CENTRO UNIVERSITÁRIO CARIOCA

DAVID ALVES DE ARAUJO

DEVOPS INFRAESTRUTURA COMO CÓDIGO

RIO DE JANEIRO

2020

DAVID ALVES DE ARAUJO

DEVOPS INFRAESTRUTURA COMO CÓDIGO

Trabalho de Conclusão de Curso apresentado ao Centro Universitário Carioca, como requisito parcial à obtenção do grau Bacharel em Ciência da Computação.

Orientador: Prof. David Bom Zanetti

RIO DE JANEIRO

2020

A658d Araujo, David Alves de
Devops infraestrutura como código / David Alves de Araujo. – Rio de Janeiro, 2020.
87 f.

Orientador: David Bom Zanetti
Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Centro Universitário UniCarioca, Rio de Janeiro, 2020

1.DevOps. 2. Infraestrutura como código. 3. Automatização de infraestrutura de TI. I. Zanetti, David Bom, prof. orient. II. Título.

CDD 005.1

DAVID ALVES DE ARAUJO

DEVOPS INFRAESTRUTURA COMO CÓDIGO

Trabalho de Conclusão de Curso apresentado ao Centro Universitário Carioca, como requisito parcial à obtenção do grau Bacharel em Ciência da Computação.

Aprovada em 2020

Banca Examinadora

Prof. David Bom Zanetti - Orientador

Centro Universitário Carioca

Prof. André Luiz Avelino Sobral

Centro Universitário Carioca

Prof. Sergio dos Santos Cardoso Silva - M.Sc.

Centro Universitário Carioca

Aos familiares e amigos, que me deram o suporte necessário durante esta longa jornada.

AGRADECIMENTOS

Gostaria de agradecer às seguintes pessoas, sem as quais eu não teria sido capaz de concluir este trabalho.

Em primeiro lugar a Deus no qual se fez cumprir a sua vontade em minha vida e me permitiu chegar até aqui.

A todo corpo docente da UniCarioca por todo aprendizado e apoio durante esse curso, em especial ao meu orientador, David Bom Zanetti, cuja visão e conhecimento me guiou nesta pesquisa.

Resumo

O mercado atual de TI é cada vez mais dominado pela "necessidade de velocidade". Essa necessidade é refletida no uso frequente de técnicas ágeis que reduzem o ciclo de desenvolvimento de software e também misturam atividades de desenvolvimento de software com operações de TI, essa tendência de usar abordagem de engenharia de software que reduz tempo e esforços entre o desenvolvimento e as operações de software, bem como a distância técnica e organizacional entre esses dois tipos de equipes de software, é conhecida como DevOps. Como parte do menu DevOps, muitas práticas envolvem automatização da criação ambientes computacionais utilizando infraestrutura como código, os tornando facilmente replicáveis, reutilizáveis e escaláveis. Esta pesquisa irá abordar a fundo essas práticas abordando os conceitos e suas ferramentas, desenvolvendo e entendendo como automatizar o provisionamento de recursos computacionais e como automatizar suas configurações.

Palavras-Chave

DevOps; Infraestrutura como código; Gerenciamentos de recursos em nuvem; automatização de infraestrutura de TI; software.

Abstract

The current IT market is increasingly dominated by the "need for speed". This need is reflected in the frequent use of agile techniques that reduce the software development cycle and also mix software development activities with IT operations, this tendency to use software engineering approach that reduce time and effort between software development and operations, as well as the technical and organizational distance between these two types of software teams , is known as DevOps. As part of the DevOps menu, many practices involve automating computer environments using infrastructure as code, making them easily replicable, reusable, and scalable. This research will address these practices in depth by addressing concepts and their tools, developing and understanding how to automate the provisioning of computational resources, and how to automate their settings.

Keywords

DevOps; Infrastructure as a Code; Cloud resource management; IT Infrastructure Automation; software.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
AWS	*Amazon Web Services*
DNS	*Domain Name System*
DSL	*Domain Specific Language*
HCL	*Hashicorp Configuration Language*
HTTPS	*Hypertext Transfer Protocol Secure*
IaC	*Infrastructure as Code*
IDE	*Integrated Development Environment*
JSON	*JavaScript Object Notation*
REST	*Representational State Transfer*
SDC	*Software Defined Computing*
SDI	*Software Defined Infrastructure*
SSH	*Secure Shell*
TCP	*Transmission Control Protocol*
VCS	*Version Control Systems*
VPC	*Virtual private cloud*
WEB	*www – World Wibe Web*
WinRM	*Windows Remote Management*
XML	*eXtensible Markup Language*
YAML	YAML *Ain't Markup Language*

SUMÁRIO

1. INTRODUÇÃO

1.1 Declaração do Problema

Configurar infraestruturas de TI sempre foi um procedimento longo e desafiador, especialmente no passado, no qual costumava ser um processo manual e tedioso. Os servidores eram fisicamente instalados rack por rack e os componentes de hardware eram configurados manualmente de acordo com os requisitos dos sistemas operacionais e dos aplicativos em funcionamento.

