

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Victor Silva Santos

**Desenvolvimento da infraestrutura de uma
plataforma voltada ao apoio e proteção de
animais baseada em computação em nuvem,
contêineres e microsserviços**

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Victor Silva Santos

**Desenvolvimento da infraestrutura de uma plataforma
voltada ao apoio e proteção de animais baseada em
computação em nuvem, contêineres e microsserviços**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, como parte dos requi-
sitos exigidos para a obtenção título de Ba-
charel em Sistemas de Informação.

Orientador: Pedro Frosi Rosa

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2023

Victor Silva Santos

Desenvolvimento da infraestrutura de uma plataforma voltada ao apoio e proteção de animais baseada em computação em nuvem, contêineres e microsserviços

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 07 de novembro de 2023:

Pedro Frosi Rosa
Orientador

Rodrigo Sanches Miani

Ronaldo Castro de Oliveira

Uberlândia, Brasil
2023

Agradecimentos

Agradeço primeiramente a Deus por estar sempre comigo, mesmo em momentos complicados em que me faltava fé.

Agradeço aos meus pais por todos os sacrifícios que fizeram para eu poder receber uma educação de qualidade e por todo o apoio que me deram durante todos os anos de minha vida.

Agradeço também à minha irmã por todo o amor e carinho mesmo com a minha frequente falta de paciência.

Agradeço à minha melhor amiga, Laix, por todo o apoio e paciência comigo durante os períodos finais do curso e pelas dicas também.

Agradeço ao meu orientador Prof. Dr. Pedro Frosi Rosa por me acompanhar nessa trajetória, por toda preocupação, apoio e paciência também.

Resumo

O objetivo deste trabalho é explorar a aplicação de tecnologias de *cloud computing*, contêineres e microsserviços no desenvolvimento da infraestrutura de uma plataforma para a integração do ecossistema envolvido em apoiar e proteger animais. À medida que a tecnologia avança, arquiteturas monolíticas estão perdendo espaço, sendo substituídas por arquiteturas baseadas em microsserviços, que estão se consolidando no mercado por meio de tecnologias de contêineres, tais como Kubernetes. Ao adotar essa abordagem, é possível construir uma infraestrutura flexível e escalável, essencial para atender às demandas que a plataforma visa suprir para o referido ecossistema. A combinação dessas tecnologias proporciona uma base sólida para uma plataforma ágil, capaz de se adaptar de forma eficiente a uma variedade de desafios, contribuindo assim para a melhoria da proteção e apoio aos animais em nossa sociedade.

Palavras-chave: *Cloud Computing*, *Containers*, Kubernetes, Microsserviços, Terraform, AWS.

Lista de ilustrações

Figura 1 – Comparativo entre arquitetura de monolito e arquitetura de microserviços	16
Figura 2 – Arquitetura do kubernetes	18
Figura 3 – Definição do Terraform	20
Figura 4 – Plataforma ADOTE: Visão Geral.	24
Figura 5 – <i>Tiers</i> da Plataforma ADOTE	25
Figura 6 – Infraestrutura Genérica	26
Figura 7 – Diagrama da Infraestrutura	28
Figura 8 – AWS Provider	30
Figura 9 – Security Group Module	30
Figura 10 – Parameter Group Module	30
Figura 11 – Subnet Group Module	31
Figura 12 – RDS Resource	31
Figura 13 – EKS Module	32
Figura 14 – RDS Criado	32
Figura 15 – EKS Criado	32

Lista de abreviaturas e siglas

APA	Associação de Proteção Animal
TI	Tecnologia da Informação
CSP	<i>Cloud Service Provider</i>
AWS	<i>Amazon Web Services</i>
IaaS	<i>Infrastructure as a Service</i>
PaaS	<i>Platform as a Service</i>
SaaS	<i>Software as a Service</i>
SPoF	<i>Single Point of Failure</i>
API	<i>Application Programming Interface</i>
VM	<i>Virtual Machine</i>
SO	Sistema Operacional
IaC	<i>Infrastructure as Code</i>
DNS	<i>Domain Name System</i>
RPC	<i>Remote Procedure Call</i>
AZ	<i>Availability Zones</i>
CDN	<i>Content Delivery Network</i>
EKS	<i>Elastic Kubernetes Service</i>
AKS	<i>Azure Kubernetes Service</i>
GKE	<i>Google Kubernetes Engine</i>
DMZ	<i>Demilitarized Zone</i>
VPC	<i>Virtual Private Cloud</i>
REST	<i>Representational State Transfer</i>
HTTP	<i>Hypertext Transfer Protocol</i>

ACM	<i>AWS Certificate Manager</i>
TLS	<i>Transport Layer Security</i>
WAF	<i>Web Application Firewall</i>
NLB	<i>Network Load Balancer</i>
NAT	<i>Network Address Translation</i>
RDS	<i>Relational Database Service</i>
CLI	<i>Command Line Interface</i>

Sumário

1	INTRODUÇÃO	9
1.1	Objetivos	10
1.2	Justificativa	10
1.3	Metodologia	11
2	REFERENCIAL TEÓRICO E CONTEXTO ATUAL	12
2.1	Fundamentação teórica	12
2.1.1	Computação em Nuvem	12
2.1.1.1	Nuvem Pública	13
2.1.1.2	Nuvem Privada	14
2.1.1.3	Nuvem Híbrida	14
2.1.1.4	Multi-nuvem e Multi-nuvem Híbrida	14
2.1.2	Microserviços	15
2.1.3	Kubernetes	17
2.1.4	IaC	19
2.1.4.1	Terraform	19
2.2	Estado da Arte	20
2.2.1	Cloud Computing	21
2.2.1.1	AWS	21
2.2.1.2	Azure	21
2.2.1.3	Google Cloud	22
2.2.2	Microserviços e Kubernetes	22
3	PLATAFORMA ADOTE: MODELO CONCEITUAL	23
3.1	Visão Geral da plataforma ADOTE	23
3.2	Tiers da plataforma ADOTE	25
3.3	Arquitetura conceitual da plataforma ADOTE	26
4	DESENVOLVIMENTO	28
4.1	Definição da Arquitetura de Infraestrutura	28
4.2	Configuração dos Recursos Terraform	29
4.3	Criação dos Recursos na AWS	32
4.4	Disponibilização via Github	32
5	CONCLUSÃO	34
	REFERÊNCIAS	35

1 Introdução

Várias cidades enfrentam problemas de abandono, maus tratos e acidentes envolvendo animais. A maior parte das pessoas, ao se deparar com esses problemas, nem sempre são capazes de promover soluções ou tomar atitudes eficazes a fim de tirá-los das ruas e oferecer-lhes um tratamento adequado e confortável.

Já existem organizações que trabalham nessa área prestando assistência, como ONGs, porém, na maioria das vezes, enfrentam dificuldades financeiras e também, não possuem meios eficientes de comunicação para notificar pessoas interessadas em adotar ou envolvidas em apoiar campanhas criadas para esta finalidade (TOCANTINS, 2021).

O projeto ADOTE, então, surge com o objetivo de ser uma plataforma de software para integrar esses diversos atores, envolvidos no ciclo de vida de animais urbanos, visando o acolhimento de animais abandonados ou maltratados. A plataforma pretende criar uma rede de colaboração que inclui, mas não se limita a, ONGs, Ministério Público, Prefeituras, Prestadores de Serviços, Clínicas, Lojas, simpatizantes com a causa etc.

Neste trabalho de conclusão de curso, será apresentado o desenvolvimento da infraestrutura de uma plataforma de adoção de animais, utilizando conceitos atuais de infraestrutura tecnológica, como *cloud computing*, kubernetes e microsserviços. O objetivo é construir uma plataforma moderna e eficiente, que possibilite a adoção de animais de forma mais ágil e fácil, tanto para as pessoas que buscam adotar, quanto para as organizações que disponibilizam animais para adoção.

A utilização de *cloud computing* nesse projeto se justifica pela necessidade de escalabilidade e flexibilidade da plataforma, uma vez que o acolhimento de animais em situação de risco sensibiliza muitas pessoas e pode gerar um grande volume de acessos em determinados momentos. Visto que este é um problema que extrapola territórios, a plataforma é projetada para ser expandida para outras cidades do País, além de Uberlândia. Portanto, com o uso de serviços em nuvem, a plataforma poderá se adaptar facilmente às demandas crescentes, sem a necessidade de investimentos em infraestrutura física que poderiam ser onerosos (NIKITA, 2021).

Devido à complexidade que se espera da aplicação e suas funcionalidades, o uso de microsserviços em detrimento de um único monólito se justifica pela modularidade e facilidade de manutenção, permitindo que a plataforma seja composta por diversos serviços independentes, cada um com uma responsabilidade específica. Outro requisito importante para que os serviços sejam isolados é que se algum deles falhar, o sistema ainda continua em funcionamento (LIMA, 2018).

Já o uso de Kubernetes é justificado pela sua capacidade de orquestração de contêineres, permitindo uma gestão eficiente de recursos e uma maior facilidade na distribuição de tarefas entre os microsserviços que comporão a plataforma. Visto que será utilizada uma plataforma de *cloud*, é possível utilizar também o serviço de Kubernetes gerenciado que elas disponibilizam, de maneira que todo o gerenciamento e provisionamento de máquinas fica a cargo da própria *cloud* (ONYSZKO, 2020).

