Ambiente em nuvem para a plataforma e-Cattle utilizando multi-tenant

Ygo Aquino Brito

Ambiente em nuvem para a plataforma e-Cattle utilizando multi-tenant

Ygo Aquino Brito

Orientadora: *Profª Drª Hana Karina Salles Rubinsztejn*
Coorientador: *Drº Camilo Carromeu*

Dissertação entregue a Faculdade de Computação da Universidade Federal de Mato Grosso do Sul - FACOM-UFMS como parte dos requisitos para a obtenção do título de Mestre em Computação Aplicada.

FACOM - Campo Grande
agosto/2023

Agradecimentos

Gostaria de agradecer primeiramente aos meus pais, pelo constante apoio e incentivo ao longo de toda a minha jornada acadêmica.

À minha irmã, por estar ao meu lado em todas as etapas de minha vida.

À minha esposa, pelo companheirismo, motivação e compreensão durante todos os momentos.

Aos meus orientadores, professora Drª Hana Karina Salles Rubinsztejn e Drº Camilo Carromeu, por toda a orientação, sabedoria, paciência e contribuições valiosas ao meu trabalho.

Aos amigos, que sempre estiveram ao meu lado, compartilhando alegrias e desafios.

Enfim, expresso minha gratidão a todos aqueles que, de alguma forma, contribuíram para o sucesso deste trabalho e para minha formação acadêmica. Cada pessoa mencionada e todas aquelas que estiveram presentes nessa jornada são peças fundamentais. Muito obrigado!

Abstract

This work presents the development of a cloud environment for data synchronization of the e-Cattle platform. e-Cattle is a platform created for beef farms, which aggregates data from different IoT devices managed through a middleware called BigBoxx. The cloud environment has been designed with a focus on supporting the e-Cattle platform while allowing integration with existing infrastructure. In order to achieve data synchronization efficiently, the multi-tenant architecture was adopted, aiming to isolate the data and services of each rural property, creating an environment with greater security and scalability. In addition, this research provided the synchronization of multiple BigBoxx for a farm, an important upgrade for e-Cattle. The cloud environment developed proved to be a solution for data synchronization of an existing IoT platform. The adoption of modern, scalable and multi-tenant technologies enabled an optimized management of resources, resulting in an environment prepared to handle the growing volume of data generated by the multiple devices present in e-Cattle.

Resumo

Este trabalho apresenta o desenvolvimento de um ambiente em nuvem para a sincronização de dados da plataforma e-Cattle. O e-Cattle é uma plataforma criada para propriedades produtoras de carne bovina, que agrega dados de diferentes dispositivos IoT gerenciados através de um *middleware* intitulado BigBoxx. O ambiente em nuvem foi projetado com foco em oferecer suporte à plataforma e-Cattle, permitindo a integração com a infraestrutura existente. Para alcançar a sincronização de dados de maneira eficiente, foi adotada a arquitetura de *multi-tenant*, com o objetivo de isolar os dados e serviços de cada propriedade rural, criando um ambiente com maior segurança e escalabilidade. Ademais, esta pesquisa proporcionou a sincronização de múltiplos BigBoxx para uma fazenda, uma atualização importante para o e-Cattle. O ambiente em nuvem desenvolvido demonstrou ser uma solução para a sincronização de dados de uma plataforma IoT já existente. A adoção de tecnologias modernas, escaláveis e *multi-tenant* possibilitou uma gestão otimizada dos recursos, resultando em um ambiente preparado para lidar com o crescente volume de dados gerado pelos múltiplos dispositivos presentes no e-Cattle.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

API *Application Programming Interface*

BSD Berkeley Software Distribution

EMBRAPA Empresa Brasileira de Pesquisa Agropecuária

FACOM Faculdade de Computação

GUI *Graphical User Interface*

HTTP *Hypertext Transfer Protocol*

IoT *Internet of Things*

JSON *JavaScript Object Notation*

JWT *JSON Web Token*

LTS *Long-term Support*

NoSQL *Not Only SQL*

NPM *Node Package Manager*

PIN *Personal Identification Number*

SaaS *Software-as-a-Service*

SMTP *Simple Mail Transfer Protocol*

TI Tecnologia da Informação

UFMS Fundação Universidade Federal de Mato Grosso do Sul

VM *Virtual Machine*

XML *Extensible Markup Language*

YAML *YAML Ain't Markup Language*

CAPÍTULO

1

Introdução

A Tecnologia da Informação (TI), aliada à gestão do conhecimento, tem exercido um papel de influência positiva no desempenho, competitividade, planejamento e inovação nas instituições modernas (Ramos et al., 2020). É necessário que as principais atividades econômicas do país utilizem estrategicamente a TI para otimizar as tomadas de decisões. A pecuária, como atividade de destaque no setor econômico, pode se beneficiar da TI, uma vez que a utilização de tecnologias pode melhorar os processos de gestão e tomada de decisão, aumentando a eficiência em todas as etapas da cadeia produtiva (JUNIOR, 2020). No Brasil, a pecuária de corte tem desempenhado um papel fundamental na economia. Em 2022, o país foi o maior exportador de carne bovina do mundo. E no ano de 2023, estima-se que possua cerca de 202 milhões de cabeças de gado, representando 12,18% do rebanho mundial (ABIEC, 2023).

A Internet das Coisas, do inglês *Internet of Things* (IoT), aliada à nuvem é uma das tecnologias que auxiliaria a pecuária digital. Estima-se que no ano de 2030 tenhamos mais de 29 bilhões de dispositivos IoT (Vailshery, 2022). Samuel and Sipes (2019) definem IoT como a coleção de dispositivos conectados entre si e com serviços, utilizando padrões de protocolos de comunicação para compartilhamento de dados.

Segundo Madakam et al. (2015), existem inúmeras utilidades de IoT que estão presentes em diferentes domínios e tornam nossas vidas mais simples e confortáveis, mesmo utilizando diversas tecnologias e aplicações.

Com a sólida adoção ao paradigma de IoT criou-se soluções inovadoras de problemas previamente conhecidos, como mensuração de dados climáticos e acompanhamento do desenvolvimento na criação animal. No entanto, surgiram inúmeros desafios, com destaque para as áreas de segurança, integração

de múltiplas plataformas e padrões de arquiteturas de *hardware* (Dudhe et al., 2017).

1.1 Motivação

No Brasil, segundo Nery and Britto (2022), cerca de 74,7% das propriedades em áreas rurais possuem acesso à internet, um crescimento de 16,9% entre 2019 e 2021. Esse aumento se deve pelo aumento da oferta de serviços de internet como Starlink, contribuindo também com a redução de custos.

Em parceria com a Empresa Brasileira de Pesquisa Agropecuária (EMBRAPA), a Faculdade de Computação (FACOM) da Fundação Universidade Federal de Mato Grosso do Sul (UFMS) tem desenvolvido diversos trabalhos relacionados a pecuária de precisão, com destaque ao projeto intitulado e-Cattle, que consiste numa plataforma de agregação de dados oriundos dos dispositivos desenvolvidos ao longo dos anos dessa cooperação entre as instituições.

O e-Cattle é uma plataforma IoT voltada à pecuária de precisão que visa agregar semanticamente e centralizar os dados coletados pelos dispositivos sensoriais, através de um *middleware* denominado BigBoxx (Carromeu, 2019). O BigBoxx é alocado fisicamente na propriedade rural e disponibiliza as informações aos usuários somente *in loco*. Criando assim, a necessidade em disponibilizar essas informações em nuvem, permitindo o acesso e gerenciamento remoto aos proprietários e administradores. No estágio atual da plataforma, cada BigBoxx está vinculado a exatamente uma propriedade, dificultando a implantação da plataforma em propriedades com extensa área territorial.

Esta pesquisa propõe mitigar o desafio de gerenciamento remoto das informações coletadas pela plataforma *e-Cattle*, sincronizando os dados em nuvem de forma eficiente, permitindo também o vínculo de mais de um BigBoxx por propriedade, e ainda o acesso por vários usuários, com permissões definidas através de seus papéis no sistema.

1.2 Objetivos

O objetivo geral deste trabalho foi desenvolver um ambiente em nuvem para a plataforma e-Cattle utilizando a arquitetura *Multi-tenant*. A solução implementada garante o acesso adequado aos envolvidos, por meio da implementação de papéis de usuários nos sistemas. Com essa solução, será possível otimizar a gestão dos dados da plataforma e-Cattle, permitindo um controle centralizado e facilitando o monitoramento remoto da propriedade.

Objetivos específicos:

- Criação de uma aplicação gestora *web* para o gerenciamento remoto das propriedades e gestores da plataforma e-Cattle;
- Criação de um portal *web* para o monitoramento remoto de propriedades e gerenciamento de múltiplos *middlewares* BigBoxx;
- Implementação do ambiente em nuvem para a plataforma e-Cattle, transformando e disponibilizando as aplicações desenvolvidas em imagens de *container*;
- Desenvolvimento de mecanismos para realizar a instanciação parametrizada de *containers*, criando um ambiente isolado para cada propriedade utilizando a arquitetura *multi-tenant*.

1.3 Organização

A estrutura deste trabalho é organizada da seguinte forma: no Capítulo 2, são apresentados os conceitos sobre o desenvolvimento de *software* baseado em *container* e a arquitetura de desenvolvimento *multi-tenant*, os trabalhos correlatos e a plataforma e-Cattle. No Capítulo 3, é mostrado o principal objeto desenvolvido neste trabalho, o ambiente em nuvem do e-Cattle, que é composto pelas aplicações *Cloud* API, Gestor de Inquilinos, Portal *Web* e *Scheduler Job*. O Capítulo 4 apresenta os principais resultados alcançados por esta pesquisa, os artefatos que compõe o ambiente em nuvem do e-Cattle. Por fim, no Capítulo 5, é realizada a conclusão do trabalho, indicando os principais resultados, limitações e sugestões de trabalhos futuros.

CAPÍTULO

2

Fundamentação Teórica

Este capítulo tem como objetivo apresentar uma revisão bibliográfica das tecnologias utilizadas no desenvolvimento deste trabalho. Iniciando pela definição e utilização de *containers* na Seção 2.1, em seguida são apresentados a arquitetura *multi-tenant* e trabalhos relacionados, na Seção 2.2 e Seção 2.3 respectivamente, e por fim é detalhada a arquitetura da plataforma e-Cattle na Seção 2.4.

2.1 Desenvolvimento baseado em container

Esta seção apresenta as definições de conceitos básicos de *containers* juntamente com a plataforma Docker[1], em seguida, a Subseção 2.1.3 aborda o tema orquestração de *containers*, mostrando sua finalidade e realizando um comparativo entre as duas principais ferramentas de orquestração disponíveis no mercado.

2.1.1 Container

Os *containers* empacotam aplicações, arquivos e códigos-fonte em um único ambiente. Eles se utilizam dos recursos do sistema operacional hospedeiro, denominado *host*, e são executados como processos isolados, garantindo assim confiabilidade e estabilidade, independente do ambiente de execução (Marathe et al., 2019).

Uma imagem de *container* é um pacote de *software* executável, que se torna um *container* durante sua execução. A imagem contém todos os artefatos

[1] https://www.docker.com/

necessários para a execução da aplicação, como bibliotecas e configurações (Docker, 2023b).

Modak et al. (2018) afirmam que pelo fato do *container* isolar a aplicação do ambiente do *host*, os conflitos na mesma infraestrutura são minimizados, pois, mesmo que existam diversas aplicações sendo executadas em um único *host* cada uma possui seu ambiente isolado, sem interferir uma à outra.

2.1.2 Docker

A plataforma Docker oferece a capacidade de empacotar e executar aplicações num ambiente isolado, chamado *container*. Através do isolamento e segurança é possível executar diversos *containers* simultaneamente no mesmo *host* (Docker, 2023a).

O Docker agiliza o desenvolvimento, permitindo que os desenvolvedores criem e testem aplicativos em *containers* isolados. A capacidade de empacotar todas as dependências em um *container* torna a configuração mais fácil e eficiente (Poulton, 2020).

Uma característica importante do Docker, é que os *containers* são portáteis, garantindo que a aplicação funcione da mesma forma em qualquer *host* Docker (2023b).

2.1.3 Orquestração de Container

O gerenciamento de múltiplos serviços utilizando diversos *containers* é chamado de orquestração de *containers*. Atualmente, as principais ferramentas disponíveis para realizar a orquestração são Kubernetes[2] e Docker Swarm[3] , ambas *open source* e com funções similares (Modak et al., 2018).

Kubernetes é uma plataforma de código aberto, flexível e expansível para o controle de tarefas e serviços distribuídos em *containers*, que realiza a automação de implantação e escalonamento de aplicativos em *containers*. Ademais, a plataforma é derivada de outro projeto intitulado Borg, criado e mantido pela Google (Kubernetes, 2023; Li et al., 2021).

O Docker Swarm é um modo disponibilizado em toda instalação Docker, no entanto, por padrão este modo é desabilitado já que seu foco é ser empregado apenas em *clusters*. A principal vantagem do Docker Swarm é a integração com o Docker, permitindo criar um agrupamento dos sistemas e o compartilhamento de informações entre os *containers* (Jutadhamakorn et al., 2017).

