. Primeiramente são descritos os passos necessários para que a aplicação consiga realizar o *offloading* de dados e posteriormente o recebimento de dados.

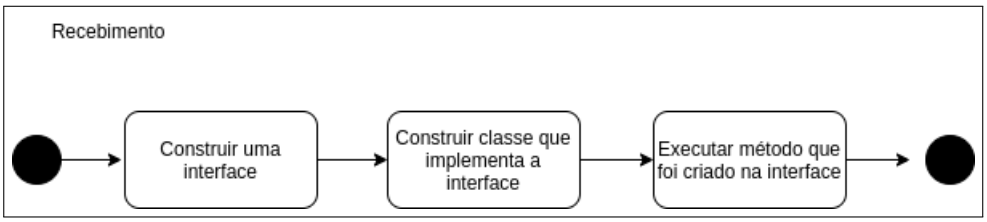
A seguir são descritos cada um dos passos necessários para realizar a configuração na aplicação para que ela execute o envio dos dados, assim como consta no diagrama 15. No

Figura 15 – Etapas para configuração dos dados a serem enviados no processo de *offloading*



Fonte: Elaborado pela autora

Figura 16 – Configuração para o recebimento de dados



Fonte: Elaborado pela autora

passo (*i*) o desenvolvedor precisa definir a classe que representará os dados a serem migrados por meio do *offloading*, e também quais dados serão anonimizados. Em seguida, (*ii*) na classe principal é necessário utilizar a anotação `@DataOffloading`, que faz referência aos dados a serem migrados. Para realizar a configuração é necessário informar para onde os dados serão migrados (*endpoint* do *cloudlet*), como eles serão migrados (qual política de sincronização utilizada), e também iniciar todos os serviços (dentro do `onCreate` da aplicação. E por fim, (*iii*) criar uma instância do objeto a ser migrado, recuperar os dados a serem migrados e encapsular no objeto criado.

A seguir são descritos cada um dos passos necessários para realizar a configuração na aplicação para que ela execute o recebimento dos dados, assim como consta no diagrama 16. O *offloading* de dados não necessariamente precisa do *offloading* de processamento para ser executado, pois tais serviços são independentes, como detalhado na seção 4.3. Desta forma, a abordagem apresentada é herdada do CAOS-MS. O passo (*i*) é a definição de uma interface, que indicará quais métodos de uma determinada classe poderão ser executados em um ambiente remoto. Em seguida, (*ii*) é implementada uma classe concreta que implementa a interface, e indicará como os filtros serão realmente executados. Todos os métodos que serão realizados *offloading* devem estar dentro dessa classe. Por fim, (*iii*) será chamado o método com a implementação desenvolvida.

No trecho de código apresentado na listagem 2 é criada uma interface chamada de `IRead` para servir como contrato entre o cliente e o servidor. A interface `IRead` define assinatura

de todos os métodos passiveis de *offloading*. Nessa classe serão informados todos os métodos criados pelo desenvolvedor. O DOP-MS, como explicado na capítulo 4, possui uma política de decisão dos filtros. O serviço de *offloading* de dados está provisionado para ser executado seguindo a definição da decisão de execução dos filtros descrita anteriormente. A parte de processamento dos dados vai depender do serviço de *offloading* está ativo ou não (caso positivo, será processado no *cloudlet*, caso negativo, será processado no dispositivo móvel)

Listagem 2 – Implementação da interface responsável por invocar o método `Offlodable`

```
1 public interface IRead {
2    @Offloadable
3    public int getQuantidadeDeDiagnosticos(String appId, String diagnostico);
4 }
```

De acordo com a proposta do DOP-MS, o local de execução de um filtro permite então utilizar ou não informações compartilhadas de outros usuários que enviam dados para o mesmo *host* em que o filtro esteja sendo executado. A Listagem 3 ilustra como o DOP-MS viabiliza esta possibilidade. Um filtro na aplicação Medical App foi implementado para buscar quais usuários estão diagnosticados com uma doença específica. O desenvolvedor pode ainda enriquecer este filtro para adicionar restrições temporais, fazendo com que o DOP-MS recupere prontuários de pacientes diagnosticados com a determinada doença, porém dentro de um certo intervalo de tempo.

Listagem 3 – Trecho de Código-Fonte da utilização de filtros na Aplicação Medical App

```
1  public int doFilter(String[] params) {
2      Pattern pattern = (Pattern) new Pattern().addField("campo0", params[0]);
3
4      String param1 = params[1];
5      String param2 = params[2];
6      String param3 = params[3];
7
8      IFilter filter = new IFilter() {
9              // Execucao local
10             @Override
11             public Expression localFilter() {
12                 StringVariable field1 = new StringVariable("campo1");
13                 Expression exp1 = eq(field1, new StringConstant(param1));
14                 StringVariable field2 = new StringVariable("campo2");
15                 Expression exp2 = eq(field2, new StringConstant(param2));
16
17                 return eq(exp1, exp2);
18             }
19
20             // Execucao remota
21             @Override
22             public Expression remoteFilter() {
23                 StringVariable field1 = new StringVariable("campo1");
24                 Expression exp1 = eq(field1, new StringConstant(param2));
25                 StringVariable field2 = new StringVariable("campo2");
26                 Expression exp2 = eq(field2, new StringConstant(param3));
27
28                 return eq(exp1, exp2);
29             }
30     };
31
32     List<Tuple> list = (List<Tuple>) Caos.getInstance().filter(pattern, filter);
33
34     return list.size();
35 }
```

O desenvolvedor pode escolher quais filtros ele deseja implementar. Os filtros que são executados localmente são filtros mais simples e que requerem menos poder de processamento. Já os filtros que são executados remotamente pode aplicar algoritmos mais complexos como algoritmos de recomendação e *machine learning*. Para criar um filtro, é necessário primeiro definir um `Pattern`. Essa abordagem foi herdada do COP, em que um `Pattern` é utilizado quando o desenvolvedor deseja realizar buscas onde não são aplicados critérios mais refinados. A Listagem 4 apresenta as configurações que são necessárias para utilizar a solução. A linha

1 é utilizada para indicar para onde os dados devem ser migrados, i.e, onde está localizado o *cloudlet* que deve ser usado. A linha 15 mostra o método `makeData()`, que serve para gerar um estado de um dado. O estado de um dado é preenchido por meio um formulário na aplicação (recupera o valor do *EditText*).

Listagem 4 – Configuração da classe cadastro de paciente

```
1  @CaosConfig(primaryEndpoint = "10.0.2.2")
2  public class MainActivity extends AppCompatActivity {
3      @DataOffloading
4      Usuario usuario;
5
6      @Inject(Read.class)
7      IRead read;
8
9      String cpf;
10     date dataNascimento;
11     String nome;
12
13     ...
14     usuario = new Usuario(cpf, dataNascimento, nome);
15     Caos.getInstance().makeData();
16     ...
17 }
```

A Listagem 5 mostra como foi implementada a classe Prontuário, que representa os dados que devem ser migrados entre o dispositivo e o *cloudlet*. Dentre os dados que compõem o prontuário, incluem-se o CPF do paciente, sintomas associados, medicações prescritas e diagnósticos fornecidos. Na Linha 9, 10 e 11 são descritos os dados dos sensores do tipo de dado Sensor.

Listagem 5 – Implementação da classe onde são definidos os dados que devem ser migrados

```
1  public class Prontuario {
2      @NotOffloadable
3      public String cpf;
4
5      public List<String> sintomas;
6      public List<String> medicoes;
7      public List<String> diagnosticos;
8
9      public Sensor sensorTemperatura;
10     public Sensor sensorLocalizacao;
11     public Sensor sensorBatimento;
12 }
```

Na solução implementada, os dados dos sensores são simulados como detalhado na listagem 6. O sensor de localização utilizou dados simulados, gerando dois números aleatórios do tipo *float*.

Listagem 6 – Implementação simulada do sensor de localização

```
1      ...
2      ArrayList<Double> list = new ArrayList<>();
3      list.add(BigDecimal.valueOf(new Random().nextDouble() * (3) + 2).setScale(2,
           RoundingMode.HALF_UP)
4                          .doubleValue());
5      list.add(BigDecimal.valueOf(new Random().nextDouble() * (10) + 30)
6                          .setScale(2, RoundingMode.HALF_UP)
7                          .doubleValue());
8
9      medical.sensorLocalizacao = new Sensor("smartwatch.location", list);
10 }
```

A seguir é exemplificado a representação dos dados utilizado na aplicação Medical App, com uma leitura do dado “Informações do paciente” no formato de tupla para o DOP-MS. Nessa estrutura, o dispositivo móvel tem identificação “`745744441`” e utiliza uma aplicação com identificação “`br.ufc.mdcc.medical`”.

$$
\begin{aligned}
t = \langle & (\texttt{IdApp}, \text{“}\texttt{br.ufc.mdcc.medical}\text{”}) \\
& (\texttt{IdClass}, \text{“}\texttt{br.ufc.mdcc.medical.user}\text{”}) \\
& (\texttt{IdDevice}, \text{“}\texttt{745744441}\text{”}) \\
& (\texttt{Timestamp}, \texttt{1610551098}) \\
& (\texttt{ObjectData}, \qquad (5.1) \\
& \quad \texttt{cpf}, \text{“}\texttt{123.456.789} - \texttt{10}\text{”} \\
& \quad \texttt{name}, \text{“}\texttt{Joo da Silva}\text{”} \\
& \quad \texttt{age}, \texttt{20}) \\
& (\texttt{Sensor}, \qquad (5.2) \\
& \quad \texttt{type}, \text{“}\texttt{sensor.heart}\text{”} \\
& \quad \texttt{value}, \text{“}\texttt{131}\text{”} \\
& \quad \texttt{timestamp}, \texttt{1610531098}) \rangle
\end{aligned}
$$

5.2.2 Descrição do Experimento

Para avaliar a arquitetura do DOP-MS, três tipos de experimentos foram realizados. O primeiro experimento foi uma validação da solução utilizando a aplicação Medical App, com objetivo de validar a corretude dos filtros quando da realização do *offloading* de dados. Em seguida, um segundo experimento comparou o DOP-MS com o COP, para avaliar se a abordagem de microsserviços trouxe vantagens para as operações de *offloading* de dados, especialmente em relação à escalabilidade dos serviços. Por fim foram realizados testes para avaliar a execução da solução realizando *offloading* de processamento e dados.