1.1 Objetivos

O objetivo geral deste trabalho é desenvolver a infraestrutura da Plataforma ADOTE, utilizando conceitos atuais de infraestrutura tecnológica. Para alcançar este objetivo, serão estabelecidos os seguintes objetivos específicos:

- Identificar as funcionalidades necessárias para a plataforma;
- Desenvolver o modelo conceitual;
- Selecionar as tecnologias de infraestrutura mais adequadas para o desenvolvimento da plataforma;
- Desenvolver a arquitetura da plataforma, utilizando conceitos de microsserviços, *cloud computing* e kubernetes;
- Implementar a infraestrutura, utilizando tecnologias de *Infrastructure as Code*.

1.2 Justificativa

Atualmente, ONGs, que são um dos principais atores do sistema, possuem uma grande dificuldade de comunicação com o público geral Maceió (2021). Na maioria das vezes possuem apenas um *website*, com as ações que executam.

Algumas ONGs largamente conhecidas são: Associação de Proteção Animal em Uberlândia (APA) (<<https://www.apauberlandia.org.br>>), Ampara Animal (<<https://amparanimal.org.br>>), Cães Sem Dono (<<https://www.caosemdono.com.br>>), ProAnima (<<https://www.proanima.org.br>>), etc.

Vale ressaltar que durante o desenvolvimento deste projeto não foram encontradas plataformas semelhantes sobre o assunto, que permitissem completar este capítulo em termos de trabalhos correlatos, apenas páginas como aquelas referenciadas.

ADOTE tem por premissa, ser uma plataforma que integrará ONGs, Público geral, Hospitais veterinários, Poder Público etc, de maneira que a comunicação entre essas

entidades se torne mais simples e, com isso, diminuindo a distância entre animais em situação de risco e seu atendimento.

Além da causa em si, cabe uma segunda, e não menos importante, justificativa em termos do trabalho de conclusão, pois é uma etapa fundamental na formação do egresso. Neste sentido, o trabalho navega em termos extremamente atuais, com desafios importantes, visando alta disponibilidade, escalabilidade e balanceamento de carga para altas demandas de processamento. Além disso, computação em nuvem, bem como o uso de contêineres são temas bastante discutidos em ambientes empresariais. Isto permite que a experiência do egresso seja enriquecida e se alinhe a demandas correntes e futuras.

1.3 Metodologia

Para o desenvolvimento deste trabalho, será adotada uma abordagem híbrida, combinando o *Unified Process* com metodologias ágeis, focada na criação da infraestrutura.

O *Unified Process* é um processo de desenvolvimento de software iterativo e incremental, que fornece uma estrutura flexível para o planejamento, execução e controle do projeto. Ele consiste em quatro fases principais: iniciação, elaboração, construção e transição.

A fase Iniciação consiste em uma pesquisa exploratória para compreender os requisitos e objetivos do projeto de infraestrutura de forma mais abrangente. Essa etapa permite estabelecer uma visão clara do que está sendo desenvolvido e a identificar os possíveis riscos e restrições relacionados à infraestrutura.

Durante a fase Elaboração são desenvolvidos modelos (conceitual e de dados), a arquitetura (camadas, componentes, integrações etc) e são especificadas tecnologias relativas a *cloud computing*, microserviços, contêineres e linguagens para o desenvolvimento da infraestrutura. Essas tecnologias serão analisadas em termos de seus pontos positivos e negativos, considerando aspectos como escalabilidade, desempenho e segurança, a fim de garantir a escolha adequada para suportar aplicações e seus acessos.

Na fase Construção serão utilizados os artefatos da fase Elaboração e linguagens de IaC (*Infrastructure as Code*) para desenvolver a infraestrutura que comportará o projeto. Essas linguagens permitirão a descrição da infraestrutura desejada de forma programática, proporcionando automação e facilitando a implantação e o gerenciamento dos recursos necessários.

Essa abordagem integrada entre o *Unified Process* e metodologias ágeis permitirá uma maior flexibilidade e adaptabilidade ao longo do projeto de criação da infraestrutura, garantindo entregas incrementais e permitindo ajustes com base no *feedback* contínuo.

2 Referencial Teórico e Contexto Atual

Este capítulo oferece um posicionamento de contexto em termos do estado da arte teórico e tecnológico. Em particular, serão explicadas as tecnologias utilizadas na construção da infraestrutura da Plataforma ADOTE.

2.1 Fundamentação teórica

Esta seção tem o objetivo de aprofundar no conhecimento dos fundamentos teóricos que embasaram o desenvolvimento das atividades, abordando temas como Computação em Nuvem, Microsserviços, Kubernetes e IaC.

2.1.1 Computação em Nuvem

A computação em Nuvem (*cloud computing*) consiste na oferta de recursos sob demanda, com balanceamento de carga, escalabilidade, disponibilidade e segurança, disponibilizados em infraestrutura acessível via Internet. Esses recursos podem ser servidores, físicos ou virtuais, para processamento ou armazenamento de dados e podem incluir ferramentas de desenvolvimento, monitoramento, gestão etc.

Plataformas responsáveis pela disponibilidades de serviços em nuvem são conhecidas por CSP (*Cloud Services Provider* - Provedor de Serviços em Nuvem). CSPs realizam a cobrança dos recursos utilizados com base no tempo de uso e na quantidade utilizada (IBM, 2023a).

Comparada com o modelo antigo de computação, em que as empresas possuíam seus próprios servidores, o uso da computação em nuvem possui algumas vantagens (AWS, 2023):

- Elasticidade - Não é necessário provisionar mais recursos do que o necessário para a aplicação funcionar e, ao invés disso, é possível disponibilizar apenas a quantidade que o serviço precisa e, se necessário, aloca mais recurso ou desaloca recurso, sendo possível fazer de maneira instantânea e automatizada;
- Custos - na computação em nuvem, o contratante não possui um custo fixo mensal de uso, ou seja, ele paga apenas pelos recursos e serviços utilizados no mês, sendo possível um estudo de gastos para poder reduzir ao máximo, sem que a aplicação rodando seja afetada; e

- Agilidade - Visto que apenas com alguns cliques é possível criar um novo servidor, isso torna a disponibilidade das aplicações criadas muito mais fáceis e rápidas, permitindo com que a equipe de desenvolvimento possa focar de fato apenas na criação e evolução da aplicação.

Existem 3 tipos de serviços para a computação em nuvem. São eles, Infraestrutura como Serviço (IaaS - *Infrastructure as a Service*), Plataforma como Serviço (PaaS - *Platform as a Service*) e Software como Serviço (SaaS - *Software as a Service*). Cada um deles difere no nível de controle, flexibilidade e gerenciamento (AWS, 2023):

- IaaS - Consiste em oferecer componentes básicos de computação em nuvem, normalmente recursos de rede, servidores (virtuais ou físicos) e espaço para armazenamento de dados, sendo um modelo que disponibiliza um nível de flexibilidade e gerenciamento sobre os recursos de TI mais alto que os demais;
- PaaS - É um modelo com foco no desenvolvimento e gerenciamento de aplicações, sem ter de se preocupar com a infraestrutura adjacente (sistemas operacionais e hardware), e, dessa forma, o contratante não precisa se preocupar com alocação de recursos, planejamento de capacidade ou qualquer outro trabalho repetitivo; e
- SaaS - É um serviço completo, oferecido e gerenciado pelo próprio CSP, que é comumente acessado via interface WEB, existindo, todavia, versões que disponibilizam API's (*Application Programming Interface*) para serem consumidas, sendo que este modelo descarta a necessidade de instalar a aplicação na própria infraestrutura, pelo próprio contratante (usuário), não sendo necessárias ações pelo usuário final para poder utilizar versões mais recentes lançadas do serviço.

Quaisquer dos modelos (IaaS, PaaS ou SaaS) podem ser implementados em diferentes tipos de nuvens, podendo ser: Nuvem Pública, Nuvem Privada, Nuvem Híbrida ou Multi-nuvem/Multi-nuvem híbrida.

2.1.1.1 Nuvem Pública

Nuvem pública é um tipo de computação em nuvem em que o CSP deixa todos os recursos de computação disponíveis para uso via internet. Tais recursos são comumente cobrados pelo tempo de uso (IBM, 2023a).

Provedores de *cloud* mantêm, gerenciam e assumem a responsabilidade por toda a infraestrutura utilizada pelos clientes. Eles normalmente disponibilizam larguras de banda altíssimas para garantir o desempenho requerido pelas aplicações sendo executadas (IBM, 2023a).

A Nuvem Pública é baseada no conceito *Multi-tenant*, ou seja, toda a infraestrutura do CSP é compartilhada pelos clientes, que são segregados de maneira lógica. Os serviços de Nuvem Pública mais utilizados atualmente são AWS, Google Cloud, IBM Cloud, Microsoft Azure and Oracle Cloud (IBM, 2023a).

2.1.1.2 Nuvem Privada

Nuvens Privadas são um tipo de computação em nuvem em que recursos computacionais são dedicados exclusivamente a uma única organização. Ao contrário da Nuvem Pública, em que a infraestrutura é compartilhada entre vários clientes, na Nuvem Privada os recursos são utilizados exclusivamente pela organização proprietária (IBM, 2023a).