A arquitetura do Docker Swarm possui dois tipos de nós, os *managers* e *workers*. O nó *manager* é o responsável por agendar e distribuir as tarefas

[2] https://kubernetes.io
[3] https://docs.docker.com/engine/swarm/

para os nós *workers*. Uma aplicação rodando em um *container* é conhecida por *service* e o conjunto de *services* é chamado de *stack* (SUSE, 2019).

A Tabela 2.1 apresenta uma breve comparação das principais características de ferramentas de orquestração de *containers*.

Característica	Docker Swarm	Kubernetes
Instalação e Configuração	Instalação facilitada	Instalação complexa
Interface Gráfica	Não possui *Graphical User Interface* (GUI) nativamente, somente com aplicações de terceiros	Possui GUI chamada Kubernetes Dashboard
Escalabilidade	Altamente escalável e escalabilidade cinco vezes mais rápida que kubernetes	Altamente Escalável e escalabilidade otimizada
Escalonamento automático	Não oferece escalonamento automático	Possui escalonamento automático
Balanceamento de carga	Realiza o balanceamento de carga automático de tráfego entre contêineres no cluster	É necessária a intervenção manual para realizar o balanceamento de carga de tráfego entre diferentes containers em diferentes nós
Atualizações contínuas e reversões	Permite atualizações contínuas, mas não realiza reversões automáticas	Permite atualizações contínuas e faz reversões automáticas
Volumes de Dados	Pode compartilhar volumes com qualquer outro container	Pode compartilhar os volumes apenas com outros containers no mesmo nó
Log e Monitoramento	Não possui logs e monitoramento otimizados, mas é possível utilizar aplicações de terceiros	Ferramentas integradas para registro e monitoramento

Tabela 2.1: Comparativo entre Docker Swarm e Kubernetes.
Fonte: Adaptada de Shah and Dubaria (2019)

A seguir são detalhadas as características utilizadas no comparativo entre Docker Swarm e Kubernetes:

- **Instalação e Configuração** aborda a facilidade e padronização de instalação em diferentes sistemas operacionais (Rosen, 2022). O Docker Swarm possui a vantagem de ser utilizado em qualquer *host* que já possua o Docker instalado, pois, ele é instalado por padrão com o Docker.

- **Interface Gráfica** permite aos usuários utilizarem o *software* de forma interativa, sem a necessidade de *scripts* ou comandos em terminais, proporcionando praticidade e eficiência em sua execução (Watabe et al., 2023).

- **Escalabilidade** é a capacidade de um sistema de expandir ou reduzir recursos computacionais em resposta à demandas e continuar operando e atendendo aos requisitos dos usuários (Reznik et al., 2019). É possível aumentar ou diminuir os recursos horizontalmente, adicionando *containers* ou *Virtual Machine* (VM) ao ambiente existente, ou ainda, verticalmente, realizando melhorias nos servidores que já compõe a infraestrutura (Millnert and Eker, 2020).

- **Escalonamento Automático** consiste em escalonar dinamicamente a infraestrutura em resposta às alterações na carga de trabalho, como quantidade de dados de entrada ou mudanças nos estados dos artefatos do

ambiente em nuvem, como falha de nó ou de rede (Khazaei et al., 2017). Segundo Zhao et al. (2019), a habilidade da plataforma em nuvem escalonar automaticamente sua infraestrutura diante de cargas dinâmicas tem um impacto direto no desempenho das aplicações implantadas.

- **Balanceamento de carga** tem como objetivo otimizar recursos, minimizando o tempo de resposta. Sua utilização resulta em um melhor gerenciamento e distribuição da carga de trabalho, as políticas de balanceamento de carga são aplicadas quando há uma disparidade na carga, distribuindo assim a carga de trabalho pela rede (Narale and Butey, 2018). Além disso, conforme Sree Devi et al. (2023), o balanceamento de carga ajuda a manter o tráfego da aplicação *web* estável graças à sua escalabilidade, lidando com picos de tráfego de forma coordenada.

- **Atualizações contínuas e reversões** indicam a capacidade de adotar atualizações e reverter caso ocorra alguma falha. Conforme Ayres et al. (2019), a atualização contínua permite que novas versões das aplicações possam ser integradas em *containers*, e a versão antiga pode ser desativada, evitando tempo de inatividade do subsistema. Já a reversão é a capacidade de restaurar o sistema a partir de um estado salvo anteriormente, em vez do estado inicial da infraestrutura, o que reduz significativamente a perda de dados. Esse recurso permite que as aplicações sejam restauradas imediatamente em sua capacidade total no caso de falhas, principalmente falhas de servidor (Cui et al., 2016).

- **Volumes de dados** permite o compartilhamento de diretórios entre diferentes *containers* que estão sendo executados na mesma máquina física, ou ainda, pode se tratar do compartilhamento de volumes entre vários dispositivos físicos (Bachiega et al., 2020).

- **Log e monitoramento** permitem que os administradores do sistema analisem transações ou detectem atividades suspeitas nas aplicações, podendo replicar essas ações posteriormente (Maneenual and Vasupongayya, 2018). Locke et al. (2022) indicam que os *logs* possuem informações valiosas sobre o sistema e que sua análise pode ser utilizada para a compreensão da aplicação, implicando na melhoria contínua do *software*.

Diante das características apresentadas, nota-se a solidez de ambos orquestradores, sendo indicado o kubernetes para projetos mais robustos, com maiores cargas de trabalhos e necessidades diversas. Já para projetos que necessitem de orquestração, mas com uma curva de aprendizagem baixa, e

com possibilidade de adição de ferramentas de terceiros ao longo do tempo, o recomendado é o Docker Swarm.

2.2 Multi-tenant

Atualmente, a maioria dos *softwares* desenvolvidos utiliza a arquitetura três camadas, constituída por três elementos: *frontend*, *backend* e armazenamento de dados, conforme mostra Figura 2.1 (Fowler, 2017). As solicitações são realizadas para a aplicação através do *frontend*; já o *backend* é responsável pelo trabalho com os dados que são armazenados, seja temporariamente na memória ou na base de dados.

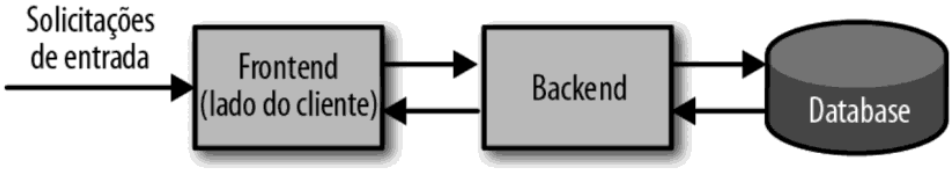

Figura 2.1: Arquitetura três camadas
Fonte: Fowler (2017)

A abordagem de desenvolvimento de software que é frequentemente implantada e utiliza a arquitetura de três camadas é a de aplicações monolíticas, que consiste em reunir toda a lógica de programação em um único pacote de aplicativos, que é então distribuído de forma integrada, sendo chamado de monólito (Fowler, 2017). Ao atualizar esse pacote, é possível que a atualização seja bem-sucedida, resultando no funcionamento correto de toda a aplicação. No entanto, a outra possibilidade é que a atualização falhe, o que significa que a aplicação não pode ser executada corretamente ou, até mesmo, não pode ser utilizada. Uma solução para resolver esse problema é dividir o pacote de aplicativos em pequenas aplicações, conhecida como microsserviços (Prasandy et al., 2020). Fowler (2017) define o conceito de microsserviço como uma aplicação pequena e que executa apenas uma única tarefa com eficiência.

Ainda segundo Prasandy et al. (2020), utilizar microsserviços permite maior flexibilidade e escalabilidade, deixando o sistema modular e facilitando o gerenciamento através das aplicações individuais. Conforme a Figura 2.2, nota-se que a arquitetura de três camadas ainda é utilizada na abordagem de microsserviços, no entanto, com a particularidade de ser organizada nos diversos serviços disponíveis na infraestrutura.

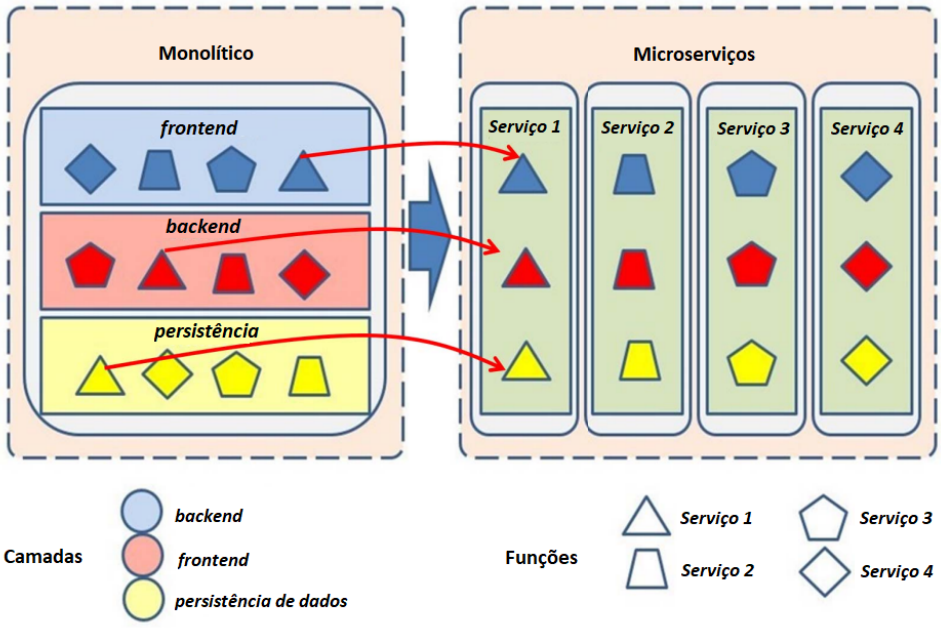

Figura 2.2: Abstração de sistema monolítico e microsserviços
Fonte: Adaptada de Velepucha and Flores (2021)

Além dos microsserviços, existe a possibilidade de utilizar uma instância de *software* que atenda diversos clientes ou sistemas, uma isolada da outra, essa abordagem é chamada *multi-tenant* (Hat, 2020). A arquitetura *multi-tenant* é utilizada em sistemas *Software-as-a-Service* (SaaS), pois cada item armazenado na infraestrutura do sistema é gerenciado como um inquilino (Tsai et al., 2016). Ademais, a arquitetura permite a alocação de diversos inquilinos que são criados como instâncias de um módulo do *software*. Conforme mostrado na Figura 2.3, cada inquilino pode interagir com a aplicação como se fosse o único usuário dela, onde nenhum inquilino pode acessar ou visualizar os dados de outro, pois os inquilinos são isolados uns dos outros (Mietzner et al., 2008; Espadas et al., 2013).

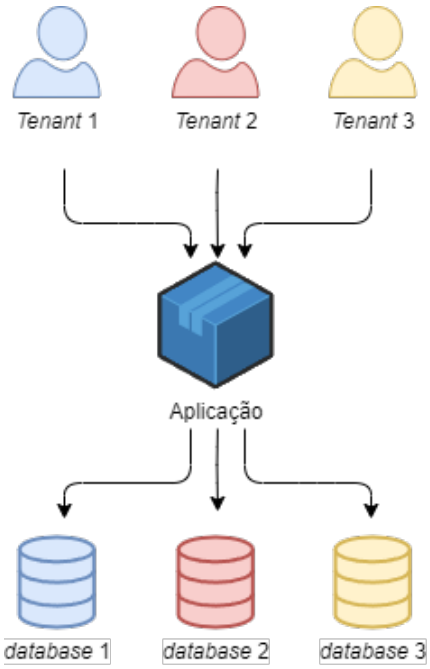

Figura 2.3: Arquitetura *multi-tenant*
Fonte: Elaborada pelo autor (2023)

As vantagens de se criar um sistema *multi-tenant* vão desde a segurança até os custos operacionais e de desenvolvimento (Kuppusamy et al., 2015). Buyya and Srirama (2019) afirmam que o conceito de *multi-tenant* pode otimizar a integração de dispositivos IoT com a nuvem, conhecida como *edge* ou névoa, pois, através do uso de *container* é possível estabelecer uma divisão de carga de trabalho e poder computacional entre os dispositivos e plataformas envolvidas.

2.3 Trabalhos Relacionados

Nesta subseção são mostrados os trabalhos correlatos, buscando explorar abordagens utilizadas em projetos similares, com foco em IoT, *multi-tenant* e ambiente em nuvem. Ao analisar esses trabalhos, buscamos identificar oportunidades de aprimoramento em nossa pesquisa, visando desenvolver o ambiente em nuvem para a gestão de dados na plataforma e-Cattle.

No contexto da agricultura de precisão, Tan and Wortman (2014) projetaram um sistema de análise da eficiência do rendimento da colheita em pomares. Esse sistema utiliza uma plataforma de computação baseada em nuvem para adquirir e analisar dados de rendimento. Após a colheita, os dados são enviados para um servidor através de uma balança digital. No entanto, uma grande preocupação apontada entre os usuários de sistemas de informação agrícola baseados em nuvem é a privacidade dos dados. Para enfrentar esse desafio, os autores desenvolveram uma arquitetura de *software multi-tenant* para o seu sistema.