Para realizar os testes foi preparado um ambiente configurado em um *laptop* modelo Acer Nitro 5, 4 cores, 2.4 GigaHertz, I5 (9º geração), 16GB de Ram e SSD (512GB) com o sistema operacional Mint 20 que atuou como um *cloudlet* nos experimentos. Este mesmo laptop foi utilizado para virtualizar emuladores de dispositivos móveis Android 29 que atuaram como clientes do serviço de *offloading*. Cada emulador foi configurado para ter 1GB de Ram. O uso de emuladores foi uma estratégia para se obter maior controle dos experimentos (dispositivos com a mesma configuração de hardware e software).

Para a realização dos experimentos, foi necessário a utilização de duas aplicações.

Além da Medical App, utilizou-se uma aplicação já existente chamada de *Integers*, já tinha sido utilizada por Gomes *et al.* (2017a) para experimentos com o COP. Esta aplicação foi escolhida por possibilitar um maior controle sobre os dados do experimento de escalabilidade.

5.2.3 Teste de validação da solução

Para a realização do teste foi instalada a aplicação Medical App em um emulador executado no laptop utilizado nos testes. Na seção 5.2.1 é detalhado o processo de implementação dessa aplicação.

Para validar os experimentos foram necessários criar dados relativos a médicos, pacientes e prontuários de atendimento. Foram cadastrados três médicos, dez pacientes e para cada paciente foi realizado o cadastro de dez prontuários de atendimento, com isso totalizando cem prontuários de atendimento. Isso foi necessário para que quando fosse realizado os testes de corretude pudesse ser verificado se esses valores estavam de acordo com os valores que foram armazenados no banco de dados. A figura 17 ilustra o registro de um prontuário de atendimento médico.

Figura 17 – Tela de um registro de Prontuário - Medical App

Fonte: Elaborado pela autora

A Figura 17 mostra a realização de um registro de prontuário médico. É possível observar na figura que o campo CPF tem um botão abaixo que possibilita que o médico busque por um paciente com base no CPF fornecido. Caso esse paciente não esteja cadastrado o médico

deverá fazer o cadastro do paciente para obter as informações e relacionar o prontuário de atendimento ao paciente.

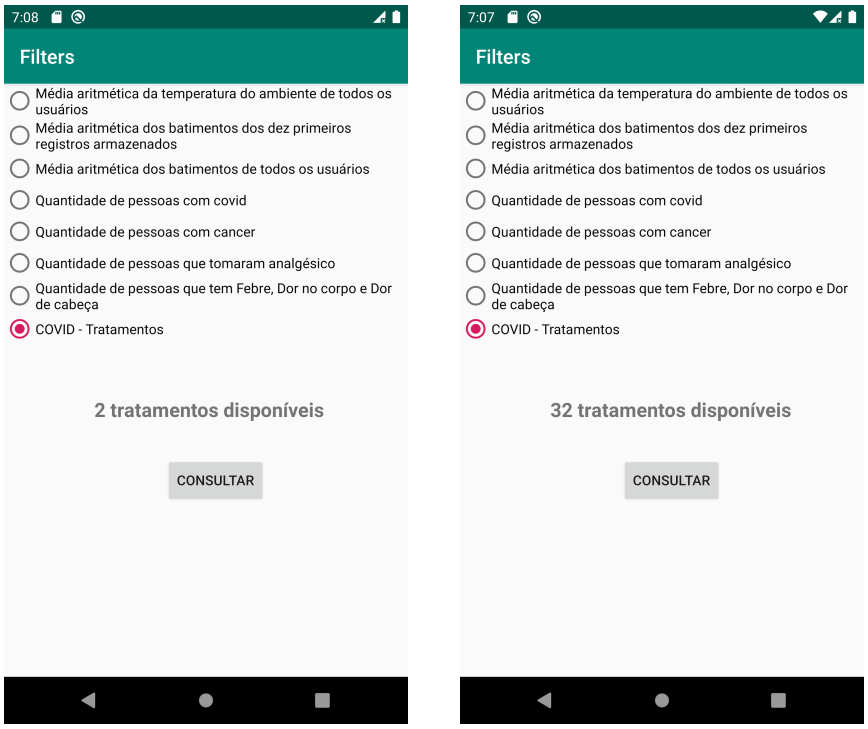
Para realização dos testes de validação, também foi proposto o seguinte cenário de caso de uso onde três médicos utilizam a aplicação Medical App em um hospital para registrar seus atendimentos, e utilizam *offloading* de dados para o *cloudlet* do hospital. Nesse caso de uso foram implementado dois filtros que podem ser utilizados para inferir situações e os médicos podem realizar os filtros localmente ou utilizando a base de dados disponível no *cloudlet*.

O primeiro filtro foi utilizado para buscar tratamentos disponíveis para uma pessoa diagnosticada com covid, enquanto o segundo foi utilizado para com base em um conjunto de sintomas de um paciente buscar com quais doenças foram diagnosticas as pessoas que tiveram aquele conjunto de sintomas semelhantes. Esses dois filtros foram executados localmente e remotamente, sendo que no segundo caso, utilizou-se o banco de dados hospedado no *cloudlet* para verificar os resultados obtidos.

A Figura 18 apresenta o resultado obtido com a execução do primeiro filtro, em que simula a busca por quais tratamentos foram oferecidos a uma pessoa diagnosticada com covid. A figura 18 mostra a quantidade de tratamentos, bem como a possibilidade de se visualizar os tratamentos fornecidos para os casos encontrados. A Figura 18 (a) apresenta o resultado da execução do filtro realizado localmente que retornou um total de 02 tratamentos disponíveis para consulta, enquanto realizado de forma remota utilizando o serviço de *offloading* como pode ser observado na Figura 18 (b) foi obtido um total de 32 tratamentos disponíveis para consulta pelo médico. Ao final do teste foi validado a corretude dos filtros através da verificação dos dados armazenados no espaço de tuplas local e no *cloudlet*, com isso foi verificado que o filtro estava funcionando corretamente.

A Figura 19 apresenta o resultado obtido com a execução do segundo filtro, utilizado para que um médico com base em sintomas descritos pelo paciente possa realizar uma busca e verificar com quais doenças foram diagnosticas as pessoas que tiveram os mesmos sintomas que o paciente. Nesse filtro, mais especificamente, o médico verifica pacientes que tem dois sintomas "Febre e dor no corpo". A Figuras 19 apresenta as doenças com base nos sintomas informados pelo médico. A Figura 19 (a) apresenta o resultado da execução do filtro realizado localmente que retornou um total de 02 doenças identificadas, e quando realizado de forma remota utilizando o serviço de *offloading* como pode ser observado na Figura 19 (b) foi obtido um total de 4 doenças identificadas.

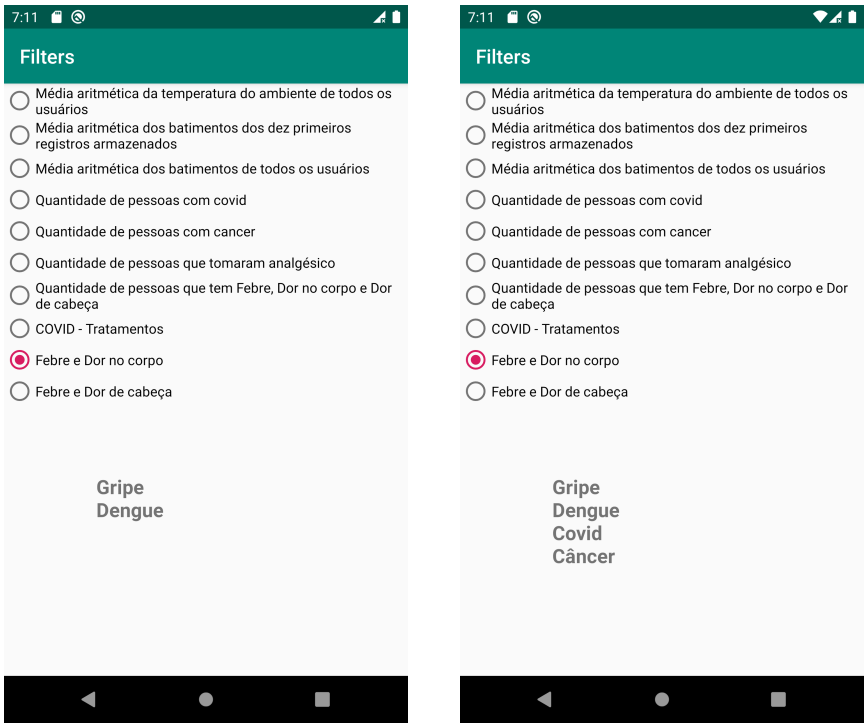
Figura 18 – *Screenshots* da Execução do Filtro de Consulta por tratamento ao Covid19



(a) (b)

Fonte: Elaborado pela autora

Figura 19 – *Screenshots* da Execução do Filtro 02



(a) (b)

Fonte: Elaborado pela autora

Com o resultado da execução desses dois filtros, pode-se observar que quando é utilizado o serviço de *offloading* para agregar as informações que estão centralizadas no *cloudlet*, obtêm-se respostas mais ricas, pois é utilizado uma quantidade maior de informações.

Também foi realizado um teste para verificar se a politica de anonimização de dados funcionou conforme esperado. A listagem 7 representa uma tupla no servidor do DOP-MS, onde verifica-se que o CPF não foi enviado na requisição, e portanto não foi armazenado no banco de dados do *cloudlet*. Conforme apresentado na listagem **??**, o atributo `cpf` foi anotado com a anotação `@NotOffloadable`, indicando que tal dado não deveria ser migrado para o *cloudlet*. Após a migração do dado foi realizado uma consulta no *cloudlet* para verificar se o dado realmente foi anonimizado. Com isso foi possível observar que a politica de anonimização de dados funcionou conforme esperado.