Uma Nuvem Privada é normalmente hospedada em servidores locais na infraestrutura do próprio cliente, sendo que, todavia, pode também ser hospedada em uma Nuvem Pública, como por exemplo AWS *Outposts* (IBM, 2023a).

Muitas companhias optam pelo uso de Nuvem Privada para poderem atender requisitos rígidos de segurança. Outras escolhem devido ao tipo de dado que armazenam, que costuma ser sigiloso (IBM, 2023a).

2.1.1.3 Nuvem Híbrida

Como o nome já indica, a Nuvem Híbrida combina os dois tipos de nuvens acima. Com isso o cliente possui uma flexibilidade maior para adequar suas necessidades entre as nuvens públicas e privadas que possui, de forma que possa otimizar também os gastos (IBM, 2023a).

2.1.1.4 Multi-nuvem e Multi-nuvem Híbrida

A abordagem multi-nuvem consiste em usar dois ou mais CSPs. Portanto, as empresas que utilizam a abordagem Multi-nuvem, estão também fazendo uso de serviços IaaS, PaaS ou SaaS (IBM, 2023a).

A abordagem Multi-nuvem Híbrida abarca o uso de Nuvem Privada entre Nuvens Públicas.

Os clientes, quando optam pelo uso de mais de um CSP, o fazem para evitar depender de apenas um único CSP, isto é, para evitar SPoF (*Single Point of Failure* - Único Ponto de Falha). Entretanto, com essa abordagem, a complexidade para se gerenciar aplicações em cada uma das nuvens aumenta consideravelmente (IBM, 2023a).

2.1.2 Microsserviços

Microsserviços são um padrão arquitetural, baseado na arquitetura orientada a serviços, que consiste na arquitetura de software ser dividida em pequenos componentes autônomos e independentes, que se comunicam entre si por meio de APIs (*Application Programming Interface* - Interface de Programação de Aplicação) bem definidas. Por serem segregados e independentes, microsserviços possuem algumas vantagens como escalabilidade, rastreabilidade etc ([LIMA, 2018](#)).

Com a adoção do padrão de microsserviços, o sistema é fragmentado em componentes menores que possuem uma responsabilidade mais específica. Por exemplo, um serviço responsável pelo cadastro de usuários, outro serviço responsável pelo cadastro de produtos etc. Dessa forma, cada equipe tem mais liberdade para trabalhar no microsserviço sob sua responsabilidade, de maneira que eles possam ser escritos em linguagens diferentes, utilizar bancos diferentes, etc ([REDHAT, 2023](#)).

Para entender as vantagens de microsserviços, é preciso primeiro entender sobre outro padrão de arquitetura, o Monolito. Um Monolito é um padrão mais simples de arquitetura e que dominou a área de desenvolvimento de software por muitos anos. O padrão consiste em acoplar todas as regras de negócios e funcionalidades do sistema em uma única aplicação e, portanto, no caso do exemplo acima, os cadastros de usuários e produtos seria realizado pelo mesmo software.

No caso de um sistema pequeno, o padrão Monolito ainda é uma opção melhor para evitar as complexidades acarretadas pelo uso de microsserviços, visto que não há necessidade de fragmentar em porções menores um sistema que já é pequeno.

Contudo, no cenário de grandes sistemas, o uso do Monolito se torna uma grande dor de cabeça, pois qualquer alteração em uma parte do código pode afetar outra(s) parte(s), e se a aplicação parar de funcionar, todo o sistema ficará fora do ar. Outro problema também é na questão da escalabilidade, uma vez que você precisa replicar todo o software para resolver o gargalo de apenas uma parte dele.

Por estas razões, como alternativa a esses problemas, surgiu o padrão de arquitetura de Microsserviços, que se tornou um dos principais métodos de desenvolvimento de software ([FOWLER, 2014](#)). A Fig. 1 compara os padrões Monolito e Microsserviços. Em que no Monolito, todos os serviços estão acoplados em uma única aplicação, enquanto que os microsserviços, cada aplicação tem uma responsabilidade específica e um banco de dados específico também.

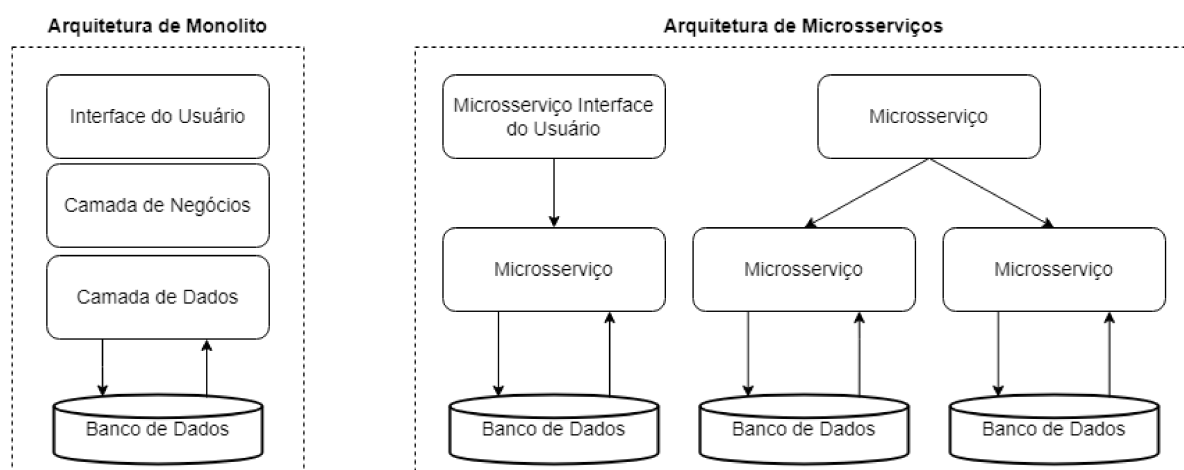


Figura 1 – Comparativo entre arquitetura de monolito e arquitetura de microsserviços. Fonte: Adaptado de (SUSE, 2021).

Diante disso, é possível elencar os benefícios do uso do padrão arquitetural de Microsserviços (SQUADRA, 2021):

- Independência para escalabilidade - A arquitetura de Microsserviços oferece autonomia e escalabilidade, por meio da capacidade de implantar e alterar servidores independentes para atender à demanda de serviços específicos, sendo que cada microsserviço pode ser escalado de forma adequada, resultando em uma infraestrutura mais eficiente e econômica devido à sua escalabilidade horizontal;
- Aceleração dos ciclos de desenvolvimento - A arquitetura de Microsserviços permite que vários desenvolvedores trabalhem simultaneamente em diferentes componentes independentes de uma aplicação e isso resulta em ciclos de desenvolvimento mais rápidos, lançamentos no mercado mais ágeis e economia de tempo valioso para a empresa;
- Facilidade para experimentação - A arquitetura de Microsserviços permite entregas contínuas, facilitando o teste e a possibilidade de descartar ideias que não funcionem em um componente específico e, como cada microsserviço opera de forma independente, eventuais problemas não têm impacto significativo para a empresa, e as atualizações de código são eficientes, tornando mais fácil a experimentação no sistema, proporcionando maior flexibilidade e agilidade para a empresa explorar novas ideias;
- Abertura para a escolha de ferramentas e tecnologias - Microsserviços oferecem a desenvolvedores, a liberdade de escolher a melhor ferramenta ou recurso para solucionar problemas, resultando em uma abordagem personalizada e adaptável e

isso permite que a equipe selecione as tecnologias mais indicadas para cada caso, oferecendo maior flexibilidade e eficiência no desenvolvimento de software;

- Reutilização de aplicativos - A modularidade de um software dividido em módulos menores e bem definidos permite que a equipe utilize a mesma aplicação para diferentes finalidades, significando que um recurso desenvolvido para uma função específica pode ser reutilizado como componente em outros recursos, evitando assim a necessidade de escrever novo código, culminando que essa reutilização promove eficiência e agilidade no desenvolvimento de software;
- Maior tolerância a falhas - A independência de microsserviços fortalece a resiliência da aplicação em relação a possíveis falhas, ao contrário do formato monolítico, em que um erro em um componente pode causar a falha de todo o sistema, sendo que os microsserviços são independentes e não exigem que todo o aplicativo seja interrompido em caso de problema, significando que as falhas são isoladas e não impactam o funcionamento global da aplicação, garantindo maior estabilidade e disponibilidade do sistema;
- Referência para domínios - Microsserviços fornecem uma referência clara e bem definida para domínios de uma aplicação, significando que um componente específico é responsável por gerenciar e atualizar dados relacionados a um usuário e, então, quando outro serviço precisa acessar as informações desse usuário, ele deve buscar esses dados diretamente no microsserviço responsável por eles - sendo um exemplo comum disso é quando um usuário faz login em um site usando suas credenciais do Twitter ou do Facebook, onde as informações do usuário são buscadas a partir desses microsserviços específicos.

2.1.3 Kubernetes

Kubernetes é uma ferramenta de orquestração de contêineres que facilita o agendamento e a automatização da implantação, administração e escalabilidade de aplicativos contidos em contêineres ([KUBERNETES, 2023](#)).

Kubernetes e o ecossistema de contêineres estão se consolidando como uma plataforma de computação e ecossistema de propósito geral que rivaliza às máquinas virtuais (VMs) como os blocos básicos da infraestrutura e dos aplicativos em nuvem modernos ([IBM, 2023b](#)).