Conforme Figura 2.4, nessa arquitetura, o banco de dados é organizado em dois conjuntos: (a) dados globais que compreendem tabelas de dados acessíveis a todos os usuários, de acordo com o acesso apropriado; e (b) dados específicos do produtor, compreendendo tabelas de dados que são acessíveis apenas a um produtor específico.

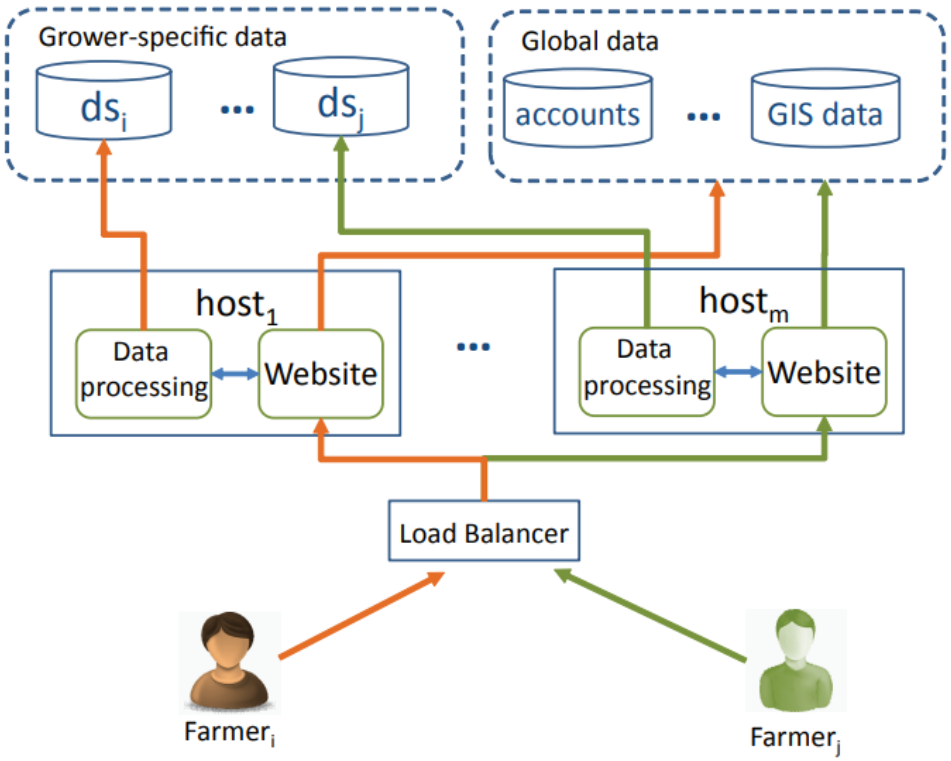

Figura 2.4: Exemplo de Arquitetura *multi-tenant* para agricultura
Fonte: Tan and Wortman (2014)

Segundo os autores, a arquitetura de *software multi-tenant* implementada traz benefícios, tais como: aproveitar a computação em nuvem para uma melhor escalabilidade, permitindo que a quantidade de *hosts* seja alterada sob demanda, com base na quantidade de proprietários; melhorar a manutenção do sistema, uma vez que todos os *hosts* estão executando exatamente o mesmo aplicativo, exigindo apenas a manutenção de uma única base de código para o aplicativo; e, por fim, aprimorar a privacidade dos dados, pois os dados de produção e outras operações fazem parte dos dados específicos do produtor, acessíveis apenas a um produtor específico.

Zhengtao Liu and Xia (2020) e Naji Saif et al. (2023) destacam em seus trabalhos abordagens para o controle de acesso em ambientes *multi-tenant*, através de relações de confiança entre os *tenants* e garantindo o acesso adequado, ao mesmo tempo em que se protege contra ataques maliciosos. Os autores destacam a importância crítica do controle de acesso em ambiente *multi-tenant*, uma vez que diferentes *tenants* podem compartilhar recursos, mas precisam ser isolados para proteger a confidencialidade e a integridade

dos dados. As abordagens propostas podem melhorar a governança de acesso, reduzir o risco de vazamento de informações e garantir que cada *tenant* tenha acesso apenas aos recursos e serviços para os quais possui permissão. A implementação efetiva do controle de acesso em ambientes *multi-tenant* pode aumentar a confiança dos usuários e empresas na computação em nuvem, impulsionando a adoção dessa tecnologia.

Em Masmoudi et al. (2019), os autores também se preocupam com a responsabilidade e proteção de dados em ambientes de computação em nuvem *multi-tenant*. A proposta apresentada visa garantir a responsabilidade dos serviços em nuvem, permitindo a verificação de conformidade e detecção de violações de responsabilidade. A abordagem proposta permite que *tenants* utilizem serviços de prestação de contas sem serem afetados por outros. Isso é de suma importância, pois muitas empresas e organizações compartilham recursos em nuvem e é essencial garantir que os dados de cada *tenant* sejam protegidos e não acessem informações confidenciais de outros *tenants* do ambiente. A abordagem proposta pode melhorar consideravelmente a segurança e a confiança em ambientes *multi-tenant*, promovendo a adoção da computação em nuvem.

Sellami et al. (2020) e Batista et al. (2022) apresentam arquiteturas de microsserviços *multi-tenant*, oferecendo soluções escaláveis, confiáveis e eficientes para empresas e provedores de serviços em nuvem. As arquiteturas propostas visam promover a flexibilidade e a facilidade de implantação de serviços em um ambiente *multi-tenant*. A arquitetura de microsserviços é amplamente adotada no desenvolvimento de aplicativos modernos, e a sua adaptação à ambientes *multi-tenant* é essencial para garantir a entrega ágil de serviços e a redução de custos operacionais. Essas abordagens de arquitetura podem beneficiar provedores de nuvem e empresas que buscam oferecer serviços de alta qualidade e resposta rápida aos seus clientes, além de melhorar a manutenção e atualização das aplicações.

É importante garantir a eficiência no escalonamento de *containers* em ambientes *multi-tenant*, mantendo baixa latência e alta utilização de recursos. A eficiência e o escalonamento adequado são fundamentais para garantir que os recursos em nuvem sejam utilizados de forma otimizada, evitando desperdícios. A abordagem proposta pode melhorar a capacidade de resposta dos serviços em nuvem, especialmente em períodos de alto tráfego, evitando a sobrecarga de servidores e garantindo a disponibilidade contínua dos serviços. Além disso, o eficiente escalonamento de *containers* também pode resultar em redução de custos para os provedores de nuvem, tornando seus serviços mais competitivos no mercado. Portanto, a eficiência e o escalonamento em nuvem *multi-tenant* têm um impacto significativo na qualidade dos serviços e na

economia de recursos, beneficiando tanto os provedores em nuvem quanto os *tenants* de serviços (Naji Saif et al., 2023).

Nesta subseção foram apresentados os principais trabalhos relacionados, que enfatizam a importância da escalabilidade e *multi-tenant* para lidar com o aumento no volume de dados gerado pelos dispositivos. Ademais, reforçam a importância da segurança dos dados armazenados em nuvem dentro da arquitetura *multi-tenant*.

2.4 Plataforma e-Cattle

A parceria entre EMBRAPA e FACOM proporcionou a criação de diversas soluções IoT de coleta de dados para pecuária de precisão, no entanto, essas pesquisas foram desenvolvidas isoladamente e não possuíam integração. Assim, Carromeu (2019) desenvolveu uma plataforma IoT denominada e-Cattle[4], responsável por agregar todos esses dados coletados de diferentes dispositivos e disponibilizar através de uma interface amigável e intuitiva ao usuário, permitindo um monitoramento da propriedade mais preciso com informações atualizadas. O e-Cattle consiste em um *gateway* IoT, um dispositivo implantado na borda da rede que integra outros dispositivos IoT heterogêneos (Krylovskiy, 2015).

Atualmente, o e-Cattle possui sua arquitetura definida conforme Figura 2.5 e encontra-se com todas as camadas implementadas, com exceção dos componentes de sincronização em nuvem, objeto de pesquisa deste trabalho.

[4] https://github.com/e-cattle

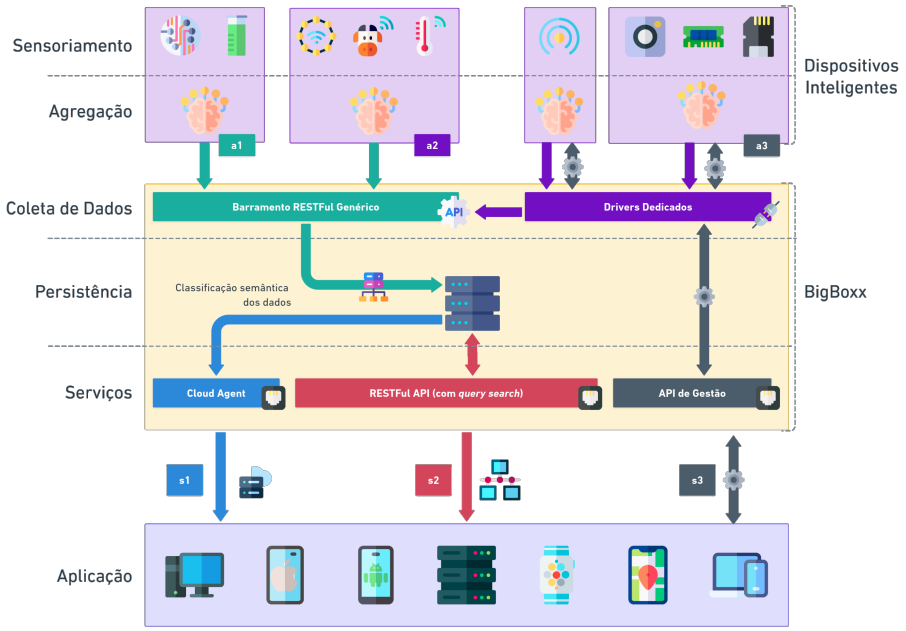

Figura 2.5: Arquitetura do e-Cattle
Fonte: Carromeu (2019)

A arquitetura da plataforma e-Cattle representada pela Figura 2.5 é dividida em seis camadas, são elas:

- **Camada de Sensores** consiste em artefatos de *hardware* e *software* responsáveis pela coleta de dados. Nessa camada encontram-se as diferentes soluções condicionadas a parceria EMBRAPA e FACOM, portanto, não há uma padronização de formato ou frequência das informações coletadas;

- **Camada de Agregação** é responsável pelo envio dos dados coletados na camada anterior, seja por meio de *software* ou *hardware*. As camadas de **Sensores** e **Agregação** formam os **Objetos Inteligentes** da arquitetura;

- **Camada de Coleta de Dados** é composta por um barramento de serviços *RESTful* responsável por receber requisições com os dados oriundos dos **Objetos Inteligentes**;

- **Camada de persistência** é responsável pelo armazenamento dos dados recebidos da camada de **Coleta de Dados**, a persistência é realizada em banco de dados *Not Only SQL* (NoSQL), após uma organização semântica;

- **Camada de Serviços** disponibiliza os dados parametrizados às aplicações de alto nível da camada de **Aplicação** através de uma *Application Programming Interface* (API) *GraphQL*. A junção das camadas **Coleta de**

Dados, **Persistência** e **Serviços** são implementadas no *middleware* BigBoxx, implantado com o *hardware* Raspberry Pi, conhecido por ser um computador de placa única de baixo custo e pouco consumo de energia. Os dados mostrados ao proprietário são dispostos através de interface gráfica no formato de *dashboard*;

- **Camada de Aplicação** é composta pelas aplicações de alto nível que consomem os dados disponibilizados pela camada anterior de **Serviços**. Essa camada é utilizada para tomada de decisão estratégica na propriedade.

A plataforma e-Cattle foi construída com tecnologias modernas e consolidadas, garantindo robustez dentro do paradigma de *software* livre (Carromeu, 2019). Para o armazenamento dos dados locais e em nuvem, foi adotado o banco de dados não-relacional MongoDB, pois, conforme Bicevska and Oditis (2017) os bancos NoSQL são associados ao desenvolvimento flexível, altas taxas de leitura e escrita e escaláveis em larga escala. As aplicações *web* foram implementadas utilizando NodeJS, um ambiente JavaScript *server-side*, focado em performance e consumo baixo de memória, facilitando assim o desenvolvimento conhecido como *full stack*, portanto, com apenas uma linguagem de programação é possível criar sistemas que executem tanto do lado do cliente quanto do lado do servidor (Tilkov and Vinoski, 2010). Além disso, o *framework* VueJS é o responsável pelo *frontend*. O transporte de dados é realizado através de APIs criadas com o *framework web* Express. Já o padrão de formato na transferência de dados é o *JavaScript Object Notation* (JSON), que segundo Cheong (2019) utiliza o conceito de chave-valor e é uma alternativa mais amigável ao *Extensible Markup Language* (XML). As consultas personalizadas de dados no BigBoxx são realizadas através do *GraphQL*, uma linguagem de consulta em tempo de execução Cha et al. (2020).