A instalação manual, configuração e manutenção da infraestrutura é um processo caro e demorado com alta chance de erro. Equipes especializadas são necessárias para realizar o trabalho de configuração, por exemplo, analistas de rede para configurar a rede, analistas de armazenamento para manter os drivers físicos etc. e imóveis devem ser adquiridos para receber todo este equipamento de hardware.

Além disso, esses enormes data centers precisam de manutenção, o que aumenta os custos extras de segurança e custos operacionais, como eletricidade e refrigeração.

Os servidores também estão sujeitos a erros de configuração e tendem a ser inconsistentes, uma vez que são fornecidos por muitos engenheiros, analistas e técnicos diferentes que não estão em comunicação constante entre si e não compartilham o mesmo escopo e metas. Isso muitas vezes leva a anormalidades e erros de configuração indesejados, que podem ser cruciais para a funcionalidade adequada de todo o sistema. Nesta infraestrutura tradicional, criar um ambiente isolado para teste, recuperação de desastres e simulações é muito caro e demorado para ser uma estratégia viável, e a única maneira para testar e melhorar o sistema é realmente experimentar um desastre, o que é altamente arriscado e estressante.

A introdução da computação definida por software apareceu como uma solução promissora para muitos desses problemas mencionados anteriormente. A ascensão da

tecnologia em nuvem está altamente relacionada a evolução da tecnologia de virtualização, na qual um aplicativo é abstraído do hardware que é emulado por uma camada de software chamada hipervisor. A combinação dessas tecnologias oferece uma maneira mais eficiente de instalar e configurar um ambiente computacional, de uma forma relativamente simples, o que resolveria os problemas de escalabilidade e agilidade da infraestrutura.

No entanto, muitas organizações de TI ainda enfrentam problemas com a inconsistência de configuração desses sistemas. Eles tendem a usar processos e estruturas que eram usadas para gerenciar software antes da introdução da computação em nuvem, e na maioria das vezes as ferramentas usadas são incapazes de acompanhar o curto tempo de provisionamento (segundos ou minutos) exigido pelos novos sistemas.

1.2 Motivação

A motivação surgiu ao estudar o conceito de Devops e infraestrutura como código, ao se deparar com as dificuldades que as organizações encontram para gerenciar sua infraestrutura computacional e integrar as equipes de desenvolvimento e operações, resolveu-se então realizar uma pesquisa com intuito de demostrar as possíveis soluções para esses problemas e os desafios para aplicação dessas soluções.

1.3 Objetivo

O objetivo deste projeto de pesquisa foi investigar a tecnologia de infraestrutura definida por software e sugerir como ela pode ser usada para melhorar a infraestrutura de TI estática de uma organização. O estudo da literatura desta pesquisa concentra-se nos conceitos e nas ferramentas de infraestrutura definidas por software disponíveis em cada camada de uma infraestrutura de TI. Com base no conhecimento adquirido a partir do estudo da literatura, foi proposta um estudo de caso de referência que visa demonstrar a automatização

de um sistema genérico a nível de infraestrutura, nível de rede, nível de computação e implantação, afim demostrar as diferentes ferramentas e conceitos definidos por código.

1.4 Estrutura do texto

O presente capítulo trata de uma breve introdução ao assunto chave deste trabalho: Infraestrutura como Código, além do objetivo e da motivação para a confecção do mesmo.

No capítulo 2 serão abordados temas fundamentais para a contextualização deste trabalho. A saber:

- Devops
- Computação Definida por Software e seus benefícios
- Computação em nuvem – Cloud
- Infraestrutura Definida por Código seus benefícios

No capítulo 3 serão abordadas as ferramentas para infraestrutura como código mais utilizadas no mercado e um comparativo entre essas tecnologias.

No capítulo 4 com o conhecimento adquirido nesta pesquisa será desenvolvido um estudo de caso, projetando um sistema definido por software do zero, automatizando o provisionamento da infraestrutura e das configurações de um sistema alvo genérico.