Um contêiner é uma unidade de software convencional que encapsula o código e todas as suas dependências, permitindo que o aplicativo seja executado de maneira ágil e confiável em diversos ambientes de computação. Por utilizarem da virtualização do sistema operacional, os contêineres simplificam a partilha de recursos, como CPU, memó-

ria, armazenamento e rede, proporcionando um método lógico de encapsular aplicativos, tornando-os independentes do ambiente no qual são executados. (IBM, 2023b).

O Kubernetes oferece várias vantagens para ambientes de desenvolvimento de aplicativos em nuvem, permitindo agendar e executar contêineres em *clusters* de máquinas virtuais ou físicas. Ele possibilita a implementação de uma infraestrutura baseada em contêineres e automatiza tarefas operacionais (REDHAT, 2020).

A Fig. 2 apresenta a arquitetura do orquestrador de contêineres Kubernetes.



Figura 2 – Arquitetura do kubernetes. Fonte: Adaptado de (REDHAT, 2020).

Os elementos do kubernetes presentes na fig. 2 são o seguinte (KUBERNETES, 2023):

- *Control Plane* - O Plano de Controle é uma coleção de processos que controlam os nós do Kubernetes, sendo a partir dessa coleção que todas as atribuições de tarefas têm origem;
- *kube-apiserver* - É responsável por expor a API do Kubernetes
- *kube-scheduler* - Responsável por observar quais *pods* recém-criadas estão sem nó, e selecionar um nó que atenda aos requisitos para executar a *pod*.
- *kube-controller-manager* - É o componente responsável por executar os processos de controlador, como controlador de nó, controlador de *job*, etc.

- *etcd* - Banco de dados do tipo chave-valor que armazena todos os artefatos criados dentro do Kubernetes.
- *Nós de dados* - São máquinas que executam as tarefas solicitadas atribuídas pelo Plano de Controle;
- *kubelet* - É um serviço executado nos *Nodes*, que lê os manifestos dos contêineres e garante que os contêineres definidos sejam iniciados e executados;
- *kube-proxy* - É um *proxy* de rede executado em todos os nós do *cluster*, para que seja possível a comunicação entre os serviços instalados entre si e com o exterior.
- *Pod* - Um "*Pod*" é um conjunto de um ou mais contêineres implantados em um único nó, sendo que todos os contêineres em um *Pod* compartilham um endereço IP, nome de *host* e outros recursos, e os *Pods* abstraem a rede e o armazenamento do contêiner subjacente, o que facilita a movimentação dos contêineres pelo *cluster*;
- Persistência de Dados - Comumente é um disco rígido atrelado a um dos nós do *cluster*, para que sirva de armazenamento para alguma *pod*.
- Repositório de Imagens - Local utilizado para armazenar todas as imagens que serão utilizadas para a criação de contêineres e consequentemente *Pods* dentro do Kubernetes.

2.1.4 IaC

A Infraestrutura como Código (IaC) é uma abordagem que consiste no gerenciamento e provisionamento da infraestrutura por meio de código em vez de utilizar processos manuais (REDHAT, 2022).

A IaC disponibiliza arquivos de configuração que incluem detalhes sobre a estrutura da sua infraestrutura, simplificando a capacidade de ajustar e disseminar essas configurações. Adicionalmente, essa abordagem assegura a criação consistente do mesmo ambiente sempre que for requerido. Ao converter as especificações de configuração em código e documentá-las, a IaC apoia a administração das configurações e impede a ocorrência de mudanças de configuração que não estejam registradas ou tenham sido realizadas de forma improvisada (REDHAT, 2022).

2.1.4.1 Terraform

O Terraform é uma ferramenta de IaC que permite criar recursos em nuvem ou locais (*on-premises*) utilizando arquivos de configuração legíveis por humanos, que podem ser versionados, reutilizados e compartilhados. Ele é capaz de gerenciar componentes

de baixo nível, como recursos de computação, armazenamento e redes, bem como componentes de alto nível, como entradas DNS (*Domain Name System*) e recursos de SaaS (SHAFIQUE, 2023).

Os arquivos de configuração do Terraform são declarativos, ou seja, eles descrevem o estado final da infraestrutura. Não é necessário escrever instruções passo a passo para criar recursos, pois o Terraform lida com a lógica subjacente. Ele constrói um grafo de recursos para determinar dependências entre eles e cria ou modifica recursos não dependentes em paralelo. Isso permite que o Terraform provisione recursos de forma eficiente (HASHICORP, 2023), cujo provisionamento pode ser dividido em duas partes principais:

- Terraform *Core*: que avalia o estado atual da infraestrutura e compara com novas configurações a ser aplicadas, com isso ele propõe um plano para adicionar ou remover componentes (SHAFIQUE, 2023); e
- Terraform *Plugin*: que fornece um mecanismo para o Terraform *Core* se comunicar com o CSP, por exemplo AWS, Azure ou Google Cloud, sendo o Terraform *Providers* um exemplo de *plugin*, que realiza essa comunicação com o Terraform *Core* via RPC (*Remote Procedure Call*) (SHAFIQUE, 2023).

A Fig. 3 demonstra o que foi explicado acima acerca da interação entre o Terraform *Core* e Terraform *Plugin* para se comunicarem com os provedores de infraestrutura.

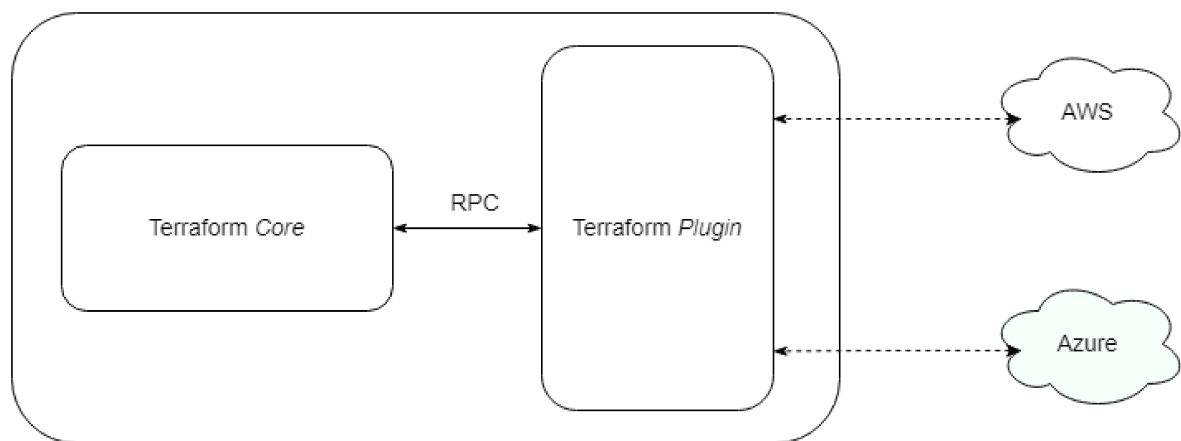


Figura 3 – Definição do Terraform. Fonte: Adaptado de (SHAFIQUE, 2023).

2.2 Estado da Arte

Esta seção tem por objetivo mostrar o estado atual das tecnologias descritas na Seção 2.1, a respeito dos fundamentos na área do projeto.

2.2.1 Cloud Computing

Atualmente existem 3 CSPs que se destacam no mercado, sendo eles: AWS da Amazon; Azure da Microsoft; e Google *Cloud Platform* (GCP) da Google. Cada uma desses CSPs possui suas características e clientes que se destacam. Por exemplo, um cliente de destaque da AWS é a Netflix, com sua infraestrutura de *streaming* de mídia. Já a Azure possui a Coca-Cola como cliente e na GCP, que apesar de ser mais nova que as outras plataformas, possui a SpaceX como cliente (PLETCHER, 2022).

2.2.1.1 AWS

A AWS é dividida entre Regiões, Zonas de Disponibilidade (AZ - *Availability Zones*) e os chamados *Edge Locations*. Ao todo, a AWS possui 22 regiões, localizadas em todo o mundo, 14 AZ's e 114 *Edge Locations* (BUCKBEE, 2022).

As regiões cobrem uma área geográfica como um estado ou país, e as AZ's são *Data Centers* dentro das regiões. As zonas de disponibilidade estão localizadas o mais distante possível umas das outras dentro de sua região para garantir que não haja interrupções no serviço caso uma AZ seja afetada por um desastre natural ou outro tipo de evento em grande escala.

As zonas de borda são *caches* que atuam de maneira semelhante às redes de distribuição de conteúdo (CDN - *Content Delivery Network*), armazenando em *cache* o conteúdo da Web mais próxima à localização do usuário para entrega e tempos de resposta mais rápidos (BUCKBEE, 2022).

A AWS oferece suporte a todos os sistemas operacionais e geralmente é classificada como a principal plataforma IaaS em termos de disponibilidade, desempenho confiabilidade e quantidade de aplicativos (BUCKBEE, 2022).

2.2.1.2 Azure

Azure é conhecida como uma plataforma sólida e integrada para empresas que já dependem da padronização baseada em Windows e seus aplicativos Sharepoint, Office 365 etc. A Azure superou alguns obstáculos para competir diretamente com a AWS. Uma característica surpreendente é a sua compatibilidade com plataformas de contêineres Linux (BUCKBEE, 2022).

