Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

MFOG: Uma Arquitetura em Névoa para a Infraestrutura de Aplicações do CCOp Mv

Marcos Francisco da Silva

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Orientadora
Prof.a Dr.a Aletéia Patrícia Favacho de Araújo

Brasília
2022

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

da Silva, Marcos Francisco
dM321m MFOG: Uma Arquitetura em Névoa para a Infraestrutura de Aplicações do CCOp Mv / Marcos Francisco da Silva; orientador Aletéia Patrícia Favacho de Araújo. -- Brasília, 2022.
73 p.

Dissertação(Mestrado Profissional em Computação Aplicada) - Universidade de Brasília, 2022.

1. Computação em Névoa. 2. Resiliência em aplicações. 3. Orquestração. 4. Modelos de Migração. 5. MFog. I. Favacho de Araújo, Aletéia Patrícia, orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

MFOG: Uma Arquitetura em Névoa para a Infraestrutura de Aplicações do CCOp Mv

Marcos Francisco da Silva

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof.a Dr.a Aletéia Patrícia Favacho de Araújo (Orientadora)
CIC/UnB

Prof. Dr. Marcelo Antonio Marotta
CIC/UnB

Prof. Dr. Alexandre da Costa Sena
UERJ

Prof. Dr. Marcelo Ladeira
Coordenador do Programa de Pós-graduação em Computação Aplicada

Brasília, 20 de outubro de 2022

Dedicatória

Dedico esse trabalho à minha esposa Rafaela Firmino Silva por suportar a espera pelo marido ausente e me dar apoio em todos os momentos da minha vida. Muito obrigado!

Agradecimentos

Agradeço a minha orientadora pela paciência e pelos conselhos objetivos e esclarecedores. Agradeço também aos professores Marcos F Caetano, Marcelo A Marotta, Lucas Bondan, Geraldo P Rocha Filho pelas inúmeras orientações e dicas ao longo do caminho do curso de mestrado. Por fim, agradeço ao Exército Brasileiro que, através do programa CCOp Mv e da parceira com a UNB, possibilitou a realização deste trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Em Operações de Resposta a Desastres é necessária uma grande coordenação de vários agentes e instituições. O uso de software tático e operacional é de grande importância para o êxito da operação. No entanto, as operações podem ocorrer em locais adversos com recursos escassos de acesso à Internet. Essa limitação pode gerar problemas no fornecimento de serviços, uma vez que tais aplicações utilizam processamento remoto, o qual é tipicamente realizado em *datacenter* distantes. Neste contexto, esta dissertação propõe uma arquitetura, denominada *MFog*, que explora os conceitos de computação em névoa com foco na resiliência das aplicações. Essa arquitetura pode ser utilizada em lugares onde, apesar de existirem estruturas com recursos computacionais próximos ao usuário final, o acesso à Internet é limitado (similar ao que ocorre nessas operações). Para isso, são definidos componentes mínimos da arquitetura e mecanismos de orquestração com base em modelos de migração de aplicações em contêiner. Os resultados obtidos, a partir de um estudo de caso, indicam que a arquitetura proposta consegue ser resiliente aos problemas enfrentados nos cenários dessas operações, garantindo a disponibilidade das aplicações demandadas em uma Operação de Reposta a Desastres.

Palavras-chave: Computação em Névoa, Resiliência, Orquestração, Modelos de Migração, MFog.

Abstract

In Disaster Response Operations, a great deal of coordination from various agents and institutions is necessary. The use of tactical and operational software is of great importance for the success of the operation. However, operations can take place in adverse locations with scarce Internet access resources. This limitation can generate problems in the provision of services, since such applications use remote processing, which is typically performed in distant datacenters. In this context, this dissertation proposes an architecture, called *MFog*, that explores the concepts of fog computing with a focus on application resilience. This architecture can be used in places where, although there are structures with computational resources close to the end user, access to the Internet is limited (similar to what happens in these operations). To this end, minimal architectural components and orchestration mechanisms are defined based on containerized application migration models. The results obtained, from a case study, indicate that the proposed architecture can be resilient to the problems faced in the scenarios of these operations, guaranteeing the availability of the applications demanded in a Disaster Response Operation.

Keywords: Fog Computing, Resilience, Container Migration, MFog.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contextualização e Problema

Operações de Resposta a Desastres são deflagradas para atender ocorrências de furacões, enchentes e outros desastres naturais. Para coordenar essas operações, de modo geral, é necessário uso de softwares táticos e operacionais que auxiliam as equipes na tomada de decisões. No entanto, em muitas situações, a região de ocorrência do desastre não possui infraestrutura de telecomunicações adequada. Assim sendo, geralmente, pode existir grande dificuldade no estabelecimento de comunicação e, por consequência, na atuação de forma coordenada pelos agentes que compõem os Centros de Operações. Esses Centros possuem a função de integrar o comando de diversas agências que irão interagir para fornecer soluções para os problemas que surgem durante a operação [1].

Assim, é importante ressaltar que esses os agentes que integram estes Centros e os que estão espalhados na operação utilizam softwares de apoio a operação. No entanto, esses softwares apresentam arquitetura de comunicação do tipo cliente-servidor e exploram os conceitos de computação em nuvem para operar. Destaca-se que a computação em nuvem (*cloud computing*) é um modelo para permitir acesso de rede onipresente e sob demanda a um conjunto de recursos de computação [2]. Neste contexto, o software está instalado em um local distante da operação, sendo o acesso possibilitado por meio de *link* com a Internet. Esse acesso é feito por aplicações do tipo *browser* ou por algum aplicativo instalado nos equipamentos dos agentes, gerando uma grande dependência com a capacidade do *link* disponível no local da catástrofe.

Uma possível solução para a perda de acesso aos serviços em locais com limitação de acesso à nuvem é a transferência de parte do processamento da aplicação para próximo do usuário final. Essa estratégia fornece maior resiliência durante os períodos de indisponibilidade de conexão. Esse conceito é uma das bases da computação em névoa (*fog computing*), que transfere parte da capacidade de processamento e de armazenamento

para dispositivos próximos do usuário final [3]. Essa transferência pode ser para a borda da rede ou no caminho entre o cliente e a nuvem, ou seja, em qualquer equipamento utilizado para fazer a interligação entre o dispositivo do usuário final e a nuvem.

Essa transferência pode ocorrer com a utilização de aplicações estruturadas por meio de componentes que estão em contêineres, nos quais, cada componente fornece um determinado conjunto de funcionalidades para a aplicação como um todo. Dessa forma, componentes que exigem mais processamento ficarão na nuvem, e os demais poderão ser instalados na camada de névoa. No entanto, para o usuário final, a aplicação fornecerá os serviços de forma transparente, mesmo com vários módulos do sistema distribuídos por diversos equipamentos. Nesse contexto, cada dispositivo, ou um conjunto de dispositivos interligados, são considerados nós da névoa.

Um exemplo dessa infraestrutura, alocada próxima ao usuário final, pode ser visto no projeto do Exército Brasileiro (EB), denominado Centro de Coordenação de Operações Móvel (CCOp Mv)[1]. Essa infraestrutura possibilita a coordenação da operação de resposta a desastres, próximo ao local de um evento, fornecendo aos agentes acesso a diversas aplicações e softwares, por meio de viaturas (caminhões e caminhonetes) que possuem computadores, servidores, equipamentos de rede e telecomunicações. Nesse projeto, as viaturas atuam como ponto de acesso dos clientes à Internet e a sistemas na nuvem que podem ser utilizadas como nós de uma arquitetura em névoa.

Para fazer uso dessa infraestrutura, diferentes arquiteturas em névoa foram propostas com uso de técnicas de migração de componentes de aplicação. Na maior parte desses estudos [4, 5, 6, 7], a solução proposta possibilita a distribuição dos componentes da aplicação pelos nós da névoa e também pela nuvem. Assim, os recursos de hardware são utilizados de maneira mais eficiente, fornecendo serviços com melhor desempenho. Entretanto, nesses trabalhos, os nós que compõem a arquitetura em névoa não são responsáveis pelo acesso dos clientes à rede de dados. Dessa forma, em caso de desconexão, os clientes de um nó isolado perdem todo o acesso aos sistemas. Além disso, caso algum nó sofra limitação de recursos de hardware, essas arquiteturas não preveem uma realocação dinâmica das aplicações.

Diante do exposto, este trabalho propõe uma arquitetura para infraestrutura de aplicações em névoa, com suporte a migração de componentes de aplicação da nuvem para a camada de névoa. Para isso, a arquitetura proposta usa mecanismos de resiliência a problemas de conexão entre os nós da névoa e a nuvem, fornecendo aplicações para os clientes mesmo em caso de nós isolados. Além disso, a arquitetura prevê mecanismos de alocação dinâmica caso o sistema esteja na névoa e algum nó sofra limitação de recursos. Esse mecanismo trata os problemas em aberto apresentados nos trabalhos da literatura.

[1] `www.epex.eb.mil.br/index.php/proteger`

1.2 Objetivos

O objetivo principal deste trabalho é desenvolver uma arquitetura resiliente, para computação em névoa, a ser utilizada pelas aplicações em infraestruturas localizadas próximas ao usuário final, em operações de resposta a desastres. Essa implementação adotará mecanismos que garantam o fornecimento do serviço, mesmo em situações de perda de conexão entre os nós. Dessa forma, em um cenário no qual um nó fica isolado, os mecanismos existentes na implementação da arquitetura devem garantir que a infraestrutura continuará fornecendo o serviço para os clientes conectados no nó de acesso.

Para alcançar esse objetivo geral, esta pesquisa apresenta os seguintes objetivos específicos:

- Definir os componentes necessários para uma implementação de arquitetura resiliente para infraestrutura de computação em névoa;
- Desenvolver um mecanismo de orquestração para aplicações que funcionam dentro de contêineres;
- Implementar um mecanismo de resiliência contra problemas de conexão, problemas relacionados à baixa capacidade nos *links* e limitação nos recursos de hardware nos nós;
- Testar a arquitetura proposta em ambiente simulado a fim de avaliar a integração entre os componentes e os mecanismos propostos.

1.3 Estrutura deste Trabalho

O restante deste documento está organizado da seguinte maneira. No Capítulo 2 é apresentada a fundamentação teórica para este trabalho, visando oferecer a base necessária para a compreensão do assunto abordado e seus desafios. No Capítulo 3 são apresentados trabalhos relacionados, os quais desenvolveram implementações de arquiteturas para infraestrutura em névoa. Em seguida, no Capítulo 4 é apresentada a proposta de implementação de arquitetura em névoa, alvo deste projeto de pesquisa. No Capítulo 5 são apresentados e discutidos a metodologia para validação da implementação proposta e os resultados alcançados, destacando as principais contribuições obtidas. Por último, no Capítulo 6 estão apresentadas as considerações finais e os trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta conceitos que serão utilizados neste trabalho. Inicialmente é apresentado o Centro de Coordenação de Operação Móvel, que pode ser tomado como exemplo de infraestrutura de computação próxima ao usuário final. Posteriormente, são apresentadas algumas limitações da computação em nuvem e uma visão geral sobre computação em névoa e seus benefícios para utilização quando se possui recursos de Tecnologia de Informação e Comunicação (TIC) descentralizados e em locais remotos. Além disso, são apresentados pontos importantes sobre resiliência e orquestração em aplicações. Por último, são apresentados modelos arquiteturais de névoa existentes na literatura.

2.1 Centro de Coordenação de Operações Móvel

O Centro de Coordenação de Operações Móvel (CCOp Mv) é um projeto do Exército Brasileiro que integra o Programa Proteger[1]. Ele é composto por um conjunto de viaturas com equipamentos de comunicação que fornecem serviços de TIC para o comando e elementos da tropa, possibilitando a gestão da consciência situacional no teatro de operações onde está inserido [8]. O CCOP Mv é composto por 9 veículos com recursos de TIC, um dos veículos pode ser considerado o nó central, possuindo mais recursos computacionais em relação aos demais, e os outros são denominados nós de acesso.

A Figura 2.1 apresenta uma visão geral do projeto CCOp Mv, a viatura centralizada é o Modulo Gerenciador de Comunicações (MGC), a qual é a viatura com maior capacidade de recursos de TIC no cenário. As viaturas de 1 a 7 representam os nós de comunicação, as quais fornecem para os clientes (tropas espalhadas no terreno) acesso a recursos de TIC por meio de *links* sem fio e 4G/LTE.

Analisando a Figura 2.1, é possível observar que o MGC (nó central) se conecta a EBNET, a qual é a rede privada do Exército, fornecendo aplicações que estão em *datacenter*

[1] `http://www.epex.eb.mil.br/index.php/proteger`

distantes. Além disso, existem vários cenários possíveis para o emprego do CCOp Mv. No primeiro cenário os nós se conectam ao nó central (MGC) por meio de outros nós (caso do nó 1). Em um segundo cenário os nós se conectam diretamente ao nó central (caso do nó 6). Em um terceiro cenário os nós podem se conectar ao nó central com utilização de *links* sem garantia de qualidade (caso do nó 7). Por fim, o cenário mais extremo é representado pelo nó 5, o qual está isolado e se conecta diretamente à EBNET sem uso do MGC.

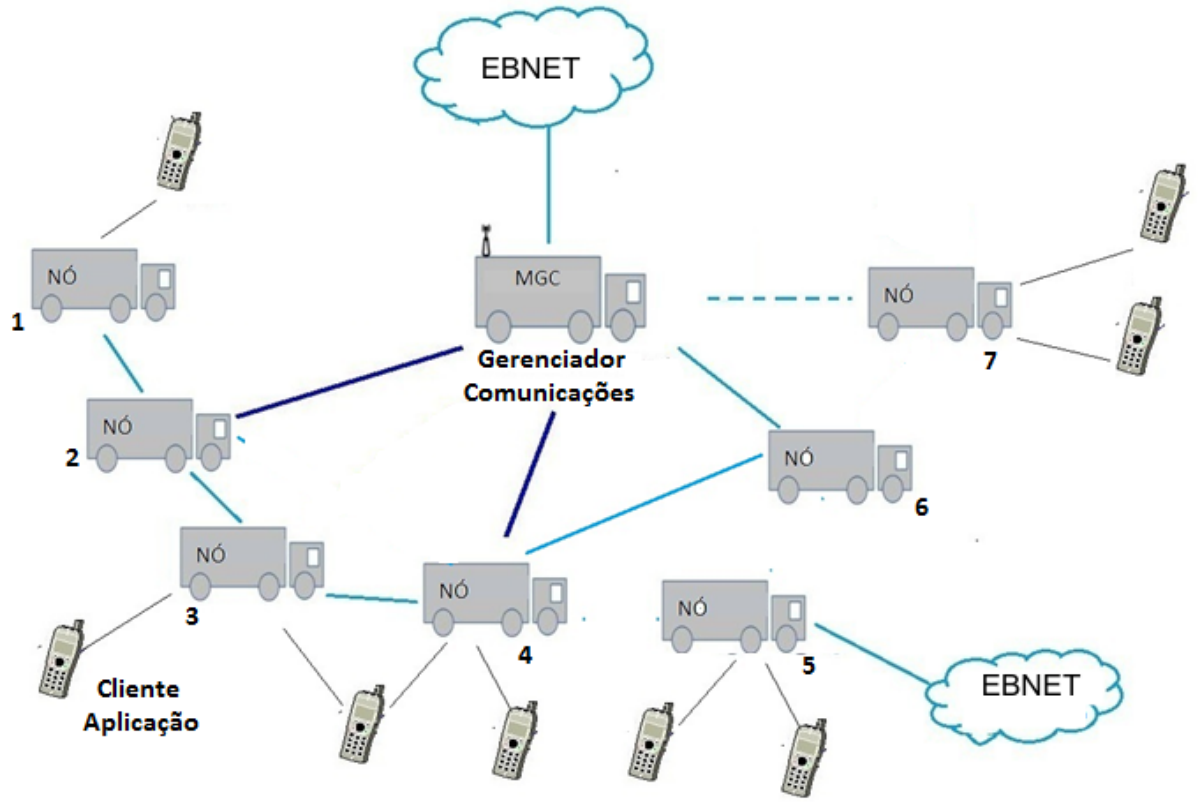

Figura 2.1: Visão Geral do Projeto CCOp Mv (Fonte: Elaboração Própria)

No cenário atual, os agentes acessam as aplicações de suporte às operações por meio de equipamentos como *tablets*, celulares e *notebooks*. Os softwares estão localizados na nuvem, que no caso do Exército Brasileiro são *datacenter* espalhados por algumas cidades no Brasil. Esse projeto tem grande importância nas operações militares porque as Forças Armadas atuam em diversas situações críticas.

Dentro de suas atribuições o Exército atua em diversas operações, em destaque estão as operações de garantia da lei e da ordem, garantia de votação e apuração nas eleições em locais remotos, socorro às vítimas durante a ocorrência de catástrofes naturais, etc. A Figura 2.2 mostra militares atuando em área isolada após ocorrência de enchente[2].

[2] `www.eb.mil.br/web/noticias/noticiario-do-exercito`

Figura 2.2: Exemplo de emprego de tropas em ambiente de catástrofe (Fonte: www.eb.mil.br)

Nesses cenários, a infraestrutura de TIC é, por vezes, precária. Assim, o CCOp Mv fornece a possibilidade dos agentes utilizarem recursos computacionais, contribuindo de sobremaneira com o sucesso das operações, inclusive possibilitando o resgate de vidas, pois fornece aos militares uma gama de informações de forma célere e coordenada. A Figura 2.3 apresenta uma visão resumida das viaturas que compõem o CCOp Mv, na qual a viatura exemplificada é uma viatura isolada (nó de acesso) conectada à nuvem por meio de *link* satelital.

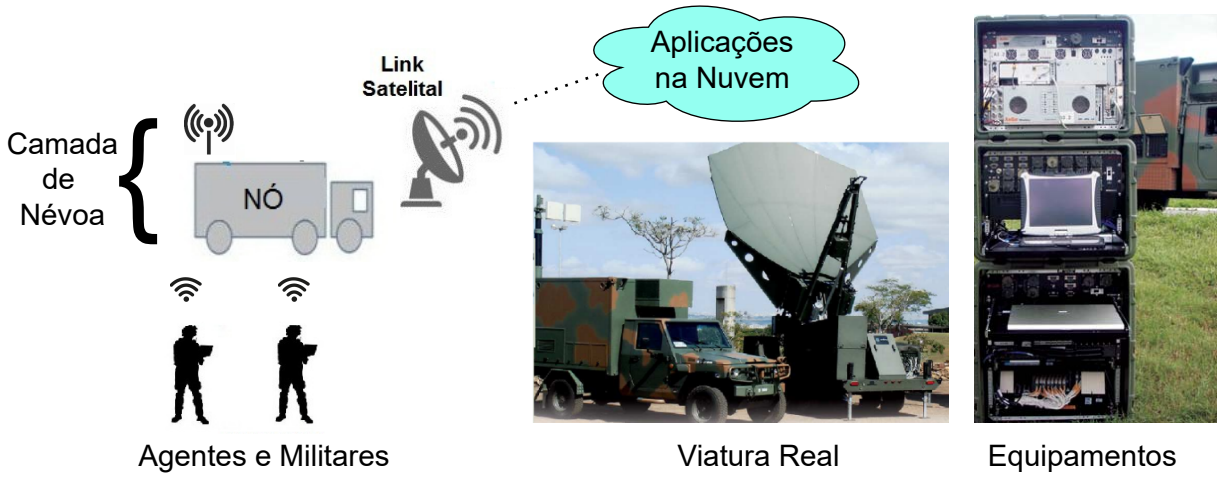

Figura 2.3: Componentes do CCOp Mv (Fonte: Elaboração Própria)

Assim, quando as operações ocorrem em locais sem infraestrutura de comunicação com a *Internet*, o CCOp Mv usa comunicação satelital. Atualmente, as Forças Armadas possuem a sua disposição alguns tipos de terminais satelitais para essa tarefa. Esses equipamentos podem ser dos seguintes tipos: portáteis (TP), leves (TL), transportáveis

Tabela 2.1: Capacidade dos equipamentos (Fonte: Adaptado de [9])

Equipamento	**Capacidade (Mbps)**
Terminal Rebocável	10
Terminal Transportável	2
Terminal Leve	1
Móvel Naval	1
Terminal Portátil	0,128

(TT), rebocáveis(TR) e móveis navais (MN) [9]. A Figura 2.4 apresenta exemplos desses equipamentos e a Tabela 2.1 apresenta a capacidade de transmissão de cada equipamento.