Listagem 7 – Tupla do DOP-MS - Register

```
{
   "id_app":"com.example.register",
   "id_class":"com.example.register.Register",
   "id_device":"4a773bd6de9d3cf9",
   "name":"Rafaela Silva",
   "dataNascimento":"08/05/1992",
}
```

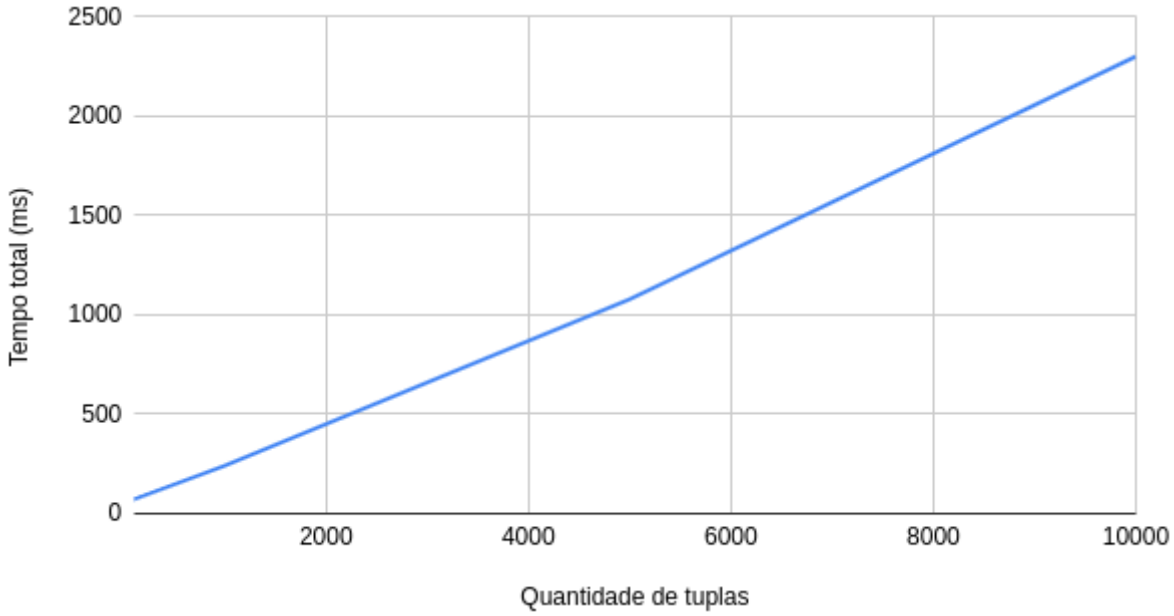
Foi realizado um teste para verificar o tempo que é gasto com o envio de dados. Para isso foi realizado a migração de 0 até 10000 mil tuplas, com intervalos de envio de 200 tuplas por vez. O gráfico com o resultado desse teste pode ser observado na figura 20.

Realizando a analise do gráfico pode ser percebido que conforme aumenta a quantidade de tuplas, o tempo gasto para realização do *offloading* também aumenta. Porém como o crescimento acontece de forma linear, é mais fácil realizar uma estimativa de custo/necessidade de acordo com o aumento de uso do serviço.

5.2.4 Teste de comparação do tempo de execução do offloading de dados no COP e DOP-MS

Foram realizados testes comparativos entre as duas soluções COP e DOP-MS. O objetivo desse teste é verificar se a arquitetura de microsserviços trouxe ganhos relacionados a desempenho e escalabilidade quando comparado com a arquitetura monolítica utilizada no COP. Para a realização desse teste foi utilizado a aplicação *Integers* nos emuladores. Essa aplicação já

Figura 20 – Tempo Gasto no Envio de Dados pelo DOP-MS



Fonte: Elaborado pela autora

foi utilizada no trabalho de (GOMES *et al.*, 2017a) seu propósito exclusivo é a realização de experimentos de *benchmark* realizados no grupo de pesquisa GREat. Essa aplicação gera tuplas de dados para um intervalo definido pelo usuário. Por exemplo, se receber o valor 0 (zero) como limiar inferior e o valor 100 (cem) como limiar superior, ele irá gerar tuplas a partir de um laço contendo esse intervalo e incrementando o valor até chegar ao limiar superior (i.e., 0, 1, 2, ..., 100).

A aplicação *Integers* é simples, tanto em termos de concepção como de implementação, mas ela é útil para o experimento de *offloading* de dados. Como o usuário consegue controlar a quantidade de tuplas geradas e controlar a quantidade de tuplas que serão recuperadas a partir dos filtros, é possível saber exatamente a quantidade de dados recuperada do serviço proposto. A partir da execução dos filtros na aplicação, foi medido o tempo gasto para realização das operações de *offloading*.

Para realizar os testes comparativos entre as duas soluções foram utilizadas as configurações de hardware já apresentadas anteriormente. Para calcular o tempo de *offloading* e de execução dos filtros foi medido o tempo gasto a partir da chamada ao método responsável pela recuperação das tuplas, até o retorno da execução desse método.

Na Tabela 3 são apresentados os cenários de testes estabelecidos. Foram utilizados dispositivos simulados e durante a execução do teste foi aumentada a quantidade de instâncias do serviço de *offloading* de dados. Para iniciar o teste foi requisitado a criação de 5000 mil tuplas, e

as consultas foram realizadas iniciando em 0 e aumentando de 200 em 200 até chegar em 4000 mil tuplas. Para cada execução foi medido o tempo gasto, e foram realizadas trinta repetições de cada cenário proposto.

Tabela 3 – Especificação dos Cenários de Teste

Quant. de dispositivos	**Quant. de instâncias do serviço**
01	01
02	01
03	01
01	02
02	02
03	02
01	03
02	03
03	03

Fonte: Elaborado pela autora

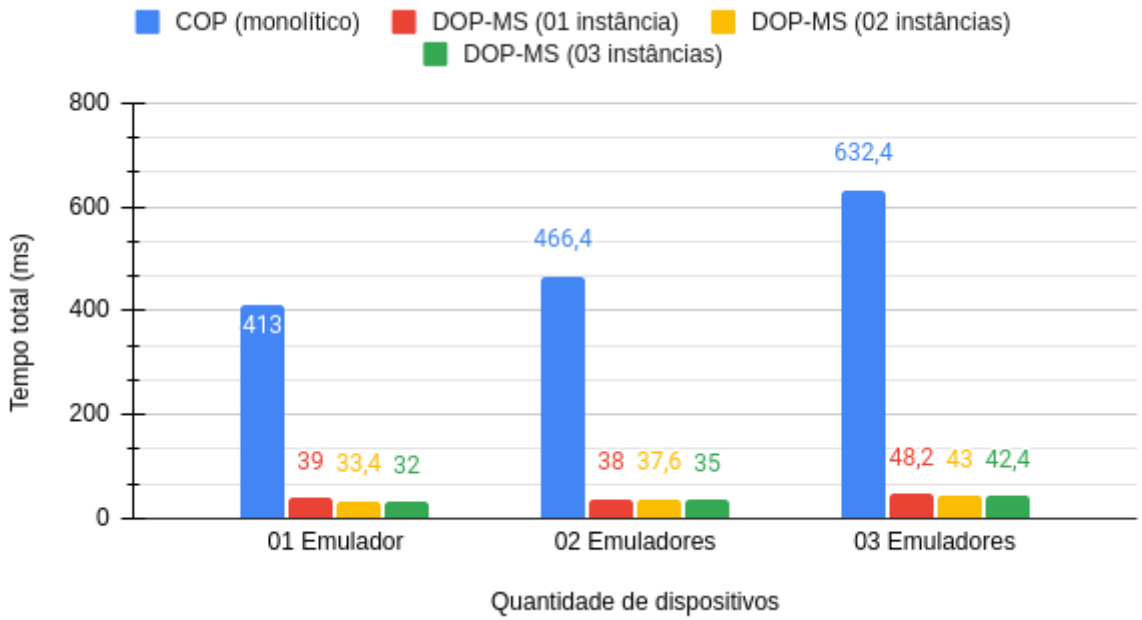
Os dados que foram obtidos a partir dos testes com as duas soluções foram coletados e armazenados em uma planilha, para posterior análise. Foi feita a média aritmética do tempo gasto nas 30 vezes em que os experimentos foram executados.

Na Figura 21 é apresentado o gráfico de comparação entre o tempo de execução do *offloading* de dados no COP e no DOP-MS. Este teste foi realizado para que fosse possível verificar os ganhos obtidos com a utilização de uma arquitetura de microsserviços em relação à sua versão monolítica. São apresentados os resultados obtidos com 01, 02 e 03 dispositivos e com 200 tuplas. A escolha desta quantidade de informação se deu pois com isso são necessários poucos recursos para realizar a migração desses dados. Durante a execução do teste também foi realizado o aumento da quantidade de instâncias do serviço de *offloading* de dados (01, 02 e 03 instâncias).

Ao realizar a analise do gráfico, é observado que o DOP-MS tem um desempenho melhor que o COP, e que conforme aumenta-se a quantidade dispositivos, também aumenta o tempo necessário para a execução do *offloading* de dados. Durante a execução dos testes também foi aumentado a quantidade de instâncias do serviço de *offloading* de dados e, com esse procedimento, reduziu-se o tempo de execução do offloading de dados. Então pode-se verificar que em todos os cenários em que o teste foi realizado, o DOP-MS obteve um resultado melhor na execução do *offloading* de dados e conseguiu reduzir seu tempo de execução conforme era aumentado a quantidade de instâncias do serviço de *offloading*.