No capítulo 5 será apresentada a implementação do estudo de caso, serão mostrados os detalhes dos códigos desenvolvidos para cada ferramenta de infraestrutura como código, a integração e a interoperabilidade entre essas ferramentas e uma visão do geral do ambiente computacional automatizado.

No Capítulo 6 é apresentada uma conclusão para o trabalho, sendo mostrado o aprendizado com a implementação e oportunidades de melhoria.

2. CONCEITOS BÁSICOS

Neste capítulo serão apresentados os principais conceitos relacionados a este trabalho.

2.1 Devops

A necessidade cada vez maior de aumentar o ciclo de vida dos lançamentos de software e entregar atualizações cada vez mais rápidas, com o objetivo de alcançar maior satisfação do cliente estão mudando o pensamento das empresas.

Entretanto, as empresas de TI estão sofrendo com a falta de integração entre os departamentos, o que atrasa o processo de entrega final do software aos clientes (HUMMER; et al., 2013).

Envolvidos diretamente nessa entrega estão duas equipes distintas, a de desenvolvimento (DEV) responsável por desenvolver novos produtos de softwares, e a de Operações (OPS) que recebe esses softwares e são responsáveis por implementar esses softwares na infraestrutura, escalar a infraestrutura se necessário o que por sua vez inclui solicitar orçamento para os novos recursos de hardware e configurar toda essa nova infraestrutura (BRIKMAN, 2017).

Nos últimos anos, os sistemas em nuvem abstraíram o gerenciamento de muitos hardwares em camadas ajudando a equipe de operações a se concentrar em níveis mais elevados no ambiente de infraestrutura estando mais perto da camada de aplicação. Foi quando o movimento DevOps começou a se popularizar. Essa tendência de usar abordagem de engenharia de software que reduzem tempo e esforços entre o desenvolvimento e as operações de software, bem como a distância técnica e organizacional entre esses dois tipos de equipes de software, é conhecida como DevOps

DevOps baseia-se na ideia de que as equipes de desenvolvimento (Dev) e operações (Ops) trabalhem em conjunto em todo o ciclo de vida de desenvolvimento do software, desde o planejamento, passando pelo desenvolvimento até a implantação e melhoria contínua.

Portanto, a natureza do trabalho da equipe de operações foi modificada para ter as mesmas metodologias e técnicas dos desenvolvedores.

Segundo (EBERT; et al., 2016, p. 94, tradução nossa)[1] "DevOps significa uma mudança de cultura em direção à colaboração entre desenvolvimento, garantia de qualidade e operações.".

A partir de diferentes definições na literatura, DevOps pode ser considerado como uma cultura ou movimento que emergiu da metodologia ágil.

A metodologia ágil é considerada um fator significativo que habilita e apoia o conceito de DevOps em diferentes literaturas. (HOSONO, 2012) e (BANG; et al., 2013) acredita que a adoção do pensamento DevOps pode ser alcançada de forma eficaz por meio implementação de métodos e princípios ágeis, já que a metodologia ágil incentiva o compartilhamento de conhecimento, colaboração entre equipes, valores e processos e ferramentas.

(USTINOVA; JAMSHIDI, 2015) em seu estudo, viu DevOps como uma extensão da metodologia ágil em termos e princípios. Eles argumentaram que as atividades DevOps através do relacionamento próximo entre desenvolvedores e operadoras estendem e alavancam os princípios ágeis para todo a esteira de processos de entrega de software.

(HUMMER; et al., 2013, p. 94, tradução nossa)[2] considerou que "Um dos pilares do DevOps é a noção de Infraestrutura como Código (IaC)". Infraestrutura como código (IaC) é uma nova abordagem para configurar e gerenciar a infraestrutura de TI através de códigos em vez de processos manuais e procedimentos operacionais. A ideia do IaC é tratar os recursos de infraestrutura como se fossem software em um código, que permitam ao IaC utilizar as práticas de desenvolvimento de software e implantar ambientes de infraestrutura de forma rápida e consistente (MORRIS, 2016).

Esta pesquisa irá destacar o conceito de infraestrutura como código ou infraestrutura definida por software. Essa prática visa a automatização do provisionamento e das

[1] "DevOps means a culture shift toward collaboration between development, quality assurance, and operations."

[2] "One of the pillars of DevOps is the notion of Infrastructure as Code (IaC)"

configurações de ambientes computacionais, serão apresentados os seus níveis de abstração e as ferramentas que possibilitam esse processo de automatização, assim como o gerenciamento desses recursos.