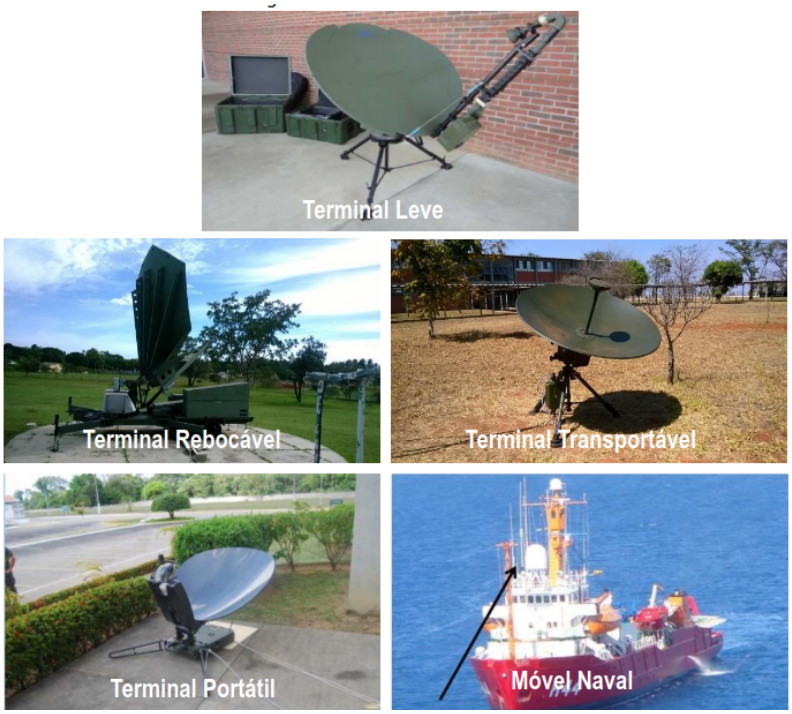

Figura 2.4: Terminais para *link* satelital em utilização pelas Forças Armadas (Fonte: Adaptado de [9])

Com base nos dados da Tabela 2.1 é possível verificar que a capacidade dos terminais satelitais é limitada quando comparada a capacidade de *links* de uso comercial. Essa limitação pode causar problemas quando os agentes, que utilizam aplicações na nuvem, fazem o acesso através dos equipamentos satelitais ligados ao CCOp Mv.

Importante destacar que essa centralização das aplicações em *datacenter* e a dependência da conexão com a Internet são pontos fracos do modelo de computação em nuvem.

2.2 Computação em Nuvem

Na computação em nuvem, os recursos de TI são fornecidos como um serviço, permitindo aos usuários acessarem os serviços sob demanda e independente de localização, aumentando a quantidade de serviços disponíveis [10].

Segundo Mell *et al.* [11], a computação em nuvem pode ser descrita como um modelo para habilitar o acesso por rede ubíquo, conveniente e sob demanda a um conjunto compartilhado de recursos de computação, por exemplo: redes, servidores, armazenamento, aplicações e softwares. Esses recursos podem ser rapidamente provisionados e liberados com o mínimo de esforço de gerenciamento, ou interação com o provedor de serviços. Atualmente esse paradigma está consolidado, pois os serviços são ofertados por grandes empresas que usam *datacenter* espalhados pelo mundo.

Esse modelo revolucionou a forma das pessoas usarem a Internet ao permitir um acesso dinâmico aos recursos computacionais[11]. Nesse paradigma a principal vantagem é a sua acessibilidade e conveniência, o que significa que os usuários podem acessar os aplicativos e dados em qualquer computador ou dispositivos móveis através da Internet. Outras vantagens são o maior espaço de armazenamento e a capacidade de compartilhar arquivos com outras pessoas a qualquer hora e em qualquer lugar. Por fim, outro ponto importante na utilização da nuvem é o fato dos usuários não precisarem mais instalar algum software que consome recursos em seus equipamentos [12].

No entanto, existem alguns óbices no uso da computação em nuvem, Habibi *et al.* [3] elencam as principais limitações desse modelo, sendo elas:

- Falta de segurança e privacidade, os dados trafegam por diversos dispositivos até chegarem à nuvem;
- Flexibilidade limitada, essa tecnologia tem grande dependência de acesso à Internet;
- Centralização de recursos, pois todo o processamento ocorre na nuvem;
- *Downtime*, existem muitos dispositivos no caminho até a nuvem, que podem apresentar lentidão ou falhas.

Para resolver algumas dessas limitações surgiu um novo paradigma de computação, a chamada computação em névoa (*fog computing*) [3], que utiliza recursos computacionais mais próximos do usuário final, conforme será descrito na seção a seguir.

2.3 Computação em Névoa

A computação em névoa ou *fog computing* é um novo paradigma de computação que apresenta solução para algumas limitações apresentadas na computação em nuvem [3].

Segundo Ren *et al.* [13], nesse paradigma a capacidade de processamento e de armazenamento são alocados em dispositivos mais próximos do usuário final, na borda da rede ou no caminho entre o cliente e a nuvem. Conforme mostrado na Figura 2.5, uma camada de névoa é fisicamente colocada entre a nuvem e os dispositivos do usuário final (camada de borda), para permitir computação e armazenamento. A camada de névoa consiste em um grande número de servidores heterogêneos, que variam desde dispositivos dedicados, como servidores e roteadores de borda até dispositivos temporários como telefones inteligentes, sensores de última geração e veículos[13]. Uma arquitetura comum para computação em névoa é apresentada por Hu *et al.* [14], na qual são listadas três camadas:

- Camada de nuvem: inclui computação forte e servidores de armazenamento com a capacidade de armazenamento de uma enorme quantidade de dados e análise de computação extensiva para vários serviços de aplicativos, se necessário;
- Camada de névoa: situada na borda da rede. É uma combinação de um grande número de nós de névoa (físicos ou virtuais). Esses nós de névoa são amplamente distribuídos entre a nuvem e os dispositivos finais. Eles podem ser móveis ou fixos em um só lugar. Os nós de névoa têm a capacidade de processar, transmitir e armazenar temporariamente os dados recebidos. Além disso, os nós de névoa têm a possibilidade de conexão e interação com nós de névoa vizinhos;
- Camada de dispositivo do usuário (borda): é a mais próxima do ambiente físico e do usuário final, incluindo dispositivos e sensores IoT.

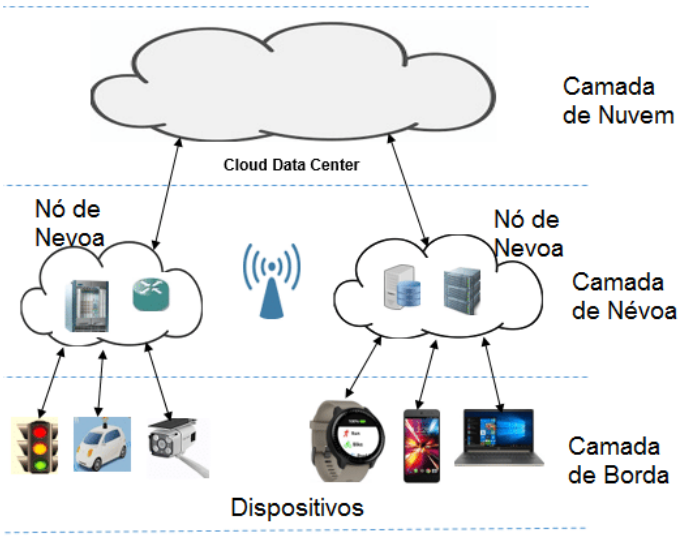

Figura 2.5: Arquitetura básica para computação em névoa (Fonte: Adaptado de miro.medium.com)

Os recursos computacionais a serem utilizados para computação em névoa são alocados em entidades denominadas nós. Mahmud *et al.* [15] descrevem cinco tipos de equipamentos que podem atuar como nós da névoa: servidores, dispositivos de comunicação, *cloudlets*, estações base e veículos. Nesse trabalho, os autores conceituam nós de névoa como nós computacionais com arquitetura e configurações heterogêneas que conseguem fornecer infraestrutura para computação na borda de rede. Nesse contexto, um ambiente de computação em névoa é composto de componentes de rede tradicionais, tais como roteadores, *switches*, servidores *proxy*, estações base, etc [16].

Naha *et al.* [17] apresentam uma taxonomia que classifica esses dispositivos em três categorias: dispositivos IoT, dispositivos de processamento e dispositivos de *gateway*. Bachiega *et al.* [16] estabelecem uma relação entre a classificação proposta por [17] e as três camadas da arquitetura de computação em névoa apresentada na Figura 2.5. Considerando essa relação, os dispositivos IoT pertencem à camada de borda, os dispositivos de processamento estão na camada de nuvem e os dispositivos do tipo *gateway* são alocados na camada de névoa [16].

A maneira mais comum de fornecer recursos por meio dos nós de névoa é utilizando a virtualização, uma vez que fornece a capacidade de dividir recursos de uma máquina física (CPU, memória, rede, armazenamento) e realocá-los em uma ou mais máquinas virtuais, que podem ser usadas em um número infinito de tarefas[16]. Mais recentemente, o método de virtualização migrou para o uso de contêineres, pois oferecem um mecanismo de isolamento mais aderente a um ambiente de computação em névoa. Os contêineres executam virtualização na camada do sistema operacional (virtualização leve) e não mais na camada de hardware, como ocorre com as máquinas virtuais. Uma grande vantagem de contêineres é que eles podem executar a plataforma de forma independente, devido ao compartilhamento do *kernel* hospedeiro, favorecendo, por exemplo, a migração entre servidores [16]. A Figura 2.6 apresenta a comparação entre a utilização de recursos diretamente sobre o hardware, a utilização de nós com máquinas virtuais e a utilização de contêineres. É possível observar na figura que a utilização de contêineres possui um número menor de camadas em relação à virtualização com máquinas virtuais.

A computação em névoa pode ser abordada sob várias perspectivas. No trabalho apresentado por Habibi *et al.* [3], são listadas as propriedades que, do ponto de vista dos autores, são comuns nas diversas perspectivas para computação em névoa e podem ajudar na compreensão e diferenciação entre este modelo de computação e outros. Essas propriedades estão apresentadas a seguir:

- A computação em névoa tem como base um cenário no qual um grande número de dispositivos heterogêneos ubíquos e descentralizados executam funções ou aplicativos de rede de forma colaborativa;

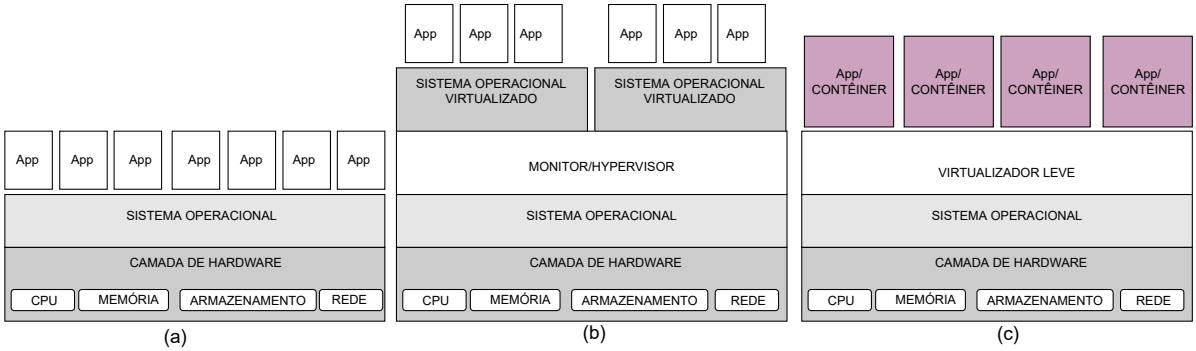

Figura 2.6: Comparação entre tecnologias de virtualização (Fonte: Adaptado de [16])

- O termo névoa não se restringe a uma determinada área de tecnologia. Ela suporta heterogeneidade de dispositivo e de interfaces;
- Geralmente, consiste em uma plataforma virtualizada de suporte a funções básicas de rede ou novos serviços e aplicativos, que são executados em um ambiente temporário;
- Oferece suporte a recursos como gerência de mobilidade, comunicação programável e descoberta de localização.

Assim sendo, nota-se que a computação em névoa resolve alguns problemas apresentados pela computação em nuvem, algumas das vantagens na utilização desse paradigma estão apresentadas na próxima seção.

2.3.1 Benefícios da Computação em Névoa

Esta seção apresenta alguns benefícios existentes no uso da computação em névoa, do ponto de vista de sua aplicação em infraestrutura distribuídas próximas ao usuário. Além disso, para exemplificar, são apresentadas algumas aplicações e tecnologias que podem ser empregadas em infraestruturas militares, como no caso do CCOp Mv.

Latência

Um dos principais benefícios fornecidos pela utilização da computação em névoa é a diminuição da latência, pois o processamento pode ocorrer mais próximo do usuário final. Dessa forma, os clientes e os dispositivos recebem respostas mais rápidas.

Isso pode ser útil para aplicações que possuem alta dependência em relação ao tempo de resposta, como controle de veículos, aplicações de realidade aumentada e telemedicina. Dentro do emprego militar, diversos são os sistemas com dependência do tempo de

resposta, como sensores de radares antiaéreos, rastreamento de viaturas blindadas para defesa e ataque, entre outros.

Diminuição no Tráfego

Na computação em névoa o processamento da informação ocorre próximo ao usuário final, não há a necessidade de todos os dados serem enviados para a nuvem. Isso diminui a dependência da Internet e de *links* com alta qualidade. Além disso, com o advento da tecnologia 5G uma grande quantidade de dados terá de ser processada, caso todo esse processamento seja realizado na nuvem, será necessária uma alta capacidade de banda de dados, o que pode inviabilizar seu uso.

Assim, aplicações militares como processamento de imagem e vídeos capturados por câmeras instaladas nas viaturas, ou no próprio uniforme do militar, que buscam por criminosos ou suspeitos ou para vigilância, podem fazer uso desse benefício. Além disso, em operações militares é muito comum não haver recursos disponíveis em abundância, com isso o uso racional de banda de dados para Internet, por exemplo, é de vital importância para o sucesso da operação.

Consciência de Localização, Segurança e Privacidade

Devido à distribuição geográfica e o acesso local nos equipamentos pertencentes a computação em névoa, é possível determinar a localização do usuário de forma mais precisa. Essa informação é útil para decisões em sistemas que podem alocar recursos segundo a posição de seus clientes ou dispositivos.

Em aplicações militares, um exemplo disso é o uso de cartas topográficas para navegação e planejamento de operações. Isso ocorre porque conhecendo a localização é possível alocar para o nó específico, apenas o conteúdo digital referente a região onde está localizado o nó.

Na arquitetura de névoa existem mecanismos de tomada de decisão sobre quais dados são enviados para a nuvem, de forma que seja possível aumentar a segurança com dados, realizando o seu tráfego apenas em determinadas áreas. Dessa forma, os dados sigilosos não serão expostos desnecessariamente.

2.4 Resiliência em Aplicações

Segundo Sterbenz *et al.* [18], resiliência é definida como a capacidade da rede de fornecer e manter um nível aceitável de serviço em face a várias falhas e desafios para a operação

normal. Prokhorenko *et al.* [19] apresentam três classes para a resiliência com base nos tipos de falhas:

- Orientado à capacidade: cujo objetivo é suportar o uso legítimo do sistema, no qual a capacidade refere-se à grande quantidade de dados sendo processados e ao grande número de usuários atendidos;
- Orientado à confiabilidade: a qual visa a prevenção contra usos indesejados ou não autorizados do sistema; e
- Orientado à eficiência: que visa prevenir problemas devido a restrições e limitações físicas, como *links* de conexões instáveis, baterias de baixa capacidade e outros problemas de conexão.

Sterbenz *et al.* [18] avaliaram que um mecanismo que trata a resiliência deve possuir, no mínimo, duas camadas em sua arquitetura: uma camada lógica, que é responsável pelo funcionamento normal do sistema; e uma camada, que os autores denominaram meta-camada, que é responsável pelo monitoramento e recuperação das falhas. Para o caso de infraestruturas de computação que se encontram em locais remotos, os problemas relacionados à conexão são os principais riscos em relação ao funcionamento normal das aplicações.

Portanto, em ambientes onde existem problemas relativos à estabilidade de conexão entre nós, nos quais existe a possibilidade de um nó ficar isolado dos demais, a resiliência a ser alcançada é aquela orientada à eficiência. Segundo esses autores, esse tipo de resiliência pode ser obtido com técnicas de migração e orquestração de componentes da aplicação.

2.5 Orquestração de Aplicações

Para o funcionamento efetivo de um sistema distribuído é necessária a utilização de orquestradores, pois são eles que determinam onde alocar os componentes e recursos necessários para o funcionamento de um sistema. Por meio de monitoramento contínuo, os orquestradores recebem informações sobre o ecossistema geral da arquitetura, e podem migrar os componentes da aplicação para atender a uma determinada regra pré-determinada.

A decisão de onde alocar os recursos é realizada com uso de algoritmos definidos para os orquestradores, os quais possuem estratégias que podem usar o hardware de maneira mais eficiente, considerando, por exemplo, a economia de energia, ou a diluição da latência nos serviços, dentre outras características [20].

Mijuskovic *et al.* [20] fazem uma revisão sistemática dentre os diversos algoritmos relacionados às tecnologias de névoa. Os autores criaram uma classificação para os vários algoritmos existentes, a seguir é apresentada esta classificação:

- *Off-Loading*: determina onde os dados devem ser armazenados para reduzir as despesas de transmissão, e o atraso entre a camada de computação em nuvem e os dispositivos IoT;
- Balanceamento de Carga: distribui a carga de trabalho nos recursos disponíveis, para tornar as operações mais eficientes, evitando congestionamento, carga baixa e sobrecarga;
- Colocação: distribui as tarefas de computação de entrada para os recursos de névoa/borda apropriados, ou seja, busca colocar os componentes em locais adequados a sua necessidade computacional;
- *QoS*: esses algoritmos focam em políticas de Qualidade de Serviço (QoS);
- Gerenciamento de Energia: tem foco em economizar a energia a ser consumida na infraestrutura.

Portanto, dependendo de qual for o objetivo desejado no projeto do sistema, pode-se usar técnicas mais adequadas, para cada necessidade. Por exemplo, é possível usar técnicas de balanceamento de carga para distribuir o processamento de forma mais eficiente entre o hardware, ou usar algoritmos que focam na economia de energia para hardware móvel.

Capítulo 3

Arquitetura de Referência para Névoa

Este capítulo apresenta arquiteturas propostas para serem utilizadas em névoa e seus principais componentes. Além disso, aborda alguns trabalhos que possuem relação direta com o tema deste projeto e mostram o estado atual das pesquisas relacionadas.

3.1 Arquiteturas de Referência

Segundo Jaakkola *et al.* [21], uma arquitetura é um modelo conceitual que define a estrutura, o comportamento e as formas de se visualizar um sistema. A descrição da arquitetura é sua representação formal com a definição dos componentes, suas funções, propriedades internas e externas, e os relacionamentos entre eles.

Uma das principais iniciativas com objetivo de padronizar a utilização de recursos na computação em névoa é o *OpenFog Consortium*[1], composto por entidades como ARM, Microsoft Corp., Intel, Cisco, Dell e Universidade de Princeton. O *OpenFog Consortium* define a computação em névoa como "uma arquitetura horizontal ao nível de sistema que usa funções de computação, armazenamento, controle e rede mais perto dos usuários ao longo de um *continuum* nuvem-coisa" [22]. O OpenFog identificou alguns "pilares" para distinguir a computação em névoa da computação em nuvem, a saber [22]:

- Baixa latência, com implantações e cálculos próximos às fontes de dados (ou seja, dispositivos IoT e borda);
- Evitar custos de migração (ou seja, evitar gastos com transmissões desnecessárias para a nuvem);
- Comunicações locais ao invés de comunicações com nós remotos.

[1] `opcfoundation.org`

Segundo Antonini *et al.* [22], a arquitetura proposta *OpenFog* possui uma pilha de três camadas (borda, névoa e nuvem) onde a camada de névoa, composta pelos nós de névoa, é dividida em quatro subcamadas principais. No nó da névoa a subcamada inferior é a plataforma sobre o hardware físico, a próxima subcamada é responsável pelo gerenciamento geral de nós e comunicações entre terminais (por exemplo, sistemas remotos em nuvem, dispositivos de borda, outros nós de névoa).