Na Figura 22 é apresentado o gráfico de comparação entre o tempo de execução do

Figura 21 – Tempo de execução do *offloading* de dados com 200 tuplas



Fonte: Elaborado pela autora

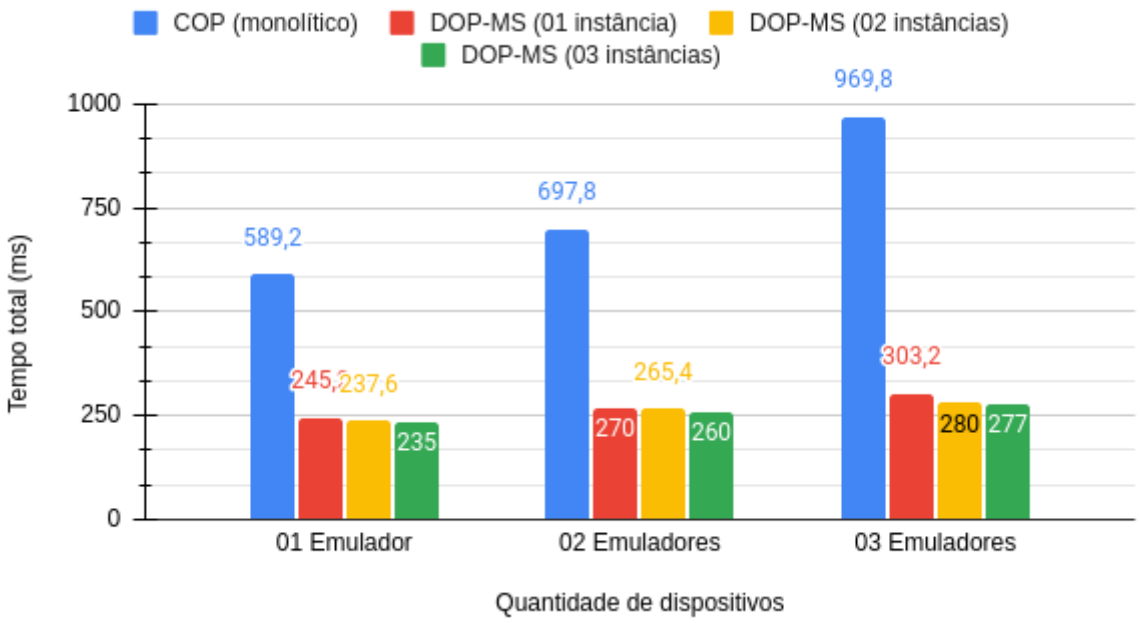
offloading de dados no COP e no DOP-MS agora com 4000 mil tuplas. São apresentados os resultados obtidos com 01, 02 e 03 dispositivos, é uma quantidade bem superior de dados se comparado a figura 21, e com isso são necessários mais recursos para realizar a migração desses dados. Durante a execução do teste também foi realizado o aumento da quantidade de instâncias do *offloading* de dados (01, 02 e 03 instâncias).

O resultado obtido com o *offloading* de 4000 mil tuplas teve um gasto de tempo bem superior ao teste realizado com 200 tuplas, pois é percebido um crescimento no tempo de execução, mas ainda o desempenho é melhor com o DOP-MS. Conforme aumenta a quantidade de instâncias do serviço de *offloading*, tem uma melhora significativa no seu desempenho. Então mais uma vez, é possível perceber que a arquitetura de microsserviços traz um ganho significativo para a solução, pois ela possibilita que seja adicionado instâncias do serviço conforme a necessidade de consumo.

5.2.5 *Teste de execução dos filtros utilizando o serviço de offloading de processamento do CAOS-MS*

Por fim, foi realizado um teste para verificar o comportamento da solução quando utiliza o *microsserviço Offloading* do CAOS-MS. O objetivo desse teste é analisar o comportamento da solução quando acontece a realização do *offloading* de processamento e de dados de

Figura 22 – Tempo de execução do *offloading* de dados com 4000 mil tuplas



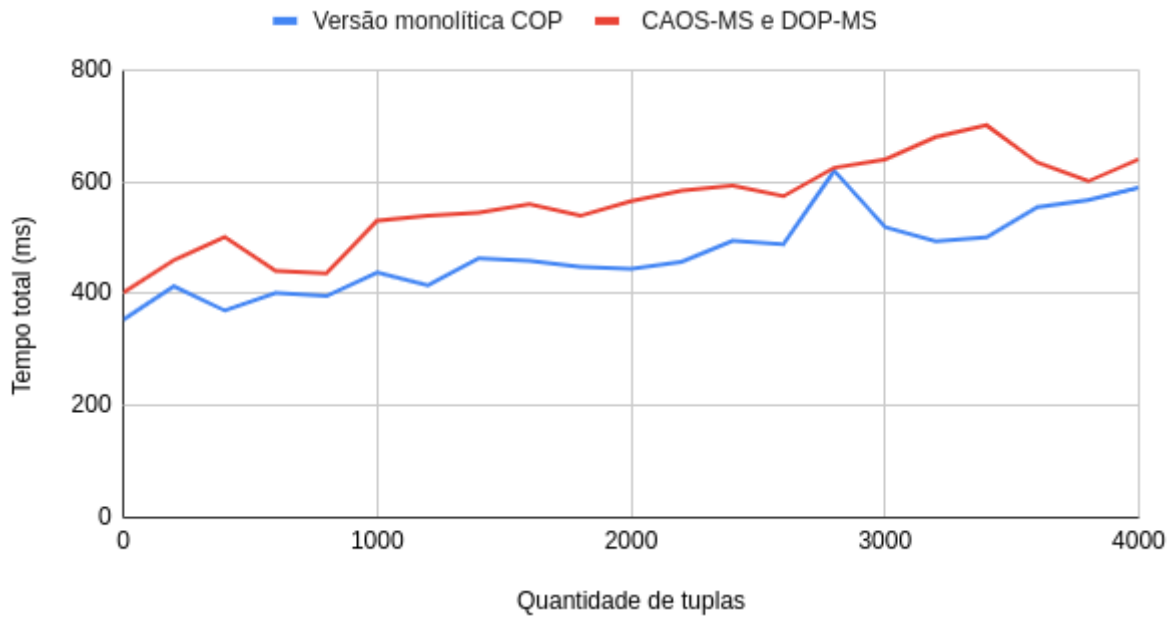
Fonte: Elaborado pela autora

forma conjunta. A aplicação utilizada neste teste foi a *Integers*, com as mesmas configurações de *hardware* apresentadas anteriormente.

Para execução desse teste foi realizado o *offloading* com 01 dispositivo (emulador) e com 5000 mil tuplas armazenadas no *cloudlet*. O filtro foi realizado de 200 em 200 tuplas, até chegar em 4000 mil tuplas (1-200, 1-400, ..., 1-4000). Este procedimento foi realizado na versão do COP (a arquitetura monolítica) e na DOP-MS com CAOS-MS.

A figura 23 apresenta o gráfico de comparação entre o tempo de execução do *offloading* de processamento no COP e no DOP-MS utilizando o serviço de *offloading* de processamento do CAOS-MS. Nesse teste, tanto o COP como o DOP-MS realiza o processamento de dados remotamente.

Figura 23 – Tempo de Execução do *Offloading* de Dados



Fonte: Elaborado pela autora

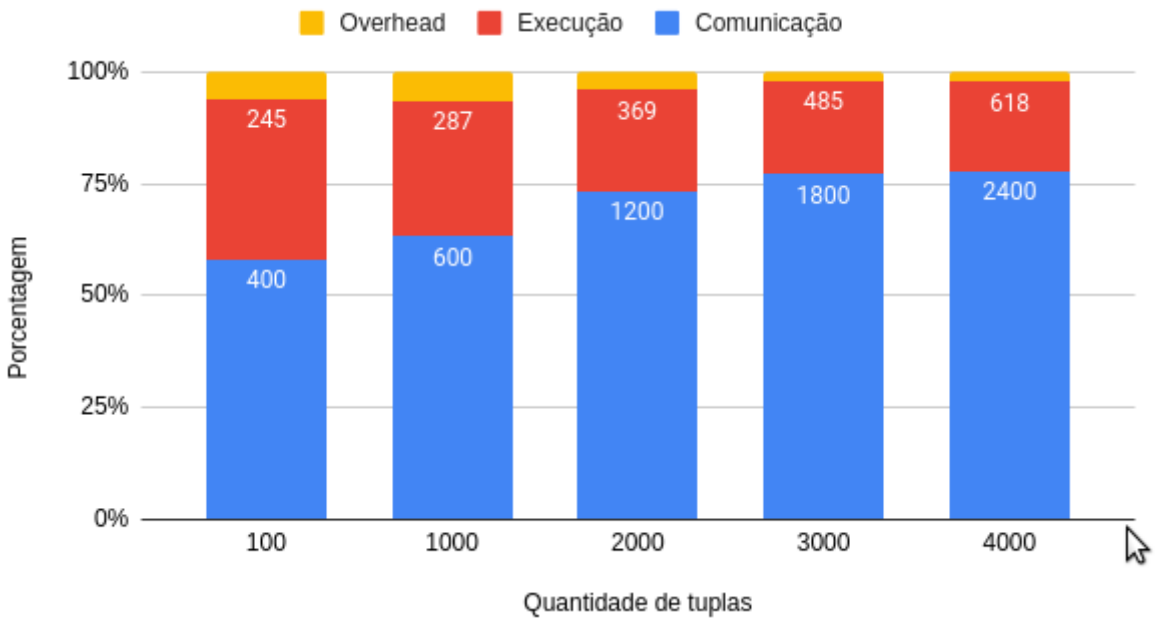
Ao realizar a analise do gráfico é observado que quando realizado a comparação no tempo de execução do processamento utilizando o CAOS-MS, o tempo gasto na realização do processamento entre as duas soluções é bem similar. Porém a arquitetura de microsserviços permite que seja criada mais instâncias do serviço de *offloading*.