A força da Azure sempre foi como provedor de IaaS. Além disso, ela oferece aplicativos de servidor embutidos e prontos para serem executados que suportam uma variedade de linguagens, incluindo .NET, Java, PHP, Node.js e Python. A plataforma está disponível em 54 regiões ao redor do mundo, com serviços projetados para aumentar a produtividade ao implementar a tecnologia mais atual. Também é uma das nuvens corporativas mais fáceis de configurar e operar (BUCKBEE, 2022).

2.2.1.3 Google Cloud

Em termos de provedores de infraestrutura como serviço (IaaS), Google *Cloud Platform* é relativamente novo. Ele oferece suporte a várias gerações do Linux, além das versões do servidor Windows até 2016. Em 2018, ele havia expandido para 21 regiões, cada uma dividida em no mínimo três zonas.

Isso lhe dá uma abrangência menor em comparação com os outros dois provedores, mas o Google está tentando compensar essa limitação de outras maneiras (BUCKBEE, 2022). Uma delas é a inovação na implantação de servidores submarinos, com um sistema exclusivo de cabos que parte da Ilha de Guam e se conecta a servidores na Austrália, no Pacífico Sul, na Ásia, no Japão e no continente dos EUA (BUCKBEE, 2022).

2.2.2 Microserviços e Kubernetes

Como visto na Seção 2.1, Microserviços é uma arquitetura de desenvolvimento de software que consiste em dividir o sistema em partes menores e independentes entre si. Porém com o crescimento da aplicação, gerenciar todos os microserviços pode se tornar um problema. Com isso, surgiu o Kubernetes, o orquestrador de contêineres que facilita o gerenciamento dos serviços do sistema.

Com os CSPs, não é mais necessário implantar um *cluster* Kubernetes do zero. Eles possuem suas próprias implementações, que agilizam e facilitam o processo de desenvolvimento e entrega, sendo que: Azure disponibiliza o AKS (*Azure Kubernetes Service*); AWS possui o EKS (*Elastic Kubernetes Service*); e Google Cloud tem o GKE (*Google Kubernetes Engine*).

Todos são implementações de Kubernetes e, portanto, possuem todas as *features* disponíveis pela ferramenta. Com isso é possível gerenciar os microserviços que são provisionados no *cluster* de maneira mais simples (POTASNICK, 2021).

3 Plataforma ADOTE: Modelo Conceitual

Modelos conceituais têm o objetivo de apresentar ideias, conceitos, construções etc, em um alto grau de abstração, de tal modo que atores (*Stakeholders*)¹ no processo de desenvolvimento do objeto possam compreender com clareza e sem ambiguidade aquilo que se objetiva.

Este capítulo tem o objetivo de apresentar a proposta de infraestrutura necessária para suportar o desenvolvimento da plataforma ADOTE, cujo objetivo foi apresentado no capítulo de introdução. Deste modo, serão apresentados diferentes diagramas que explicitam diferentes visões do projeto, estratificado em camadas (*layers*) e *tiers*, que constituirão o projeto (*design*) da infraestrutura de forma agnóstica.

3.1 Visão Geral da plataforma ADOTE

Inicialmente, é preciso entender quem são os atores que farão uso da plataforma. O ecossistema que a plataforma pretende integrar passa por pessoas, que de algum modo interagem com animais, por clínicas veterinárias, poder público (MP, Prefeitura, etc), terceiro setor (ONGs, OSs etc), lojas de animais (*Pet Shop*), entre outros. Diante disso, a Fig. 4 apresenta uma visão de alto nível dos tipos de atores, aos quais as funcionalidades da plataforma são destinadas.

A Fig. 4 oferece uma concepção inicial do que se espera da plataforma, tal como, ter um (*API Gateway*) por meio da qual diferentes atores podem realizar seus respectivos acessos, seja por integrações de sistemas legados, seja por meio de Aplicações Web (público geral).

O Ministério Público de Minas Gerais (MPMG), por meio da **Coordenadoria Estadual de Defesa dos Animais** (CEDA – criada pela Resolução PGJ N.º 50/2021), tem a missão de fortalecer e integrar a atuação dos Promotores de Justiça na defesa de animais “**como seres sencientes, sujeitos de direito despersonificados**”, fazendo jus à tutela jurisdicional em caso de violação de seus direitos, como forma de garantir o bem-estar desses animais, sempre em parceria com os órgãos públicos e com a sociedade civil.

A região de Uberlândia conta com inúmeras **Organizações Não Governamentais** (ONG) e Organizações Sociais (OS) reconhecidas por seus compromissos com animais, tais como, por exemplo, **APA** (Associação Protetora de Animais), **Projeto Ajuda**

¹ **Atores** são todos **aqueles** de algum modo **terão interações com a plataforma**, podendo ser pessoas, instituições, negócios, profissionais etc.

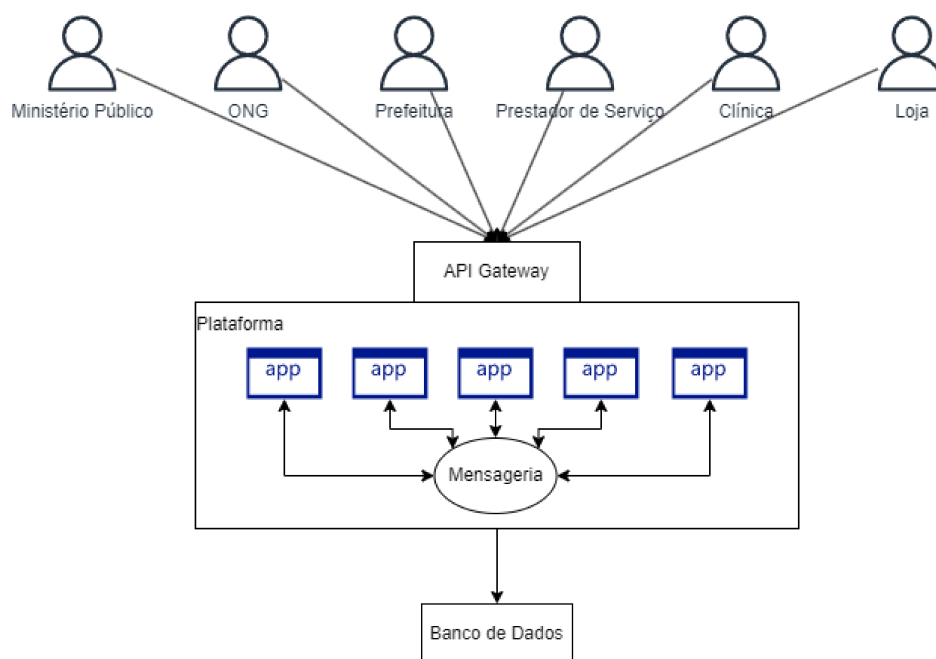


Figura 4 – Plataforma ADOTE: Visão Geral.

4 Patas, Liga das Patinhas, entre outras. Ao todo, são mais de 250 organizações e pessoas individuais, que formam uma rede de proteção, somente em Uberlândia.

A Prefeitura de Uberlândia (PMU), por meio do Centro de Zoonoses, é responsável por promover e executar ações voltadas para o controle de doenças causadas por animais, sendo, portanto, um órgão fundamental nesta rede. Hoje, laboratórios trabalham com diagnósticos de endemias, como é o caso da Malária e da leishmaniose canina, e com pesquisas para a captura e identificação de vetores².

No âmbito da sociedade civil, há pessoas físicas e jurídicas (**prestadores de serviços, farmácias, clínicas veterinárias, lojas** etc) que prestam atendimento médico, vendem medicamentos, vendem insumos domésticos para animais etc. Estes são aquelas pessoas que de algum modo estão envolvidos na recepção, tratamento, acolhimento e adoção de animais em situação de risco.

Há que se incluir neste rol, um tipo de pessoa, que hoje não se envolve por falta de meios, que são aquelas pessoas que se movem pela cidade e que muitas vezes são testemunhas de ocorrências de abandonos, animais soltos/perdidos, maus tratos e atropelamentos. Este tipo de pessoa é extremamente importante para notificar ocorrências para a rede de atenção.

Observe que se tratam de diferentes atores, com diferentes responsabilidades, que requerem diferentes serviços. A Fig. 4 apresenta esses diferentes **serviços**, com funções específicas, representados por **Módulos** do tipo "app".

² Vetores são organismos que transmitem doenças infecciosas entre seres humanos e animais

Para **desacoplar Módulos** do tipo **app**, tornando-os independentes uns dos outros, de tal forma que um eventual mal funcionamento ou queda não impacte nos demais, é inserida uma **camada de Mensageria** (*Middleware*), que será a responsável por receber, armazenar e entregar mensagens entre os referidos módulos.

3.2 Tiers da plataforma ADOTE

O Modelo de *Tiers* separa a organização de componentes de um sistema em camadas distintas (planos), em que cada uma desempenha um papel específico e tem uma responsabilidade clara. O objetivo principal desse modelo é separar preocupações e tornar o sistema mais escalável, flexível e fácil de manter. A Fig. 5 apresenta os *tiers* da Plataforma ADOTE.