Essa subcamada é transversal as demais e garante a transparência na utilização dos recursos físicos pelas camadas lógicas. A Camada de Suporte do aplicativo é uma coleção de serviços utilizados por aplicativos instanciados nos nós. Esses módulos incluem bancos de dados, gerenciadores de armazenamento, pilhas de rede, módulos de segurança, etc. O último módulo é a camada de aplicações e serviços, onde um dos itens principais é a interface com o usuário final. A figura 3.1 apresenta uma visão geral dessa proposta.

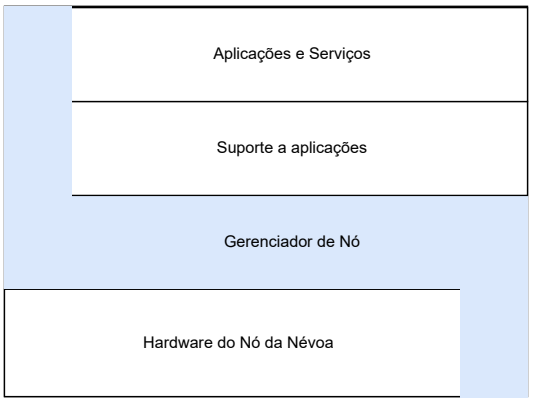

Figura 3.1: Visão geral em camadas proposta *Open Fog* (Fonte: Adaptado de [22])

Uma pesquisa abrangente sobre diversos modelos arquiteturais para infraestrutura de computação em névoa foi realizada por Habibi *et al.* [3], na qual os autores mostram que existem alguns modelos de referência desenvolvidos para computação em névoa a partir de várias perspectivas.

3.1.1 Clouds Lab

Essa arquitetura de referência foi proposta por Dastjerdi *et al.* [23], e ela é formada por cinco camadas: (i) Acesso, (ii) Rede, (iii) Serviços e Recursos em Nuvem, (iv) Gerenciamento de Recursos Definidos por Software e (v) Aplicativos IoT.

Na camada mais baixa, estão os dispositivos finais com seus aplicativos. A camada de rede oferece meios para a conexão entre as camadas. A camada de serviços e recursos oferece uma plataforma para gerenciamento de recursos e aplicativos. A camada de gerenciamento de recursos orquestra os recursos por toda a arquitetura sendo baseada em

software. Por fim, a camada superior contém os aplicativos IoT que usam a computação em névoa para fornecer serviços para os usuários finais [3].

Portanto, nesta arquitetura, cada camada não faz referência ao hardware disponível e sim a uma visão abstrata do problema. Por exemplo, na Camada de Aplicativos, é possível encontrar os softwares disponíveis para serem consumidos pelos clientes. No entanto, os componentes desses softwares podem estar distribuídos ao longo dos nós da estrutura.

3.1.2 Cisco-Bonomi

Esta arquitetura de referência foi proposta por Bonomi *et al.* [24], e consiste em cinco componentes principais: (i) a Camada de Recursos Físicos Heterogêneos, (ii) a Camada de Abstração de Névoa, (iii) a Camada de Orquestração de Serviço de Névoa, (iv) Serviços de IoT e (v) Barramento de Mensagem Distribuída [3].

Neste modelo a Camada de Recursos Físicos contém todos os hardwares envolvidos na névoa incluindo servidores, roteadores, pontos de acesso, sensores, etc. A Camada de Abstração tem a responsabilidade de ocultar a natureza heterogênea dos dispositivos, e fornecer uma interface programável a fim de permitir o gerenciamento dos recursos. A Camada de Orquestração de Serviço foi projetada para gerenciar o ambiente distribuído e fazer a orquestração dos serviços. O Barramento de Mensagens é responsável por transportar mensagem entre a orquestração e o gerenciamento, já os Serviços de IoT contém as aplicações disponibilizadas para os clientes. Assim sendo, similar a arquitetura *Cloud Labs*, o modelo Cisco-Bonomi apresenta componentes abstratos, no qual a hierarquia das camadas não representa a diferença na complexidade de computação, e sim níveis de complexidade na abstração dos recursos. Na camada mais baixa está o hardware e nas camadas superiores estão as aplicações, sendo que nas camadas intermediárias estão os mecanismos de orquestração e gerenciamento dos recursos.

3.1.3 AUT

Habibi *et al.* [25] propuseram uma arquitetura de referência para computação em névoa (AUT), baseada em padrões de função de rede virtualizada (FRV) e redes definidas por software (SDN). Nesse modelo existem cinco componentes principais [3], os quais são: (i) camada de infraestrutura, (ii) camada de abstração de recursos, (iii) controle e gerenciamento, (iv) aplicativos e serviços e (v) camada de orquestração.

Nessa arquitetura os equipamentos estão na camada de infraestrutura, a heterogeneidade dos equipamentos é abstraída através da camada de abstração que fornece APIs de alto nível para que as camadas subsequentes possam utilizar os recursos.

Essa utilização é feita por meio de controle e gerenciamento de componentes de rede e controladores de redes definidas por software (SDN), e pela orquestração dos serviços, na qual, dentro da gama de serviços, estão as funções de rede virtualizadas.

Os modelos, *Clouds Lab*, *Cisco-Bonomi* e *AUT*, possuem uma característica em comum que é a abstração por níveis, nos quais os níveis mais baixos estão próximos ao hardware, e os níveis mais altos são as aplicações. É importante observar que nessas arquiteturas, os hardwares que compõem cada nó da névoa são gerenciados por uma camada de abstração. Para isso, os componentes de orquestração nas camadas superiores alocam os serviços sobre o hardware de maneira transparente, ou seja, os nós não são tratados como elementos da arquitetura de forma individualizada, eles são absorvidos pela camada de abstração, que assume a responsabilidade de fornecer os recursos para as camadas superiores. A Figura 3.2 apresenta a estrutura das três arquiteturas.

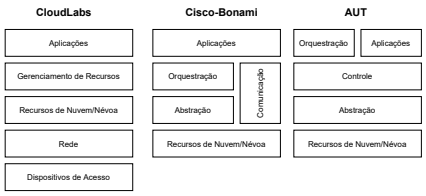

Figura 3.2: Visão geral em camadas das arquiteturas Cloudlabs, Cisco-Bonami e AUT (Fonte: Elaboração Própria)

3.1.4 SORTS

Velasquez *et al.* [26] propuseram uma abordagem híbrida para orquestração de serviço que é chamada de SORTS. Esse modelo consiste em três camadas, na qual a camada mais baixa compreende os dispositivos dos usuários, a intermediária corresponde aos nós da névoa e a mais alta corresponde a nuvem. Nesse modelo todos os nós recebem os mesmos elementos do orquestrador, permitindo a criação e o funcionamento de mecanismos de coreografia distribuídos. No entanto, é explicitada a existência de um nó que irá gerenciar os demais de forma centralizada. Portanto, na arquitetura SORTS, os nós são considerados de forma individual, mas a orquestração ocorre de forma centralizada. A Figura 3.3 apresenta uma visão geral dessa arquitetura.

3.1.5 SOAFI

SOAFI, proposto por Brito *et al.* [27], aproveita o TOSCA (a qual é uma linguagem padrão para descrever uma topologia de serviços da *web* baseados em nuvem[2]) e padrões

[2]`https://www.oasis-open.org/committees/tosca/faq.php`

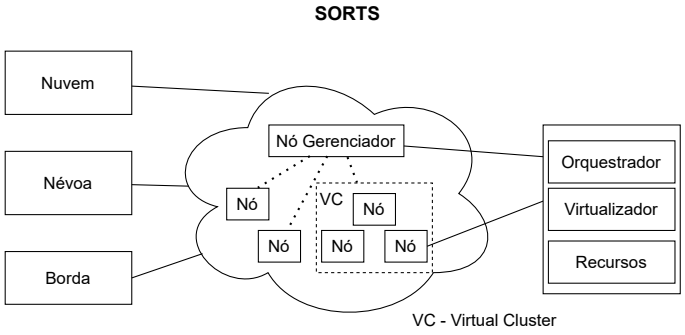

Figura 3.3: Visão geral das arquiteturas SORTS (Fonte: Elaboração Própria)

arquiteturais de funções de rede virtualizadas (NFV) para construir uma arquitetura de referência para a computação em névoa. Essa proposta apresenta um modelo federado, no qual cada nó possui autonomia para gerenciar seus recursos. A coordenação do sistema é feita por uma entidade denominada *Fog Orchestrator* (FO), que pode estar alocada em qualquer nó do sistema. Nesse modelo não é apresentada uma estrutura de camadas hierárquicas, cada nó avalia sua quantidade de recursos disponíveis e informa ao FO para tomar as ações necessárias. Nessa arquitetura os nós são considerados de forma individualizada e o sistema é coordenado de forma federada. A Figura 3.4 apresenta uma visão geral desse modelo.

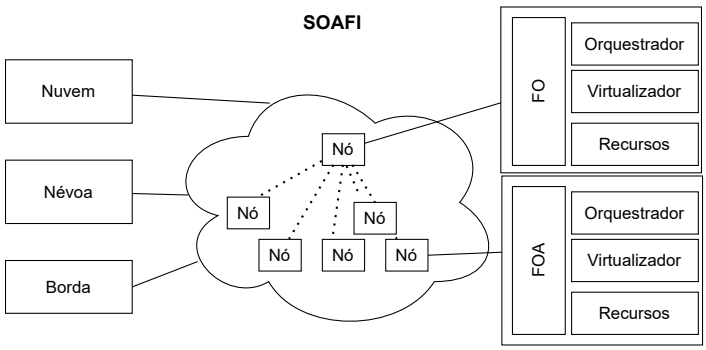

Figura 3.4: Visão geral das arquiteturas SOAFI (Fonte: Elaboração Própria)

Diante do exposto é possível verificar que os modelos apresentados se dividem em dois grupos. No primeiro modelo, a infraestrutura de computação em névoa, fornecida pelos nós, é vista como uma camada a ser abstraída, e as camadas superiores fazem uso desse recurso de forma transparente. No segundo grupo, cada nó da névoa é visto como uma entidade que possuirá os mesmos elementos dos demais nós, e a orquestração do sistema é realizada por um agente central, que também é um nó.

3.2 Recursos Existentes em Arquiteturas para Névoa

Ainda não existe um consenso sobre a melhor estrutura de arquitetura a ser utilizada por computação em névoa. Costa *et al.* [28] apresentaram uma revisão sistemática, na qual os autores analisaram diversos modelos propostos sob múltiplas óticas, em especial, avaliaram as funcionalidades disponibilizadas nos modelos arquiteturais, essas funcionalidades são:

- Controle de Acesso: essa funcionalidade pode verificar as credenciais de um solicitante antes de atendê-lo, sendo uma etapa adicional para preservar a privacidade e verificar se o solicitante possui autorização adequada para executar o serviço solicitado;
- Gerenciamento de Serviços e gerenciamento de recurso: um serviço pode ser uma aplicação monolítica, um módulo, ou outro tipo de instância que forneça saídas de interesse para um usuário ou para outras aplicações. O gerenciamento de serviços tenta garantir que esses serviços tenham os recursos necessários para seu funcionamento, ao se deparar com alguma restrição, que possa afetar o funcionamento adequado desses elementos. Além disso, ele deve solicitar ao gerenciador de recursos que tome ações para manter o QoS, dentre essas ações é possível citar: (i) melhoria no recurso de hardware disponibilizado para o serviço, (ii) transferência do serviço (migração) para um nó mais rico em recursos ou para a nuvem,(iii) aceitar e fornecer o serviço com qualidade abaixo do esperado;
- Monitoramento de Recursos: os recursos devem ser monitorados com frequência para garantir informações atualizadas sobre a condição da infraestrutura. Além de verificar a disponibilidade, o monitoramento também trata das medidas necessárias para manter SLA e QoS;
- Busca por Otimização: a otimização em um orquestrador de névoa implementa algoritmos e técnicas para minimizar algumas métricas (por exemplo, latência, carga de rede) e/ou maximizar outras (por exemplo, disponibilidade de recursos), melhorando a qualidade de experiência, qualidade de serviço e contribuindo com o SLA;
- Gerenciamento de Comunicação: controla e possibilita as comunicações entre os nós da névoa; e
- Segurança: funcionalidade focada em segurança da informação, como uso de criptografia e *Virutal Private Network* (VPN) .

Assim, por meio dessa revisão, Costa *et al.* [28] mostraram que alguns desafios ainda não são bem abordados na literatura. Dentre esses é possível citar o controle de acesso, a

gerência de serviços, e a busca por otimização e segurança da informação. Diante disso, é importante destacar que a arquitetura, objeto deste trabalho, apresenta mecanismos que atuam no gerenciamento de serviços e no controle de acesso.

3.3 Trabalhos Relacionados

Com intuito de encontrar trabalhos que apresentaram implementação de arquiteturas para aplicações que fazem uso de infraestrutura em névoa, foi realizada uma pesquisa bibliográfica nas bases de dados *Web of Science* e *IEEE*. Os termos usados nesta pesquisa foram "*fog architecture and orchestration*" e "*fog architetural and orchestration*". Como critério de inclusão na busca foi definida a janela de tempo de trabalhos no período de 2018 a 2021. Essa pesquisa resultou inicialmente em cerca de 100 artigos. Após uma análise de títulos e *abstract*, resultaram 7 artigos, os quais apresentaram diversos pontos em comum com a proposta deste projeto de pesquisa, o resultado dessa análise está descrito a seguir.

Uma das implementações arquiteturais que usa componentes de aplicações (microsserviços) em névoa é apresentada por Taherizadeh *et al.* [7]. Nesse trabalho os autores apresentam uma estrutura capaz de alocar recursos computacionais de acordo com diversas métricas de hardware e de rede. O principal objetivo é oferecer suporte a aplicativos com uso intensivo de processamento ou dados executados na camada de dispositivo. Na implementação arquitetural proposta existem mecanismos que fazem a migração de componentes da aplicação da camada de borda para a névoa. Assim, dependendo de métricas previamente definidas, isso garante o uso eficiente de recursos no *cluster* formado pelos nós de névoa. Todavia, o trabalho não explora as situações nas quais o ponto de acesso à Internet é também um nó da camada de névoa. Assim, caso um nó desse tipo fique desconectado dos demais, os mecanismos da arquitetura irão migrar os componentes que estavam no nó isolado para os demais nós. Porém, os clientes do nó desconectado perderão acesso aos serviços.

Outra abordagem é apresentada no estudo de Rosário *et al.* [6], no qual é analisada uma implantação, para arquitetura em névoa, capaz de suportar a migração de aplicações de multimídia da nuvem para a névoa. O objetivo é garantir métricas relativas à Qualidade na Experiência (QoE). No entanto, nesse trabalho o controlador do orquestrador está alocado na nuvem, e não são avaliados mecanismos de resiliência quanto aos problemas de conexão (eficiência).

Santos *et al.* [5] propuseram um modelo denominado SRFog para uso em redes de última geração baseadas em névoa, com aplicação para sistemas de realidade virtual, no qual o tempo de resposta da aplicação tem grande importância. A implantação tem

como base o desenho arquitetural padrão do *kubernetes*[3], o qual é um produto *open source* utilizado para automatizar a implantação, o dimensionamento e o gerenciamento de aplicativos em contêiner. No SRFog existem dois tipos de nós: o nó mestre e o nó trabalhador, sendo que o nó mestre fica alocado na nuvem e o controle de alocação de recursos é responsabilidade desse nó. Em trabalho anterior, Santos *et al.* [4] propuseram modificações no algoritmo padrão do orquestrador *kubernetes*, com foco na redução da latência, diminuindo o tempo de resposta global da aplicação.

Outro trabalho utilizando *kubernetes* é apresentado por Nguyen *et al.* [29], no qual os autores desenvolveram uma estrutura denominada *ElasticFog*, que tem o objetivo de fornecer recursos de aplicativos de forma eficaz na névoa. Essa proposta considera a capacidade de tráfego entre os nós para a decisão de onde alocar os componentes da aplicação. Esses três trabalhos, [4] [5] [29], utilizam os elementos padrão do sistema de orquestração *kubernetes*. No entanto, nesse padrão existe uma grande dependência do nó mestre e os autores não avaliam mecanismos de resiliência quanto à perda de comunicação com esse nó.

Mouradian *et al.* [30] propuseram uma implantação para plataforma como serviço (PaaS), que engloba nuvem e névoa de forma híbrida, implementando funções de rede virtual (VNFs). A plataforma automatiza o provisionamento de aplicativos em componentes espalhados pela nuvem e névoa, de tal maneira que os serviços são alocados nos nós mais apropriados para seu funcionamento, e o usuário utiliza a aplicação de forma transparente. No entanto, os autores não avaliaram o problema de desconexão de pontos de acesso que também são nós da arquitetura apresentada.

Uma proposta de uma nova implantação arquitetural baseada em contêiner para orquestração de camada de névoa, com base em *kubernetes*, é apresentada por Bakhshi *et al.* [31]. Nessa proposta os mecanismos do orquestrador são alterados para fornecerem um melhor tratamento de erros, e um subsistema para armazenamento de arquivos permanente tolerante a falhas. Uma arquitetura que usa serviço de *blockchain* é proposta por Núñez-Gómez *et al.* [32]. Nesse trabalho os autores apresentam a HIDRA, uma arquitetura baseada em *blockchain* distribuída para ambientes de computação de névoa. A HIDRA usa a plataforma descentralizada para automatizar o controle, e o gerenciamento dos recursos e serviços dentro do *cluster*. Nesse caso, cada nó consegue fazer sua própria gerência e atualizar seu estado conforme os contratos previstos no serviço de *blockchain*. A arquitetura foi desenhada para ser tolerante a falhas, segura e auditável. No entanto, em ambas as arquiteturas não existem previsão de funcionamento em nós totalmente isolados.

Com base no trabalho de Costa *et al.* [28] foi realizada uma comparação entre os trabalhos analisados e a proposta apresentada. Essa análise está apresentada na Tabela

[3] `kubernetes.io`

Tabela 3.1: Comparação entre os trabalhos relacionados (Fonte: Elaboração Própria)

Trabalho	**Ano**	**Gerência de Serviço**	**Controle de Acesso**	**Resiliência**	**Migração**	**Suporte a nó isolado**
[7]	2018	Sim	N/A	Capacidade	Aplicação	N/A
[6]	2018	Sim	N/A	Capacidade	Aplicação	N/A
[29]	2020	N/A	Sim	Capacidade	Aplicação	N/A
[30]	2020	Sim	N/A	Capacidade	Aplicação	N/A
[5]	2021	N/A	Sim	Capacidade	Aplicação	N/A
[32]	2021	Sim	Sim	Todas	Aplicação	N/A
[31]	2021	Sim	Sim	Capacidade	Aplicação/Arquivos	N/A
Este Trabalho	2022	Sim	Sim	Capacidade/Eficiência	Aplicação/Arquivos	Sim

Legenda: N/A- não apresenta ou não prevê.

3.1. Nessa tabela a primeira coluna indica os trabalhos relacionados. A segunda coluna indica qual é o ano da publicação. A terceira coluna indica se a arquitetura possui funcionalidades de gerência de serviços. A quarta coluna indica se a arquitetura possui controle de acesso aos recursos. A quinta coluna traz informações sobre a classe de resiliência na qual o trabalho pode ser enquadrado, conforme a classificação apresentada na Seção 2.4. A sexta coluna indica qual é o tipo de objeto que está sendo migrado pelo orquestrador. Por fim, a última coluna indica se a arquitetura funciona em nó isolado (após uma desconexão) ou não.

Assim, diante dos trabalhos analisados, nota-se que a principal contribuição desta pesquisa é apresentar uma arquitetura resiliente que adote mecanismos que possam garantir o fornecimento do serviço, mesmo em situações de perda de conexão entre os nós. Dessa forma, mesmo no pior cenário, no qual um nó ficará isolado dos demais, a arquitetura proposta neste trabalho garantirá que a infraestrutura continuará fornecendo o serviço para os clientes conectados no nó de acesso. Esta arquitetura será descrita, em detalhes, no próximo capítulo.

Capítulo 4

Arquitetura Proposta: MFOG

Neste capítulo é apresentada a proposta deste trabalho, a qual é a implementação de uma arquitetura resiliente para a infraestrutura que receberá as aplicações (softwares) em nós de névoa com conexões instáveis. Essa arquitetura visa diminuir a dependência da conexão do conjunto de nós com a nuvem (trazendo as aplicações para mais próximos do usuário final), e também aumentar a resiliência em relação a problemas de conexão. Para isso, são apresentados os elementos essenciais para este tipo de implementação, com base em modelos de referência. Em seguida, é apresentada a arquitetura proposta, em detalhes, com todos seus componentes.