Para detalhar melhor o tempo gasto na solução quando realiza o *offloading* de dados e processamento de forma conjunta, foi realizado um teste para especificar o tempo gasto durante a realização do *offloading*. Foi analisado cada detalhe da sua execução e contabilizado o tempo gasto. Com isso foi obtido o tempo que é gasto com comunicação, execução e *overhead*[1]. As informações obtidas são descritas no gráfico da figura 24.

No gráfico é descrito o tempo gasto com o *offloading* de 100, 1000, 2000, 3000 e 4000 mil tuplas. Com 100 tuplas podemos observar que 58,2% do tempo foi gasto comunicação, 35,7% com execução e 6,1% com *overhead*. Com 1000 tuplas podemos observar que 63,2% do tempo foi gasto comunicação, 30,2% com execução e 6,5% com *overhead*. Com 2000 tuplas podemos observar que 73,4% do tempo foi gasto comunicação, 22,6% com execução e 4,0% com *overhead*. Com 3000 tuplas podemos observar que 77,1% do tempo foi gasto comunicação, 20,8% com execução e 2,1% com *overhead* e com 4000 tuplas podemos observar que 77,7% do tempo foi gasto comunicação, 20,0% com execução e 2,3% com *overhead*.

[1] tempo que não é gasto com comunicação e nem com execução (ex: tempo da passagem de parâmetros e passagem entre os métodos)

Figura 24 – Tempo Detalhado



Fonte: Elaborado pela autora

Com o resultado desse teste é observado um gargalo ocasionado por um problema de comunicação que ocorre entre a VM (do CAOS-MS) e o serviço de *offloading* de dados. Outra informação que pode ser observada com a execução desse teste é que quanto maior a quantidade de dados, maior a influência da comunicação no tempo total de execução do *offloading*. Porém, as duas soluções de forma integradas conseguem funcionar e oferecer seus serviços de forma satisfatória possibilitando a sua utilização para aplicações que realizam *offloading* de dados e processamento.

5.2.6 Análise Estatística

Foram realizados testes estatísticos para analisar o desempenho da solução proposta quando é variado a quantidade de instâncias do microsserviço. No cenário de teste elaborado foi solicitado a criação de 15 mil tuplas de dados. Foram elaborados cinco cenários: 05 emuladores / 01 instância do serviço, 05 emuladores / 02 instâncias do serviço, 05 emuladores / 03 instâncias do serviço, 05 emuladores / 04 instâncias do serviço e 05 emuladores / 05 instâncias do serviço.

Para cada cenário foram realizadas 40 repetições e avaliado o tempo gasto na realização do *offloading*, que é composto por tempo gasto com execução, comunicação e *overheard*. O tempo gasto com execução é o tempo calculado desde a requisição, execução no ambiente servidor e o instante em que chega a resposta. A comunicação é o tempo gasto quando é realizado

upload da requisição, transferência entre os microsserviços e *download* da resposta. *Overheard* é o tempo durante o processo de *offloading* que é gasto com passagem de parâmetros e métodos.

Visando reforçar a relevância dos resultados e das conclusões apresentadas anteriormente, foram aplicados testes estatísticos ANOVA e T-Student (BRUCE; BRUCE, 2019) nos resultados obtidos com os experimentos realizados. Ambos os testes consistiram em avaliar estatisticamente se as médias calculadas são diferentes entre si. O teste ANOVA faz a avaliação considerando todos os componentes de uma vez, enquanto o T-Student faz a avaliação desses componentes em pares. Para isso, ambos os testes definem uma hipótese nula (H0) que todas as médias avaliadas são iguais. Após a aplicação do teste, com base no resultado obtido, é possível rejeitar ou não tal hipótese. Caso a hipótese nula seja rejeitada, o teste indica que nem todas as médias são iguais, assim, dentre os componentes avaliados, podem ou não existir médias estatisticamente diferentes umas das outras. Assim, o objetivo desta avaliação é identificar quais médias são estatisticamente diferentes e relevantes na análise dos resultados.

De início, os resultados dos cinco cenários foram separados em grupos com base na métrica (Comunicação, Execução, Overhead e Total) e na quantidade de tuplas (1000 e 15000) avaliadas. Por exemplo, considerando a primeira métrica (Comunicação), um grupo é formado pelos resultados dos cenários nesta métrica para uma quantidade de 1000 tuplas e outro grupo é composto pelos resultados na mesma métrica, porém para uma quantidade de 15000 tuplas. Tal mecanismo se repete para todas as demais métricas. Ao final deste passo, foram produzidos oito grupos.

Feita essa separação inicial, o teste estatístico ANOVA foi aplicado em cada um dos grupos e seus resultados são exibidos na Tabela 5. Em todas as métricas, a hipótese nula foi rejeitada, o que indica que nem todas as médias são iguais entre si. Continuando a avaliação, o teste T-Student foi aplicado para cada par de componentes dentro de cada grupo e seus resultados são apresentados na Tabela 4. Ao contrário do observado com os resultados do teste ANOVA, foi constatado que, em alguns casos, todos eles relacionados a métrica de Overhead, não foi possível rejeitar a hipótese nula, o que impede afirmar que as médias são diferentes nestes casos. Acredita-se que este comportamento ocorre porque a ação associada ao Overhead é rápida e relativamente simples quando comparada as demais métricas, o que resulta em tempos de execução bastante próximos uns dos outros. Para todos os demais casos, a hipótese nula foi rejeitada, o que confirma que as médias são estatisticamente diferentes, relevantes e reforçam as observações apresentadas anteriormente nos testes.

Tabela 4 – Resultados dos testes ANOVA

	1000				15000			
	05Emu/01Ins X 05Emu/02Ins X 05Emu/03Ins X 05Emu/04Ins X 05Emu/05Ins							
Métrica	**Comu**	**Exec**	**Over**	**Tot**	**Comu**	**Exec**	**Over**	**Tot**
ANOVA	$7.4{\cdot}10^{-134}$	$1.8{\cdot}10^{-197}$	$8.1{\cdot}10^{-23}$	$1.5{\cdot}10^{-46}$	$1.7{\cdot}10^{-150}$	$6.8{\cdot}10^{-178}$	$7.0{\cdot}10^{-24}$	$3.6{\cdot}10^{-147}$
Rejeita H0?	51	51	51	51	51	51	51	51

Fonte: Elaborado pela autora

Tabela 5 – Resultados dos testes T-Student

	1000				15000			
	05Emu/01Ins X 05Emu/02Ins							
Métrica	**Comu**	**Exec**	**Over**	**Tot**	**Comu**	**Exec**	**Over**	**Tot**
T-Student	$4.0{\cdot}10^{-26}$	$4.1{\cdot}10^{-61}$	$2.3{\cdot}10^{-01}$	$1.5{\cdot}10^{-17}$	$3.1{\cdot}10^{-36}$	$9.1{\cdot}10^{-34}$	$1.1{\cdot}10^{-01}$	$2.0{\cdot}10^{-17}$
Rejeita H0?	51	51	55	51	51	51	55	51
	05Emu/01Ins X 05Emu/03Ins							
Métrica	**Comu**	**Exec**	**Over**	**Tot**	**Comu**	**Exec**	**Over**	**Tot**
T-Student	$2.4{\cdot}10^{-35}$	$5.1{\cdot}10^{-71}$	$1.0{\cdot}10^{-01}$	$5.8{\cdot}10^{-27}$	$4.1{\cdot}10^{-36}$	$2.3{\cdot}10^{-49}$	$3.3{\cdot}10^{-02}$	$2.1{\cdot}10^{-40}$
Rejeita H0?	51	51	55	51	51	51	51	51
	05Emu/01Ins X 05Emu/04Ins							
Métrica	**Comu**	**Exec**	**Over**	**Tot**	**Comu**	**Exec**	**Over**	**Tot**
T-Student	$1.2{\cdot}10^{-71}$	$3.7{\cdot}10^{-81}$	$8.6{\cdot}10^{-02}$	$1.1{\cdot}10^{-11}$	$7.8{\cdot}10^{-64}$	$6.4{\cdot}10^{-105}$	$1.8{\cdot}10^{-01}$	$2.3{\cdot}10^{-79}$
Rejeita H0?	51	51	55	51	51	51	55	51
	05Emu/01Ins X 05Emu/05Ins							
Métrica	**Comu**	**Exec**	**Over**	**Tot**	**Comu**	**Exec**	**Over**	**Tot**
T-Student	$4.6{\cdot}10^{-78}$	$2.5{\cdot}10^{-91}$	$1.3{\cdot}10^{22}$	$7.4{\cdot}10^{-33}$	$1.7{\cdot}10^{-86}$	$1.7{\cdot}10^{-111}$	$1.1{\cdot}10^{-21}$	$6.7{\cdot}10^{-92}$
Rejeita H0?	51	51	51	51	51	51	51	51
	05Emu/02Ins X 05Emu/03Ins							
Métrica	**Comu**	**Exec**	**Over**	**Tot**	**Comu**	**Exec**	**Over**	**Tot**
T-Student	$7.5{\cdot}10^{-05}$	$3.4{\cdot}10^{-33}$	$2.8{\cdot}10^{-01}$	$1.5{\cdot}10^{-04}$	$2.8{\cdot}10^{-02}$	$9.2{\cdot}10^{-33}$	$2.0{\cdot}10^{-03}$	$2.2{\cdot}10^{-30}$
Rejeita H0?	51	51	55	51	51	51	51	51
	05Emu/02Ins X 05Emu/04Ins							
Métrica	**Comu**	**Exec**	**Over**	**Tot**	**Comu**	**Exec**	**Over**	**Tot**
T-Student	$4.4{\cdot}10^{-43}$	$7.6{\cdot}10^{-60}$	$2.2{\cdot}10^{-01}$	$1.2{\cdot}10^{-04}$	$5.9{\cdot}10^{-47}$	$1.7{\cdot}10^{-78}$	$4.4{\cdot}10^{-01}$	$3.1{\cdot}10^{-66}$
Rejeita H0?	51	51	55	51	51	51	51	51
	05Emu/02Ins X 05Emu/05Ins							
Métrica	**Comu**	**Exec**	**Over**	**Tot**	**Comu**	**Exec**	**Over**	**Tot**
T-Student	$1.9{\cdot}10^{-52}$	$1.7{\cdot}10^{-80}$	$3.2{\cdot}10^{-21}$	$7.9{\cdot}10^{-07}$	$5.6{\cdot}10^{-69}$	$2.0{\cdot}10^{-85}$	$1.3{\cdot}10^{-19}$	$8.7{\cdot}10^{-77}$
Rejeita H0?	51	51	51	51	51	51	51	51
	05Emu/03Ins X 05Emu/04Ins							
Métrica	**Comu**	**Exec**	**Over**	**Tot**	**Comu**	**Exec**	**Over**	**Tot**
T-Student	$3.9{\cdot}10^{-39}$	$4.1{\cdot}10^{-41}$	$4.0{\cdot}10^{-01}$	$5.9{\cdot}10^{-13}$	$5.2{\cdot}10^{-43}$	$5.2{\cdot}10^{-40}$	$8.1{\cdot}10^{-03}$	$1.4{\cdot}10^{-24}$
Rejeita H0?	51	51	55	51	51	51	51	51
	05Emu/03Ins X 05Emu/05Ins							
Métrica	**Comu**	**Exec**	**Over**	**Tot**	**Comu**	**Exec**	**Over**	**Tot**
T-Student	$6.6{\cdot}10^{-50}$	$1.3{\cdot}10^{-71}$	$3.3{\cdot}10^{-19}$	$1.5{\cdot}10^{-01}$	$3.1{\cdot}10^{-63}$	$1.0{\cdot}10^{-50}$	$1.2{\cdot}10^{-21}$	$9.6{\cdot}10^{-37}$
Rejeita H0?	51	51	51	51	51	51	51	51
	05Emu/04Ins X 05Emu/05Ins							
Métrica	**Comu**	**Exec**	**Over**	**Tot**	**Comu**	**Exec**	**Over**	**Tot**
T-Student	$2.2{\cdot}10^{-23}$	$1.0{\cdot}10^{-58}$	$1.2{\cdot}10^{-14}$	$6.1{\cdot}10^{-18}$	$5.8{\cdot}10^{-23}$	$3.4{\cdot}10^{-60}$	$3.2{\cdot}10^{-14}$	$1.6{\cdot}10^{-32}$
Rejeita H0?	51	51	51	51	51	51	51	51