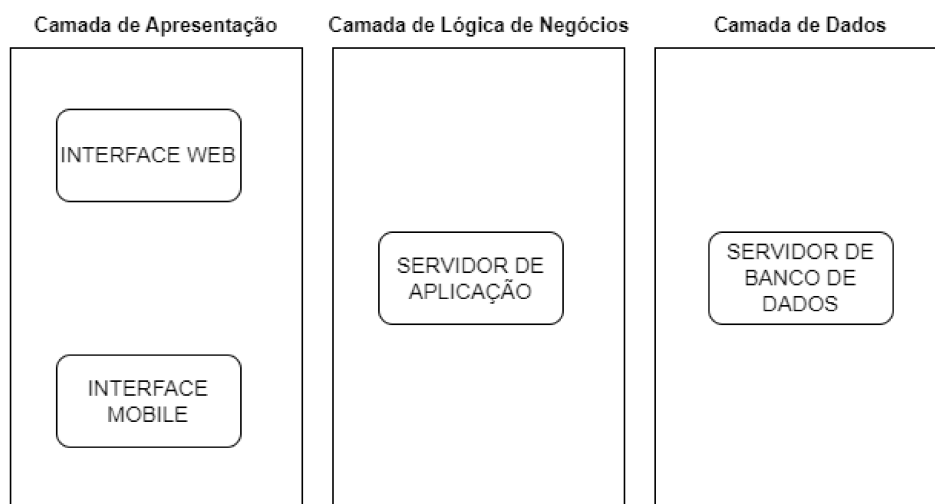


Figura 5 – *Tiers* da Plataforma ADOTE

A camada de Apresentação é responsável por endereçar os aspectos relativos à interface do Usuário. Ela lida com a apresentação de informações ao usuário e recebe entradas de dados. Isso inclui, páginas da web, aplicativos móveis etc.

A camada de Lógica de Negócios processa as solicitações da camada de Apresentação, realiza cálculos, executa acessos a bancos de dados e toma decisões de negócios. É nela que as funcionalidades do sistema são processadas.

A camada de Dados, conceitualmente referenciada como camada de Persistência, lida com o armazenamento e a recuperação de dados. Isso geralmente envolve bancos de dados, sistemas de arquivos etc. Ela fornece acesso a dados necessários para a camada de Lógica de Negócios.

Este trabalho foca na criação da infraestrutura que comporta as camadas de Lógica de Negócios e de Dados. Especifica em que parte do Servidor de Aplicação se encontram os

Microserviços, Mensageria³, *API Gateway* etc, como apresentado na Fig. 4. O servidor de Banco de Dados será responsável por armazenar as informações necessárias para o funcionamento da plataforma.

3.3 Arquitetura conceitual da plataforma ADOTE

Por fim, a Fig. 6 apresenta uma visão genérica de como se espera que seja a infraestrutura para suportar essa plataforma. O diagrama serve de base para a criação da arquitetura utilizando um CSP específico.

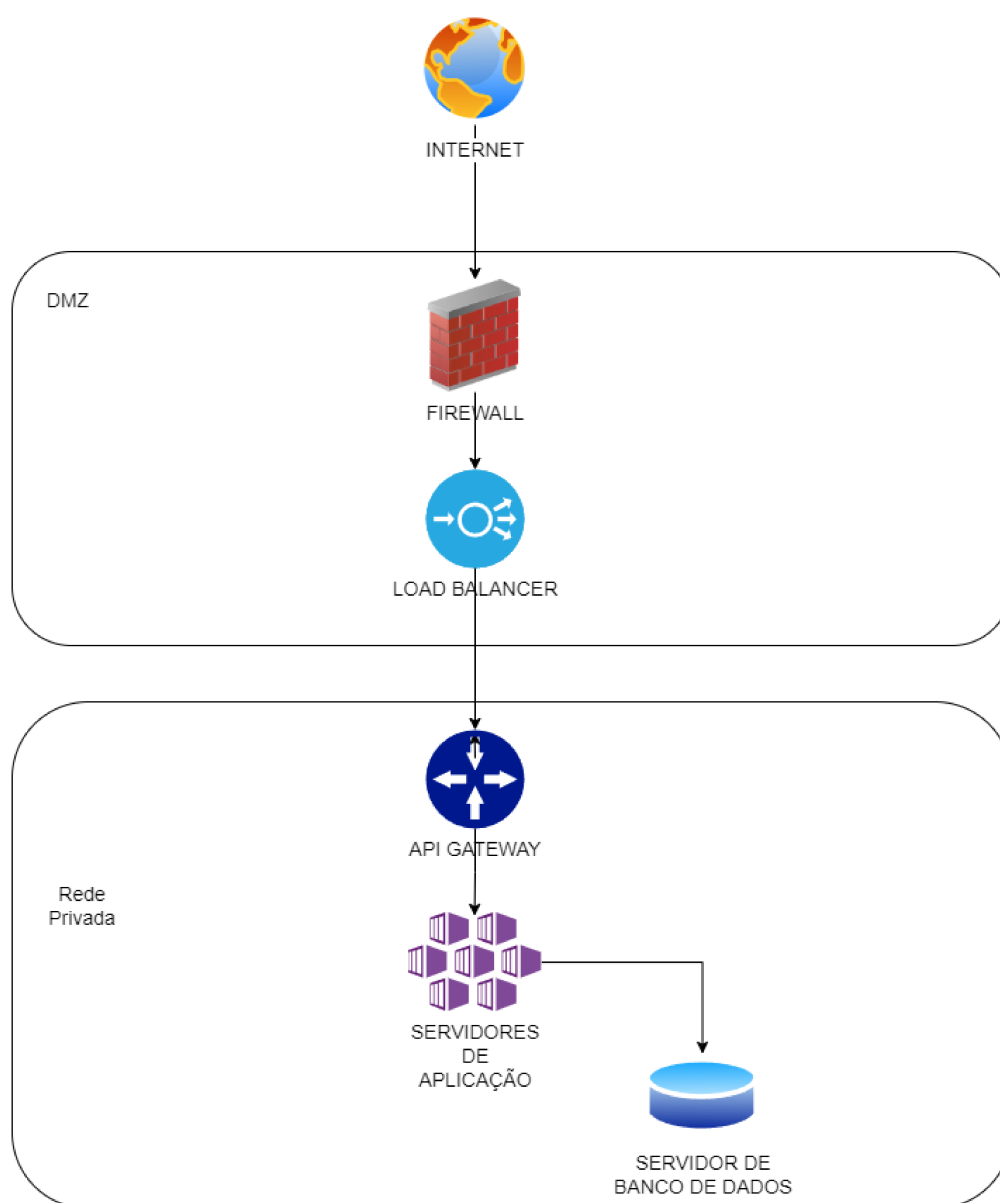


Figura 6 – Infraestrutura Genérica.

³ Por Mensageria entenda-se Serviços de Passagem de Mensagens, *Publisher/Subscriber*, entre outros

A DMZ (*Demilitarized Zone* - Zona Desmilitarizada) desempenha um papel crucial como uma camada intermediária entre a rede interna e a internet, atuando como uma linha de defesa adicional. Seu papel na arquitetura do projeto é essencial para mitigar potenciais ameaças externas, filtrar e monitorar o tráfego de dados, proporcionando uma barreira de proteção robusta para a rede interna. Ao criar essa zona desmilitarizada, a arquitetura ganha em termos de segurança, prevenindo efetivamente ataques antes que alcancem a infraestrutura interna.

O *Firewall*, integrado à arquitetura do projeto, opera como um guardião virtual que estabelece regras de segurança para delimitar e controlar o acesso à rede privada. Sua função é crucial na proteção da arquitetura contra tentativas não autorizadas de acesso, agindo como uma barreira essencial. Com isso, contribui para a integridade e segurança dos dados armazenados na rede interna, desempenhando um papel central na arquitetura de segurança do projeto.

O *Load Balancer*, integrado à arquitetura, assume a responsabilidade de otimizar a distribuição da carga de tráfego de dados entre as aplicações instaladas no ambiente. Seu papel é vital na arquitetura do projeto, proporcionando eficiência operacional e resistência do sistema em face de picos de demanda. Ao equilibrar a carga, contribui significativamente para a estabilidade e desempenho geral do projeto.

O *API Gateway*, como componente central da arquitetura, age como um elemento intermediário estratégico entre os clientes e os microsserviços. Ele desempenha um papel fundamental na exposição controlada das APIs, simplificando o gerenciamento e fortalecendo a segurança na arquitetura do projeto. Além disso, proporciona governança sobre as interações entre clientes e serviços, integrando funcionalidades como autenticação e autorização.

Os Servidores de Aplicação, parte essencial da arquitetura, são necessários para hospedar os microsserviços. Seu papel é crítico na garantia da disponibilidade, desempenho e escalabilidade das aplicações. Ao fornecer o ambiente de execução necessário, contribuem para a eficiência operacional e o sucesso geral do projeto.

Por fim, o Servidor de Banco de Dados, integrado à arquitetura, desempenha a persistência no armazenamento seguro e eficiente dos dados. Sua função é central na arquitetura do projeto, atuando como o repositório central para informações críticas. Ao separar os servidores de aplicação e banco de dados, a arquitetura otimiza o desempenho geral do sistema e adota uma abordagem mais segura em relação ao armazenamento de dados sensíveis.

4 Desenvolvimento

Este capítulo abordará o desenvolvimento da infraestrutura do projeto, de forma a simular como seria em um ambiente de produção.