2.2 Computação Definida por Software

Computação definida por software (SDC) usa técnicas de virtualização para calcular funções necessárias ao sistema (SDXCENTRAL, 2020). A tecnologia de virtualização desacopla CPU e recursos de memória do hardware físico, resultando em agrupamentos de recursos que podem ser usados onde quer que sejam necessários. A indústria usa duas tecnologias populares para alcançar a transformação SDC em seus sistemas: máquinas virtuais (VMs) e contêineres.

Uma VM é definida como uma estrutura isolada de software que executa seu próprio sistema operacional e aplicativos como uma máquina. Uma VM para funcionar requer um hipervisor, que é um software que é colocado entre o hardware e o sistema operacional.

O hipervisor fornece a capacidade para o hardware de compartilhar seus recursos entre as VMs executadas nele. O hipervisor é também responsável por gerenciar e monitorar as VMs.

Um contêiner por outro lado é uma unidade de software que empacota o código e todas as suas dependências, para que o aplicativo pode ser executado de forma rápida e confiável de um ambiente de computação para outro. Os contêineres isolam o software do ambiente e garantem a funcionalidade do aplicativo, independentemente das possíveis diferenças no ambiente computacional. É possível observar na figura 1 uma vantagem da tecnologia de conteinerização sobre VMs que consiste em não precisar de um sistema operacional totalmente funcional para funcionar, como as máquinas virtuais. (BERNSTEIN, 2014).

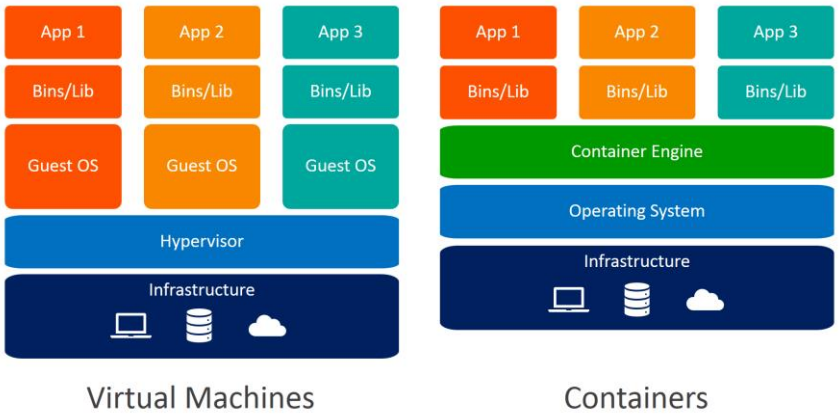

Figura 1 – Comparação de hipervisor e implantações baseadas em contêiner
Fonte: https://blogs.bmc.com/wp-content/uploads/2018/07/containers-vs-virtual-machines.jpg

2.2.1 Benefícios da Computação Definida por Software

SDC abstraiu o data center do hardware físico e adicionou flexibilidade e automação ao sistema, que era um grande desafio há uma década. Alguns dos benefícios mais importantes do SDC são (WEI, 2020).

- Agrupamento de recursos compartilhados - Uma das maiores vantagens do SDC é a elasticidade que oferece ao sistema. Por adicionar uma camada virtual entre o hardware físico e o sistema operacional, a utilização dos recursos torna-se mais fácil e muito mais flexível. Os recursos são agrupados em pools lógicos e as máquinas / contêineres virtuais podem usar e compartilhar o mesmo recurso físico. Mediante um aumento na demanda, as VMs / contêineres podem ser configurados para usar mais recursos, em contraste com os servidores físicos onde os recursos são limitados e é difícil e às vezes até impossível atualizar o hardware.
- Consolidação de servidores em qualquer lugar - Um servidor virtual é basicamente um arquivo e como arquivo pode ser implantado em qualquer lugar de uma maneira fácil. Pode ser implantado em um data center e atender quase todo o tráfego no laptop de um desenvolvedor de software para executar testes e desenvolver código ou pode ser

executado na nuvem, onde usuários em todo o mundo podem gerenciá-lo por meio de um navegador.