Fonte: Elaborado pela autora

5.3 Considerações Finais

Este capítulo apresentou uma aplicação móvel que é baseada no cenário motivador apresentado na seção 4.1, e testes de validação para verificar o funcionamento da aplicação desenvolvida. Também foram realizados testes desempenho na solução em comparação com o COP, para verificar se houve ganhos com a utilização da arquitetura de microsserviços. Por fim, foram executados testes para verificar o comportamento da solução quando utiliza o serviço de processamento do CAOS-MS.

Com a prova de conceito foi possível perceber como é importante a migração dos dados para um ambiente centralizador, e com isso se utilizar dos benefícios provisionados pela migração de dados. Essa migração é vantajosa tanto em termos de obter mais espaço para armazenamento, acesso a informações que estão centralizadas no *cloudlet* e também a possibilidade de inferir situações com base nos dados disponíveis no *cloudlet* que fornecem uma grande quantidade de informações juntamente com a utilização dos filtros.

O principal objetivo dessa aplicação é registrar os prontuários de atendimento médico dos pacientes que procuram a unidade de saúde. Quando a aplicação faz uso da solução proposta nessa dissertação ela obtém ganhos significativos: *(i)* mais espaço para armazenamento (e.g., os prontuários podem ser armazenados no banco de dados do *cloudlet*), *(ii)* acesso a informações dos pacientes que foram atendidos por outros médicos (e.g., acessando o histórico de atendimento médico), e também *(iii)* a possibilidade de inferir situações com base nos dados disponíveis no *cloudlet* que fornecem uma grande quantidade de informações juntamente com a utilização dos filtros (e.g., quantidade de pessoas diagnosticas com uma doença, tratamentos que foram realizados para doenças especificas e com base em um conjunto de sintomas diagnosticar possíveis doenças).

Além da prova de conceito, também foi utilizada uma aplicação chamada *Integers*, para realização dos testes de desempenho. O experimento consistia em medir o tempo gasto na realização do *offloading* e realizar uma comparação com o COP. O experimento detectou que a solução com uma arquitetura de microsserviços tem um maior desempenho na realização do *offloading* de dados.

Durante os testes foi verificado o que ocorre quando a solução utiliza o *offloading* de processamento do CAOS-MS. Com os testes foi percebido que a solução integrada tem um tempo de execução superior ocasionado por um gargalo de comunicação. Porém as duas soluções funcionando em conjunto oferecem seus serviços de forma satisfatória para os desenvolvedores

de aplicações móveis que realizam *offloading*.

Por fim, foi realizada uma analise estatística que confirma que as médias são estatisticamente diferentes, relevantes e reforçam as observações apresentadas no resultado dos testes.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este capítulo resume o que foi discutido e desenvolvido nesta dissertação de mestrado. A seção 6.1 apresenta os resultados alcançados com a realização da proposta, enquanto na seção 6.2 são apresentadas as principais limitação encontrada na solução. Por fim, a seção 6.3 discorre sobre os trabalhos futuros para melhorias deste trabalho.

6.1 Resultados Alcançados

Esse trabalho apresentou um serviço de *offloading* de dados com suporte a anonimização de dados desenvolvido em uma abordagem de microsserviços, essa solução é denominada de DOP-MS. Esse serviço amplia as possibilidades para os desenvolvedores de aplicações móveis que utilizam o compartilhamento de dados dos usuários do sistema. O DOP-MS evolui uma solução monolítica previamente construída, porém com restrições de escalabilidade e com forte acoplamento chamada COP. Nesse estudo foi detalhado o processo de concepção do DOP-MS.

O trabalho proposto desenvolveu uma solução que fornece suporte ao *offloading* de dados e permite que o desenvolvedor escolha quais dados serão migrados e quais dados serão anonimizados. Foram incluídas funcionalidades que permitem lidar com: *(i)* armazenamento dos dados do domínio da aplicação e dados contextuais gerados pelos usuários do sistema; *(ii)* a disseminação de dados entre o dispositivo móvel e o ambiente de *cloudlet* de forma transparente e configurável; e *(iii)* a anonimização dos dados de forma ajustável, para restringir a divulgação e difusão de dados de cada usuário do sistema.

Em relação à política de privacidade da solução proposta, foi adotada uma politica de anonimização de dados chamada de supressão, onde o desenvolvedor irá marca os dados que deseja anonimizar com a anotação chamada `@NotOffloadble`.

Em relação ao momento em que os dados são enviados do dispositivo móvel para o *cloudlet*, foram utilizadas as politicas definidas no COP. Atualmente na solução, três estratégias de envio encontram-se implementadas: por tempo, por quantidade de tuplas e orientado a conexão *Wi-Fi*.

Como prova de conceito nesse trabalho de dissertação, foi apresentada a aplicação Medical App, construída sobre o DOP-MS, e também apresentado como ela foi desenvolvida. Na implementação da prova de conceito, foram seguidos alguns passos necessários para toda a construção e execução da aplicação. Também foi utilizada outra aplicação chamada *Integers* para

realização dos testes de desempenho e escalabilidade. Depois de toda a implementação, foram realizados os testes de validação da solução desenvolvida. Durante os testes, percebeu-se que quando é utilizado uma base de dados centralizadas, os resultados são mais ricos na execução dos filtros, e também a arquitetura de microsserviços possibilitou um melhor desempenho se comparado com a arquitetura monolítica do COP.

No capitulo 3 foi apresentada uma tabela comparativa entre os trabalhos relacionados ao trabalho de mestrado apresentado nessa dissertação. A tabela 6 posiciona as características do DOP-MS em relação aos trabalhos relacionados.

Tabela 6 – Comparativo Entre os Trabalhos Relacionados e a Solução DOP-MS

Trabalho	**Plataforma**	**Arquitetura**	**Modelo de Dados**	**Política de Sincronização**	**Privacidade**
COP	Android	Monolítico	Chave-Valor	Flexível	Anonimização
CAOS-MS	Android	Microsserviços	Não possui	Não possui	Não possui
SMAll	Android WEB	Microsserviços	Chave-Valor	Não encontrado	Criptografia
SEMAR	WEB	Microsserviços	Chave-Valor	Controlada	Não possui
PUMA	Android WEB	Microsserviços	Chave-Valor	Não encontrado	Criptografia
(SILVA *et al.*, 2019a)	Android	Monolítico	Não encontrado	Não encontrado	*Blockchain* Anonimização Autenticação
DOP-MS	Android	Microsserviços	Chave-Valor	Flexível	Anonimização

Fonte: Elaborado pela autora

A solução apresentada nessa dissertação é utilizada por desenvolvedores Android, e foi desenvolvida utilizando uma arquitetura de microsserviços que possibilita a escalabilidade de serviços específicos no momento em que recebem uma quantidade maior de requisições. É utilizado o modelo de dados chave-valor, foi percebido no levantamento dos trabalhos relacionados que esse é o modelo mais comum entre aplicações que permite migração e compartilhamento de dados. O trabalho também apresenta uma politica de sincronização flexível, que permite ao desenvolvedor da aplicação seleciona a politica utilizada e configurar conforme a necessidade da aplicação que vai ser desenvolvida. Por fim, a privacidade de dados é garantida através de uma politica de anonimização de dados também flexível que permite ao desenvolvedor escolher quais dados deseja migrar. É importante ressaltar que o DOP-MS permite a migração de dados do domínio da aplicação e também dados contextuais.

6.2 Limitações

Apesar dos benefícios do *offloading* de dados, o trabalho apresenta limitações. Uma delas foi o baixo desempenho do DOP-MS quando integrado à um serviço de *offloading* de processamento, no caso, o CAOS-MS. Além disso, a dependência da proposta a certas tecnologias também traz problemas, especialmente do kubernetes.