4.1 Definição da Arquitetura de Infraestrutura

Note que a arquitetura de infraestrutura desenvolvida para a plataforma é uma especificação do diagrama mostrado na Fig. 6, apresentado no final do Capítulo anterior. Contudo, agora especificando as tecnologias utilizadas, em uma *Cloud* específica, conforme se pode ver na Fig. 7.

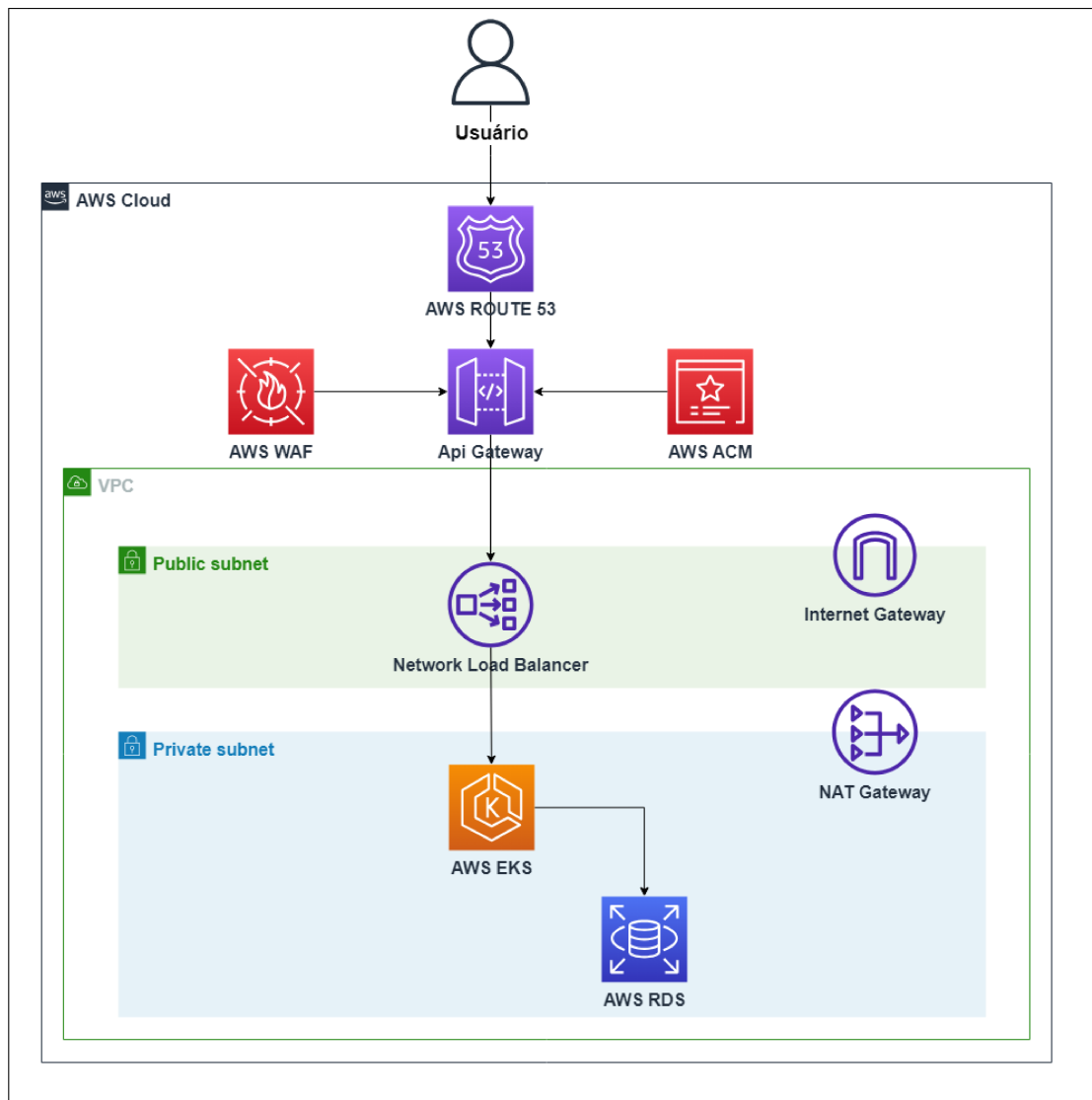


Figura 7 – Diagrama da Infraestrutura.

Os componentes representados na Fig. 7, são todos eles serviços AWS, descritos a seguir:

- **AWS VPC:** VPC (*Virtual Private Cloud*) é o principal serviço para definir a rede da infraestrutura, especificando endereços de IP, gateways, *subnets*, etc;
- **AWS Subnet:** Serviço da VPC responsável por gerenciar as faixas de rede utilizadas e especificar se serão redes públicas ou privadas;
- **AWS Route53:** Serviço responsável por criar as zonas de DNS e roteamento;
- **AWS API Gateway:** Serviço responsável por ser o ponto de entrada para aplicações REST (*Representational State Transfer*) ou serviços HTTP (*Hypertext Transfer Protocol*);
- **AWS ACM:** ACM (*AWS Certificate Manager*) é o gerenciador de certificados TLS (*Transport Layer Security*);
- **AWS WAF:** WAF (*Web Application Firewall*) é o serviço de *firewall* de aplicação responsável por filtrar endereços IP e regiões que podem acessar o sistema;
- **AWS NLB:** NLB (*Network Load Balancer*) é um balanceador de carga que trabalha na camada de Transporte da Arquitetura Internet, ou seja, na camada 4;
- **AWS Internet Gateway:** desempenha o papel de *gateway* responsável por permitir tráfego de entrada e saída entre a *subnet* pública e a Internet;
- **AWS NAT Gateway:** desempenha o papel de um *gateway* responsável por permitir que aplicações na *subnet* privada consigam acessar a Internet, sendo que a Internet não consegue acessar as aplicações;
- **AWS EKS:** EKS (*Elastic Kubernetes Service*) é o serviço gerenciado para Kubernetes;
- **AWS RDS:** RDS (*Relational Database Service*) é o serviço gerenciado para bancos relacionais.

Para fins de demonstração, neste projeto foram criados apenas o RDS, EKS e *Network Load Balancer*.

4.2 Configuração dos Recursos Terraform

Como a plataforma é baseada em arquitetura de Microserviços, é necessária a criação de um *cluster* kubernetes para suportar as aplicações e, com isso, será utilizado o EKS, serviço de kubernetes gerenciado pela AWS.

Outro recurso necessário é um banco de dados relacional, para persistir os dados da plataforma, sendo que para esta finalidade será utilizado o RDS com Postgres, que é também um serviço gerenciado da AWS.

Ambos os recursos serão criados utilizando Terraform e, dentro de cada arquivo, que fará o provisionamento dos recursos supracitados, é necessário configurar qual o CSP especificado, sendo que nesse caso foi definida a AWS, assim como mostra a Fig. 8.

```
provider "aws" {  
  region                = "us-east-1"  
  shared_credentials_files = ["$HOME/.aws/credentials"]  
  profile                = "default"  
}
```

Figura 8 – AWS Provider.

Para o banco de dados especificado para o projeto, o RDS Postgres, são especificadas as configurações a seguir:

- Primeiro, foi criado um grupo de segurança (*Security Group*), outro serviço da AWS responsável por agir como um *firewall*, especificando quais Portas (TCP *Ports*) e quais endereços IP poderão acessar o serviço, conforme mostra a Fig. 9.

```
module "tcc_security_group" {  
  source = "../../modules/security-group"  
  
  name      = "rds-tcc-sg"  
  vpc_id    = "vpc-0250de01ab975f659"  
  cidr      = "0.0.0.0/0"  
}
```

Figura 9 – Security Group Module.

- Segundo, foi criado um grupo de parâmetros (*Parameter Group*), com as configurações especificadas para o RDS Postgres, conforme mostra a Fig. 10.

```
module "tcc_parameter_group" {  
  source = "../../modules/rds/parameter-group"  
  
  name      = "tcc-parameter-group"  
  family    = "postgres15"  
}
```

Figura 10 – Parameter Group Module.

- Terceiro, foi criado um grupo de sub-redes (*Subnet Group*), para definir quais redes o RDS Postgres poderá utilizar, conforme mostra a Fig. 11.

```
module "tcc_subnet_group" {  
  source = "../../modules/rds/subnet-group"  
  
  name           = "tcc-subnet-group"  
  vpc_id         = "vpc-0250de01ab975f659"  
  subnet_tag     = "pub-*"  
}
```

Figura 11 – Subnet Group Module.

- Por último, em quarto, é criado o RDS Postgres de fato, com as configurações de versão da *engine* PostgreSQL, o tipo de máquina, o usuário e senha *master*, tamanho de disco, etc, conforme mostra a Fig. 12.

```
resource "aws_db_instance" "default" {  
  identifier      = "rds-tcc-facom"  
  db_name         = "tcc"  
  allocated_storage = 10  
  engine          = "postgres"  
  engine_version  = "15.3"  
  instance_class  = "db.t3.micro"  
  username        = "tccadmin"  
  password        = "tccadmin"  
  vpc_security_group_ids = [module.tcc_security_group.security_group_id]  
  parameter_group_name = module.tcc_parameter_group.parameter_group_name  
  db_subnet_group_name = module.tcc_subnet_group.subnet_group_name  
  skip_final_snapshot = true  
  publicly_accessible = true  
  multi_az           = false  
}
```

Figura 12 – RDS Resource.