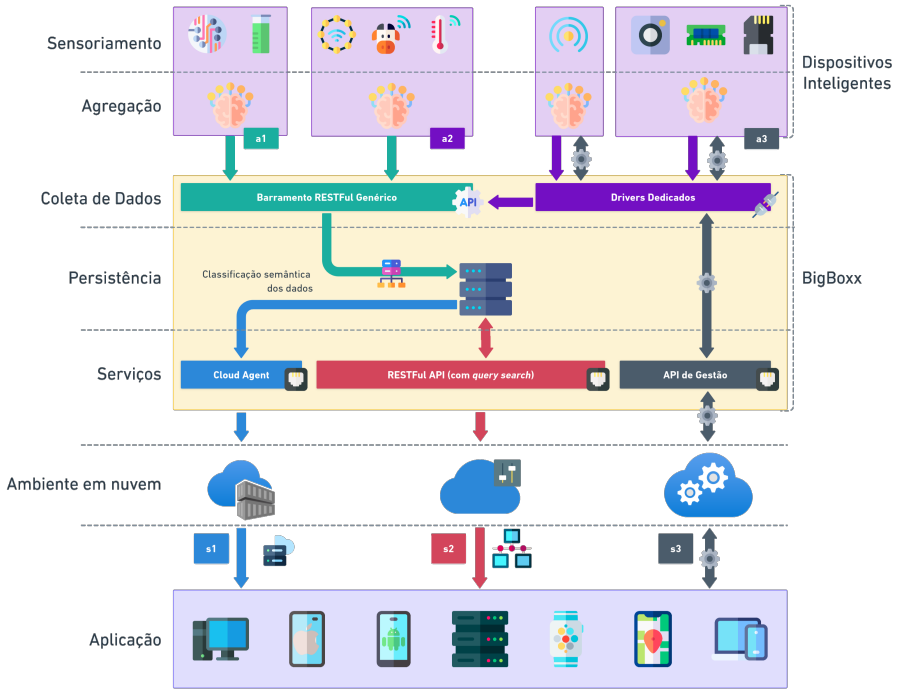

Figura 2.6: Ambiente em nuvem dentro da arquitetura do e-Cattle
Fonte: Adaptada de (Carromeu, 2019)

O presente trabalho implementou a camada intitulada “Ambiente em Nuvem” para a persistência em nuvem dos dados armazenados localmente no *middleware* BigBoxx, possibilitando o gerenciamento remoto das propriedades. Conforme destacado na Figura 2.6, o ambiente em nuvem foi implementado entre as camadas de **Serviços** e **Aplicação** da plataforma e-Cattle, com sistemas orquestrados em *containers* através do Docker Swarm.

CAPÍTULO

3

Metodologia

Neste capítulo, é apresentado o processo utilizado para realizar a pesquisa, incluindo os procedimentos e ferramentas adotadas. Na Seção 3.1, é mostrado um panorama geral do ambiente em nuvem do e-Cattle, seguida pela Seção 3.2, que apresenta a aplicação *Cloud* API. A Seção 3.3 aborda a aplicação *web* gestora de inquilinos. Na Seção 3.4, é apresentada a aplicação *web* utilizada pelos produtores rurais, finalizando com a Seção 3.5 responsável pela criação das *stacks* de cada propriedade rural.

3.1 Visão geral do ambiente em nuvem do e-Cattle

Atualmente, a plataforma e-Cattle utiliza um e somente um BigBoxx para cada propriedade rural, o que dificulta a implantação em uma fazenda com extensa área territorial, com grande volume de dados sensoriais coletados, e pode ter o espaço de armazenamento esgotado rapidamente.

Diante disso, propomos um ambiente em nuvem para a plataforma e-Cattle, que permite a sincronização em nuvem das informações armazenadas localmente no *middleware* BigBoxx. O ambiente possibilita também a utilização de múltiplos BigBoxx numa mesma propriedade.

Para implementar o ambiente em nuvem do e-Cattle[1], optamos pela arquitetura *multi-tenant* juntamente com o Docker Swarm para o gerenciamento dos múltiplos *containers*.

Conforme Figura 3.1, o ambiente em nuvem do e-Cattle, possui diversos serviços organizados numa *stack*, dentre eles destacamos a aplicação Gestor

[1] https://github.com/e-cattle/swarm

de Inquilinos, que permite criar as fazendas que estarão aptas a receberem dados de *middlewares* BigBoxx. Para tal, por meio da aplicação *Scheduler Job*, é instanciada uma *stack* parametrizada com dados da propriedade, como código identificador e nome, além das portas que serão expostas para os serviços desta fazenda. Outra aplicação que compõem a *stack* do ambiente em nuvem consiste no Portal *Web*, que permite gerenciar o vínculo dos *middlewares* BigBoxx à *stack* de uma propriedade, por meio da aplicação *Cloud* API, detalhada na Seção 3.2.

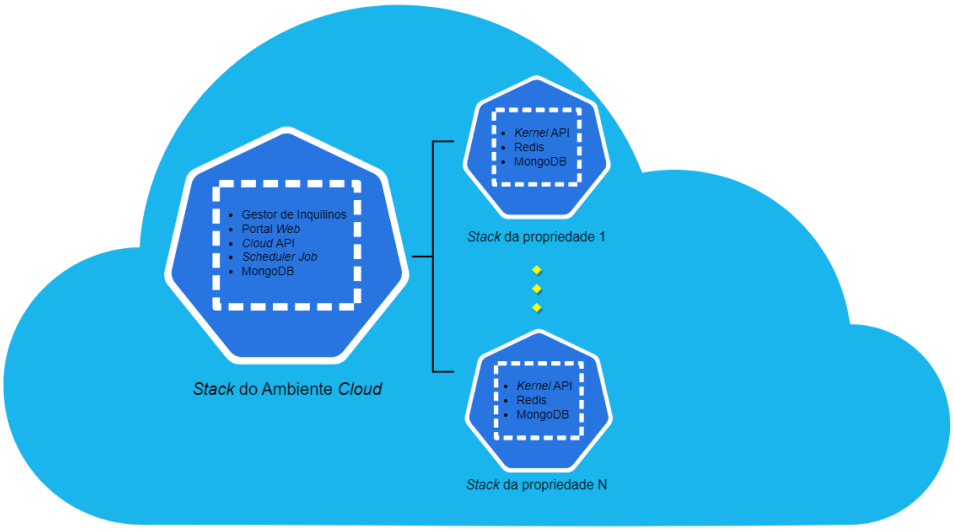

Figura 3.1: Arquitetura do ambiente em nuvem do e-Cattle
Fonte: Elaborada pelo autor (2023)

Cada propriedade rural cadastrada no ambiente em nuvem possui seu conjunto de *containers* responsável por armazenar suas informações e disponibilizar seus serviços, isolado das *stacks* de outras propriedades, conforme mostrado na Figura 3.2. Além disso, é possível observar que quando as aplicações *web* requisitam dados de uma propriedade específica, elas interagem diretamente com a *stack* da propriedade envolvida, aplicando a abordagem de *multi-tenant*.

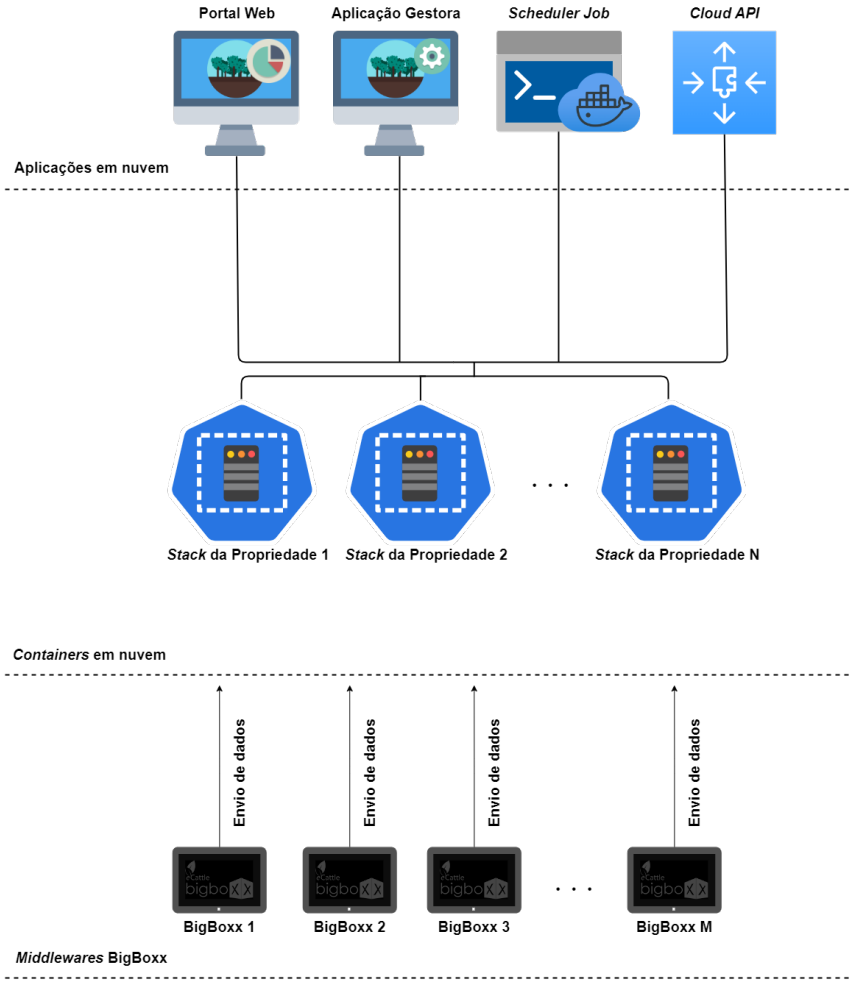

Figura 3.2: Arquitetura do sincronismo em nuvem do e-Cattle
Fonte: Elaborada pelo autor (2023)

A Figura 3.3 mostra o passo a passo da sincronização no ambiente em nuvem do e-Cattle, objeto deste trabalho. Na primeira etapa, o Gestor da propriedade indica no *middleware* BigBoxx que deseja realizar a sincronização na nuvem, em seguida, na aplicação Portal *Web*, esse pedido é aprovado pelo usuário que possui papel de *manager* ou *owner*, detalhados na Seção 3.3. Após a aprovação, o BigBoxx está apto a enviar seus dados diretamente para a *stack* específica da propriedade.

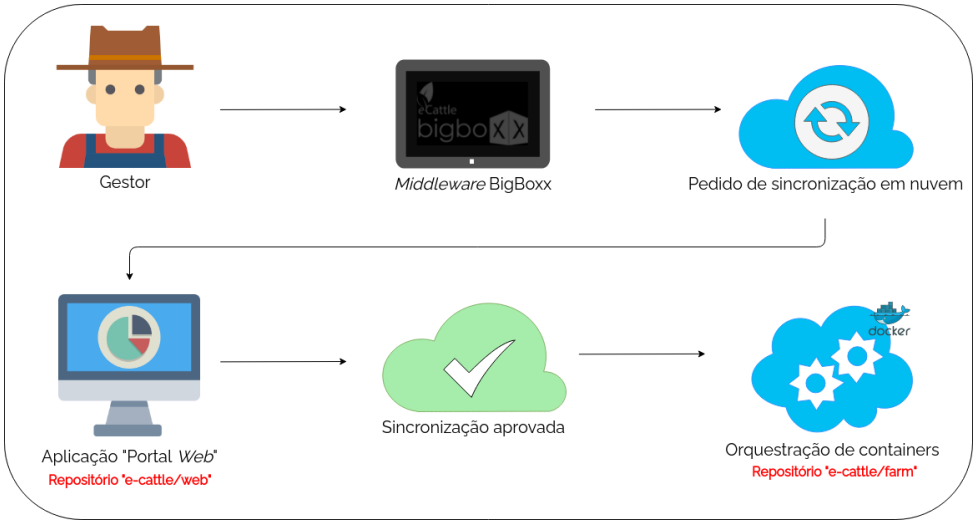

Figura 3.3: Fluxograma da sincronização no ambiente em nuvem do e-Cattle
Fonte: Elaborada pelo autor (2023)

3.2 Aplicação Cloud API

O ambiente em nuvem do e-Cattle possui uma aplicação intitulada *Cloud* API[2], responsável por fornecer os *endpoints* de gerenciamento de propriedades, usuários e vínculos de *middlewares* BigBoxx. As interações das aplicações e dispositivos com a *Cloud* API são apresentadas na Tabela 3.1.

Dispositivo ou Aplicação	**Atividade realizada via *Cloud* API**
BigBoxx	*Middleware* da propriedade que solicita e envia os dados armazenados nele para a nuvem, mediante aprovação feita no Portal Web
Gestão de Inquilinos	Aplicação web em NodeJS que permite criar propriedades e gestores na nuvem
Portal Web	Aplicação web em NodeJS que permite a aprovação do envio dos dados locais do BigBoxx para a nuvem
Scheduler Job	Aplicação NodeJS de tarefas agendadas, responsável por criar a *stack* de uma nova propriedade na nuvem

Tabela 3.1: Aplicações e Dispositivos que consomem a *Cloud* API
Fonte: Elaborada pelo autor (2023)

Como medida de segurança, toda interação com a aplicação *Cloud* API é realizada mediante autenticação, garantindo assim, que somente as outras

[2] https://github.com/e-cattle/cloud

aplicações que fazem parte do ambiente em nuvem do e-Cattle tenham acesso aos *endpoints* fornecidos pela *Cloud* API. A autenticação dos dispositivos e sistemas com a *Cloud* API é realizada através de *token JSON Web Token* (JWT), que segundo Alkhulaifi and El-Alfy (2020), é um conjunto de dados alfanuméricos separados por três campos: o cabeçalho contendo informações como o algoritmo utilizado para criação do *token*; o *payload* que contém os dados que foram criptografados, e por fim o campo assinatura que possui os dados dos outros dois campos encriptados. Para gerar um *token* JWT é necessário informar uma chave de segurança no momento de sua criação, essa mesma chave é utilizada para decriptar as informações posteriormente.