4.1 *MFOG* - Arquitetura Resiliente para Aplicações em Névoa

Neste trabalho é proposto uma arquitetura para infraestrutura de aplicações em névoa denominada *MFOG*, um acrônimo para a junção das palavras *mobilidade* e *fog*. Assim sendo, conforme apresentado na Figura 4.1, a arquitetura é composta por quatro camadas, as quais são:(i) Camada de Aplicação, (ii) Meta-camada, (iii) Camada de Recursos e (iv) Camada de Acesso. A Figura 4.1 mostra uma visão geral desta arquitetura.

A Camada de Aplicação contém componentes que possibilitam o funcionamento da aplicação, é nela que está contido o sistema de virtualização e os contêineres contendo os serviços que serão disponibilizados aos usuários finais. A Meta-camada contém componentes responsáveis pelos mecanismos de orquestração e resiliência, os componentes dessa camada fornecem recursos para a migração de aplicação, monitoramento do estado e dos recursos disponíveis nos nós, busca de serviços, além do próprio orquestrador da arquitetura. Essa camada é responsável pelo gerenciamento de toda a arquitetura. A Camada de Recursos contém os recursos físicos e o sistema operacional dos nós, ela representa a

abstração desses elementos, recursos de memória RAM, armazenamento, rede e etc. O sistema operacional contido nesta camada é o responsável pela gerência e disponibilização desse recurso para as camadas superiores. Por fim, a Camada de Acesso contém elementos de controle de acesso a arquitetura, além do controle das urls das aplicações.

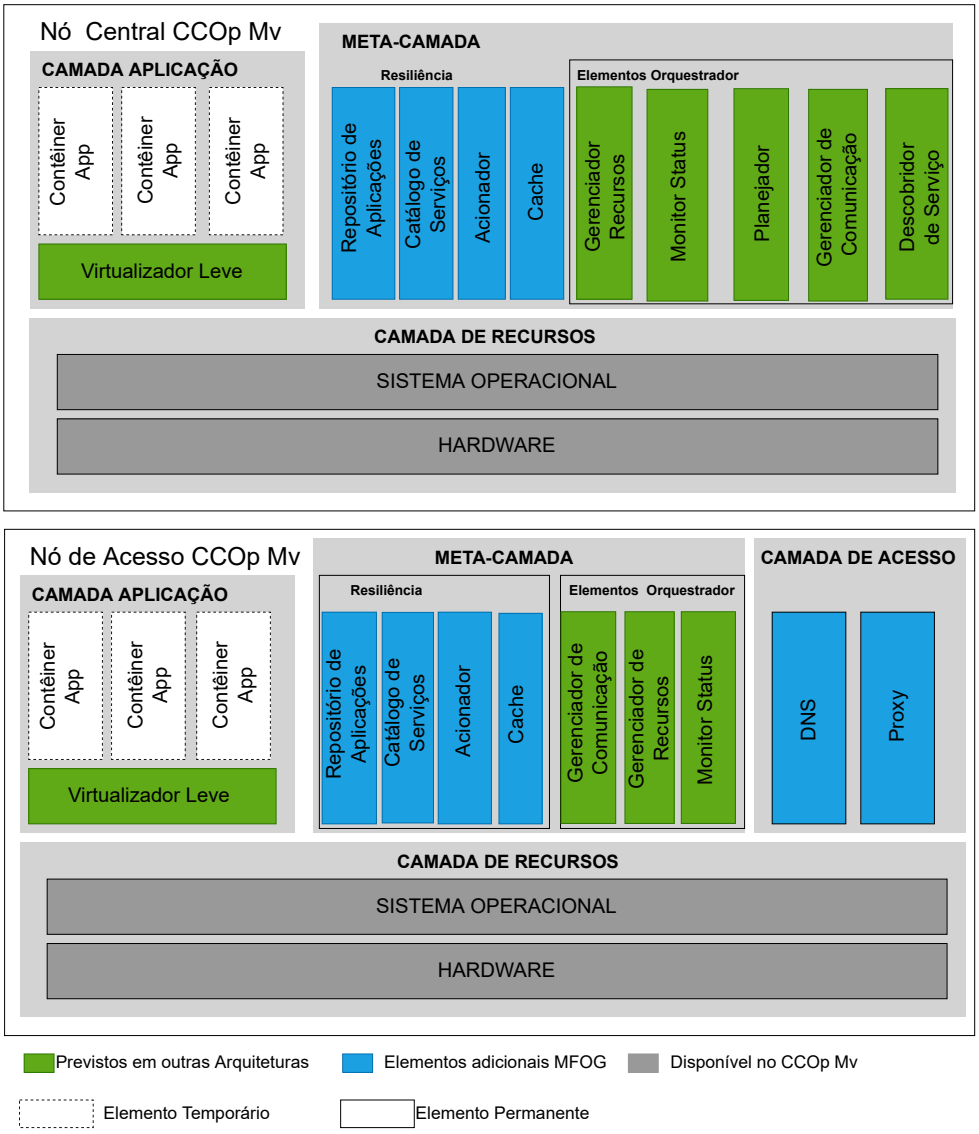

Figura 4.1: Modelo proposto para arquitetura *MFOG* (Fonte: Elaboração Própria)

4.2 Componentes do Modelo Proposto

Segundo Prokhorenko *et al.* [19], um mecanismo de resiliência para sistemas deve possuir, no mínimo, duas camadas (ou planos). A primeira é responsável pelo funcionamento normal da aplicação, e a segunda (Meta-camada) é responsável pelo monitoramento e recuperação de falhas.

As aplicações alvo deste trabalho terão seu funcionamento dentro de contêineres. Portanto, na camada responsável pelo funcionamento normal da aplicação deve existir um virtualizador que operacionalize tal tecnologia, de forma que os contêineres possam ser executados em todos os nós.

Na Meta-camada deve existir, no mínimo, um elemento responsável pelo monitoramento e outro responsável pela recuperação de falhas. A recuperação de falhas em relação a problemas de conexão pode ser alcançada por meio de técnicas de migração das aplicações [19]. Assim sendo, nessa camada também deve existir elementos responsáveis pela orquestração e migração de componentes da aplicação.

Além dessas duas camadas deve-se adicionar a camada relativa aos recursos de hardware e sistema operacional, denominada camada de recursos, pois é nessa camada que toda a estrutura será montada. Ademais, para nodos que fornecem acesso à rede de dados (nodos que são ponto de acesso), é necessário a existência de uma Camada de Acesso, a qual deve controlar a entrada dos usuários no sistema como um todo.

É importante salientar que outros modelos arquiteturais contemplam os elementos responsáveis pela orquestração e migração de aplicações. Todavia, este trabalho tem como foco a criação de mecanismos de recuperação de falhas com intuito de aumentar a resiliência do sistema como um todo. Com isso, foi proposta uma subcamada dentro da Meta-camada, denominada resiliência, com componentes necessários para o funcionamento desses mecanismos, sendo eles:

- Mecanismo de monitoramento para verificar situação da conexão com a nuvem e situação dos recursos disponíveis nos nós;
- Mecanismo de alocação dinâmica que faz a migração da nuvem para a névoa em caso de problemas de conexão (mecanismo de resiliência orientado à eficiência);
- Mecanismo de alocação dinâmica que faz a migração de um nó para outro com mais recursos disponíveis, em caso de problemas com recursos em um nó (mecanismo de resiliência orientado à capacidade).

Dessa forma, a implementação foi realizada sobre uma arquitetura composta por quatro camadas, as quais são: Camada de Aplicação, Meta-camada, Camada de Recursos e Camada de Acesso. Essas camadas serão apresentadas nas próximas seções.

4.2.1 Camada de Aplicação

A Camada de Aplicação é responsável pelo funcionamento normal da aplicação. Neste sentido, é nela que os componentes da aplicação serão alocados. Assim sendo, no caso em que se utilizam componentes que estão em contêineres, cada componente irá funcionar

sobre um virtualizador. Essa camada é instanciada de forma idêntica em todos os nós da camada de névoa, inclusive no nó central (pois o nó central também receberá componentes das aplicações). Para tal, esta camada será composta pelos seguintes elementos:

- Virtualizador Leve: responsável pelo funcionamento das aplicações e componentes dentro de contêineres. A virtualização leve, ou virtualização baseada em contêineres, fornece um nível diferente de abstração em termos de virtualização e isolamento de processo ao nível do sistema operacional. Assim, para cada instância nova não é criada uma pilha completa de sistema operacional dedicado. Contêineres executam sobre o mesmo *kernel* do sistema operacional hospedeiro em um servidor físico, sendo que um ou mais processos podem executar dentro de cada contêiner [33];
- Componentes de aplicação (Contêiner App): contêineres contendo as aplicações que fornecerão os serviços para os clientes, esses componentes são virtualizados de tal forma que funcionam sobre o virtualizador leve da camada.

A Figura 4.2 apresenta os elementos presentes na Camada de Aplicação, notar que no contêiner (1) existe uma aplicação java com sua JVM, seu arquivo binário e bibliotecas. No contêiner (2) existe uma aplicação em PHP com servidor de aplicação, arquivos PHP e bibliotecas. No contêiner (3) existe uma página html, servidor de aplicação e bibliotecas. Por fim, no contêiner (4) existe um banco de dados na tecnologia MySql. O virtualizador leve está representado na camada através da tecnologia *docker*.

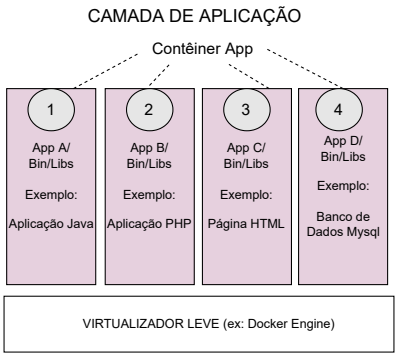

Figura 4.2: Representação de elementos da Camada de Aplicação (Fonte: Elaboração Própria)

4.2.2 Camada de Acesso

Os nós, além de fornecerem recursos computacionais, também podem ser ponto de acesso dos usuários aos sistemas. Assim, faz-se necessário a existência de uma camada que

gerencie e controle o acesso à infraestrutura. Dessa forma, é possível usar mecanismo de autenticação e autorização. Os componentes desta camada estão apresentados a seguir:

- *Proxy* de acesso: monitora e controla todos os acessos dos clientes. Assim sendo, além de prover métricas de tempo de resposta das aplicações solicitadas, essas métricas são utilizadas pelo monitor de recursos para tomar decisão de migração da nuvem para a névoa, ou entre os nós da névoa;
- DNS: fornece o endereço IP das aplicações. Esse endereço IP é alterado segundo a localização da aplicação de forma dinâmica, caso ocorra migração realizada pelos mecanismos de resiliência.

4.2.3 Meta-camada

A Meta-camada é responsável pela orquestração e pelo monitoramento do funcionamento das aplicações e de consumo de recursos. Além disso, é nessa camada que são implantados os mecanismos de resiliência. Ela é instanciada por completo no nó central (nó que será responsável por coordenar os demais nós da névoa e que possui o papel de gerenciador do sistema). Para o caso dos nós de acesso a Meta-camada é instanciada com alguns componentes. A descrição dos componentes desta camada está apresentada a seguir:

- Repositório com aplicações: repositório contendo as imagens, para contêineres, das aplicações e seus componentes que serão usadas pelo orquestrador para efetivar as alocações e distribuições, a Figura 4.3 apresenta um exemplo de um repositório instalado em sistema operacional ubuntu;
- Catálogo de Serviços: lista dos serviços que serão ofertados aos clientes. O catálogo contém os comandos necessários para realizar o processo de *deploy* dos componentes do serviço em ambiente de névoa. Além disso, o catálogo também deve conter as restrições em relação ao tempo de resposta limite, quantidade miníma de memória, CPU e armazenamento, e especificar a localização dos itens que necessitam de armazenamento permanente (para serem armazenados no cache);
- Acionador: baseado nas métricas fornecidas pelo Gerenciador de Recursos, ele pode acionar o Planejador para melhorar o desempenho do sistema, ou acionar os mecanismos de resiliência em caso de perda de conexão;
- *Cache*: repositório contendo arquivos que poderão ser usados pelas aplicações, caso ocorra perda de conexão com os demais nós;

- Monitor de Recursos: elemento responsável por coletar as métricas relativas à utilização de recursos no nó, além de verificar métricas relativas à rede (perda de conexão, lentidão, etc);
- Monitor de *Status*: mantém registro do estado das aplicações. Assim, caso alguma aplicação apresente falha no funcionamento, essa informação será utilizada pelo planejador para reestabelecer o funcionamento do sistema problemático;
- Planejador: elemento responsável por efetuar as alocações das migrações necessárias para o funcionamento eficiente e correto da aplicação;
- Gerenciador de Comunicação: responsável por estabelecer a comunicação entre os nós. Isso significa que é por meio dele que as informações são repassadas entre o nó central e cada nó gerenciado;
- Descoberta de Serviços: ponto de acesso dos serviços, ou seja, recebe a solicitação do serviço e encaminha a um determinado nó que possua uma instância daquele serviço.

Figura 4.3: Exemplo de repositório com aplicações, a seta indica uma imagem contendo banco de dados mysql na versão 8.0 (Fonte: Adaptado de www.mend.io)

A Figura 4.4 mostra os elementos da Meta-camada, na figura estão representadas as etapas do processo para instanciar uma aplicação na névoa, são eles:

- 1- O Monitor de Recursos coleta métricas indicando que ocorreu problemas na comunicação com a nuvem;
- 2- O Acionador, baseado nas métricas, decide por enviar comandos ao Planejador para instanciar a aplicação na névoa;
- 3- O Planejador, após decidir qual nó receberá a aplicação, envia comandos para o nó de acesso para instanciar a aplicação;
- 4 e 5- A comunicação entre os nós ocorre por meio do Gerenciador de Comunicação;
- 6- O Virtualizador Leve recebe os comandos e instancia a aplicação;
- 7- A Aplicação passa a funcionar sobre o Virtualizador Leve no nó de destino; e

- 8- Agentes (clientes) fazem requisições para a aplicação por meio do Descobridor de Serviço do Nó Central.

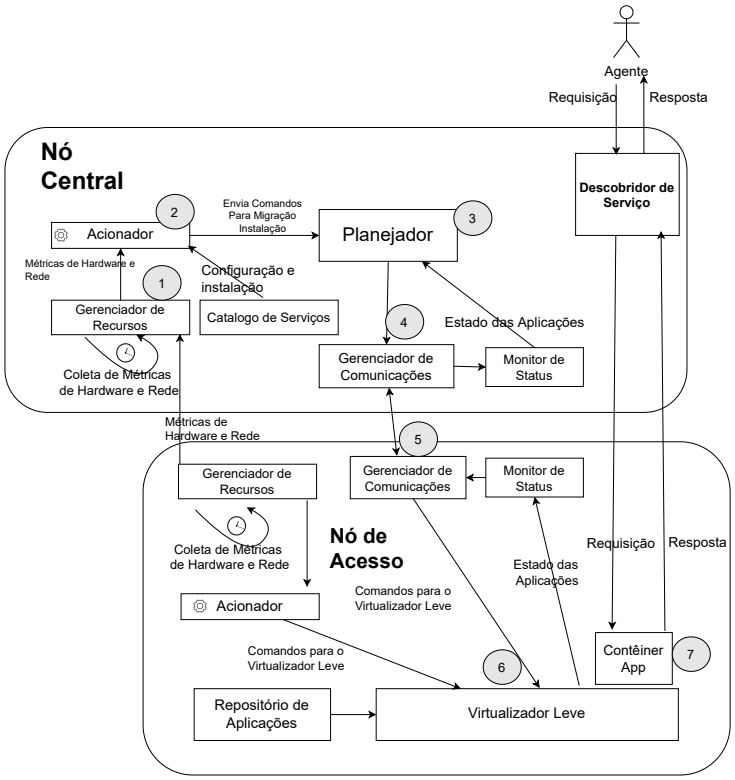

Figura 4.4: Representação de elementos da Meta-camada e o processo de instanciar aplicação (Fonte: Elaboração Própria)

Os usuários passam a acessar a aplicação através do nó central, esse acesso é feito através do *proxy* (pertencente a camada de acesso), importante ressaltar que nesse processo o Acionador muda a configuração de DNS de todos os nós para que a *url* da aplicação aponte para o nó central.

A Figura 4.5 mostra o processo para manter uma aplicação em funcionamento sobre o Virtualizador Leve:

- 1- O Virtualizador Leve fornece métrica sobre o estado da aplicação para o Monitor de Status;
- 2, 3 e 4- O Monitor de Status do nó de acesso envia informações para o Monitor de Status do nó central por meio do Gerenciador de Comunicações;
- 5- Caso ocorra algum problema na aplicação, tal como a aplicação parar de funcionar, o Planejador é acionado para instanciar novamente a aplicação no nó;
- 6 e 7- Planejador envia comandos para reestabelecer a aplicação;

- 8- Virtualizador Leve toma ações para instanciar a aplicação no nó novamente;
- 9- A Aplicação passa a funcionar sobre o Virtualizador Leve no nó de destino; e
- 10- Agentes (clientes) fazem requisições para a aplicação por meio do Descobridor de Serviço do nó central.

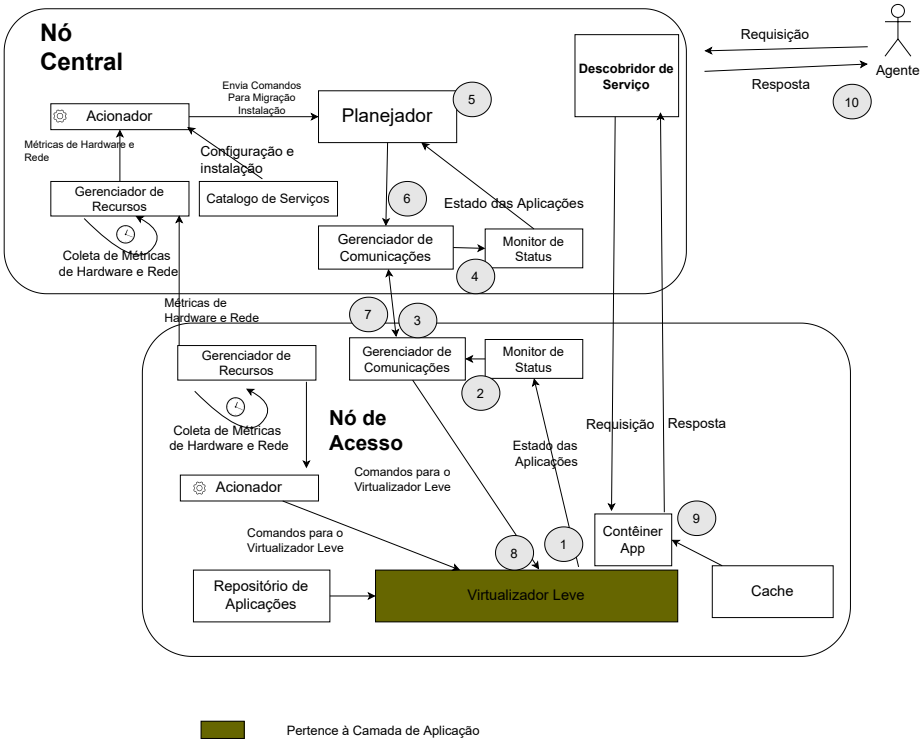

Figura 4.5: Representação de elementos da Meta-camada e o processo de manutenção de estado da aplicação (Fonte: Elaboração Própria)

A Figura 4.6 mostra o processo de resiliência contra problemas de limitação de recursos de hardware:

- 1- O Gerenciador de Recursos do nó de acesso fornece métricas de recursos de hardware para o gerenciador de recursos de nó central;
- 2- O Gerenciador de Recursos do nó central recebe métricas sobre recursos nos nós de acesso;
- 3 e 4- O Acionador avalia as métricas, e caso verifique alguma limitação de recursos no nó de acesso, envia comandos para o Planejador migrar aplicação para outro nó;
- 5, 6 e 7- Planejador envia comandos para o nó de destino e para o nó de origem para efetuar a migração;

- 8- Virtualizador Leve toma ações para instanciar a aplicação no nó de destino, e para remover a aplicação do nó de origem;
- 9- A Aplicação passa a funcionar sobre o Virtualizador Leve no nó de destino; e
- 10- Agentes (clientes) fazem requisições para a aplicação por meio do Descobridor de Serviço do nó central.