Com essas especificações configuradas, é possível criar um RDS Postgres na AWS em apenas alguns minutos.

Para o EKS, as configurações especificadas foram um pouco diferentes. Foi definido *module*, em detrimento de *resource*, pois dessa maneira é possível enviar todos os argumentos necessários para a criação do *cluster* uma única vez, sem precisar repetir valores. A Fig. 13 mostra esta configuração.

```

module "aws-eks" {
  source = "../../modules/eks"

  eks_cluster_role_name = "tcc-eks-cluster-role"
  aws_vpc_id             = "vpc-0250de01ab975f659"
  security_group_name    = "tcc-sg"
  cidr_blocks            = ["10.0.0.0/16"]
  eks_cluster_name       = "tcc-eks"
  eks_cluster_subnet_ids = ["subnet-07122fb09143f9895", "subnet-080aba3196de29235", "subnet-0f42628bacd64cd43"]
  security_group_name_nodes = "tcc-eks-nodes-sg"
  eks_nodes_role_name    = "tcc-eks-node-role"
  eks_nodes_subnet_ids   = ["subnet-07122fb09143f9895", "subnet-080aba3196de29235", "subnet-0f42628bacd64cd43"]
  node_group_name        = "tcc-eks-nodes"
  disk_size              = "20"
  instance_types         = ["t2.micro"]
  pvt_desired_size       = 1
  pvt_min_size           = 1
  pvt_max_size           = 1
  eks_key_name            = "tcc-key"
  region                 = "us-east-1"
  aws_profile             = "default"
}

```

Figura 13 – EKS Module.

4.3 Criação dos Recursos na AWS

Para fazer a criação de fato dos recursos, é necessária utilização da CLI (*Command Line Interface*) do terraform com os seguintes comandos:

```

1    $ terraform init
2    $ terraform plan
3    $ terraform apply

```

Caso os comandos tenham sido executados com sucesso, é possível consultar na interface da AWS os recursos criados, conforme mostram a Fig. 14 e a Fig. 15.

DB identifier	Status	Role	Engine	Region & AZ	Size	Actions	CPU	Current activity	Maintenance	VPC
rds-tcc-facom	Available	Instance	PostgreSQL	us-east-1b	db.t3.micro	-	11.72%	0 Connections	none	vpc-0250de01ab975f659

Figura 14 – RDS Criado.

Cluster name	Status	Kubernetes version	Provider
tcc-eks	Active	1.27	EKS

Figura 15 – EKS Criado.

4.4 Disponibilização via Github

Todo o desenvolvimento mostrado acima está versionado no github, assim como a aplicação criada para auxiliar nos testes da infraestrutura.

Os códigos podem ser encontrados no seguinte repositório github <<https://github.com/SantosVictor1/projeto-adote>>. Cada diretório possui seu próprio arquivo *README* que possui algumas orientações para o uso.

5 Conclusão

Este trabalho contribuiu com minha formação em várias frentes. Sem dúvida, o aspecto tecnológico foi uma delas, mas há outras frentes como a redação de um documento e a troca de ideias para o projeto. Toda a vivência em TCC1 e agora em TCC2 foram bem interessantes, nesta frente intangível.

Após o estudo sobre computação em nuvem e o orquestrador de contêineres kubernetes, pude perceber a destacada importância dessas tecnologias na indústria, bem como os benefícios do uso de Microsserviços para melhorar a escalabilidade, desempenho e disponibilidade das aplicações.

A utilização do Terraform como ferramenta de IaC demonstrou ser crucial para uma gestão eficiente e escalável da infraestrutura. Com ela, o projeto possui uma base *open ended* sólida, que permitirá futuras expansões e refinamentos.

Isto introduz uma capacidade de maneira que, caso sejam necessárias algumas modificações, basta alterar as configurações nos arquivos do Terraform, o que destaca a relevância contínua dessas tecnologias no desenvolvimento de sistemas robustos e escaláveis na era digital.

Estas disciplinas se mostraram efetivas para a consolidação da formação obtida por meio da grade curricular, bem como, mostram a importância destes momentos para pensar em planejamento, projeto, desenvolvimento, implantação e, principalmente, na manutenção dos entregáveis.

Referências

- AWS. **O que é computação em nuvem**. 2023. Acesso em 28 de abril de 2023. Disponível em: <<https://aws.amazon.com/pt/what-is-cloud-computing/>>. Citado 2 vezes nas páginas 12 e 13.
- BUCKBEE, M. **AWS vs Azure vs Google: Cloud Services Comparison**. 2022. Acesso em 20 de maio de 2023. Disponível em: <<https://www.varonis.com/blog/aws-vs-azure-vs-google>>. Citado 2 vezes nas páginas 21 e 22.
- FOWLER, M. **Microservices, a definition of this new architectural term**. março 2014. Acesso em 6 de maio de 2023. Disponível em: <<https://martinfowler.com/articles/microservices.html>>. Citado na página 15.
- HASHICORP. **What is Terraform**. 2023. Acesso em 27 de agosto de 2023. Disponível em: <<https://developer.hashicorp.com/terraform/intro>>. Citado na página 20.
- IBM. **What is cloud computing**. 2023. Acesso em 3 de maio de 2023. Disponível em: <<https://www.ibm.com/topics/cloud-computing>>. Citado 3 vezes nas páginas 12, 13 e 14.
- _____. **What is kubernetes**. 2023. Acesso em 5 de maio de 2023. Disponível em: <<https://www.ibm.com/topics/kubernetes>>. Citado 2 vezes nas páginas 17 e 18.
- KUBERNETES. **Kubernetes Overview**. 2023. Acesso em 5 de maio de 2023. Disponível em: <<https://kubernetes.io/docs/concepts/overview/>>. Citado 2 vezes nas páginas 17 e 18.
- LIMA, V. **O que são microservices?** 2018. Acesso em 28 de abril de 2023. Disponível em: <<https://blog.schoolofnet.com/o-que-sao-microservices/>>. Citado 2 vezes nas páginas 9 e 15.
- MACEIÓ, F. **Kitty revela dificuldades enfrentadas pelas ongs quanto ao abandono de animais**. **Câmara Municipal de Aracaju**, novembro 2021. Acesso em 28 de abril de 2023. Disponível em: <<https://www.aracaju.se.leg.br/institucional/noticias/kitty-revela-dificuldades-enfrentadas-pelas-ongs-quanto-ao-abandono-de-animais>>. Citado na página 10.
- NIKITA. **10 Reasons to Choose Cloud Computing**. 2021. Acesso em 28 de abril de 2023. Disponível em: <<https://www.cloudpanel.io/blog/10-reasons-to-choose-cloud-computing/>>. Citado na página 9.
- ONYSZKO, T. **Why should you use Kubernetes in 2022? Or why not?** 2020. Acesso em 28 de abril de 2023. Disponível em: <<https://www.predicagroup.com/blog/why-kubernetes-2022/>>. Citado na página 10.
- PLETCHER, S. **AWS vs Azure vs GCP: The big 3 cloud providers compared**. 2022. Acesso em 20 de maio de 2023. Disponível em: <<https://acloudguru.com/blog/engineering/aws-vs-azure-vs-gcp-the-big-3-cloud-providers-compared>>. Citado na página 21.

POTASNICK, A. **AKS vs EKS vs GKE: Managed Kubernetes services compared**. 2021. Acesso em 20 de maio de 2023. Disponível em: <<https://acloudguru.com/blog/engineering/aks-vs-eks-vs-gke-managed-kubernetes-services-compared>>. Citado na página 22.

REDHAT. **What is kubernetes?** 2020. Acesso em 5 de maio de 2023. Disponível em: <<https://www.redhat.com/en/topics/containers/what-is-kubernetes>>. Citado na página 18.

_____. **What is Infrastructure as Code (IaC)?** 2022. Acesso em 27 de agosto de 2023. Disponível em: <<https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac>>. Citado na página 19.

_____. **What are microservices?** 2023. Acesso em 6 de maio de 2023. Disponível em: <<https://www.redhat.com/en/topics/microservices/what-are-microservices>>. Citado na página 15.

SHAFIQUE, M. **What is Terraform? What is it Used For, How it Works**. 2023. Acesso em 27 de agosto de 2023. Disponível em: <<https://spacelift.io/blog/what-is-terraform>>. Citado na página 20.

SQUADRA. **Microserviços: quais são as vantagens de optar pela arquitetura?** 2021. Acesso em 6 de maio de 2023. Disponível em: <<https://www.squadra.com.br/blog/microservicos>>. Citado na página 16.

SUSE. **Microservices and Monolithic Architectures**. 2021. Image. Acesso em 6 de maio de 2023. Disponível em: <https://www.suse.com/c/wp-content/uploads/2021/09/rancher_blog_microservices-and-monolithic-architectures.jpg>. Citado na página 16.

TOCANTINS, G. Voluntários enfrentam dificuldades para manter abrigos de animais resgatados; 'dependem de nós'. **G1 Tocantins**, dezembro 2021. Acesso em 28 de abril de 2023. Disponível em: <<https://g1.globo.com/to/tocantins/noticia/2021/12/07/voluntarios-enfrentam-dificuldades-para-manter-abrigos-de-animais-resgatados-dependem-de-nos.ghtml>>. Citado na página 9.