3.3 Aplicação Gestor de Inquilinos

A aplicação *web* Gestor de Inquilinos[3] tem como finalidade permitir o gerenciamento de propriedades e usuários no sistema. Através dela o proprietário cadastra fazendas e vincula seus gestores. Considerando o desafio de segurança apontado por Meneghello et al. (2019) em seu estudo de vulnerabilidades exploradas em ataques reais a sistemas IoT, optou-se pela utilização de níveis de acesso e permissão, representados por papéis de usuários, conforme Tabela 3.2.

Papel	Descrição
viewer	É o papel com menor permissão no sistema, o usuário está habilitado somente a visualização dos dados da propriedade que estão sincronizados na nuvem.
manager	Possui permissão maior em relação ao ***viewer***, podendo ativar *middlewares* e vincular outros usuários de mesmo papel à propriedade.
owner	Possui maior permissão entre os três papéis, além das permissões de ***manager*** também está apto ao vínculo de usuários de qualquer papel à propriedade.

Tabela 3.2: Papéis de usuários na Aplicação Gestor de Inquilinos
Fonte: Elaborada pelo autor (2023)

Ao cadastrar uma propriedade na aplicação gestora, é atribuído a ela um identificador único, também conhecido como código da fazenda. Após o cadastro da fazenda, é criada uma *stack* de *containers* docker na nuvem, com banco de dados e serviços referentes apenas a propriedade em questão. Para iniciar o processo de sincronização dos dados do BigBoxx com o *container* de uma fazenda, é necessário informar no BigBoxx o código da fazenda em que o *middleware* será vinculado.

[3] https://github.com/e-cattle/manager

3.4 Aplicação Portal Web

O Portal *Web*[4] é o sistema que mostra as informações que foram sincronizadas entre *middlewares* BigBoxx e a nuvem, sendo estas disponibilizadas graficamente aos usuários. Ademais, é por meio dele que é aprovado o vínculo de um novo BigBoxx à propriedade, de acordo com o papel do usuário, pois, usuários com permissão de acesso do tipo *viewer* não possuem privilégio de aprovação.

Por meio deste portal *web* é possível adicionar ou remover usuários vinculados à propriedade, também mediante permissão de acesso. Pelo portal, o usuário pode visualizar ou gerenciar o contexto de todas as propriedades que ele está vinculado.

Tanto o portal *web* quanto a aplicação gestora foram desenvolvidos utilizando o conceito de responsividade, permitindo assim, uma navegação compatível com dispositivos móveis como *smartphones* e *tablets*.

3.5 Aplicação Scheduler Job

A aplicação *Scheduler Job*[5] foi implementada em NodeJS e é responsável por criar a *stack* de cada propriedade rural dentro do ambiente em nuvem do e-Cattle.

Para realizar o agendamento da rotina de verificação de novas propriedades cadastradas, a aplicação utiliza a biblioteca intitulada agenda, disponibilizada como pacote pelo *Node Package Manager* (NPM).

Conforme mostrado na Figura 3.4, a criação da *stack* é feita em duas etapas, a primeira consiste em criar um arquivo de texto no formato *YAML Ain't Markup Language* (YAML) especificando quais são os *services*, portas, imagens de *containers* e demais informações da *stack*. Em seguida, na segunda etapa, as mesmas informações inseridas no arquivo YAML são utilizadas na criação da *stack*.

A criação do arquivo YAML foi uma estratégia de *backup* adotada, para manter registrado no *host* do ambiente em nuvem do e-Cattle as propriedades já cadastradas, além disso, com o arquivo YAML é possível recriar a *stack* da fazenda com as informações já persistidas até o momento, desde que o volume de dados não tenha sido removido antes do processo.

[4] https://github.com/e-cattle/web
[5] https://github.com/e-cattle/farm

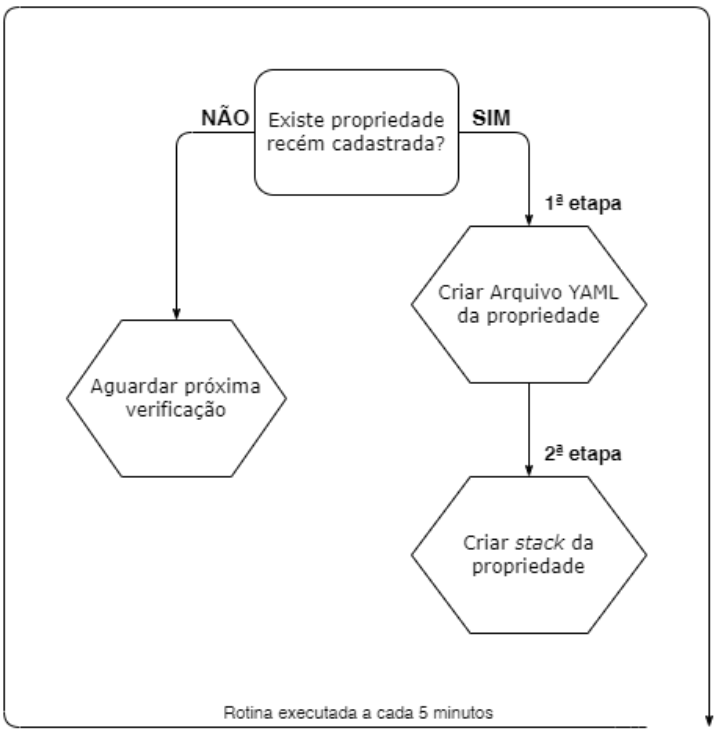

Figura 3.4: Fluxograma da rotina de criação de *stacks* das propriedades
Fonte: Elaborada pelo autor (2023)

Neste capítulo, foram apresentados os artefatos desenvolvidos nesta pesquisa, iniciando com uma visão geral abrangente do ambiente em nuvem do e-Cattle, seguida pela apresentação das aplicações Cloud API e Gestor de inquilinos. Apresentamos a aplicação *web* destinada aos produtores rurais e concluímos com o processo de criação das *stacks* específicas para cada propriedade rural.

CAPÍTULO

4

Resultados

Neste capítulo serão apresentados os resultados do desenvolvimento dos artefatos mostrados no Capítulo 3. Na Seção 4.1, abordamos o processo de desenvolvimento do ambiente em nuvem proposto, destacando as tecnologias utilizadas. Já na Seção 4.2, focamos na implantação do ambiente, detalhando os procedimentos de instalação e configuração. Por fim, na Seção 4.3, discutimos a disponibilização do ambiente em nuvem do e-Cattle para os usuários finais.

4.1 Desenvolvimento

A aplicação Gestor de Inquilinos foi implementada utilizando o VueJS[1], que consiste em um *framework JavaScript* que oferece um modelo de programação baseado em componentes que auxilia a implementação de interfaces de usuário eficientes (VueJS, 2023).

A Figura 4.1 mostra a tela de gerenciamento de propriedades. Nela é possível adicionar uma fazenda e vincular usuários já existentes no sistema. É possível também através do item "Usuários" do menu, cadastrar novo usuário e vinculá-lo em uma propriedade.

Ao cadastrar uma fazenda, é atribuído a ela um código identificador único, que é utilizado no BigBoxx para indicar qual propriedade o *middleware* será vinculado.

[1] https://vuejs.org

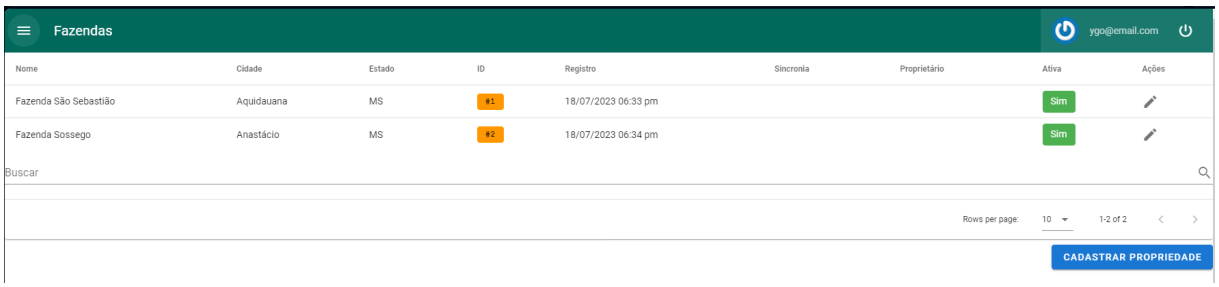

Figura 4.1: Tela de gerenciamento de propriedades na aplicação Gestor de Inquilinos

Fonte: Elaborada pelo autor (2023)

Após a propriedade ter sido cadastrada na aplicação Gestor de Inquilinos, o usuário deve ir na aba “Sincronizar” do *middleware* BigBoxx e no campo “identificador” inserir o código da fazenda informado em sua criação, após a inserção do código o usuário deve selecionar a opção “Conectar”, conforme Figura 4.2.

Figura 4.2: Tela de solicitação do sincronismo em nuvem do BigBoxx

Fonte: Elaborada pelo autor (2020)

Após o pedido de sincronização realizado no BigBoxx, é necessário que esse *middleware* tenha permissão para o envio dos dados para a nuvem; essa permissão é feita através da aplicação Portal *Web*.

A aplicação Portal *Web* também foi desenvolvida utilizando o *framework* VueJS e a autenticação em ambas as aplicações é realizada através do envio de um *Personal Identification Number* (PIN) realizado por um servidor *Simple Mail Transfer Protocol* (SMTP), conforme Figura 4.3.

Figura 4.3: Tela de autenticação da aplicação Portal *Web*
Fonte: Elaborada pelo autor (2023)

A Figura 4.4 mostra a primeira tela após o *login* na aplicação Portal *Web*; nela são listadas, em formato de *cards*, as propriedades vinculadas ao usuário logado, juntamente de seu papel atribuído a cada fazenda. Para visualizar as informações de uma propriedade o usuário deve selecionar a fazenda através do botão dentro do *card* da propriedade.

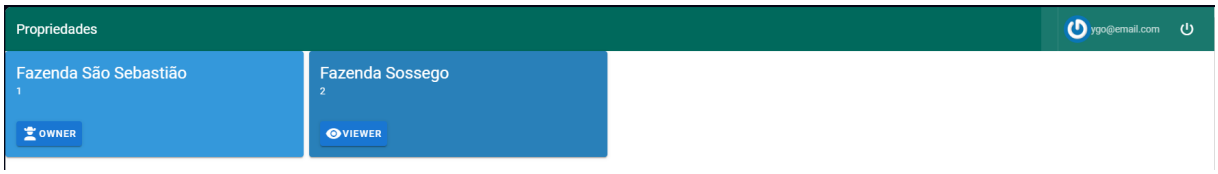

Figura 4.4: Tela de propriedades da aplicação Portal *Web*
Fonte: Elaborada pelo autor (2023)

Ao selecionar a fazenda desejada, o usuário visualiza informações somente daquela propriedade, conforme Figura 4.5. Após acessar os dados da fazenda, o usuário pode autorizar, conforme seu papel, o vínculo dos *middlewares* Big-Boxx na propriedade. Na aplicação Portal *Web*, os *middlewares* são chamados de *gateways*.

Figura 4.5: Tela de contexto da propriedade na aplicação Portal *Web*
Fonte: Elaborada pelo autor (2023)

Na Figura 4.6, é mostrada a tela de gerenciamento dos *middlewares* BigBoxx que realizaram pedido para sincronizar os dados na propriedade selecionada. Somente após a aprovação, os BigBoxx conseguem enviar os dados para a nuvem.

Figura 4.6: Tela de *gateways* da propriedade selecionada na aplicação Portal *Web*
Fonte: Elaborada pelo autor (2023)

Após a criação da propriedade na aplicação Gestor de Inquilinos, e concomitantemente com o processo de solicitação e autorização do envio de dados do BigBoxx ao ambiente em nuvem, a aplicação *Scheduler Job* possui uma tarefa agendada de verificar se existem novas fazendas recém criadas no ambiente em nuvem; em caso afirmativo ela é responsável por criar a *stack* de serviços para a propriedade. Assim, quando o usuário com devida permissão autorizar o envio dos dados do *middleware* BigBoxx para a sincronização em nuvem, o conjunto de *containers* da propriedade estará apto a receber as requisições e consequentemente os dados.