- Automação - Os níveis de automação nos novos sistemas aumentaram dramaticamente com a introdução do SDC. O sistema pode migrar automaticamente VMs e contêineres no caso de erro sem impacto real na experiência do usuário. O sistema também monitora automaticamente o desempenho e a carga de tráfego nas VMs / contêineres e faz operações necessárias para alcançar equilíbrio e eficiência entre as VMs /contêineres e os recursos solicitados. Além disso, o sistema pode iniciar automaticamente novos recursos em caso de alta demanda e desligue recursos não utilizados em períodos de baixa demanda.
- Priorização – O SDC fornece ao usuário a capacidade de dar maior prioridade a alguns servidores sobre outros. O administrador pode definir um limite mínimo para recursos de um servidor crítico, garantindo que o servidor tenha sempre os recursos necessários para executar algumas funções essenciais. Além disso, o administrador pode determinar a ordem em que os VMs/contêineres são atenuados em caso de erro. O sistema padrão não pode ser atendido por uma solução de tamanho único para tudo e o SDC pode resolver isso fornecendo a granularidade necessária para gerenciar cada servidor independentemente.

2.2.2 Ferramentas da Computação Definida por Software

Os principais elementos da tecnologia SDC, como já mencionado, são as técnicas de virtualização, como VMs e a tecnologia de conteinerização. O principal software, que cria e monitora as VMs, é o hipervisor. O hipervisor é basicamente uma camada de software que permite que várias VMs com diferentes sistemas operacionais sejam executadas simultaneamente em uma máquina física. O hipervisor pode ser dividido em duas categorias: os hipervisores Tipo 1 e Tipo 2. Hipervisores Tipo 1, são executados diretamente no hardware, enquanto o Tipo 2 é executado como um aplicativo em um sistema operacional

existente no hardware. As soluções tipo 1 mais usadas são KVM, Xen, VMWare ESXi, enquanto as soluções tipo 2 mais usadas são Microsoft Hyper-V e Oracle VirtualBox (KELSEY,2020).

A ferramenta mais popular para criar e implantar contêineres é o Docker. Outras ferramentas populares para executar contêineres são lxc, runc e rkt.

Neste projeto será utilizado o Docker como empacotador da aplicação automatizando a implantação da aplicação e do banco de dados nas máquinas virtuais na nuvem.

2.3 Computação em nuvem – Cloud

Nuvem é um modelo de computação onde servidores, redes, armazenamento, ferramentas de desenvolvimento e até mesmo aplicativos (apps) são habilitados por meio da internet. Em vez de as organizações terem que fazer grandes investimentos para comprar equipamentos, treinar funcionários e fornecer manutenção contínua, algumas ou todas essas necessidades são atendidas por um provedor de serviços em nuvem.

Existem cinco características principais de um ambiente de computação em nuvem, conforme definido pelo Instituto Nacional de Padrões e Tecnologia dos Estados Unidos da América (MELL; GRANCE, 2011):

- Autoatendimento sob demanda - Os serviços podem ser solicitados e provisionados rapidamente, sem a necessidade de instalação e configurações manuais.
- Acesso à Internet - Com um ambiente de nuvem pública, os usuários podem se conectar com os dados e aplicativos através de uma conexão à internet dando acesso a qualquer hora e qualquer lugar.
- Mensuração do serviço - Os serviços são cobrados conforme o uso dos recursos e esses recursos podem ser monitorados, controlados e relatados, proporcionando transparência para o provedor e consumidor do serviço utilizado.

- Agrupamento de recursos compartilhados - Na computação em nuvem muitas vezes é empregado o modelo multi-locação. Isso significa que um único aplicativo é compartilhado entre vários usuários. Assim, em vez de criar uma cópia do aplicativo para cada usuário, vários usuários podem configurar o aplicativo às suas necessidades específicas.
- Elasticidade - Plataformas de nuvem são elásticas. Uma organização pode escalar seus níveis de uso de recursos para cima ou para baixo rapidamente e facilmente à medida que as necessidades mudam.

A computação em nuvem é oferecida em três modelos de serviço diferentes, cada um satisfazendo um conjunto único de requisitos de negócios. Esses três modelos são conhecidos como:

- SaaS - Software como serviço: Software as a Service oferece aplicativos que são acessados pela web e não são gerenciados por sua empresa, mas pelo fornecedor do software. Isso alivia sua organização da pressão constante de manutenção de software, gerenciamento de infraestrutura, segurança de rede, disponibilidade de dados e todas as outras questões operacionais envolvidas em manter os aplicativos em funcionamento. A cobrança de SaaS é normalmente baseada em fatores como número de usuários, tempo de uso, quantidade de dados armazenados e número de transações processadas.
- PaaS - Plataforma como serviço: Este modelo está a meio caminho entre a infraestrutura como serviço (IaaS) e Software como serviço (SaaS). Ele oferece acesso a um ambiente baseado em nuvem no qual os usuários podem construir e entregar aplicativos sem a necessidade de instalar e trabalhar com IDEs (ambientes de desenvolvimento integrado), que muitas vezes são muito caros. Além disso, os usuários muitas vezes podem personalizar os recursos que desejam incluir com suas assinaturas.