6.3 Trabalhos Futuros

Como trabalhos futuros, propõe-se algumas melhorias no DOP-MS, tais como:

- Executar o DOP-MS em uma nuvem pública, para que os dados presentes no *cloudlet* seja enviado para um local único, permitindo uma capacidade de armazenamento ainda maior, assim como de processamento dos filtros;
- Realizar mais experimentos, levando em consideração mais métricas (avaliar o gasto energético em vários ambientes de execução: local, *cloudlet* e nuvem, a fim de melhorar a avaliação do serviço proposto;
- Realizar uma avaliação do DOP-MS com desenvolvedores, a fim de analisar a qualidade da solução do ponto de vista dos engenheiros de *software* em termos de facilidade de uso;

REFERÊNCIAS

AGUIARI, Davide; CONTOLI, Chiara; DELNEVO, Giovanni; MONTI, Lorenzo. **Smart mobility and sensing: Case studies based on a bike information gathering architecture.** In: SPRINGER. International Conference on Smart Objects and Technologies for Social Good. [S. l.], 2017. p. 112–121.

ÁLVAREZ, Ángel; UGARTE, Ínigo; FERNÁNDEZ, Víctor; SÁNCHEZ, Pablo. **Openmp dynamic device offloading in heterogeneous platforms.** In: SPRINGER. International Workshop on OpenMP. [S. l.], 2019. p. 109–122.

BALALAIE, Armin; HEYDARNOORI, Abbas; JAMSHIDI, Pooyan. **Migrating to cloud-native architectures using microservices:** an experience report. In: SPRINGER. European Conference on Service-Oriented and Cloud Computing. [S. l.], 2015. p. 201–215.

BALALAIE, Armin; HEYDARNOORI, Abbas; JAMSHIDI, Pooyan. **Microservices architecture enables devops:** migration to a cloud-native architecture. IEEE Software, IEEE, v. 33, n. 3, p. 42–52, 2016.

BALDWIN, Carliss Young; CLARK, Kim. **Design rules:** The power of modularity. [S. l.]: MIT press, 2000. v. 1.

BRITO, Felipe Timb; MACHADO, Javam Castro. **Preservação de privacidade de dados:** Fundamentos, técnicas e aplicações. Jornadas de atualização em informática, p. 91–130, 2017.

BRUCE, Peter; BRUCE, Andrew. **Estatística prática para cientistas de dados:** 50 conceitos essenciais. 1. ed. [S. l.]: O'Reilly Media, 2019. 392 p. ISBN 9788550806037.

CAMENISCH, Jan; FISCHER-HÜBNER, Simone; RANNENBERG, Kai. **Privacy and identity management for life.** [S. l.]: Springer Science Business Media, 2011.

CÂNDIDO, Adriano; TRINTA, Fernando; REGO, Paulo; ROCHA, Lincoln; MENDONÇA, Nabor; GARCIA, Vinicius. **Um relato sobre a migração de uma plataforma de offloading para microsserviços.** In: SBC. Anais do VII Workshop on Software Visualization, Evolution and Maintenance (VEM). [S. l.], 2019. p. 29–36.

CÂNDIDO, Adriano; TRINTA, Fernando; REGO, Paulo; ROCHA, Lincoln; MENDONÇA, Nabor; GARCIA, Vinicius. **A microservice based architecture to support offloading in mobile cloud computing.** In: Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse. [S. l.: s. n.], 2019. p. 93–102.

COSTA, Philipp; REGO, Paulo; ROCHA, Lincoln; TRINTA, Fernando SOUZA, José. **Mpos:** a multiplatform offloading system. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing. [S. l.: s. n.], 2015. p. 577–584.

CUERVO, Eduardo; BALASUBRAMANIAN, Aruna; CHO, Dae-ki; WOLMAN, Alec; SAROIU, Stefan; CHANDRA, Ranveer; BAHL, Paramvir **Maui:** making smartphones last longer with code offload. In: ACM. Proceedings of the 8th international conference on Mobile systems, applications, and services. [S. l.], 2010. p. 49–62.

DENG, Shuiguang; HUANG, Longtao; TAHERI, Javid; ZOMAYA, Albert. **Computation offloading for service workflow in mobile cloud computing.** IEEE transactions on parallel and distributed systems, IEEE, v. 26, n. 12, p. 3317–3329, 2014.

DINH, Hoang; LEE, Chonho; NIYATO, Dusit; WANG, Ping. **A survey of mobile cloud computing:** architecture, applications, and approaches. Wireless communications and mobile computing, Wiley Online Library, v. 13, n. 18, p. 1587–1611, 2013.

DRAGONI, Nicola; GIALLORENZO, Saverio; LAFUENTE, Alberto; MAZZARA, Manuel; MONTESI, Fabrizio; MUSTAFIN, Ruslan; SAFINA, Larisa. **Microservices:** yesterday, today, and tomorrow. In: Present and Ulterior Software Engineering. [S. l.]: Springer, 2017. p. 195–216.

ESPOSITO, Christian; CASTIGLIONE, Aniello; CHOO, Kim-Kwang. **Challenges in delivering software in the cloud as microservices.** IEEE Cloud Computing, IEEE, v. 3, n. 5, p. 10–14, 2016.

FAN, Chen-Yuan; MA, Shang-Pin. **Migrating monolithic mobile application to microservice architecture:** An experiment report. In: IEEE. 2017 IEEE International Conference on AI Mobile Services (AIMS). [S. l.], 2017. p. 109–112.

FEHLING, Christoph; LEYMANN, Frank; RETTER, Ralph; SCHUPECK, Walter; ARBITTER, Peter. **Cloud computing patterns:** fundamentals to design, build, and manage cloud applications. [S. l.]: Springer, 2014.

FERNANDO, Niroshinie; LOKE, Seng; RAHAYU, Wenny. **Mobile cloud computing.** Future Gener. Comput. Syst., Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 29, n. 1, p. 84–106, jan. 2013. ISSN 0167-739X. Disponível em: <http://dx.doi.org/10.1016/j.future.2012.05.023>.

FOWLER, Martin; LEWIS, James. **Microservices.** 2014. <https://martinfowler.com/articles/microservices.html>. (Acesso em: 10/01/2021).

FRIDELIN, Yohanes; ALBAAB, Mochamad; BESARI, Adnan; SUKARIDHOTO, Sritrusta; TJAHJONO, Anang. **Implementation of microservice architectures on semar extension for air quality monitoring.** In: IEEE. 2018 International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC). [S. l.], 2018. p. 218–224.

GOMES, Francisco; VIANA, Windson; ROCHA, Lincoln; TRINTA, Fernando. **On the evaluation of a contextual sensitive data offloading service:** the cop case. Journal of Information and Data Management, v. 8, n. 3, p. 197, 2017.

GOMES,Francisco; REGO, Paulo; ROCHA, Lincoln; SOUZA, José; TRINTA, Fernando. **Caos:** A context acquisition and offloading system. In: IEEE. 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC). [S. l.], 2017. v. 1, p. 957–966.

GOMES, Francisco; VIANA, Windson; ROCHA, Lincoln; TRINTA, Fernando. **A contextual data offloading service with privacy support.** In: Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web. [S. l.: s. n.], 2016. p. 23–30.

HAT, Red. **O que é Docker? |** Red Hat. 2019.<https://www.redhat.com/pt-br/topics/containers/what-is-docker>. (Acessado em 13/01/2021).

JR, Eliseu; MACHADO, Javam; MONTEIRO, Jose. **Estratégias para proteção da privacidade de dados armazenados na nuvem.** Simpósio Brasileiro de Banco de Dados. Citado na pág, v. 6, 2014.

KALSKE, Miika. **Transforming monolithic architecture towards microservice architecture.** 2019.

KUMAR, Karthik; LIU, Jibang; LU, Yung-Hsiang; BHARGAVA, Bharat. **A survey of computation offloading for mobile systems.** Mobile Networks and Applications, Springer, v. 18, n. 1, p. 129–140, 2013.

KUMAR, Karthik; LU, Yung-Hsiang. **Cloud computing for mobile users:** Can offloading computation save energy? Computer, IEEE, v. 43, n. 4, p. 51–56, 2010.

LINDVALL, Mikael; TESORIERO, Roseanne; COSTA, Patricia. **Avoiding architectural degeneration:** An evaluation process for software architecture. In: IEEE. Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on. [S. l.], 2002. p. 77–86.

LUZ, Welder; AGILAR, Everton; OLIVEIRA, Marcos; MELO, César; PINTO, Gustavo; BONIFÁCIO, Rodrigo. **An experience report on the adoption of microservices in three brazilian government institutions.** In: Proceedings of the XXXII Brazilian Symposium on Software Engineering. [S. l.: s. n.], 2018. p. 32–41.

MAIA, Marcio; FONTELES, Andre; NETO, Benedito; GADELHA, Romulo; VIANA, Windson; ANDRADE, Rossana. **Loccam-loosely coupled context acquisition middleware.** In: ACM. Proceedings of the 28th Annual ACM Symposium on Applied Computing. [S. l.], 2013. p. 534–541. MATEVELI, G. V.; MACHADO, N. G.; MORO, M. M.; JR, C. A. D. Taxonomia e desafios de recomendação para coleta de dados geográficos por cidadaos. 2015.

MEHTA, Alok; HEINEMAN, George. **Evolving legacy system features into fine-grained components.** In: Managing Corporate Information Systems Evolution and Maintenance. [S. l.]: IGI Global, 2005. p. 108–137.

MENDES, Laura; DONEDA, Danilo. **Reflexões iniciais sobre a nova lei geral de proteção de dados.** Revista de Direito do Consumidor, 2020.

MIRRI, Silvia; PRANDI, Catia; SALOMONI, Paola; CALLEGATI, Franco; MELIS, Andrea; PRANDINI, Marco. **A service-oriented approach to crowdsensing for accessible smart mobility scenarios.** Mobile Information Systems, Hindawi, v. 2016, 2016.