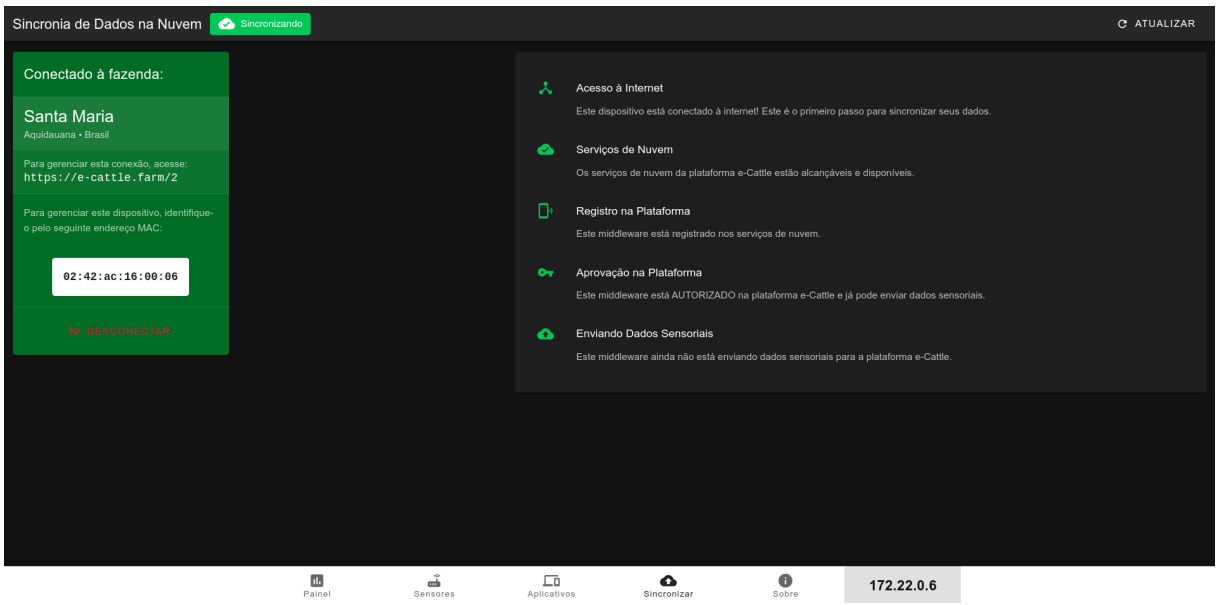

Figura 4.7: Tela do *middleware* BigBoxx apto a sincronização no ambiente em nuvem

Fonte: Elaborada pelo autor (2023)

Ao término da sequência de etapas citadas anteriormente, desde a criação da fazenda na aplicação Gestor de Inquilinos até a aprovação via aplicação Portal *Web*, o *middleware* BigBoxx apto à sincronização no ambiente em nuvem inicia o envio dos dados persistidos localmente, conforme mostrado na Figura 4.7

4.2 Implantação

Toda aplicação desenvolvida com VueJS que será publicada em modo de produção deve passar pelo processo de *build*, que consiste na criação dos arquivos minificados prontos para serem hospedados, logo, as aplicações Gestor de Inquilinos e Portal *Web* foram submetidas a este processo.

Além disso, para serem disponibilizadas como imagem de *container* houve a necessidade de utilizar um servidor *web* na mesma imagem. O servidor *web* escolhido foi o NGINX, conhecido pela sua performance, escalabilidade e ser *open-source* (NGINX, 2023).

Para realizar a orquestração de todo o ambiente em nuvem, optamos pelo Docker Swarm, no entanto, conforme indicado na Tabela 2.1 não há nativamente uma GUI no Docker Swarm. Logo, para o gerenciamento do ambiente em nuvem do e-Cattle adotamos o Portainer CE, que é um conjunto de ferramentas *open-source* que permite gerenciar facilmente *containers* no Docker e Docker Swarm através de uma interface gráfica (Portainer, 2023). A Figura 4.8 mostra a tela do Portainer, na qual é possível observar que a ferramenta gerencia as *stacks*, os serviços, containers, imagens de *containers*, redes, volumes

de dados, entre outros.

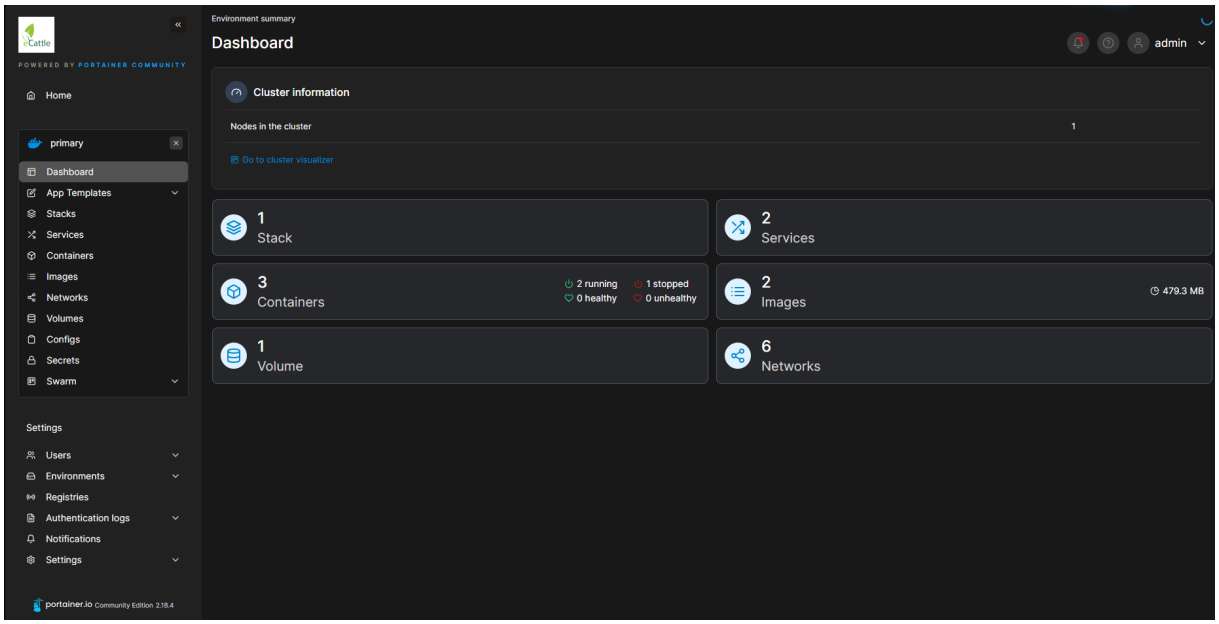

Figura 4.8: Tela do Portainer
Fonte: Elaborada pelo autor (2023)

O Portainer possui uma característica importante: ele gerencia totalmente somente as *stacks* que foram criadas através de sua interface gráfica ou API[2], como por exemplo, editar, atualizar, duplicar ou migrar uma *stack*. Portanto, optamos por utilizar a API disponibilizada pela ferramenta Portainer para a criação das *stacks* de cada fazenda. Assim, a aplicação *Scheduler Job* responsável por essa criação, realiza requisições *Hypertext Transfer Protocol* (HTTP) à API do Portainer, passando os parâmetros da propriedade rural em questão.

No Apêndice A, é possível consultar o arquivo utilizado para a configuração do ambiente em nuvem, já no Apêndice B, é mostrado o arquivo *template* para a criação das *stacks* parametrizadas de cada propriedade rural, e por fim, o Apêndice C contém um *script* criado com o objetivo de facilitar a instalação e configuração do Docker Swarm.

4.3 Distribuição

Os códigos dos artefatos desenvolvidos neste trabalho, que compõe o ambiente em nuvem do e-Cattle, mostrados no Capítulo 3, podem ser consultados no repositório público https://github.com/e-cattle, sob a licença Berkeley Software Distribution (BSD), versão 3.

Para distribuir uma imagem de *container*, é necessário utilizar uma plataforma de registro de *container*, que possui como principal finalidade o gerenci-

[2] https://app.swaggerhub.com/apis/portainer/portainer-ce/2.18.4

amento e compartilhamento de imagens de *container* através de repositórios. Nesta pesquisa optamos pela plataforma Docker Hub[3], que permite a criação ilimitada de repositórios públicos. É possível utilizar as imagens de *containers* disponíveis nos repositórios https://hub.docker.com/orgs/ecattle/repositories.

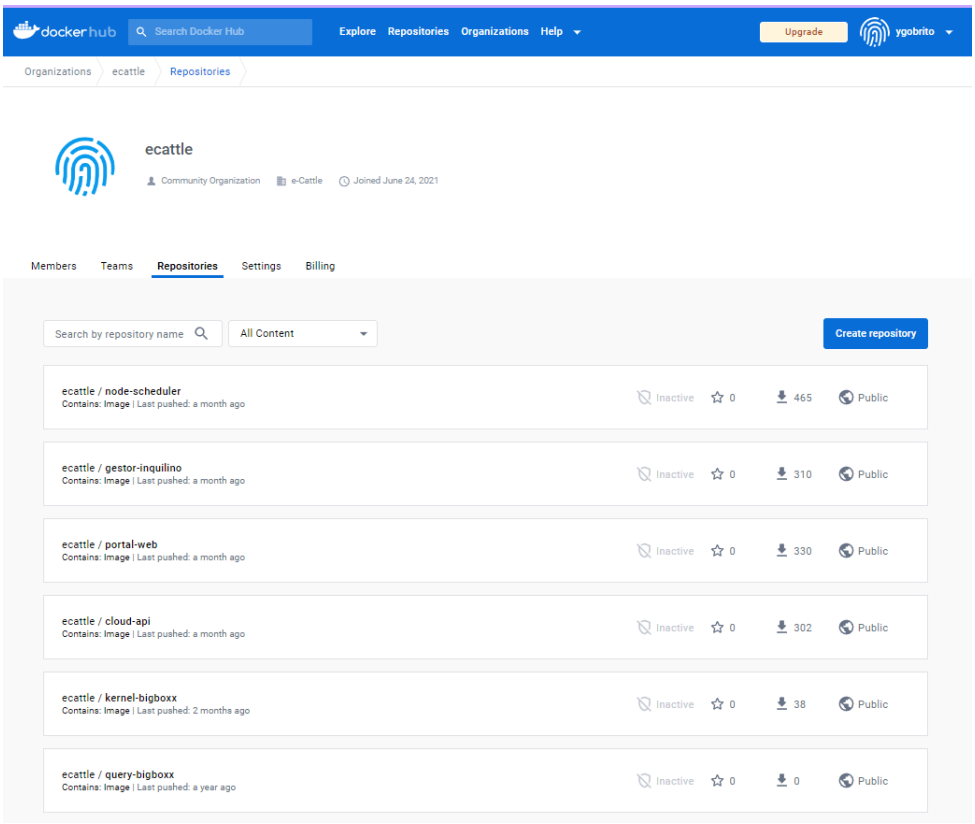

Figura 4.9: Repositórios na plataforma Docker Hub
Fonte: Elaborada pelo autor (2023)

Conforme mostrado na Figura 4.9, possuímos seis repositórios dentro da plataforma Docker Hub, quatro deles utilizados na *stack* do ambiente em nuvem do e-Cattle e dois utilizados nas *stacks* de cada propriedade.

[3] https://hub.docker.com/

CAPÍTULO

5

Conclusões

Neste capítulo são expostas as considerações finais deste trabalho. Na Seção 5.1, é feito um paralelo entre os objetivos desta pesquisa e os resultados alcançados. Na Seção 5.2, são abordadas algumas limitações das soluções propostas e, na Seção 5.3, são apresentadas algumas perspectivas de trabalhos futuros.

5.1 Resumo dos Objetivos e Principais Resultados

A plataforma e-Cattle, desenvolvida por (Carromeu, 2019) inovou a pecuária de precisão, composta por tecnologias atuais e com licença *open-source*. Esta pesquisa propôs solucionar o problema de espaço de armazenamento dos dados persistidos no *middleware* BigBoxx e o acesso dessas informações somente *in loco*, através de sincronização em nuvem. Foram criados dois sistemas *web* para criação de fazendas, usuários e vínculo de *middlewares* BigBoxx nas propriedades. As aplicações foram apresentadas nas Seção 3.3 e Seção 3.4. Também foi criada a aplicação *Scheduler Job*, responsável pela instanciação parametrizada das *stacks* das propriedades, mostrada na Seção 3.5. Por fim, a aplicação *Cloud* API, apresentada na Seção 3.2, foi implementada para disponibilizar os *endpoints* consumidos pelas demais aplicações.

As principais contribuições apresentadas neste trabalho consistem na criação do ambiente em nuvem da plataforma IoT e-Cattle utilizando a arquitetura *multi-tenant* e a orquestração de *containers* realizada com Docker Swarm. Essa pesquisa possibilitou também a sincronização de múltiplos *middlewares* BigBoxx na mesma propriedade dentro do ambiente em nuvem do e-Cattle, através das aplicações implementadas neste trabalho, Gestor de Inquilinos e

Portal *Web*. A criação das *stacks* via *Scheduler Job* pode ser considerada uma importante contribuição, pois, a aplicação é executada dentro de um *container* e possui a capacidade de executar comandos Docker, agendar tarefas e fazer requisições à API do Portainer.