- IaaS - Infraestrutura como serviço: Este modelo oferece uma maneira padronizada de adquirir recursos de computação sob demanda e pela web. Esses recursos incluem instalações de armazenamento, redes, poder de processamento e servidores virtuais privados. Eles são cobrados em um modelo “pague conforme o uso”, em que você é cobrado por fatores como a quantidade de armazenamento que você usa ou a quantidade de energia de processamento que você consome em um determinado intervalo de tempo. Nesse modelo de atendimento, o cliente não precisa dar suporte a infraestrutura, cabendo ao provedor garantir a quantidade contratada de recursos e a disponibilidade.

É possível ter uma visão pratica dos serviços prestados por cada modelo de computação em nuvem através da comparação feita na figura 2.

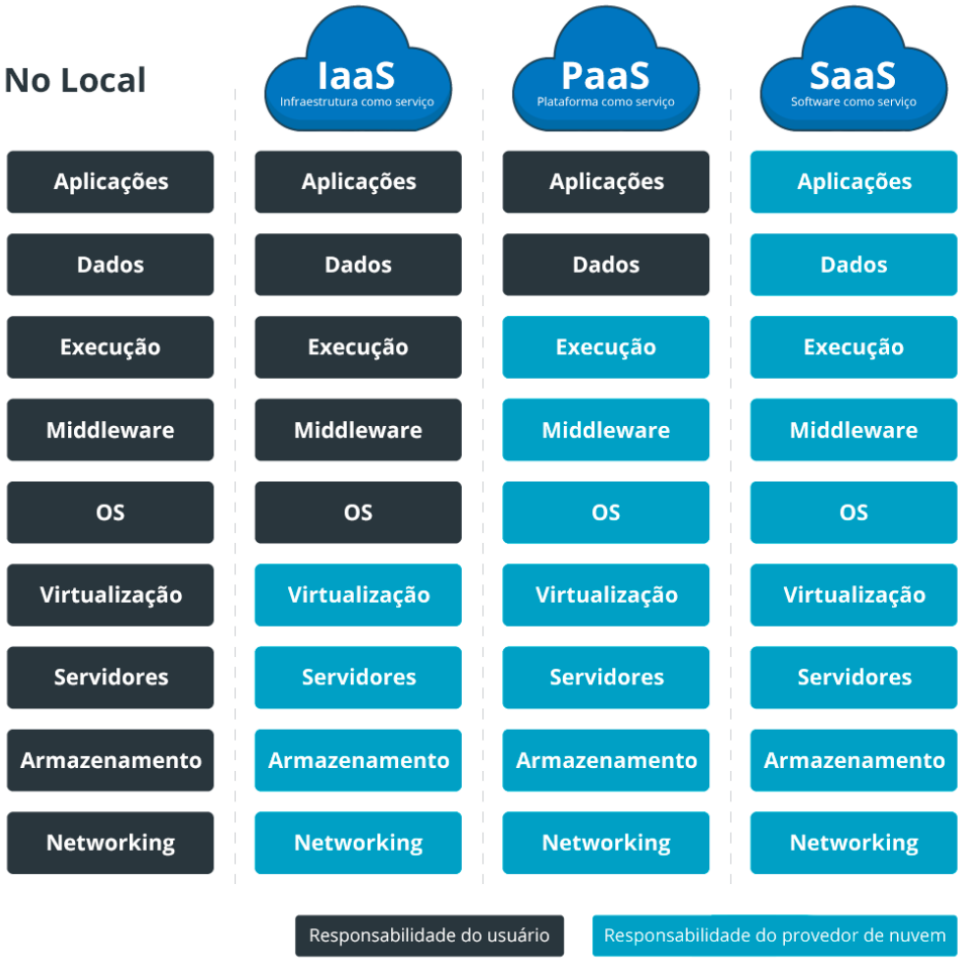

Figura 2 – Modelos de serviços em nuvem
Fonte: https://blog.trendmicro.com.br/wp-content/uploads/2019/12/Cloud-Service-Models-Comparison-01-Traduzido-1024x1024.png

2.4 Infraestrutura definida por código (SDI)

Nos últimos 5 anos, a indústria tem tentado dar o próximo grande passo em direção à melhoria da configuração e implantação de toda a infraestrutura de TI. Seu principal objetivo é se afastar completamente de qualquer dependência de hardware e criar uma plataforma de funcionalidade de software mais dinâmica e responsiva.