NAVARRO, Natália; COSTA, Cristiano; BARBOSA, Jorge; RIGHI, Rodrigo. **A context-aware spontaneous mobile social network.** In: 2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom). [S. l.: s. n.], 2015. p. 85–92.

NETFLIX. **Netflix Open Source Software Center.** 2018. <https://netflix.github.io/>. (Acesso em: 02/01/2021).

OHM, Paul. **Broken promises of privacy:** Responding to the surprising failure of anonymization. UCLA l. Rev., HeinOnline, v. 57, p. 1701, 2009.

OTHMAN, Mazliza; MADANI, Sajjad; KHAN, Samee et al. **A survey of mobile cloud computing application models.** IEEE communications surveys & tutorials, ieee, v. 16, n. 1, p. 393–413, 2013.

O'HERRIN, Jacquelyn; FOST, Norman; KUDSK, Kenneth. **Health insurance portability accountability act (hipaa) regulations:** effect on medical record research. Annals of surgery, Lippincott, Williams, and Wilkins, v. 239, n. 6, p. 772, 2004.

RICHARDSON, Chris. **Pattern:** Decompose by subdomain. 2018. <https://microservices.io/patterns/decomposition/decompose-by-subdomain.html>. (Acesso em: 11/01/2021).

SAMPAIO, Hildegarda; MATOS, Lúcia. **Dirceu:** a potência da composição performativa e suas micropolíticas. DAPesquisa, v. 14, n. 24, p. 037–051, 2019.

SANAEI, Zohreh; ABOLFAZLI, Saeid; GANI, Abdullah; BUYYA, Abdullah. **Heterogeneity in mobile cloud computing:** Taxonomy and open challenges. IEEE Communications Surveys Tutorials, v. 16, n. 1, p. 369–392, First 2014. ISSN 1553-877X.

SANTOS, Gabriel; REGO, Paulo; GOMES, Francisco; TRINTA, Fernando. **Caos d2d:** Uma solução para offloading de métodos entre dispositivos móveis. In: SBC. Anais Estendidos do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web. [S. l.], 2017. p. 164–168.

SANTOS, Gabriel; REGO, Paulo; GOMES, Francisco; TRINTA, Fernando. **Uma proposta de solução para offloading de métodos entre dispositivos móveis.** In: SBC. Anais Estendidos do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web. [S. l.], 2017. p. 76–81.

SANT'ANNA, Cláudio; FIGUEIREDO, Eduardo; GARCIA, Alessandro; LUCENA, Costa. **On the modularity assessment of software architectures:** Do my architectural concerns count. In: Proc. International Workshop on Aspects in Architecture Descriptions (AARCH. 07), AOSD. [S. l.: s. n.], 2007. v. 7.

SATYANARAYANAN, Mahadev; BAHL, Paramvir; CACERES, Ram; DAVIES, Nigel. **The case for vm-based cloudlets in mobile computing.** Pervasive Computing, IEEE, v. 8, n. 4, p. 14–23, Oct 2009. ISSN 1536-1268.

SATYANARAYANAN, Mahadev; BAHL, Paramvir; CACERES, Ram; DAVIES, Nigel. **The case for vm-based cloudlets in mobile computing.** IEEE pervasive Computing, IEEE, n. 4, p. 14–23, 2009.

SHIRAZ, Muhammad; GANI,Abdullah; KHOKHAR, Rashid; BUYYA, Rajkumar. **A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing.** Communications Surveys Tutorials, IEEE, v. 15, n. 3, p. 1294–1313, Third 2013. ISSN 1553-877X.

SILVA, Cícero; JÚNIOR, Aquino; MELO, Sávio. **A blockchain-based approach for privacy control of patient's medical records in the fog layer.** In: Proceedings of the 25th Brazillian Symposium on Multimedia and the Web. [S. l.: s. n.], 2019. p. 133–136.

SILVA, Hebert de Oliveira and others. **Uma abordagem baseada em anonimização para privacidade de dados em plataformas analíticas.** [sn], 2019.

SILVA, Hebert; BASSO, Tania; MORAES, Regina. **Privacy and data mining:** evaluating the impact of data anonymization on classification algorithms. In: IEEE. 2017 13th European Dependable Computing Conference (EDCC). [S. l.], 2017. p. 111–116.

SILVA, Wermeson. **Estudo comparativo do impacto da segurança em ambientes de mobile cloud computing.** 2019.

SOMMER, Lukas; KOCH, Andreas. **Openmp device offloading for embedded heterogeneous platforms-work-in-progress.** In: IEEE. 2020 International Conference on Embedded Software (EMSOFT). [S. l.], 2020. p. 4–6.

SOMMERVILLE, Ian. **Engenharia de software.** PEARSON BRASIL, 2011. ISBN 9788579361081. Disponível em: <https://books.google.com.br/books?id=H4u5ygAACAAJ>.

SOUSA, Gustavo. **Software product lines for multi-cloud microservices configuration.** In: Journées Cloud GdR RSD. [S. l.: s. n.], 2016.

SPRING. **Spring Cloud.** 2018. <http://projects.spring.io/spring-cloud/>. (Acesso em: 02/01/2021).

SUN, Yuqiong; NANDA, Susanta; JAEGER, Trent. **Security-as-a-service for microservices-based cloud applications.** In: IEEE. Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on. [S. l.], 2015. p. 50–57.

TAIBI, Davide; LENARDUZZI, Valentina; PAHL, Claus. **Processes, motivations, and issues for migrating to microservices architectures:** An empirical investigation. IEEE Cloud Computing, IEEE, v. 4, n. 5, p. 22–32, 2017.

THÖNES, Johannes. **Microservices.** IEEE software, IEEE, v. 32, n. 1, p. 116–116, 2015.

VILLAMIZAR, Mario; GARCÉS, Oscar; CASTRO, Harold; VERANO, Mauricio; SALAMANCA, Lorena; CASALLAS, Rubby; GIL, Santiago. **Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud.** In: IEEE. 2015 10th Computing Colombian Conference (10CCC). [S. l.], 2015. p. 583–590.

XIA, Feng; DING, Fangwei; LI, Jie; KONG, Xiangjie; YANG, Laurence; MA, Jianhua. **Phone2cloud:** Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. Information Systems Frontiers, Springer, v. 16, n. 1, p. 95–111, 2014.

YI, Shanhe; LI, Cheng; LI, Qun. **A survey of fog computing:** concepts, applications and issues. In: Proceedings of the 2015 workshop on mobile big data. [S. l.: s. n.], 2015. p. 37–42.

YURUR, Ozgu; LIU, Chi; SHENG, Zhengguo; LEUNG, Victor; MORENO, Wilfrido; LEUNG, Kin. **Context-awareness for mobile sensing:** A survey and future directions. Communications Surveys Tutorials, IEEE, PP, n. 99, p. 1–1, 2014. ISSN 1553-877X.

ZIMMERMANN, Olaf. **Microservices tenets.** Computer Science-Research and Development, Springer, v. 32, n. 3-4, p. 301–310, 2017.

APÊNDICE A – *SCRIPTS* DA EXECUÇÃO DOS SERVIÇOS

Foi necessário realizar algumas configurações no ambiente onde serão executados os microsserviços. Foi desenvolvido uma imagem *docker* e para implementação do ambiente foram criados alguns *scripts* com comandos do *kubernetes* para inicialização dos serviços.

A seguir são apresentados os *scripts* desenvolvidos para iniciar a execução dos serviços. São eles: *start*, *destroy* e *stop*. Na listagem 8 é apresentado o *script* de *start* que realiza três ações: (a) cria um serviço de descoberta de *cloudlet*, (b) cria o serviço de banco de dados e (c) inicializa os serviços de *offloading* de dados e processamento.

Listagem 8 – Trecho de *script* do serviço de *start*

```
1  !/bin/sh
2
3  Script to add/deploy all project resources.
4
5  echo "\nStart CAOS ..."
6  sleep 3
7  echo "\nStart Discovery Cloudlet ..."
8  sleep 3
9  docker run --rm -d --network host --name discovery andersonalmada/discovery-cloudlet
10 echo "\nStart Databases ..."
11 sleep 3
12 kubectl apply -f databases.yaml
13 echo "\nStart Data and Processing Offloading Microservices ..."
14 sleep 3
15 kubectl apply -f caos.yaml
```

Na listagem 9 é apresentado o *script* de *stop* que deve ser utilizado para pausar todos os serviços que estão em execução. Os serviços não são destruídos e as informações não são apagadas do banco de dados, é realizada apenas uma pausa na execução dos serviços.

Listagem 9 – Trecho de *script* do *microservive* de Stop

```
1  !/bin/sh
2
3   Script to stop all services
4
5  echo "\nStop CAOS ..."
6  sleep 3
7  echo "\nStop Data and Processing Offloading Microservices ..."
8  sleep 3
9  kubectl delete -f caos.yaml
```

```
10 echo "\nStop Discovery Cloudlet ..."
11 sleep 3
12 docker stop $(docker ps -q --filter ancestor=andersonalmada/discovery-cloudlet )
```

Na listagem 10 é apresentado o código que deve ser utilizado pelo desenvolvedor para destruir todos serviços que estão em execução, incluindo o banco de dados, e nesse caso será perdido todas as informações que estão armazenadas.

Listagem 10 – Trecho de *script* do serviço de *destroy*

```
1  !/bin/sh
2
3  Script to remove/undeploy all project resources.
4
5  echo "\nDestroy CAOS ..."
6  sleep 3
7  echo "\nDestroy Discovery Cloudlet ..."
8  sleep 3
9  docker stop $(docker ps -q --filter ancestor=andersonalmada/discovery-cloudlet )
10 echo "\nDestroy Data and Processing Offloading Microservices ..."
11 sleep 3
12 kubectl delete -f caos.yaml
13 echo "\nDestroy Databases ..."
14 sleep 3
15 kubectl delete -f databases.yaml
```