5.2 Limitações

O ambiente em nuvem do e-Cattle foi projetado para ser utilizado com a plataforma *Portainer*, para uma gestão alto nível através de uma GUI, no entanto, como mencionado na Tabela 2.1, o Docker Swarm não possui nativamente a funcionalidade de escalonamento automático. Logo, sugerimos a adoção de ferramentas de terceiros *open-source* para realizar o escalonamento automático, como *cAdvisor*[1] ou *prometheus*[2] para coletar as métricas dos *containers* e jenkins[3] para executar as tarefas de escalonamento.

É possível também adicionar ao *Scheduler Job* as tarefas de escalonamento, pois, atualmente ele possui a capacidade de executar comandos Docker e fazer requisições ao *Portainer* via API. Possibilitando assim, em tempo real, o incremento ou decremento da infraestrutura do ambiente em nuvem, conforme a demanda das requisições recebidas

5.3 Trabalhos Futuros

Atualmente a plataforma e-Cattle se encontra em fase de testes, e ciente do avanço das tecnologias empregadas ao longo do seu desenvolvimento, torna-se essencial a atualização das bibliotecas e ferramentas utilizadas.

Logo, uma possível melhoria ao e-Cattle seria a criação de uma política de atualizações, aplicada em todos os artefatos da plataforma, incluindo o ambiente em nuvem. Uma sugestão de estratégia a ser adotada na política de atualizações é implantar a versão *Long-term Support* (LTS) mais recente de todas as tecnologias envolvidas, garantindo assim a estabilidade do e-Cattle com suporte prolongado.

Como melhoria do e-Cattle, e consequentemente maior utilização do ambiente em nuvem, sugerimos também a sincronização em nuvem através de um *smartphone* nas propriedades sem conectividade com a *internet*. O *middleware* BigBoxx faria o *upload* das informações para o *smartphone* e assim que o dispositivo estiver conectado na *internet*, ele realizaria a sincronização com o ambiente em nuvem. A utilização de outros dispositivos intermediários

[1] https://github.com/google/cadvisor
[2] https://prometheus.io
[3] https://www.jenkins.io

na sincronização em nuvem é conhecida como adoção da abordagem de névoa. Segundo Mahmud and Buyya (2016), a névoa consiste num paradigma de computação distribuída que atua como um intermediário entre a nuvem e dispositivos IoT, descentralizando o processamento de dados.

Bibliografia

ABIEC (2023). Perfil da pecuária no brasil - capítulo 3 - 2023. *Associação Brasileira das Indústrias Exportadoras de Carne*. Citado na página 1.

Alkhulaifi, A. e El-Alfy, E.-S. M. (2020). Exploring lattice-based post-quantum signature for jwt authentication: Review and case study. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, páginas 1–5. Citado na página 23.

Ayres, N., Deka, L., e Passow, B. (2019). Virtualisation as a means for dynamic software update within the automotive e/e architecture. In *2019 IEEE SmartWorld, Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, páginas 154–157. Citado na página 8.

Bachiega, N. G., de Souza, P. S. L., Bruschi, S. M., e de Souza, S. d. R. S. (2020). Performance evaluation of container's shared volumes. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, páginas 114–123. Citado na página 8.

Batista, C., Proença, B., Cavalcante, E., Batista, T., Morais, F., e Medeiros, H. (2022). Towards a multi-tenant microservice architecture: An industrial experience. In *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, páginas 553–562. Citado na página 13.

Bicevska, Z. e Oditis, I. (2017). Towards nosql-based data warehouse solutions. *Procedia Computer Science*, 104:104 – 111. ICTE 2016, Riga Technical University, Latvia. Citado na página 16.

Buyya, R. e Srirama, S. (2019). *Fog and Edge Computing: Principles and Paradigms*. Wiley Series on Parallel and Distributed Computing. Wiley. Citado na página 11.

Carromeu, C. (2019). *e-Cattle: Uma Plataforma de IoT para Pecuária de Precisão*. PhD thesis, Universidade Federal de Mato Grosso do Sul, Campo Grande, MS, Brasil. Citado nas páginas 2, 14, 15, 16, 17, e 35.

Cha, A., Wittern, E., Baudart, G., Davis, J. C., Mandel, L., e Laredo, J. A. (2020). A principled approach to graphql query cost analysis. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, pagina 257–268, New York, NY, USA. Association for Computing Machinery. Citado na página 16.

Cheong, H. (2019). Translating json schema logics into owl axioms for unified data validation on a digital manufacturing platform. *Procedia Manufacturing*, 28:183 – 188. 7th International conference on Changeable, Agile, Reconfigurable and Virtual Production (CARV2018). Citado na página 16.

Cui, L., Hao, Z., Wang, C., Fei, H., e Ding, Z. (2016). Piccolo: A fast and efficient rollback system for virtual machine clusters. In *2016 45th International Conference on Parallel Processing (ICPP)*, páginas 87–92. Citado na página 8.

Docker (2023a). Docker overview. Disponível em: https://docs.docker.com/get-started/overview/. Acesso em: 21 de maio de 2023. Citado na página 6.

Docker (2023b). What is a container? Disponível em: https://www.docker.com/resources/what-container/. Acesso em: 21 de maio de 2023. Citado na página 6.

Dudhe, P. V., Kadam, N. V., Hushangabade, R. M., e Deshmukh, M. S. (2017). Internet of things (iot): An overview and its applications. In *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, páginas 2650–2653. Citado na página 2.

Espadas, J., Molina, A., Jiménez, G., Molina, M., Ramírez, R., e Concha, D. (2013). A tenant-based resource allocation model for scaling software-as-a-service applications over cloud computing infrastructures. *Future Generation Computer Systems*, 29(1):273–286. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures. Citado na página 10.

Fowler, S. (2017). *Microsserviços prontos para a produção: Construindo sistemas padronizados em uma organização de engenharia de software*. Novatec Editora. Citado na página 9.

Hat, R. (2020). Arquitetura multitenancy. Disponível em: https://www.redhat.com/pt-br/topics/cloud-computing/what-is-multitenancy. Acesso em: 15 de abril de 2023. Citado na página 10.

JUNIOR, G. B. M. (2020). Forças motrizes para a agropecuária brasileira na próxima década: implicações para a agricultura digital. *Agricultura digital: pesquisa, desenvolvimento e inovação nas cadeias produtivas.* Citado na página 1.

Jutadhamakorn, P., Pillavas, T., Visoottiviseth, V., Takano, R., Haga, J., e Kobayashi, D. (2017). A scalable and low-cost mqtt broker clustering system. In *2017 2nd International Conference on Information Technology (INCIT)*, páginas 1–5. Citado na página 6.

Khazaei, H., Bannazadeh, H., e Leon-Garcia, A. (2017). End-to-end management of iot applications. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, páginas 1–3. Citado na página 8.

Krylovskiy, A. (2015). Internet of things gateways meet linux containers: Performance evaluation and discussion. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, páginas 222–227. Citado na página 14.

Kubernetes (2023). Kubernetes documentation. Disponível em: https://kubernetes.io/docs/concepts/overview/. Acesso em: 22 de maio de 2023. Citado na página 6.

Kuppusamy, S., Kaniappan, V., e Thirupathi, D. (2015). Design and development of multi-tenant web framework. *Procedia Computer Science*, 48:180 – 191. International Conference on Computer, Communication and Convergence (ICCC 2015). Citado na página 11.

Li, D., Pyke, R., e Jiang, R. (2021). A scalable cloud-based architecture to deploy jupyterhub for computational social science research. In *Practice and Experience in Advanced Research Computing*, PEARC '21, New York, NY, USA. Association for Computing Machinery. Citado na página 6.

Locke, S., Li, H., Chen, T.-H. P., Shang, W., e Liu, W. (2022). Logassist: Assisting log analysis through log summarization. *IEEE Transactions on Software Engineering*, 48(9):3227–3241. Citado na página 8.

Madakam, S., Ramaswamy, R., e Tripathi, S. (2015). Internet of things (iot): A literature review. *Journal of Computer and Communications*, 3(05):164. Citado na página 1.

Mahmud, R. e Buyya, R. (2016). Fog computing: A taxonomy, survey and future directions. *CoRR*, abs/1611.05539. Citado na página 37.

Maneenual, P. e Vasupongayya, S. (2018). Logging mechanism for internet of things: A case study of patient monitoring system. In *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, páginas 1–6. Citado na página 8.

Marathe, N., Gandhi, A., e Shah, J. M. (2019). Docker swarm and kubernetes in cloud computing environment. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, páginas 179–184. Citado na página 5.

Masmoudi, F., Sellami, M., Loulou, M., e Hadj Kacem, A. (2019). Accountability management for multi-tenant cloud services. *International Journal of Grid and Utility Computing*, 10:141. Citado na página 13.

Meneghello, F., Calore, M., Zucchetto, D., Polese, M., e Zanella, A. (2019). Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices. *IEEE Internet of Things Journal*, 6(5):8182–8201. Citado na página 23.

Mietzner, R., Leymann, F., e Papazoglou, M. P. (2008). Defining composite configurable saas application packages using sca, variability descriptors and multi-tenancy patterns. In *2008 Third International Conference on Internet and Web Applications and Services*, páginas 156–161. Citado na página 10.

Millnert, V. e Eker, J. (2020). Holoscale: horizontal and vertical scaling of cloud resources. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, páginas 196–205. Citado na página 7.

Modak, A., Chaudhary, S. D., Paygude, P. S., e Ldate, S. R. (2018). Techniques to secure data on cloud: Docker swarm or kubernetes? In *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, páginas 7–12. Citado na página 6.

Naji Saif, M. A., Niranjan Aradhya, S., Hezam Murshed, B. A., Farhan Alnaggar, O. A. M., e Ali, I. M. S. (2023). Multi-objective container scheduling and multi-path routing for elastic business process management in autonomic multi-tenant cloud. *Concurrency and Computation: Practice and Experience*, 35(6):e7584. Citado nas páginas 12 e 14.

Narale, S. A. e Butey, P. (2018). Throttled load balancing scheduling policy assist to reduce grand total cost and data center processing time in cloud environment using cloud analyst. In *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, páginas 1464–1467. Citado na página 8.

Nery, C. e Britto, V. (2022). Pesquisa nacional por amostra de domicílios. Disponível em: https://www.ibge.gov.br/estatisticas/sociais/trabalho/9171-pesquisa-nacional-por-amostra-de-domicilios-continua-mensal.html?=&t=publicacoes. Acesso em: 25 de julho de 2023. Citado na página 2.

NGINX (2023). What is nginx? Disponível em: https://www.nginx.com/resources/glossary/nginx/. Acesso em: 12 de maio de 2023. Citado na página 31.

Portainer (2023). Portainer documentation. Disponível em: https://docs.portainer.io. Acesso em: 15 de maio de 2023. Citado na página 31.

Poulton, N. (2020). *Docker Deep Dive: Zero to Docker in a single book*. NIGEL POULTON LTD. Citado na página 6.

Prasandy, T., Titan, Murad, D. F., e Darwis, T. (2020). Migrating application from monolith to microservices. In *2020 International Conference on Information Management and Technology (ICIMTech)*, páginas 726–731. Citado na página 9.

Ramos, N., Yamaguchi, C., e Costa, U. (2020). Brazilian journal of development tecnologia da informação e gestão do conhecimento: estratégia de competitividade nas organizações information technology and knowledge management: competitiveness strategy in organizations. *Brazilian Journal of Development*, 6:144–161. Citado na página 1.

Reznik, P., Dobson, J., e Gienow, M. (2019). *Cloud native transformation: practical patterns for innovation*. O'Reilly Media. Citado na página 7.

Rosen, C. (2022). Docker swarm vs. kubernetes: A comparison. Disponível em: https://www.ibm.com/cloud/blog/docker-swarm-vs-kubernetes-a-comparison. Acesso em: 20 de maio de 2023. Citado na página 7.

Samuel, A. e Sipes, C. (2019). Making internet of things real. *IEEE Internet of Things Magazine*, 2(1):10–12. Citado na página 1.

Sellami, W., Hadj Kacem, H., e Hadj Kacem, A. (2020). Dynamic provisioning of service composition in a multi-tenant saas environment. *Journal of Network and Systems Management*, 28. Citado na página 13.

Shah, J. e Dubaria, D. (2019). Building modern clouds: Using docker, kubernetes and google cloud platform. In *2019 IEEE 9th Annual Computing*

and Communication Workshop and Conference (CCWC), páginas 0184–0189. Citado na página 7.

Sree Devi, K. D., Sumathi, D., Vignesh, V., Anilkumar, C., Kataraki, K., e Balakrishnan, S. (2023). Cloud load balancing for storing the internet of things using deep load balancer with enhanced security. *Measurement: Sensors*, 28:100818. Citado na página 8.

SUSE (2019). Kubernetes vs docker swarm: Comparison of two container orchestration tools. Disponível em: https://www.suse.com/c/rancher_blog/kubernetes-vs-docker-swarm-comparison-of-two-container-orchestration-tool Acesso em: 9 de abril de 2023. Citado na página 7.

Tan, L. e Wortman, R. (2014). Cloud-based monitoring and analysis of yield efficiency in precision farming. In *Proceedings of the 15th IEEE International Conference on Information Reuse and Integration, IRI 2014, Redwood City, CA, USA, August 13-15, 2014*, páginas 163–170. Citado nas páginas 11 e 12.