A Infraestrutura definida por software é um conceito que se refere à capacidade de controlar a infraestrutura como se fosse um software podendo alterar suas características por código (MORRIS, 2016).

A nova infraestrutura definida por software pode ser capaz de provisionar, configurar e conectar as tecnologias em diversos níveis da infraestrutura e assim finalmente suportar os aplicativos para o usuário final.

A introdução da nuvem e da virtualização foi seguida por muitas novas ferramentas e plataformas, o que levou a um grande portfólio de sistemas para as empresas de TI o que geralmente requer ainda mais tempo para manutenção. A indústria também evoluiu desde então, e a demanda por serviços mais flexíveis e facilmente acessíveis aumentou dramaticamente nos últimos anos.

Esta demanda crescente, combinada com a alta necessidade de lidar com o contínuo mundo crescente de TI forçou as organizações a buscar maneiras novas e eficazes de atender aos novos altos padrões da indústria. Como resultado, mais e mais organizações fizeram um passo na direção da abordagem definida por software. O nível da abordagem definida por software que se refere à infraestrutura de TI é a Infraestrutura definida por software (SDI) que é geralmente chamada de infraestrutura como código ou IaC pela indústria) é uma tentativa de atender à alta demanda de serviços de TI, maximizando o potencial da infraestrutura de TI atual. Uma definição para IaC dada por (MORRIS, 2016, p. 5, tradução nossa)[3] é:

> Infraestrutura como código é uma abordagem para automação de infraestrutura baseada em práticas de desenvolvimento de software. Ele enfatiza rotinas consistentes e repetíveis para provisionamento e alteração de sistemas e sua configuração. As

[3] "Infrastructure as code is an approach to infrastructure automation based on practices from software development. It emphasizes consistent, repeatable routines for provisioning and changing systems and their configuration. Changes are made to definitions and then rolled out to systems through unattended processes that include thorough validation."

alterações são feitas nas definições e, em seguida, implementadas nos sistemas por meio de processos autônomos que incluem validação completa.

2.4.1 Beneficios da infraestrutura como código (IaC).

Esta seção resume os benefícios de usar IaC para implantar, gerenciar e atualizar a Infraestrutura de TI em uma organização (MORRIS, 2016).

- Reprodução fácil e rápida de sistemas - usando IaC, os administradores (e até desenvolvedores) podem provisionar e configurar toda a infraestrutura desde o início, simplesmente escrevendo alguns scripts e definição arquivos. Cada elemento da infraestrutura pode ser reproduzido repetidamente de forma confiável e sem esforço. Os scripts IaC descrevem todas as etapas necessárias para criar e provisionar o recurso solicitado, por exemplo, qual software deve ser instalado, sua versão correta, nome do servidor etc. Escrever esses scripts é muito mais fácil em comparação com o antigo modo manual, e o desenvolvimento e a produção dos sistemas torna-se muito mais rápida e simples. Novos serviços e novos aplicativos podem ser implantados na infraestrutura facilmente, o que leva a um processo de desenvolvimento de software mais eficiente.
- Sistemas Descartáveis - IaC melhorou os antigos sistemas estáticos em sistemas dinâmicos e descartáveis, que são capazes mudar de forma rápida e fácil. Os recursos das novas infraestruturas dinâmicas são facilmente criados, destruídos, atualizados, redimensionados e realocados no sistema. No entanto, o software implantado é capaz de ser executado mesmo quando o servidor em que ele é executado, é excluído, movido ou redimensionado. Melhorias e patches na infraestrutura tornaram-se mais fáceis, pois as mudanças são tratadas sem problemas. Isso é crucial em infraestruturas em nuvem em larga escala, onde o sistema não pode depender do hardware subjacente.

- Consistência de configuração - Erros humanos sempre causaram problemas à consistência da configuração, mesmo quando os procedimentos padrões para configuração são seguidos. O fator humano pode criar alguns pequenos desvios em configurações que são desafiadoras e demoradas para depurar.

  A implementação do IAC padroniza totalmente a configuração da infraestrutura, deixando pouco espaço para erros humanos. Como resultado, as chances de encontrar problemas de incompatibilidade são muito reduzidas, e a execução dos aplicativos torna-se mais consistente e suave.
- Sistemas e processos auto documentados - As equipes de TI sempre se esforçaram para manter sua documentação útil e precisa. Atualizações e melhorias são implementadas em um ritmo alto, por isso é quase impossível que a documentação esteja em dia com elas. Além disso, muitas pessoas preferem escrever os documentos descritivos à sua maneira e, em muitos casos, ignoram explicações por considerá-las óbvias ou desnecessárias para o leitor. Portanto, a maioria dos documentos não é uma representação justa do que realmente acontece.