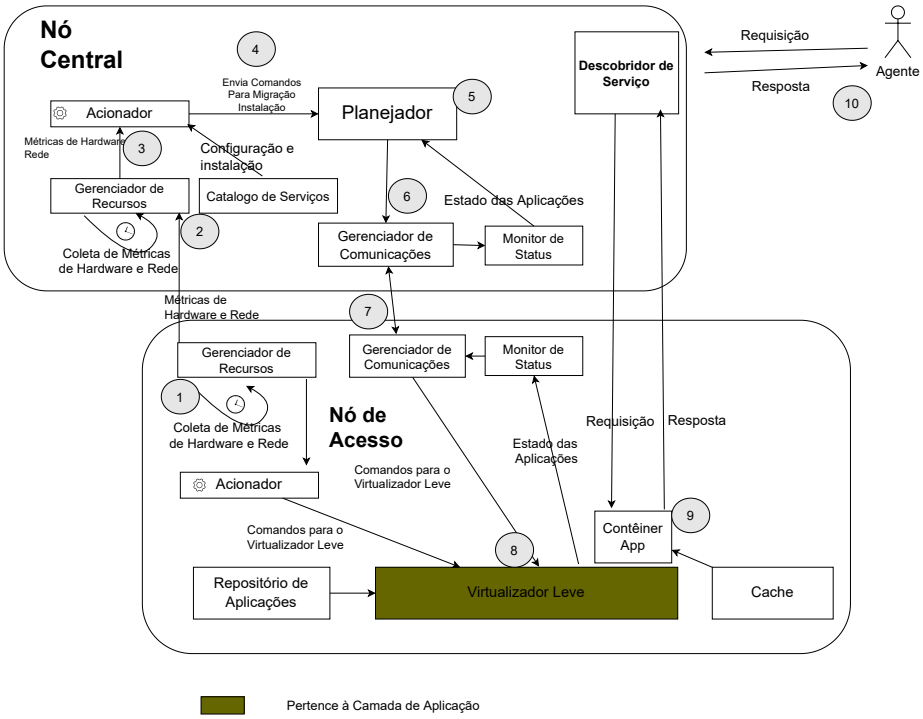

Figura 4.6: Representação de elementos da Meta-camada e o processo de resiliência contra problemas de limitação de recursos de hardware (Fonte: Elaboração Própria)

A Figura 4.7 mostra o processo de resiliência contra problemas de perda de conexão e nó isolado:

- 1- O Gerenciador de Recursos do nó de acesso fornece métricas para o Acionador do próprio nó de acesso;
- 2- O Acionador avalia as métricas, e caso verifique a perda de conexão, envia comandos para o Virtualizador Leve do próprio nó para instanciar nesse nó todos os componentes do nó central, e para migrar todos os componentes das aplicações para o nó de acesso desconectado;
- 3- O Virtualizador Leve executa os comandos recebidos do Acionador e instancia todos os componentes pertencentes ao nó central;

- 4- O Acionador envia comandos para o Planejador para instanciar a aplicação no nó;
- 5- O Planejador envia comandos para o Virtualizador Leve para instanciar a aplicação;
- 6- A Aplicação passa a funcionar sobre o Virtualizador Leve no nó de destino utilizando o conteúdo existente no *Cache*; e
- 7- Agentes (clientes) fazem requisições para a aplicação por meio do Descobridor de Serviço do nó de acesso.

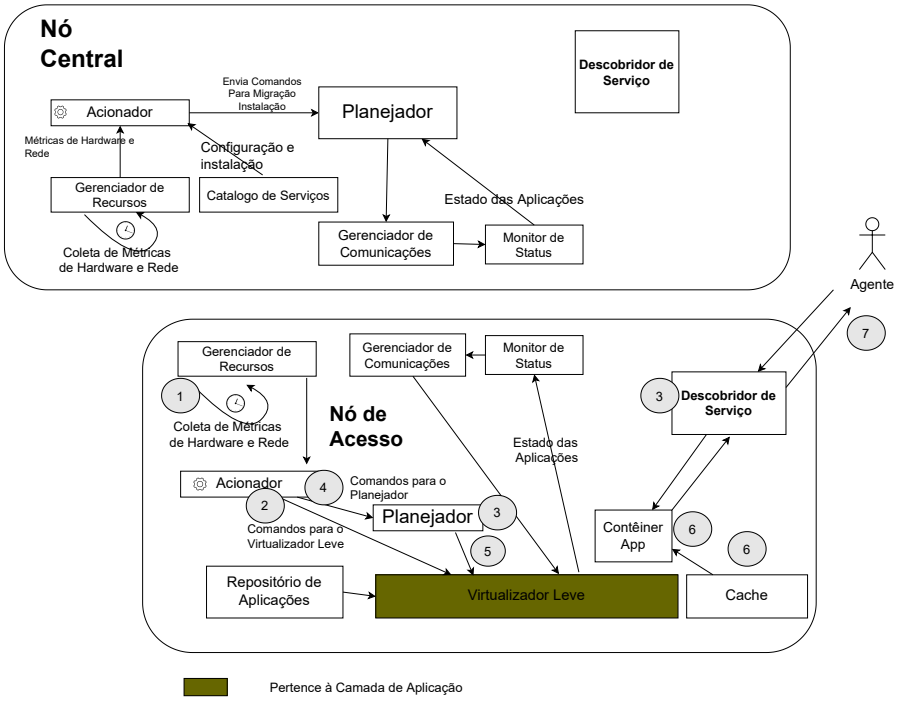

Figura 4.7: Representação de elementos da Meta-camada e o processo de resiliência contra problemas de perda de conexão e nó isolado (Fonte: Elaboração Própria)

4.2.4 Monitor de Recursos

Para a arquitetura proposta foi concebido um sistema de monitoramento para possibilitar mecanismos de otimização e resiliência. O esquema de funcionamento desse sistema está apresentando na Figura 4.8. Nesse esquema é possível observar que os monitores, alocados em cada nó, coletam métricas relativas aos recursos do sistema operacional, e métricas relativas às requisições feitas pelos clientes (oriundas do *proxy* de acesso). Essas métricas

são consolidadas e enviadas ao nó central através de protocolo HTTP. O monitor também avalia métricas relativas à conexão com o nó central (existência da conexão e latência), e recebe o *status* de conexão com a nuvem. As informações recebidas pelos monitores serão usadas pelos mecanismos de otimização e resiliência presentes na arquitetura proposta.

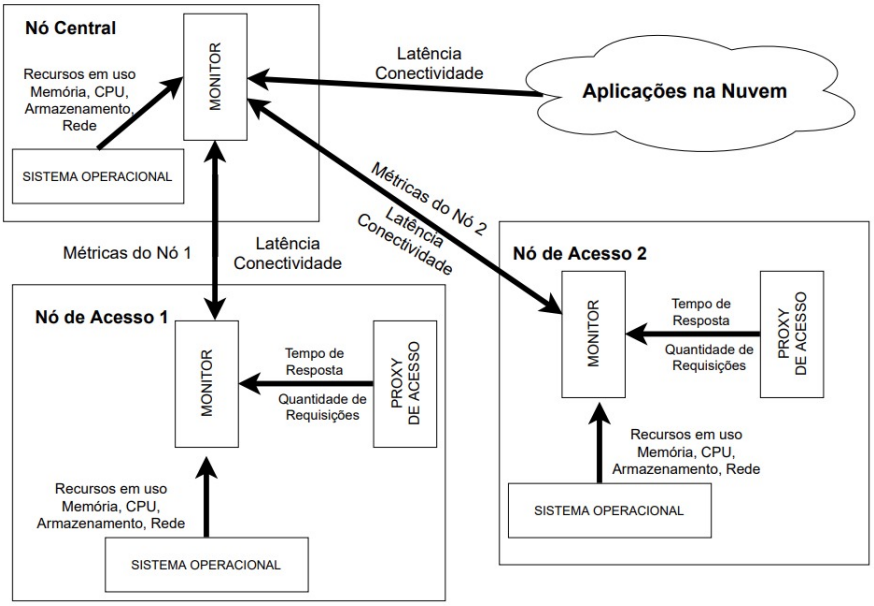

Figura 4.8: Esquema de funcionamento do Monitor de Recursos (Fonte: Elaboração Própria)

4.2.5 Camada de Recursos

A Camada de Recursos contém os recursos de hardware que existem em cada nó da névoa, por exemplo, no caso do CCOp Mv, são aqueles recursos existentes nos hardwares instalados nas viaturas. Esses recursos são relativos à memória RAM, CPU, armazenamento e rede. Além disso, nessa camada também está localizado o sistema operacional sobre o qual serão instaladas as Camadas de Aplicação, Meta-camada e Camada de Acesso. Esses componentes estão descritos a seguir:

- Hardware: fornece CPU, memória RAM, armazenamento, e rede para o sistema operacional;
- Sistema Operacional: responsável pela gerência e operacionalização dos recursos.

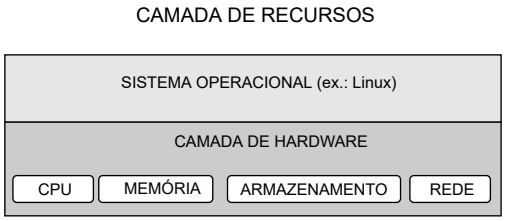

Figura 4.9: Representação de elementos da Camada de Recursos (Fonte: Elaboração Própria)

4.3 Funcionamento dos Mecanismos de Resiliência

A arquitetura proposta será implantada em diferentes cenários, nos quais os mecanismos de orquestração irão atuar. Para todos os cenários é necessário considerar o funcionamento normal dos componentes da arquitetura proposta, a saber:

- Nó central conectado à nuvem: o nó central está conectado à nuvem, e possui a capacidade de fornecer esse acesso aos demais nós;
- DNS: o DNS de todos os nós têm seus apontamentos para os locais originais da aplicação, ou seja, as aplicações que estão na nuvem;
- Monitor de Recursos: os monitores de recursos localizados em cada nó coletam métricas de consumo de recursos, tempo de resposta e conectividade, e repassam essas informações ao monitor de recurso do nó central.

Um diagrama que representa o acesso às aplicações em uma situação normal, no qual o nó central possui recursos amplos de conexão com a nuvem, está apresentado na Figura 4.10. Nesse caso os agentes estão acessando uma aplicação na nuvem que fornece três serviços distintos. Portanto, em todos os nós, os DNS apontam os endereços para os locais originais da aplicação (nuvem). O Monitor de Recursos, localizado no nó central, coletará métricas de tempo de resposta e conectividade com a nuvem. Assim, caso as métricas indiquem degradação no tempo de resposta ou perda de conexão, o Acionador enviará comandos para o Planejador, para dar início ao processo de migração.

Caso o Planejador receba os comandos para efetuar a migração da aplicação para a névoa, será dado início ao processo criando tarefas (definindo recursos como IP e restrição de hardware, como memória RAM e CPU) e, posteriormente, atribuindo tarefas aos nós de acesso, por meio do Gerenciador de Comunicação. Os nós recebem o comando para instanciar os serviços por meio do Gerenciador de Comunicação, e o serviço é instanciado no ambiente de névoa. Após a confirmação da migração, o Acionador do nó central enviará comandos para todos os DNS, mudando o endereço da aplicação para o nó central, onde

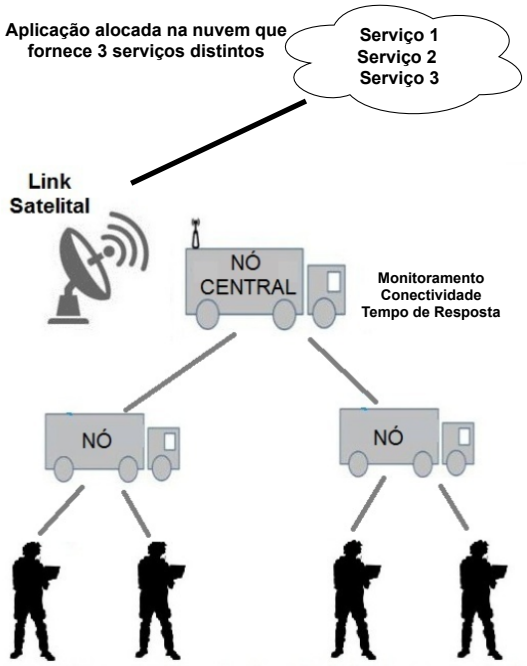

Figura 4.10: Exemplo de acesso às aplicações que estão na nuvem (Fonte: Elaboração Própria)

está localizado o Descobridor de Serviços. Desta forma, os clientes passarão a consumir o serviço que estará sendo provido pela névoa, de maneira transparente.

O acesso para uma aplicação na névoa tem o fluxo representando na Figura 4.11. Nesse cenário o cliente deseja acessar uma aplicação que fornece três serviços (Serviços 1, 2 e 3), ou seja, pode-se considerar que a aplicação possui três módulos distintos. No exemplo apresentado na Figura 4.11, o cliente faz requisições endereçadas ao módulo representado pelo Serviço 3. Os serviços estão distribuídos em três nós distintos. E essa distribuição foi realizada pelo Planejador, localizado no nó central. No entanto, para o cliente esta distribuição é desconhecida, de tal forma que ele faz a requisição endereçada para o nome da aplicação.

Na Figura 4.11 é possível observar que o cliente envia uma requisição para o *Proxy* (Camada de Acesso). Em seguida, o *Proxy*, obtém o endereço de destino da requisição por meio de consulta ao DNS. A requisição então é enviada para o nó que contem o Descobridor de Serviços, nesse caso o nó 1 (nó central). O Descobridor de Serviços conhece onde está localizado cada componente da aplicação e, por fim, enviará a requisição para o nó que possua o contêiner requerido (nesse caso, o nó 3), o qual responderá à requisição.

Após o estabelecimento do serviço na névoa, e com base nas métricas relativas ao número de requisições para um determinado serviço e da situação do *link* com a nuvem, será

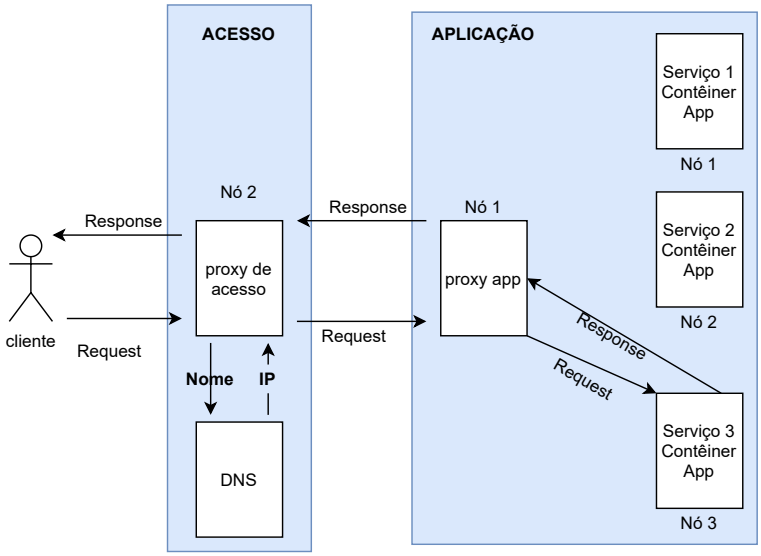

Figura 4.11: Fluxo para resposta de uma requisição para aplicação que esta na névoa (Fonte: Elaboração Própria)

avaliada dinamicamente a possibilidade de retorno do serviço para a nuvem. Caso o retorno seja possível, o Acionador enviará comandos para os nós de acesso, de tal forma que o DNS aponte seus endereços para os sites originais. Posteriormente, enviará comandos ao Planejador para remover as aplicações da névoa. Contudo, nos casos de falhas, os mecanismos existentes na arquitetura atuarão para garantir dois níveis de resiliência, sendo a Resiliência Orientada à Capacidade e a Resiliência Orientada à Eficiência, descritas nas próximas seções.

4.3.1 Resiliência Orientada à Capacidade

Resiliência Orientada à capacidade, como apresentado na Seção 2.4, tem objetivo de suportar o uso legítimo do sistema, Isso significa que a capacidade refere-se à grande quantidade de dados sendo processados e ao grande número de usuários atendidos [18].

Assim sendo, caso o Gerenciador de Recursos, localizado no nó central, avalie que alguma restrição pré-determinada, relativa à memória ou CPU, esteja sendo descumprida (por exemplo, algum nó com sobrecarga), ele enviará comandos para o Planejador. Assim, o orquestrador pode redistribuir as aplicações considerando as restrições. Dessa maneira, ele pode especificar os melhores nós para receberem os serviços, baseando essa decisão em algum algoritmo do tipo balanceador de carga.

4.3.2 Resiliência Orientada à Eficiência

Resiliência Orientada à Eficiência, como apresentado na Seção 2.4, visa prevenir problemas devido à restrições e limitações físicas, tais como *links* de conexão instáveis, baterias de baixa capacidade e outros problemas de conexão [18].

Na arquitetura proposta, caso o Acionador do nó detecte que algum nó de acesso tenha perdido a conexão com o nó central (detecção feita por meio do Monitor de Recursos), ele irá comandar o Planejador para serem criadas tarefas para reestabelecer o estado original do serviço (por exemplo, instanciando o serviço que estava no nó desconectado em outro nó). Quando a conexão com o nó é reestabelecida, o Acionador aciona o Planejador para redistribuir a carga novamente.

Do ponto de vista do nó desconectado, quando o Acionador receber informações do Monitor de Recursos local relativo à desconexão, ele executará comandos sobre o sistema operacional daquele nó. Esses comandos irão instanciar uma nova Meta-camada no nó desconectado, com todos os elementos existentes no nó central. Após isso, o Acionador irá comandar o Planejador para instanciar os serviços a serem ofertados para os clientes. Nesse caso, como existirá apenas um nó, todos os serviços serão instanciados de maneira local. Por fim, o Acionador enviará comandos para o DNS local, para mudar o endereço das aplicações para o próprio nó. Esse mecanismo garantirá a continuidade do serviço para os clientes que possuem o nó desconectado como único ponto de acesso aos sistemas.

Quando a conexão for reestabelecida, o Acionador enviará comandos para o sistema operacional, para que o nó local volte a atuar como nó gerenciado pelo nó central. A partir desse ponto, o Acionador do nó central detectará o reestabelecimento do serviço do nó desconectado e atuará conforme descrito no início desta seção.

4.4 MFOG aplicado ao CCOp Mv

A arquitetura MFOG pode ser aplicada ao CCOp Mv do Exército, os veículos do CCOp irão atuar como nós da névoa, o veículo com mais recursos será o Nó Central e os demais serão considerados Nós de Acesso. A Figura 4.12 apresenta o acesso em situação normal quando os veículos possuem acesso à Internet através do Nó Central.

A Figura 4.13 apresenta um cenário no qual o acesso à Internet apresenta problema. Nesse caso, a arquitetura MFOG instancia as aplicações na névoa de forma que os usuários finais continuam com acesso às aplicações utilizando dados obtidos do *cache* contido na MFOG.