Tilkov, S. e Vinoski, S. (2010). Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80–83. Citado na página 16.

Tsai, W.-T., Zhong, P., e Chen, Y. (2016). Tenant-centric sub-tenancy architecture in software-as-a-service. *CAAI Transactions on Intelligence Technology*, 1(2):150 – 161. Citado na página 10.

Vailshery, L. S. (2022). Number of internet of things (iot) connected devices worldwide from 2019 to 2021, with forecasts from 2022 to 2030. Disponível em: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/. Acesso em: 24 de maio de 2023. Citado na página 1.

Velepucha, V. e Flores, P. (2021). Monoliths to microservices - migration problems and challenges: A sms. In *2021 Second International Conference on Information Systems and Software Technologies (ICI2ST)*, páginas 135–142. Citado na página 10.

VueJS (2023). Documentation for vue 3. Disponível em: https://vuejs.org/guide/introduction.html. Acesso em: 30 de abril de 2023. Citado na página 27.

Watabe, H., Yu, K. N., Safakatti, N., e Shahmohammadi Beni, M. (2023). Development of an open-source gui computer program for modelling irradiation

of multi-segmented phantoms using grid-based system for phits. *Nuclear Engineering and Technology*, 55(1):373–377. Citado na página 7.

Zhao, H., Lim, H., Hanif, M., e Lee, C. (2019). Predictive container auto-scaling for cloud-native applications. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, páginas 1280–1282. Citado na página 8.

Zhengtao Liu, Yun Yang, W. G. e Xia, J. (2020). A multi-tenant usage access model for cloud computing. *Computers, Materials and Continua*, 64(2):1233–1245. Citado na página 12.

APÊNDICE

A

Template para criação do ambiente em nuvem

O arquivo `StartCloudEnvironment.yml` contém todos os serviços que o ambiente em nuvem necessita para sua execução. Cada serviço é uma imagem de *container* oficial ou criada pelo autor e disponibilizada na plataforma Docker Hub. Para a criação do ambiente em nuvem do e-Cattle, é necessário informar variáveis de ambiente que serão utilizadas pelos serviços, são elas:

NODE_ENV Especifica se o ambiente é *development, production* ou *test*

NODE_PORT Porta que o Node do *container guest* será exposta no *host*

MONGO_PORT Porta que o MongoDB do *container guest* será exposta no *host*

JWT_SECRET *Token* utilizado pelo JWT no *payload*

SMTP_HOST Nome do serviço de e-mail SMTP dentro da *stack*

SMTP_PORT Porta do serviço de e-mail SMTP dentro da *stack*

SMTP_SECURE Indica se o serviço SMTP utilizará SSL

SMTP_FROM Texto padrão do campo Assunto do e-mail de autenticação encaminhado pelo serviço SMTP

DOCKER_EMAIL Porta que o SMTP vai rodar no *host*

VUE_APP_CLOUD Endereço do serviço Cloud API

TOKEN_PORTAINER_API *Token* para consumir API do Portainer

SWARM_ID ID da rede *swarm*, utilizado nas requisições via API do Portainer

```
version: '3.8'
services:
  cloud-api:
    image: ecattle/cloud-api:latest
    hostname: cloud-api
    command: ["sh", "-c", "npm i & npm run start"]
    environment:
      - NODE_ENV=${NODE_ENV}
      - DOCKER_NODE_PORT=${NODE_PORT}
      - DOCKER_MONGO_PORT=${MONGO_PORT}
      - JWT_SECRET=${JWT_SECRET}
      - SMTP_HOST=${SMTP_HOST}
      - SMTP_PORT=${SMTP_PORT}
      - SMTP_SECURE=${SMTP_SECURE}
      - SMTP_FROM=${SMTP_FROM}
      - DOCKER_EMAIL=${DOCKER_EMAIL}

    ports:
      - "${NODE_PORT}:3000"
    networks:
      - swarm_network
    deploy:
      mode: replicated
      replicas: 2
      placement:
        constraints: [node.role == manager]

  portal-web:
    image: ecattle/portal-web:latest
    environment:
      - VUE_APP_CLOUD=${VUE_APP_CLOUD}
    ports:
      - "${WEB_PORT}:8080"
    networks:
      - swarm_network
    deploy:
      mode: replicated
      replicas: 2
      placement:
        constraints: [node.role == manager]

  gestor-inquilino:
    image: ecattle/gestor-inquilino:latest
    environment:
      - VUE_APP_CLOUD=${VUE_APP_CLOUD}
    ports:
      - "${MANAGER_PORT}:8080"
```

```
48      networks:
49        - swarm_network
50      deploy:
51        mode: replicated
52        replicas: 2
53        placement:
54          constraints: [node.role == manager]
55    database:
56        image: mongo
57        ports:
58          - "${MONGO_PORT}:27017"
59        networks:
60          - swarm_network
61        deploy:
62          mode: replicated
63          replicas: 1
64          placement:
65            constraints: [node.role == manager]
66    mail:
67        image: mailhog/mailhog
68        user: root
69        ports:
70          - "${DOCKER_EMAIL}:8025"
71        networks:
72          - swarm_network
73        deploy:
74          mode: replicated
75          replicas: 2
76          placement:
77            constraints: [node.role == manager]
78    scheduler:
79      image: ecattle/node-scheduler:latest
80      environment:
81        - URL_MONGO=database
82        - URL_API_PORTAINER=${VUE_APP_CLOUD}
83        - TOKEN_API_PORTAINER=${TOKEN_API_PORTAINER}
84        - SWARM_ID=${SWARM_ID}
85      command: ["sh", "-c", "npm i & npm run start"]
86      volumes:
87        - farms_data:/usr/src/app
88        - /var/run/docker.sock:/var/run/docker.sock
89        - /var/lib/docker/volumes:/var/lib/docker/volumes
90      networks:
91        - swarm_network
92      deploy:
93        mode: replicated
94        replicas: 1
95        placement:
96          constraints: [node.role == manager]
```

```
  agendash:
    image: agenda/agendash
    environment:
      - MONGODB_URI=mongodb://database
      - COLLECTION=schedulerJob
    ports:
      - "2999:3000"
    networks:
      - swarm_network
    deploy:
      mode: replicated
      replicas: 1
      placement:
        constraints: [node.role == manager]

networks:
  swarm_network:
    external: true

volumes:
  farms_data:
```

APÊNDICE

B

Template para criação da *stack* de cada propriedade

O arquivo `DockerfileNewEnvironmentCloud.yml` é o *template* para a criação das *stacks* de cada propriedade rural, a seguir a listagem de variáveis de ambiente utilizadas pelos serviços.

KERNEL_PORT Porta que a aplicação Kernel será exposta no *host*;

DB_CLOUD URI do serviço MongoDB da propriedade;

API_CLOUD URI do serviço Cloud API da propriedade;

REDIS_CLOUD URI do Redis da propriedade;

CODE_FARM Código identificador da propriedade que o kernel atuará;

```
1  version: "3.8"
2  services:
3    ecattle-mongo:
4      image: mongo
5      ports:
6        - ${MONGO_PORT}:27017
7      deploy:
8        replicas: 2
9        restart_policy:
10         condition: on-failure
11       placement:
12         constraints: [node.role == worker]
13   ecattle-redis:
```

```
14      image: redis:alpine
15      ports:
16        - ${REDIS_PORT}:6379
17      deploy:
18        replicas: 2
19        restart_policy:
20          condition: on-failure
21        placement:
22          constraints: [node.role == worker]
23    ecattle-kernel:
24      command: ["sh", "-c", "npm i & npm run start"]
25      image: ecattle/kernel-bigboxx:latest
26      ports:
27        - ${KERNEL_PORT}:3000
28      environment:
29        - DB_CLOUD=mongodb://ecattle-mongo/e-cattle
30        - REDIS_CLOUD=ecattle-redis
31        - API_CLOUD=ecattle-cloud
32        - CODE_FARM=${CODE_FARM}
33      deploy:
34        replicas: 1
35        restart_policy:
36          condition: on-failure
37        placement:
38          constraints: [node.role == worker]
```

APÊNDICE

C

Shell Script para criação do ambiente em nuvem via terminal

O arquivo `cloud-script.sh` é um *script* responsável pela criação do ambiente em nuvem sem interface gráfica. O *script* possui a instalação padrão do Docker para Sistemas Operacionais que utilizam pacotes DEB. Neste trabalho foi utilizada a distribuição Ubuntu Server 20.04.

Após a instalação e configuração de permissão de usuário do Docker, o *script* verifica se o nó será considerado *manager* ou *worker* no ambiente Docker Swarm; caso seja do tipo *manager*, nesta etapa são listadas as interfaces de redes presentes na máquina e o usuário escolhe qual deseja utilizar para o Docker Swarm, pois é na interface escolhida que será ativado o modo Swarm do Docker.

Em seguida, é criada uma rede virtual intitulada *swarm_network* que será utilizada pelos serviços do ambiente em nuvem do e-Cattle, e o *script* é encerrado após o *download* e instalação do Portainer. Caso a máquina seja do tipo *worker*, o *script* mostra qual passo deve ser executado no nó *manager* para inserir nós *workers* na rede Swarm.

```
1 #!/bin/bash
2
3 # Tornando script case insensitive
4 shopt -s nocasematch
5
6 # Titulo e opcoes do menu principal
7 PS3="Escolha qual tipo de node o PC tera no ambiente cloud do e-Cattle: "
8 options=("Manager" "Worker" "Sair")
9
```

```
10 # Estilizando script {verde: Informacao geral, amarelo: Informacao que
       requer atencao maior}
11 green=`tput setaf 2`
12 yellow=`tput setaf 3`
13 reset=`tput sgr0`
14
15 # Capturando nome do usuario linux e armazenando na variavel user
16 user="$(id -u -n)"
17
18 # Capturando as interfaces de rede do computador e armazenando na variavel
       interfaces
19 interfaces="$(ls /sys/class/net)"
20
21 # Instalacao de softwares utilizados no decorrer do script
22 default_install()
23 {
24     echo "${green}Atualizando repositorio...${reset}"
25     sleep "2s"
26     sudo apt-get update
27     sleep "2s"
28     echo "${green}Instalando curl...${reset}"
29     sleep "2s"
30     sudo apt-get install curl software-properties-common apt-transport-
       https -y
31     sleep "2s"
32     echo "${green}Instalando Docker...${reset}"
33     sleep "2s"
34     sudo apt-get install docker.io -y
35     echo "${green}Configurando Docker...${reset}"
36     sleep "2s"
37     sudo usermod -aG docker ${user}
38     sudo chmod 666 /var/run/docker.sock
39     sleep "2s"
40 }
41
42 # Cria a rede swarm do zero
43 create_manager()
44 {
45     PS3="Escolha qual interface sera utilizada para a rede swarm: "
46     default_install
47     echo "${green}Criando rede swarm...${reset}"
48     sleep "1s"
49     echo "${green}Escolhendo interface${reset}"
50     sleep "2s"
51     select interface in $interfaces; do
52         echo "${green}Criando rede swarm em $interface${reset}";
53         docker swarm init --advertise-addr $interface
54         sleep "2s"
55         echo "${yellow}Atencao: copie o codigo indicado acima e cole nos
```

```
      nodes Workers${reset}"
56          sleep "2s"
57          echo "${green}Criando network na rede swarm...${reset}"
58          sleep "2s"
59          docker network create swarm_network --driver overlay --attachable
60          sleep "2s"
61          echo "${green}Baixando e instalando portainer para gerenciamento da
       rede swarm...${reset}"
62          sleep "2s"
63          curl -L https://downloads.portainer.io/ce2-18/portainer-agent-stack
      .yml -o portainer-agent-stack.yml
64          docker stack deploy -c portainer-agent-stack.yml portainer
65          echo "${green}Abrir portainer em http://localhost:9000${reset}"
66          sleep "2s"
67          break
68      done
69      sleep "2s"
70      PS3="Escolha qual tipo de node o PC tera no ambiente cloud do e-Cattle:
        "
71 }
72
73 # Adiciona um node manager no ambiente swarm ja criado
74 join_manager()
75 {
76   default_install
77   sleep "2s"
78   echo "${green}Execute o seguinte comando no node Manager da rede swarm${
       reset}"
79   echo "${yellow}docker swarm join-token manager${reset}"
80   echo "${green}Cole o codigo resultante aqui${reset}"
81 }
82
83 # Cria um node worker
84 create_worker()
85 {
86   default_install
87   sleep "2s"
88   echo "${green}Execute o seguinte comando no node Manager da rede swarm${
       reset}"
89   echo "${yellow}docker swarm join-token worker${reset}"
90   echo "${green}Cole o codigo resultante aqui${reset}"
91 }
92
93 # Menu principal do script
94 select choice in "${options[@]}"; do
95      case $choice in
96          "Criar rede swarm e node Manager")
97              echo "Voce escolheu $choice"
98              create_manager
```

```
 99             break;;
100         "Adicionar node Manager na rede swarm ")
101             echo "Voce escolheu $choice"
102             join_manager
103             break;;
104         "Worker")
105             echo "Voce escolheu $choice"
106             create_worker
107             break;;
108     "Sair")
109         exit
110         ;;
111         *) echo "Opcao Invalida $REPLY";;
112     esac
113 done
```