A Figura 4.14 apresenta um cenário no qual as aplicações estão instanciadas na névoa, no entanto, um dos nós apresenta problemas de conexão. Nesse caso, a arquitetura MFOG instancia todas as aplicações no nó isolado de forma que os usuários desse nó continuam

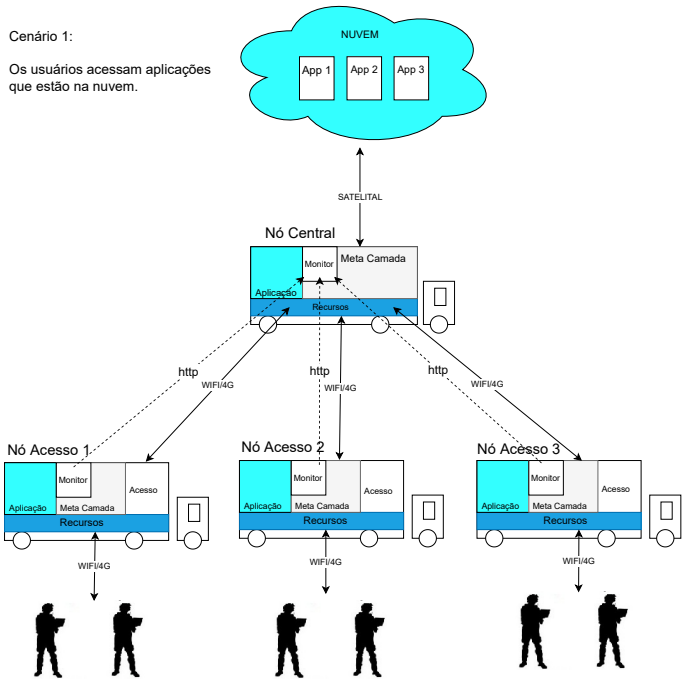

Figura 4.12: MFOG aplicada ao CCOp Mv, cenário com conexão à Internet (Fonte: Elaboração Própria)

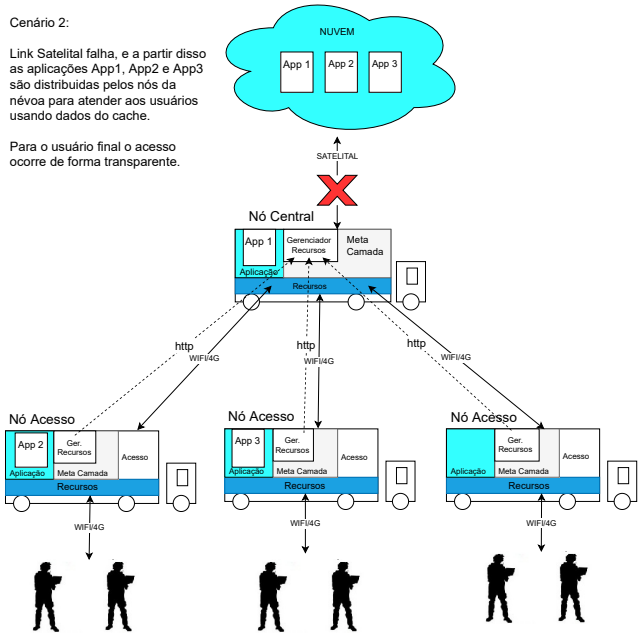

Figura 4.13: MFOG aplicada ao CCOp Mv, cenário sem acesso à Internet (Fonte: Elaboração Própria)

com acesso utilizando o cache existente no nó. A arquitetura também redistribui as aplicações pelos demais nós de forma que os usuários finais continuam com acesso às

aplicações utilizando dados obtidos do *cache* contido na MFOG.

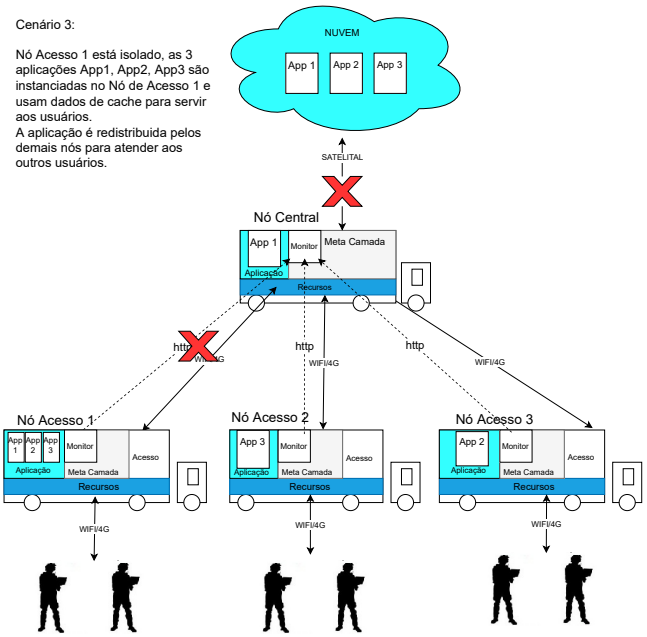

Figura 4.14: MFOG aplicada ao CCOp Mv, cenário com nó isolado (Fonte: Elaboração Própria)

Uma metodologia para experimentação e testes foi elaborado para validar o funcionamento da arquitetura proposta e esta descrita no próximo capítulo.

Capítulo 5

Validação da Arquitetura

Este capítulo apresenta os resultados de testes realizados para verificar o funcionamento e o desempenho da arquitetura *MFOG* aplicada à infraestrutura do CCOp Mv. Para isso, são apresentadas as tecnologias utilizadas, os experimentos realizados e as métricas analisadas. Assim, inicialmente, foi realizado um experimento para demonstrar a diferença de desempenho de uma aplicação alocada diretamente na nuvem, e da mesma aplicação quando alocada na névoa. Além disso, neste experimento foi testado o funcionamento do mecanismo de resiliência orientado à eficiência, proposto para ser utilizado na arquitetura *MFOG*. Outra análise foi realizada em relação à resiliência orientada à capacidade, com o objetivo de demonstrar o funcionamento do mecanismo de resiliência diante de situações de limitações de recursos de hardware nos nós.

5.1 Tecnologia para Implementação

Uma das tecnologias que podem ser utilizadas para a implementação da arquitetura *MFOG* é baseada em contêineres *docker* conhecida como *Docker Swarm*. O *Swarm* consiste em vários nós e podem atuar como gerenciador (para gerenciar associação e delegação) e trabalhadores (que executam serviços *Swarm*). Um determinado nó pode ser um gerente, um trabalhador ou desempenhar ambas as funções.

Ao definir as propriedades de uma aplicação são elencadas métricas como número de réplicas, recursos de rede e armazenamento, portas que o serviço expõe, etc. O *Swarm* trabalhará para manter esse estado desejado [34]. Algumas definições utilizadas para a tecnologia *Docker Swarm* estão listadas a seguir:

- Nós: um nó é uma instância do mecanismo *Docker* que participa do *Cluster Swarm*. As implantações geralmente incluem nós do *Docker* distribuídos em várias máquinas físicas e em nuvem;

- Nó Gerenciador: executam as funções de orquestração e gerenciamento de *cluster* necessárias para manter o estado desejado das aplicações. Esse componente pode ser instanciado no Nó Central para gerenciar os nós de acesso da MFOG.
- Nós trabalhadores: recebem e executam tarefas despachadas do nó gerenciador. No caso da MFOG, esse papel é feito pelo Nó de Acesso;
- Serviço: é a definição das tarefas a serem executadas nos nós gerenciador ou trabalhador. Os usuários do *Swarm* conseguem fazer a implementação de aplicações no *cluster* por meio de configurações e regras enviadas pelas solicitações de serviço.
- Tarefa: uma tarefa carrega um contêiner *Docker* e os comandos a serem executados dentro do contêiner. Depois que uma tarefa é atribuída a um nó, ela não pode ser movida para outro nó. Ela só pode ser executada no nó atribuído, ou falhar;
- Agente: é executado em cada nó trabalhador e relata as tarefas atribuídas a ele. O nó trabalhador notifica o nó gerenciador sobre o estado atual de suas tarefas para que o gerenciador possa manter o estado desejado de cada trabalhador. Na MFOG o elemento responsável por esse monitoramento é o Monitor de *Status*;

O *Swarm* possui um componente DNS interno que atribui automaticamente a cada serviço do *Swarm* uma entrada DNS [34]. O gerenciador usa balanceamento de carga interno para distribuir solicitações entre serviços dentro do *cluster* com base no nome DNS do serviço. Esse componente tem as mesmas atribuições do Descobridor de Serviço presente na MFOG. A Figura 5.1 apresenta os elementos pertencentes a tecnologia *Swarm*.

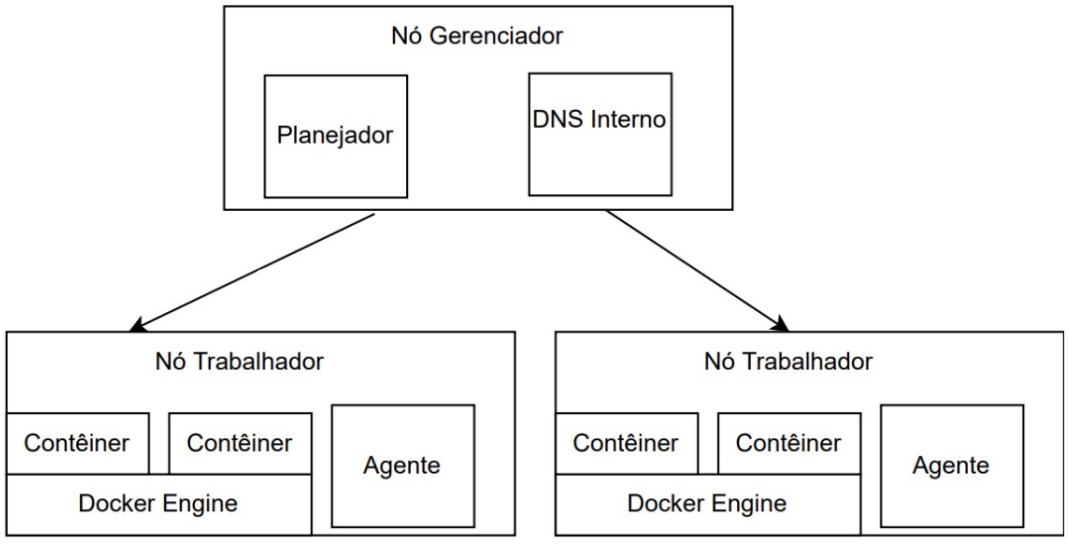

Figura 5.1: Componentes presentes na tecnologia *Swarm* (Fonte: Elaboração Própria)

Importante destacar que a tecnologia *Swarm* possui diversos elementos para a utilização de uma arquitetura baseada em contêineres em névoa, podendo ser utilizada para efetuar a implantação de diversas arquiteturas propostas.

No entanto, para o caso da MFOG, é necessário a utilização de componentes adicionais, como é o caso do Acionador e dos elementos da Camada de Acesso, ademais a tecnologia não apresenta elementos de resiliência, uma vez que as tarefas ficam presas a determinados nós, não sendo transferidas para outros nós de forma dinâmica. A Figura 5.2 apresenta uma comparação entre a arquitetura MFOG e os elementos utilizadas para implantação através do *Docker Swarm*.

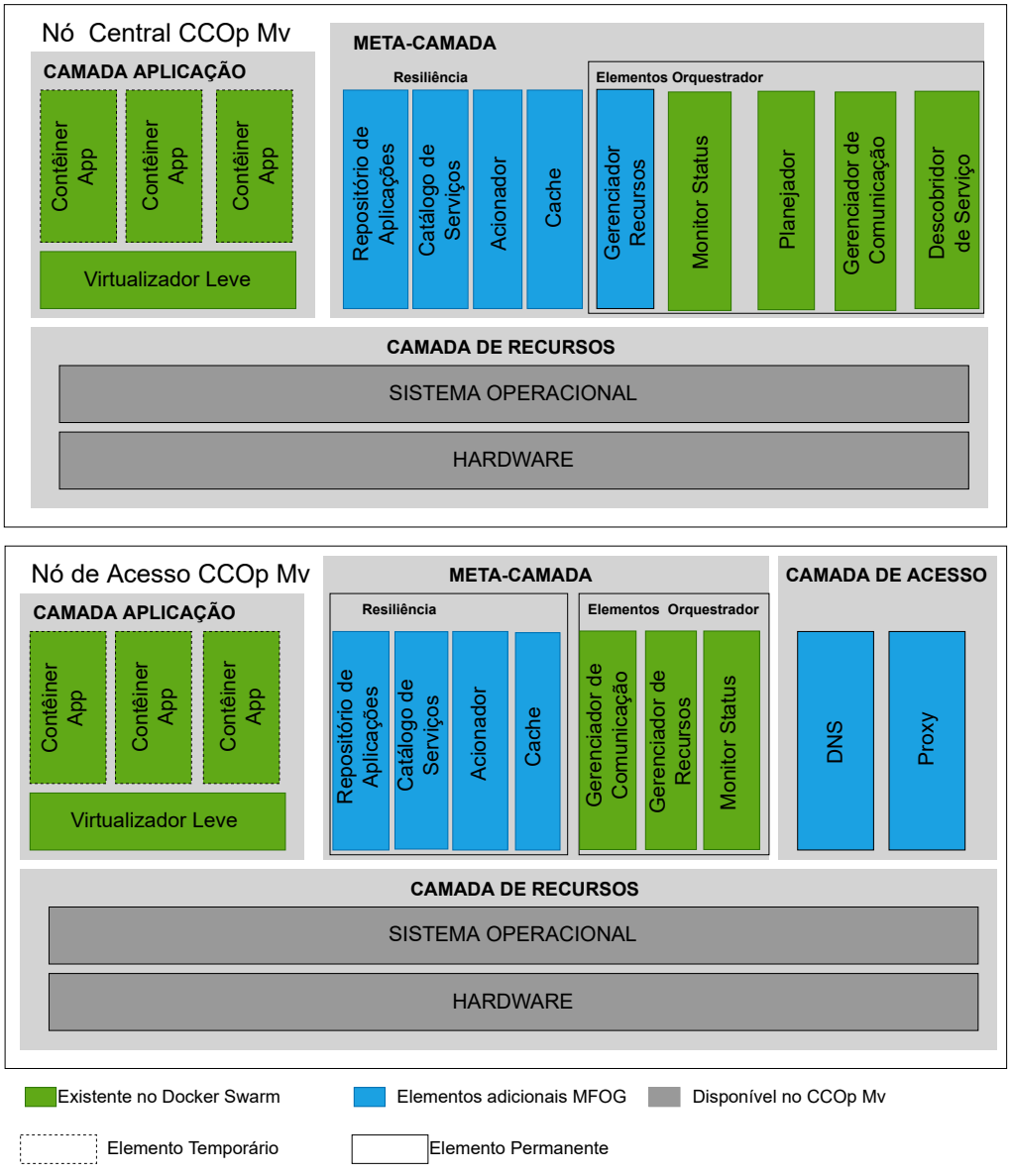

Figura 5.2: Componentes da MFOG implantados pelo *Swarm* (Fonte: Elaboração Própria)

5.2 Metodologia

Para verificar o funcionamento do mecanismo, um estudo de caso foi realizado em ambiente simulado. A aplicação em estudo foi a *Open Map Tile Server*[1], a qual é um servidor de mapas e permite visualizar mapas e dados geográficos em computadores e celulares. A escolha dessa aplicação tem como objetivo simular um sistema que forneça cartas de terreno para possibilitar aos agentes a navegação e orientação em operações de salvamento.

Além disso, a aplicação é do tipo cliente-servidor e possui dois componentes, sendo o Componente 1 (C_1) o *front-end* da aplicação, e o Componente 2 (C_2) o repositório de dados geográficos (que exige capacidade de armazenamento persistente). Um esquema com a estrutura da aplicação está apresentado na Figura 5.3.

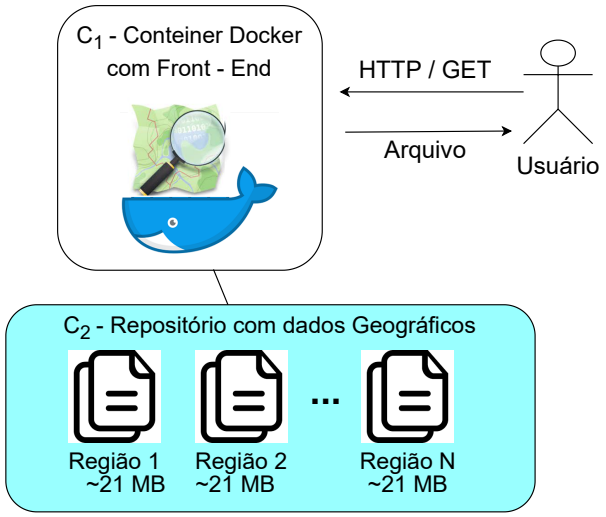

Figura 5.3: Componentes da aplicação utilizada no estudo (Fonte: Elaboração Própria)

A estrutura foi construída usando aplicações *open-source*. Assim, para o virtualizador foi utilizado o *Docker Engine* e os componentes da aplicação foram obtidos por meio do repositório *Docker Hub*. Para o DNS foi utilizada a aplicação *Bind DNS* e criado um *script* em *python* que faz o papel de Monitor de Recursos/Acionador. Além disso, ele alimenta o *Cache* com arquivos consumidos pela aplicação. O *Proxy* foi implementado utilizado a aplicação *Squid*.

A validação do mecanismo de resiliência orientado à eficiência foi feita com base em um experimento que objetivou verificar, em primeiro lugar, o desempenho da aplicação quando comparada sua alocação entre nuvem e na névoa; e em segundo lugar, a disponibilidade da aplicação para o cliente. As métricas analisadas foram o tempo e o tipo de resposta de requisições feitas para a aplicação, as quais indicam o desempenho e a disponibilidade

[1] `openmaptiles.org`

do sistema, respectivamente. O tipo de resposta possibilita checar se a requisição teve êxito em ser antedida, e o tempo de resposta é uma métrica importante para a análise de satisfação do usuário [35]. Assim sendo, os fatores que sofreram alteração no experimento foram a capacidade do *link* e o número de usuários conectados à aplicação.

Além disso, foi comparado o consumo de recursos de hardware quando a aplicação estava na nuvem e quando a aplicação estava na névoa. Para isso, as métricas avaliadas foram % de ocupação de CPU e total de uso de memória RAM, durante os períodos de acesso. Para verificar o consumo desses recursos foi instalada na máquina do nó do CCOp Mv, a aplicação *NetData*[2]. Essa aplicação é uma solução de monitoramento em tempo real, que faz medições relativas às métricas de hardware do equipamento.

A validação do mecanismo de resiliência orientado à capacidade foi feita com base em outro experimento, similar ao anterior, com o uso da mesma aplicação. Contudo, neste caso o software já está instanciado na névoa (que dispõe de 3 (três) nós) e o nó que contém a aplicação é estressado, aumentando o uso de recursos de hardware até um limite que força a arquitetura a migrar a aplicação de lugar. Nesse experimento foram monitoradas a quantidade de recursos (memória RAM) disponível, nos nós em estudo, durante a simulação.

5.3 Experimento 1- Resiliência Orientada à Eficiência

Para este estudo de caso foi concebido um cenário hipotético, representando uma operação em ambiente com recursos escassos, baseado nas seguintes premissas:

- O CCOp Mv possui um nó próximo no local do evento;
- O nó está conectado a nuvem por um *link* com capacidade de 256 Kbps, 512 Kbps, 768 Kbps, 1.024 Kbps, 2.048 Kbps, 4.096 Kbps, 8.192 Kbps, 16.834 Kbps e 32.768 Kbps; e
- Os usuários estão conectados ao nó através de um *link* com capacidade de 32.768 Kbps.

O valor de 256 Kbps à 32.768 Kbps, entre o nó central e a nuvem, simula as capacidades de *link* disponíveis para os órgãos de defesa do Brasil através de satélite. A escolha do valor de 32.768 Kbps para a conexão entre os usuários e o nó tem como objetivo simular um enlace *wirelles*. O ambiente foi montado usando uma máquina virtual (20 Gb RAM, 100 GB de HD, 20 CPU e *Ubuntu* 2018) como nó do CCOp Mv, e outra máquina virtual

[2] `www.netdata.cloud`

(20 Gb RAM, 100 GB de HD, 20 CPU e *Windows Server* 2019) usada para gerar as requisições.

O diagrama da montagem deste Experimento 1 está apresentado na Figura 5.4. Na montagem, o *Apache Jmeter*[3] simula os clientes realizando requisições. A nuvem foi implantada utilizando a nuvem pública *Microsoft Azure* com uma máquina virtual (8 Gb RAM, 100 GB de HD, 20 CPU e *Ubuntu* 2018). As restrições na capacidade do *link* foram criadas com a aplicação *Wondershaper*[4]. Por fim, foi criado um *script* que monitora o conteúdo acessado pelos clientes por meio do *Squid*, e faz cópias dos últimos arquivos acessados no nó central para o *cache* do nó do CCOp Mv. Além disso, o algoritmo monitora a conexão e toma ações baseadas no parâmetro Tempo de Espera (TE), que é o valor que o algoritmo aguarda antes de realizar algum procedimento relativo à migração da aplicação.

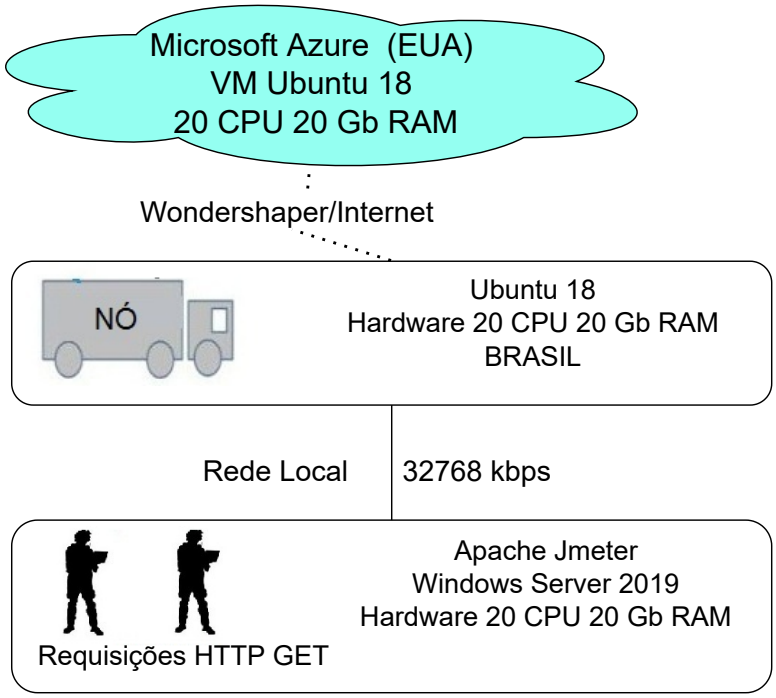

Figura 5.4: Diagrama de montagem do Experimento 1 (Fonte: Elaboração Própria)

Assim sendo, o comportamento do algoritmo em relação aos problemas de conexão está descrito a seguir:

1. A cada segundo monitora o tempo de resposta no nó, calculando a média das últimas 10 requisições (TRM) no *Squid*;

[3] `jmeter.apache.org`
[4] `manpages.ubuntu.com/manpages/trusty/man8/wondershaper.8.html`

2. Caso o tempo de resposta seja maior que a Latência Limite (LL) é acionado um contador ($Cont_1$) o qual é incrementando a cada segundo, enquanto o tempo de resposta for maior que LL;

3. Se antes de ultrapassar TE, o TRM retorne para valores abaixo de LL, o contador é zerado;

4. A cada segundo é monitorado se a conexão está ativa;

5. Caso a conexão não esteja ativa, é acionado outro contador ($Cont_2$) o qual é incrementando a cada segundo enquanto a conexão não é reestabelecida com a nuvem;

6. Se antes de ultrapassar TE, a conexão voltar a ficar ativa, o segundo contador é zerado;

7. Caso o ($Cont_1$) ou ($Cont_2$) ultrapassem TE, o algoritmo muda o apontamento da URL da aplicação para o próprio nó, sendo realizado o *deploy* da mesma, localmente.

Importante destacar que a Latência Limite (LL) é definida como sendo o tempo máximo aceitável para as respostas das requisições, e pode variar de aplicação para aplicação. Assim, para simular o acesso de um usuário com o *JMeter* foi configurada na ferramenta o valor de 15 requisições do tipo HTTP *GET*, que juntas somam 500 KB no tamanho dos arquivos de resposta, a cada 12 segundos. Esse valor tem por base os valores médios obtidos nos estudos de Braga *et al.* [36]. O Tempo de Espera (TE) foi definido com o valor de 10s para que a simulação pudesse ocorrer dentro do intervalo de 1 minuto. A própria ferramenta *JMeter* fornece o tempo de resposta de cada solicitação.

5.3.1 Comparação de Desempenho na Nuvem e na Névoa

Para verificar o funcionamento da aplicação em estudo, com sua implantação inteiramente na nuvem, para um cenário onde existem limitações na capacidade do *link*, foram realizadas diversas simulações, variando o número de usuários e a capacidade do *enlace*. Assim, foi possível obter tempo de resposta da aplicação e taxa de erro nas respostas para cada valor de capacidade de *link* estabelecido. Desta forma, o experimento foi realizado por meio dos seguintes passos:

1. C_1 e C_2 foram alocadas na nuvem;

2. O *DNS* foi configurado para a URL da aplicação apontar para a nuvem, e a simulação foi executada por um período de 1 minuto;

3. Para cada capacidade do *link* 256 Kbps, 512 Kbps, 768 Kbps, 1.024 Kbps, 2.048 Kbps, 4.096 Kbps, 8.192 Kbps, 16.834 Kbps e 32.768 Kbps foram realizadas simulações para 25, 50, 75 e 100 usuários.

A escolha do número de usuários entre 25 e 100 usuários foi feita com base na capacidade do ambiente disponibilizado executar *threads* simultâneas oriundas do *Jmeter*.

Para verificar o funcionamento da aplicação em estudo, com sua implantação inteiramente no nó da névoa (CCOp Mv), foram realizados simulações variando o número de usuários, de forma que fosse possível obter tempo de respostas da aplicação e taxa de erro nas respostas para cada caso. Desta forma, o experimento foi realizado por meio dos seguintes passos:

1. C_1 e parte de C_2 foram alocados na névoa (nó do CCOP Mv);
2. O *DNS* foi configurado para a URL da aplicação apontar para a névoa, e foram realizadas simulações para 25, 50, 75 e 100 usuários;
3. A simulação foi executada por um período de 1 minuto.

5.3.2 Teste do Mecanismo de Resiliência

Para demonstrar o funcionamento do mecanismo de resiliência, em relação à disponibilidade da aplicação, foram realizados os seguintes passos:

1. O *DNS* foi configurado para a URL da aplicação apontar para a nuvem, o *JMeter* ajustado para 50 usuários, e a capacidade do *link* definida em 8192 Kbps;
2. A simulação é iniciada e, após transcorridos 30 segundos, a capacidade do *link* entre o nó e a nuvem é alterada para 4098 Kbps, simulando uma degradação na capacidade;
3. A simulação foi realizada por um período de 2 minutos.

5.3.3 Resultados e Análise do Experimento 1

Os experimentos realizados, com as requisições dos usuários enviadas para a aplicação, forneceram dados de tempo de resposta e taxa de erro para cada caso. Esses dados foram compilados e estão apresentados na Figuras 5.5 e Figura 5.6. Cabe ressaltar que para este experimento, tempo de resposta maior do que 15 segundos foi considerado como requisição com erro. Caso contrário, as simulações poderiam ficar em execução por tempo indefinido.

A Figura 5.5 apresenta o valor médio de tempo de resposta de todas as requisições realizadas para a aplicação na nuvem. Além disso, a Figura 5.6 apresenta a taxa de erro nas respostas das requisições para cada caso e a Tabela 5.2 apresenta os valores para 95º percentil dessas requisições. A análise do resultado indica que a aplicação em estudo tem respostas adequadas, ou seja, taxas de erro próximas a zero e tempo de resposta abaixo

de 3 segundos, para valores de capacidade de *link* acima de 16384 Kbps (16 Mbps), esses valores são possíveis de serem alcançados com o *enlace* satelital disponível para emprego do CCOp Mv, mas ficam muito próximo de sua capacidade máxima.

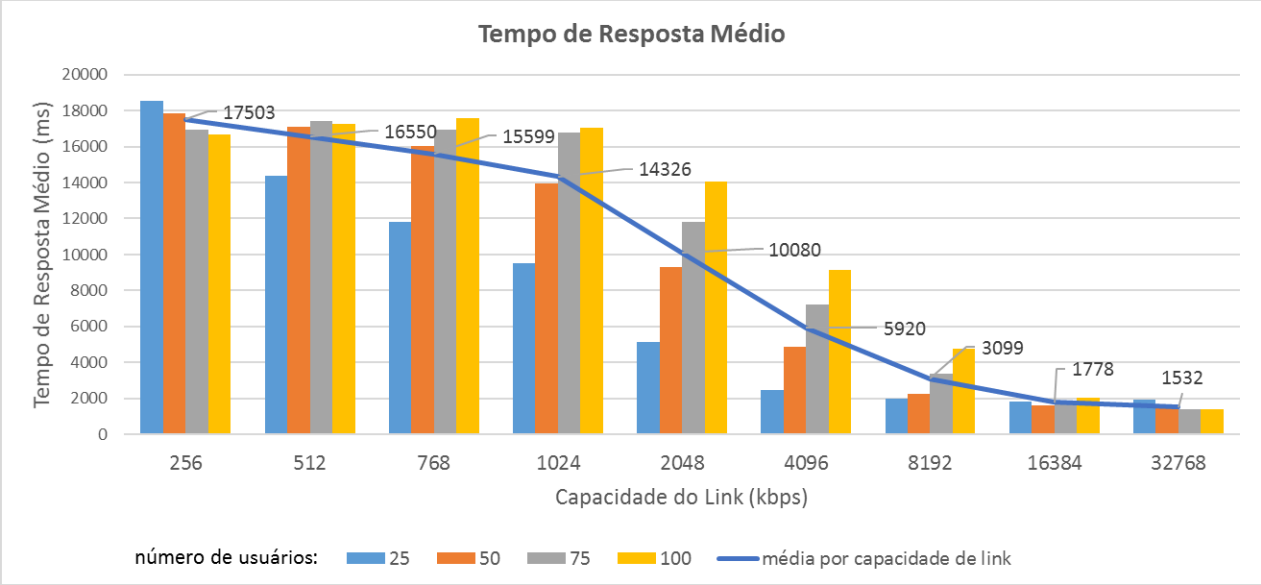

Figura 5.5: Tempo de reposta médio para aplicação na nuvem (Fonte: Elaboração Própria)

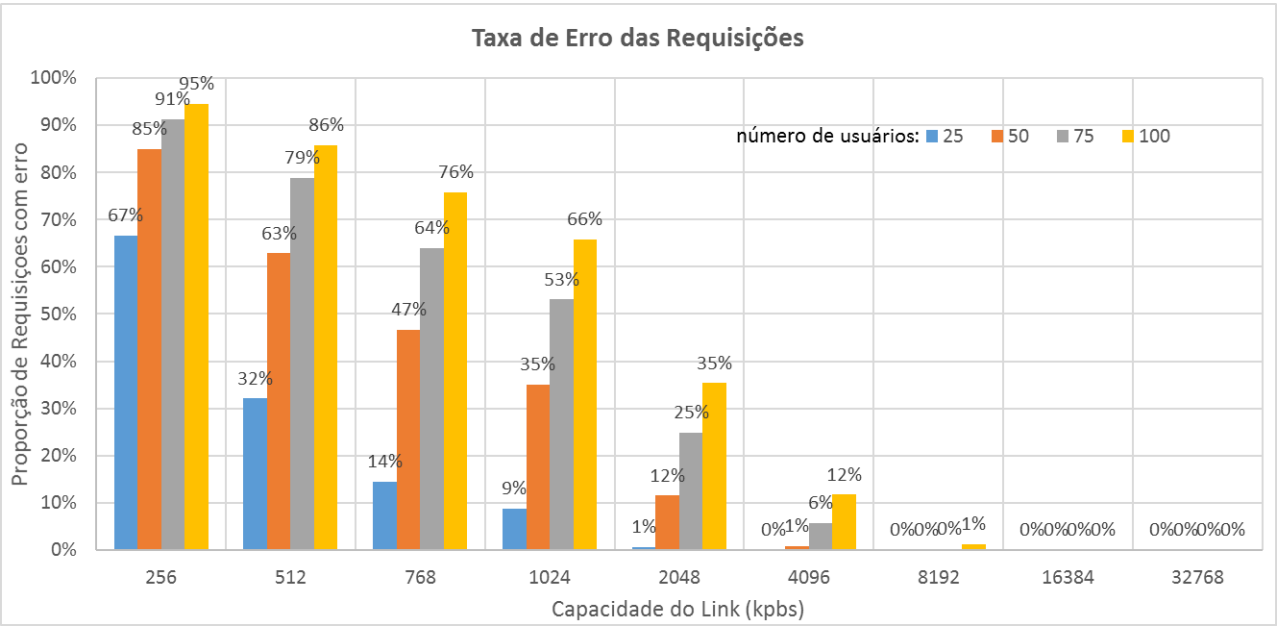

Figura 5.6: Taxa de erro nas requisições (Fonte: Elaboração Própria)

Os resultados dos experimentos realizados com as requisições dos usuários enviadas para a mesma aplicação alocada na névoa (nó do CCOp Mv), estão apresentados na

Tabela 5.1: Tempo de resposta para requisições na nuvem 95 percentil (Fonte: Elaboração Própria)

Capacidade (Kbps)	**Quantidade Usuários**	**Tempo Resposta 95º (ms)**
256	25	34833
256	50	30838
256	75	27781
256	100	25922
512	25	31681
512	50	33327
512	75	30605
512	100	28556
1024	25	26751
1024	50	30614
1024	75	34566
1024	100	32937
2048	25	14788
2048	50	25096
2048	75	27673
2048	100	31534
4096	25	4872
4096	50	13719
4096	75	20090
4096	100	24546
8192	25	3944
8192	50	4617
8192	75	7975
8192	100	13695
16384	25	3158
16384	50	3337
16384	75	3405
16384	100	4132
32768	25	2980
32768	50	3011
32768	75	3667
32768	100	3821

Tabela 5.2: Tempo de resposta para aplicação na névoa (Fonte: Elaboração Própria)

Local	**Nº Usuários**	**Média***	**Min***	**Max***	**90º Percentil***
Névoa	25	11	4	87	15
Névoa	50	10	3	135	14
Névoa	75	10	3	142	14
Névoa	100	12	3	129	17
Nuvem**	25	1820	228	5856	3380
Nuvem**	50	1593	224	5744	2601
Nuvem**	75	1678	227	5127	2408
Nuvem**	100	2023	226	7989	2379

* Os valores de tempo estão em ms. ** A capacidade do link é de 32768 Kbps

Tabela 5.2. Nesta tabela, para fins de comparação, também estão apresentados os dados de tempo de resposta para a aplicação alocada na nuvem com a maior capacidade de *link* (32768 Kbps). A taxa de erro para as repostas das requisições, quando a aplicação estava na névoa, ficaram em 0% para todos os casos.

A comparação indica que o tempo de resposta, quando a aplicação está na névoa (nó do CCOp Mv) é, em média, mais de 100 vezes menor em relação à nuvem, e a taxa de erro, para todos os casos, é nula. No entanto, deve-se levar em consideração que o servidor de nuvem está alocado em região geográfica distante do nó da névoa (para este estudo de caso o servidor foi alocado nos EUA e o nó no Brasil). Assim, é possível diminuir o tempo de resposta alocando o servidor mais próximo do CCOp Mv. No entanto, o nó da névoa está apenas a um salto de distância (os usuários estão conectados diretamente no nó), o que implica que para todos os casos o tempo de resposta do nó de névoa será menor.

Outro ponto importante é o fato da aplicação, de mapa e dados geográficos, necessitar de grande capacidade de volume para armazenar dados, o que é possível de se alcançar com os recursos existentes na nuvem. Já os nós do CCOp Mv têm capacidade limitada para armazenamento de dados, o que inviabiliza a instalação completa da aplicação de mapa nos equipamentos da viatura.

Os resultados obtidos com o mecanismo de resiliência orientado à eficiência são mostrados na Figura 5.7. Esse mecanismo faz a migração da aplicação da nuvem para a névoa, no caso de problemas com a conexão. No gráfico da é possível observar que a partir do momento que a capacidade do *link* é reduzida de 8192 Kbps para 4096 Kbps, o tempo de resposta médio (média para 10 requisições) aumenta e fica acima da latência limite, que para este estudo de caso foi fixada em 3000 ms, esse valor está dentro do limite, apresentado por Winckler *et al.* [37], como sendo o valor máximo para tempo de resposta no qual o usuário detecta lentidão.

Após 10 segundos na situação de tempo de resposta maior do que LL, o mecanismo de migração é acionado de tal forma que ele cria uma instância da aplicação no nó da névoa.

Assim sendo, é alterada a URL da aplicação para apontar para o nó e, por fim, reinicia o *Proxy* para que as configurações sejam atualizadas. Durante este processo os usuários ficam cerca de 10s sem acesso à aplicação.

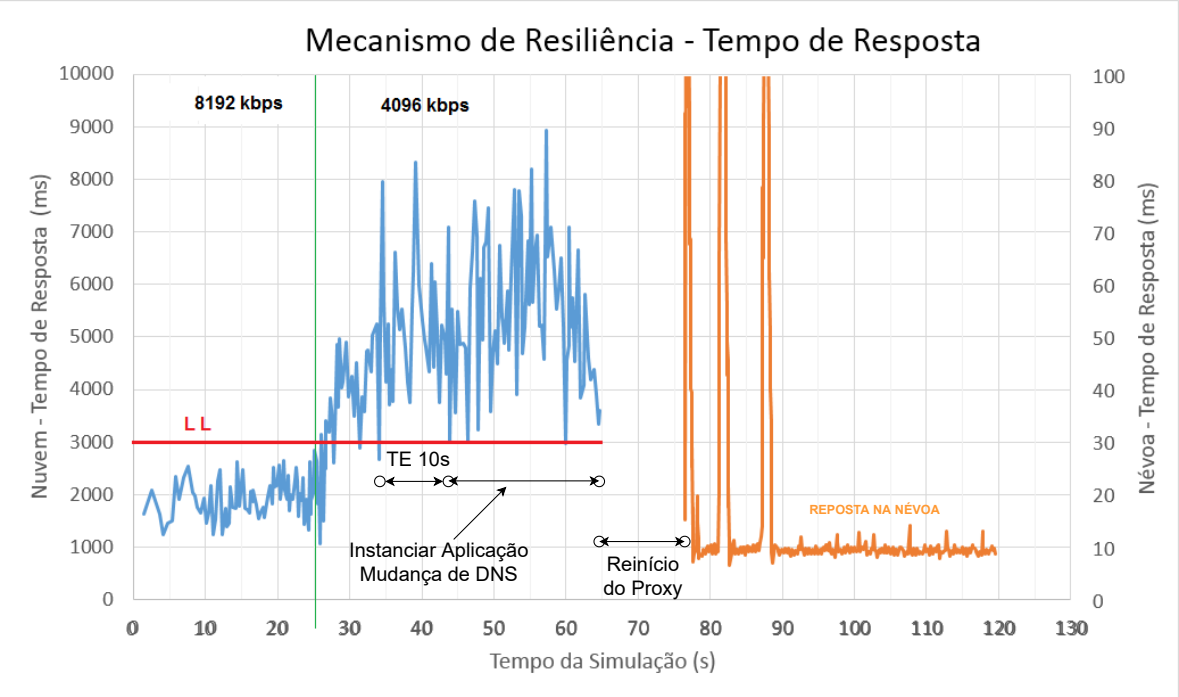

Figura 5.7: Funcionamento do mecanismo proposto (Fonte: Elaboração Própria)

O objetivo deste teste foi verificar o funcionamento do mecanismo de resiliência proposto, demonstrando ser factível o seu funcionamento, pois, a migração obteve êxito. Contudo, cabe ressaltar que o conteúdo contido no repositório da aplicação (repositório de arquivos contendo imagens de mapas e dados geográficos) foi criado anteriormente durante o acesso à nuvem, tendo efetuado a migração do conteúdo relativo apenas a uma região do globo (neste exemplo, Região 1). Assim, caso os usuários necessitassem de conteúdo de outras regiões, a aplicação não poderia suprir tal necessidade.

Outro ponto importante analisado nos testes foi o consumo de recursos de hardware. Para essa análise, foi avaliado o consumo para cada etapa deste estudo de caso, ou seja, aplicação na nuvem, aplicação na névoa e migração da aplicação com o mecanismo proposto (etapa anterior a migração e posterior a migração). O resultado desse levantamento está apresentado na Tabela 5.3. Analisando os dados dessa tabela é possível observar que quando a aplicação está instanciada no nó da névoa, o consumo de memória RAM e CPU tem um aumento significativo, isto é, mais de 4 vezes para uso de CPU e cerca de 1,5 vezes para o consumo de memória RAM.

Além disso, é possível destacar que o mecanismo proposto não afeta significativamente o consumo de recursos de hardware. Essas indicações mostram que migrar aplicações da

Tabela 5.3: Consumo de Recursos de Hardware (Fonte: Elaboração Própria)

Situação	**Uso de CPU (Max)**	**RAM (Max)**
Sem utilização	1%	730 MB
Aplicação na Nuvem	3%	740 MB
Aplicação na Névoa	14%	1100 MB
Mecanismo Antes Migração	3%	730 MB
Mecanismo Após Migração	15%	1100 MB

nuvem para a névoa podem aumentar significativamente o consumo de hardware no nó da névoa, de forma que a depender da quantidade de aplicações a migração dessas aplicações para névoa podem se tornar inviáveis.

5.4 Experimento 2- Resiliência Orientada à Capacidade

Similar ao experimento anterior, nesta avaliação foi concebido um cenário hipotético, representando uma operação em ambiente com recursos escassos, baseado nas seguintes premissas:

- O CCOp Mv possui 3 (três) nós próximo ao local do evento, 1 nó central e 2 nós de acesso;
- O nó central está desconectado da nuvem;
- A aplicação esta instanciada na névoa em um dos nós de acesso;e
- O nó de acesso que contém a aplicação passa por um processo que tem seus recursos estressados, não sendo mais capaz de suportar a operação.

O ambiente foi montado usando uma máquina virtual com 20 Gb RAM, 100 GB de HD, 20 CPU e *Ubuntu* 2018 como nó central do CCOp Mv. Outras duas máquinas virtuais com 8 Gb RAM, 100 GB de HD, 4 CPU e *Ubuntu* 2018, foram utilizados como nó de acesso do CCOp Mv. Por fim, outra máquina virtual com 20 Gb RAM, 100 GB de HD, 20 CPU e *Windows Server* 2019 foi utilizada para gerar as requisições.

O diagrama da montagem está apresentado na Figura 5.8. Nesse experimento, a arquitetura MFOG foi implantada utilizando a tecnologia *Docker Swarm*.

Na montagem, o *stress* no uso de recurso no nó de acesso que contém a aplicação foi realizado utilizando a aplicação *Stress-NG*[5]. Por fim, foi criado um *script* em *python*

[5]`manpages.ubuntu.com/manpages/bionic/man1/stress-ng.1.html`

que monitora os recursos de hardware disponíveis em cada nó, que tem as atribuições do Monitor de Recursos e do Acionador previstos na arquitetura MFOG.

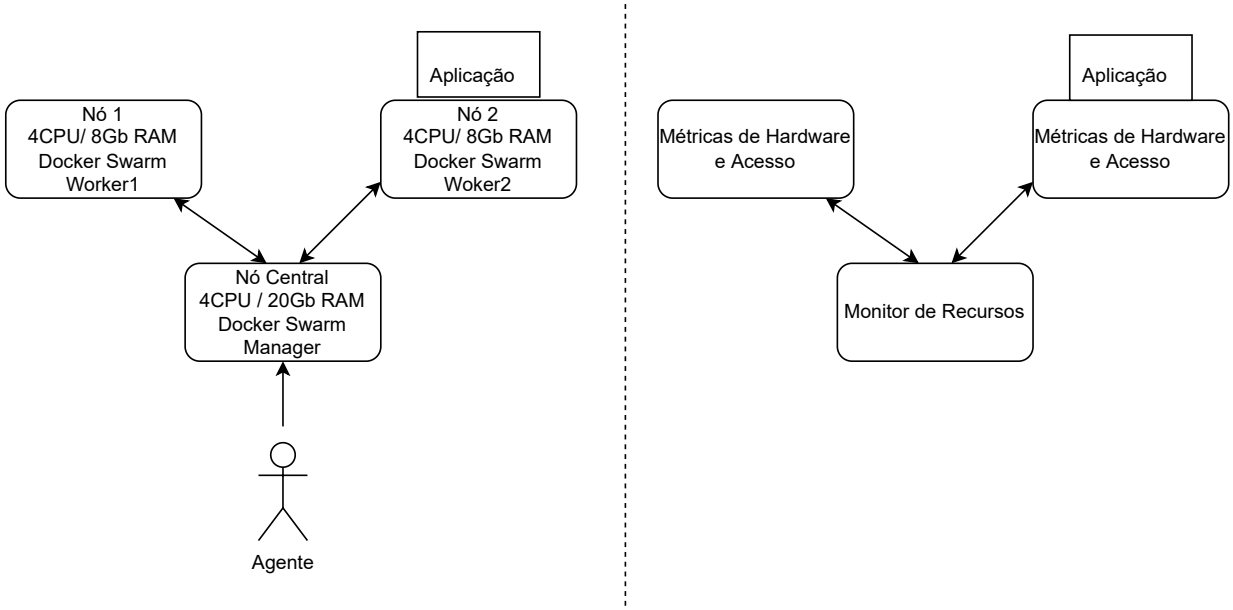

Figura 5.8: Diagrama de montagem do Experimento 2 (Fonte: Elaboração Própria)

O comportamento do algoritmo em relação aos problemas de limitação de recurso em um nó está descrito a seguir:

1. A cada segundo o monitor do nó central analisa a quantidade de recurso livre (memória RAM) em cada nó de acesso;
2. Caso a quantidade de recurso livre (memória RAM) em algum nó seja menor que um valor pré-determinado (aqui denominado valor mínimo (VL)), por tempo superior ao Tempo de Espera (TE), já utilizado no experimento anterior, o algoritmo migra a aplicação para outro nó que contenha recursos livres;
3. Caso não existam nós com recursos livres (memória RAM), o algoritmo não toma nenhuma ação.

Importante destacar que VL neste experimento foi definido como 20% do valor total existente no nó, para ser possível simular a utilização e a migração após a utilização da ferramenta *stress-ng*. O TE nessa simulação foi fixado em 5 segundos para que as ações de migração ocorressem em uma janela observável de um minuto.

5.4.1 Resultados e Análise do Experimento 2

A Figura 5.9 apresenta a memória RAM livre nos nó 1 e 2 durante a simulação. Assim, é possível observar que ao iniciar o *stress* no nó 2, no quinto segundo da simulação, o valor de memória livre no nó 2 começa a diminuir até valores abaixo de VL (2 Gb), o que ocorre no 25º segundo da simulação. Esta situação dura por 5 segundos (TE) até que o processo de migração da aplicação seja iniciado.

Após isso, a aplicação é migrada para o nó 1, como pode ser observado uma vez que o consumo de memória naquele nó aumenta, ou seja, ocorre uma diminuição da memória livre no nó. O resultado mostra que o mecanismo proposto atuou conforme o esperado, migrando a aplicação para outro nó diante de problemas de limitação de recursos.

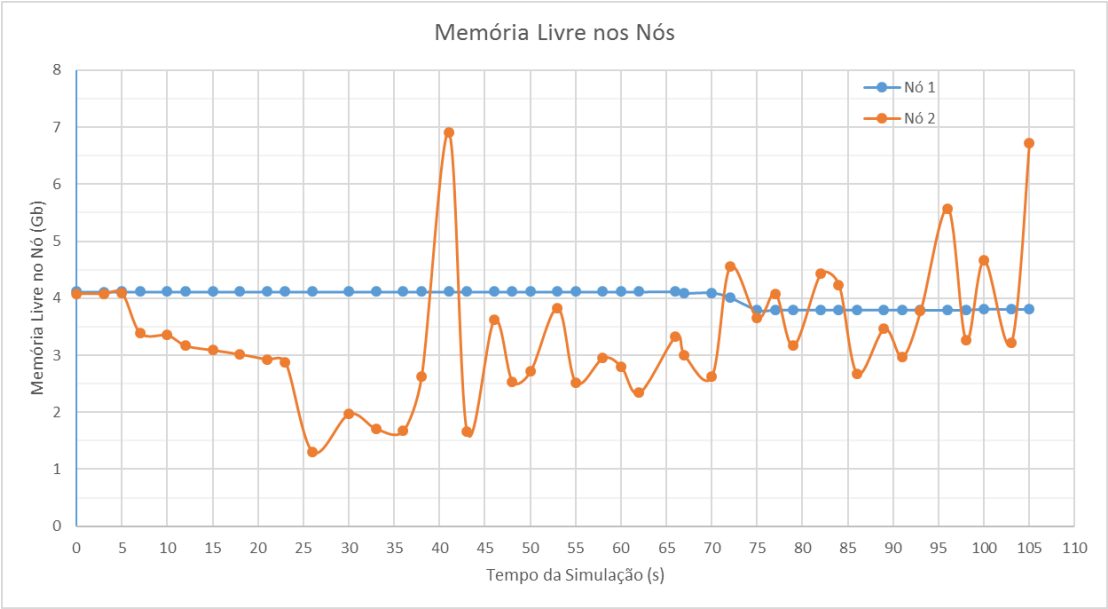

Figura 5.9: Comportamento dos nós na migração por restrição de recursos (Fonte: Elaboração Própria)

Capítulo 6

Conclusão

Este trabalho de mestrado apresentou a arquitetura *MFOG*, a qual é uma arquitetura resiliente para a implantação de aplicações em estruturas computacionais localizadas próximas ao usuário final, mas com limitação em recursos de conexão com a Internet e recursos de hardware. Os nós destas estruturas, além de serem elementos de um sistema distribuído, também são, por vezes, os únicos pontos de acesso de diversos usuários. Assim sendo, a arquitetura *MFOG* foi proposta para ser uma solução plenamente resiliente, com foco em resiliência orientado à eficiência e à capacidade, mantendo desta forma a disponibilidade das aplicações para o usuário final.

Os resultados de estudo de caso aplicado ao CCOp Mv indicaram que, quando se utiliza *links* com limitação de banda para a conexão com nuvem, pode haver um baixo desempenho em relação ao tempo de resposta da aplicação alocada na nuvem. Ademais, os resultados também indicaram que um mecanismo de resiliência contra problemas de conexão, que migra a aplicação para o nó isolado, pode fornecer uma garantia de funcionamento da aplicação mesmo diante de uma perda total de conexão por este nó.

O mecanismo que trata de problemas de capacidade de hardware nos nós também foi testado em um estudo de caso aplicado ao CCOp Mv. Nesse estudo foi possível observar que o monitoramento realizado pela arquitetura pode detectar a falta de recursos de hardware, e tomar ações de migração das aplicações para outro nó da névoa que esteja com recursos disponíveis.

O estudo de caso mostrou que a arquitetura MFOG pode ser utilizada para implantação de aplicações que estão em contêineres em estruturas distribuídas próximas ao usuário final, utilizando o paradigma da computação em névoa. Assim, a arquitetura proposta possui mecanismos de monitoramento de recursos e migração das aplicações que garantem a disponibilidade dos serviços para o usuário final.

Para a continuidade deste trabalho, pretende-se implementar no futuro mecanismos de migração, e de sincronismo de arquivos e armazenamento de dados. Esses mecanismos não

foram tratados neste trabalho, mas é possível agregar essas funcionalidades à arquitetura proposta, de forma que uma maior gama de aplicações funcionem sobre a MFOG.

Além disso, como trabalho futuro, deve-se avaliar a integração entre névoa e nuvem, de tal forma que a arquitetura possa distribuir os componentes da aplicação ao longo da camada de névoa e nuvem de forma transparente. Pois, na arquitetura atual a aplicação está na camada de névoa ou está na camada de nuvem.

Por último, é importante registrar que um artigo sobre o mecanismo de resiliência contra problemas de conexão do MFOG foi publicado na 17º Conferência Ibérica sobre Sistemas de Informação e Tecnologias (2022)[1] com o título "*Resilience Mechanism for Applications in Fog*"[38].

[1] (https://ieeexplore.ieee.org/xpl/conhome/9819948/proceeding)

Referências

[1] Brasil, Exército Brasileiro.: *Programa Proteger: Proteção da Sociedade.* Disponível em: `www.dct.eb.mil.br/images/conteudo/DSMEM/rfi/RFI_CCOp_Mv_07_MAI_21.pdf`. Acesso em: 07 junho 2021, 2019. 1

[2] Mell, Peter, Tim Grance *et al.*: *The nist definition of cloud computing.* National Institute of Standards and Technology, 2011. 1

[3] Habibi, Pooyan, Mohammad Farhoudi, Sepehr Kazemian, Siavash Khorsandi e Alberto Leon-Garcia: *Fog computing: a comprehensive architectural survey.* IEEE Access, 8:69105–69133, 2020. 2, 8, 10, 16, 17

[4] Santos, Jose, Tim Wauters, Bruno Volckaert e Filip De Turck: *Towards network-aware resource provisioning in kubernetes for fog computing applications.* Em *2019 IEEE Conference on Network Softwarization (NetSoft)*, páginas 351–359. IEEE, 2019. 2, 22

[5] Santos, José, Jeroen van der Hooft, Maria Torres Vega, Tim Wauters, Bruno Volckaert e Filip De Turck: *Srfog: A flexible architecture for virtual reality content delivery through fog computing and segment routing.* Em *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, páginas 1038–1043. IEEE, 2021. 2, 21, 22, 23

[6] Rosário, Denis, Matias Schimuneck, João Camargo, Jéferson Nobre, Cristiano Both, Juergen Rochol e Mario Gerla: *Service migration from cloud to multi-tier fog nodes for multimedia dissemination with qoe support.* Sensors, 18(2):329, 2018. 2, 21, 23

[7] Taherizadeh, Salman, Vlado Stankovski e Marko Grobelnik: *A capillary computing architecture for dynamic internet of things: Orchestration of microservices from edge devices to fog and cloud providers.* Sensors, 18(9):2938, 2018. 2, 21, 23

[8] Brasil: *Portaria nº 258-EME de 31 de outubro de 2018.* Diário Oficial da República Federativa do Brasil, 2018. 4

[9] Melo Junior, Pedro Nicolau de: *A utilização dos meios satelitais nas operações militares - trabalho de conclusão de curso (especialização em ciências militares).* Rio de Janeiro, Escola de Comando e Estado-Maior do Exército, 2019. 7

[10] Sousa, Flávio RC, Leonardo O Moreira e Javam C Machado: *Computação em nuvem: Conceitos, tecnologias, aplicações e desafios.* II Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI), páginas 150–175, 2009. 8

[11] Mell, Peter M e Timothy Grance: *Sp 800-145. the nist definition of cloud computing*, 2011. 8

[12] Lin, Angela e Nan Chou Chen: *Cloud computing as an innovation: Percepetion, attitude, and adoption*. International Journal of Information Management, 32(6):533–540, 2012. 8

[13] Ren, Ju, Deyu Zhang, Shiwen He, Yaoxue Zhang e Tao Li: *A survey on end-edge-cloud orchestrated network computing paradigms: transparent computing, mobile edge computing, fog computing, and cloudlet*. ACM Computing Surveys (CSUR), 52(6):1–36, 2019. 9

[14] Hu, Pengfei, Sahraoui Dhelim, Huansheng Ning e Tie Qiu: *Survey on fog computing: architecture, key technologies, applications and open issues*. Journal of network and computer applications, 98:27–42, 2017. 9

[15] Mahmud, Redowan, Ramamohanarao Kotagiri e Rajkumar Buyya: *Fog computing: A taxonomy, survey and future directions*. Em *Internet of everything*, páginas 103–130. Springer, 2018. 10

[16] Bachiega Jr, João, Breno Costa e Aleteia PF Araujo: *Computational perspective of the fog node*. arXiv preprint arXiv:2203.07425, 2022. 10, 11

[17] Naha, Ranesh Kumar, Saurabh Garg, Dimitrios Georgakopoulos, Prem Prakash Jayaraman, Longxiang Gao, Yong Xiang e Rajiv Ranjan: *Fog computing: Survey of trends, architectures, requirements, and research directions*. IEEE access, 6:47980–48009, 2018. 10

[18] Sterbenz, James PG, David Hutchison, Egemen K Çetinkaya, Abdul Jabbar, Justin P Rohrer, Marcus Schöller e Paul Smith: *Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines*. Computer networks, 54(8):1245–1265, 2010. 12, 13, 37, 38

[19] Prokhorenko, Victor e M Ali Babar: *Architectural resilience in cloud, fog and edge systems: A survey*. IEEE Access, 8:28078–28095, 2020. 13, 25, 26

[20] Mijuskovic, Adriana, Alessandro Chiumento, Rob Bemthuis, Adina Aldea e Paul Havinga: *Resource management techniques for cloud/fog and edge computing: An evaluation framework and classification*. Sensors, 21(5):1832, 2021. 13

[21] Jaakkola, Hannu e Bernhard Thalheim: *Architecture-driven modelling methodologies*. Em *Information Modelling and Knowledge Bases XXII*, páginas 97–116. IOS Press, 2011. 15

[22] Antonini, Mattia, Massimo Vecchio e Fabio Antonelli: *Fog computing architectures: A reference for practitioners*. IEEE Internet of Things Magazine, 2(3):19–25, 2019. 15, 16

[23] Dastjerdi, Amir Vahid, Harshit Gupta, Rodrigo N Calheiros, Soumya K Ghosh e Rajkumar Buyya: *Fog computing: Principles, architectures, and applications*. Em *Internet of things*, páginas 61–75. Elsevier, 2016. 16

[24] Bonomi, Flavio, Rodolfo Milito, Preethi Natarajan e Jiang Zhu: *Fog computing: A platform for internet of things and analytics*. Em *Big data and internet of things: A roadmap for smart environments*, páginas 169–186. Springer, 2014. 17

[25] Habibi, Pooyan, Soroush Baharlooei, Mohammd Farhoudi, Sepehr Kazemian e Siavash Khorsandi: *Virtualized sdn-based end-to-end reference architecture for fog networking*. Em *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, páginas 61–66. IEEE, 2018. 17

[26] Velasquez, Karima, David Perez Abreu, Diogo Goncalves, Luiz Bittencourt, Marilia Curado, Edmundo Monteiro e Edmundo Madeira: *Service orchestration in fog environments*. Em *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, páginas 329–336. IEEE, 2017. 18

[27] Brito, Mathias Santos de, Saiful Hoque, Thomas Magedanz, Ronald Steinke, Alexander Willner, Daniel Nehls, Oliver Keils e Florian Schreiner: *A service orchestration architecture for fog-enabled infrastructures*. Em *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, páginas 127–132. IEEE, 2017. 18

[28] Costa, Breno, Joao Bachiega Jr, Leonardo Rebouças de Carvalho e Aleteia PF Araujo: *Orchestration in fog computing: A comprehensive survey*. ACM Computing Surveys (CSUR), 55(2):1–34, 2022. 20, 22

[29] Nguyen, Nguyen Dinh, Linh An Phan, Dae Heon Park, Sehan Kim e Taehong Kim: *Elasticfog: elastic resource provisioning in container-based fog computing*. IEEE Access, 8:183879–183890, 2020. 22, 23

[30] Mouradian, Carla, Fereshteh Ebrahimnezhad, Yassine Jebbar, Jasmeen Kaur Ahluwalia, Seyedeh Negar Afrasiabi, Roch H Glitho e Ashok Moghe: *An iot platform-as-a-service for nfv-based hybrid cloud/fog systems*. IEEE Internet of Things Journal, 7(7):6102–6115, 2020. 22, 23

[31] Bakhshi, Zeinab, Guillermo Rodriguez-Navas e Hans Hansson: *Fault-tolerant permanent storage for container-based fog architectures*. Em *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, volume 1, páginas 722–729. IEEE, 2021. 22, 23

[32] Núñez-Gómez, Carlos, Blanca Caminero e Carmen Carrión: *Hidra: A distributed blockchain-based architecture for fog/edge computing environments*. IEEE Access, 2021. 22, 23

[33] Trindade, Leon Valentim Porto e Luís Henrique MK Costa: *Análise do desempenho da virtualização leve para ambientes com edge computing baseada em nfv*. Em *Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, páginas 866–879. SBC, 2018. 27

[34] Docker Inc: *Swarm mode key concepts*, 2021. `https://docs.docker.com/engine/swarm`, acesso em 1 jun. 2021. 41, 42

[35] Wiedenhoefer, Lars: *Key metrics*. Em *Digital Customer Experience Engineering*, páginas 95–116. Springer, 2021. 45

[36] Braga, Vinícius Gonçalves, Sand Luz Correa, Kleber Vieira Cardoso e Aline Carneiro Viana: *Data-driven characterization and modeling of web map system workload*. IEEE Access, 9:26983–27002, 2021. 47

[37] Winckler, Marco e Marcelo Soares Pimenta: *Avaliação de usabilidade de sites web*. Escola de Informática da SBC Sul (ERI 2002). Porto Alegre, 1:85–137, 2002. 51

[38] Silva, Marcos Francisco da, Marcos F Caetano, Marcelo A Marotta, Lucas Bondan, Geraldo P Rocha Filho e Aleteia Araujo: *Resilience mechanism for applications in fog*. Em *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*, páginas 1–7. IEEE, 2022. 57