  O IaC conseguiu resolver esse problema, incluindo e explicando todas as etapas necessárias para executar um processo nos arquivos de definição e nas ferramentas que realmente realizam o procedimento. Uma pequena documentação adicional também é necessária nesse caso. Os documentos devem estar próximos (física e significativamente) do código que eles explicam para ajudar as pessoas a adquirirem uma boa compreensão dos procedimentos subjacentes.
- Versionamento em tudo - Ter toda a infraestrutura codificada em arquivos de definição abre a oportunidade de usar técnicas de controle de versão para acompanhar as alterações confirmadas e revertê-las para versão estável quando ocorre um erro na versão anterior. O Sistema de Controle de Versão (VCS) oferece um arquivo de log com todas as mudanças implementadas, o motivo das mudanças e a entidade que as fez. Este recurso é muito útil para fins de depuração. Cada recurso do sistema é identificado por etiquetas e o número da versão, melhorando o rastreamento e correção de anormalidades mais complicadas. Além disso, o VCS suporta o início automático

de uma série de ações necessárias após a solicitação de uma nova mudança, que é uma característica da integração contínua e da abordagem de entrega contínua.

- Sistemas e processos continuamente testados - O teste automatizado é uma das práticas mais importantes que foi adicionada ao desenvolvimento de infraestrutura com a introdução do IaC. Escrever testes automatizados para uma infraestrutura em execução é desafiador, mas sua implementação correta pode levar a uma infraestrutura limpa, simples e funcional. As pessoas estarão mais confiantes em fazer mudanças se receberem feedback rápido das mudanças propostas. Os testes são realizados simultaneamente com o desenvolvimento e isso é crucial para uma infraestrutura automatizada, na qual um pequeno erro pode rapidamente causar danos significativos.
- Maior eficiência no desenvolvimento de software - O IaC aumentou a produtividade dos desenvolvedores de software. O ciclo de desenvolvimento de software se transformou em um processo mais eficiente, pois a infraestrutura de TI pode ser implantada de forma rápida, fácil e em vários estágios. Os desenvolvedores podem facilmente criar e simular a infraestrutura em suas máquinas para testar e experimentar seu código. Os testes e a verificação de segurança podem ocorrer em ambientes de testes separados, e o aplicativo pode ser simplesmente implantado e gerenciado nesses ambientes usando ferramentas IaC. Essas ferramentas também podem excluir os ambientes que não são usados, liberando um valioso poder computacional. Além disso, ao desligar todos os recursos não utilizados, o ambiente de desenvolvimento permanece limpo e simples. Isso aumenta a produtividade da equipe de engenharia, pois eles encontram um ambiente limpo e amigável para implantar, não tendo que gastar tempo apagando componentes não usados de projetos anteriores.

2.4.2 Elementos Gerais

Os principais elementos da Infraestrutura como Código são os arquivos de definição (também referidos como código), as plataformas de infraestrutura dinâmica, as ferramentas de

automação e a Interface de Programação de Aplicativos (API) esses elementos são explicados com mais detalhes a seguir (MORRIS, 2016):

- Os arquivos de definição: Os arquivos de definição são o elemento-chave do IaC. Os componentes da infraestrutura são definidos e configurados por meio desses arquivos. As ferramentas IaC usam esses arquivos como entradas para configurar / provisionar instâncias dos componentes da infraestrutura. Os componentes da infraestrutura podem ser diversas coisas como um servidor, uma parte de um servidor, uma rede, uma configuração etc. Os nomes dos arquivos de definição variam de acordo com a ferramenta utilizada, no Ansible o arquivo de definição recebe o nome de “playbook”, no Chef de “recipes” e no Puppet de “manifests”. Os arquivos de definição são basicamente arquivos de texto e são tratados como tal. Os formatos mais comuns de arquivos de definição são JSON, YAML ou XML, e algumas ferramentas definem seu próprio arquivo e linguagem específica.
- Plataforma de infraestrutura dinâmica: Uma plataforma de infraestrutura dinâmica concede as bases para provisionar e gerenciar os principais recursos da infraestrutura, como servidores, armazenamento e componentes de rede, e garante que eles possam ser programáveis. Existem várias plataformas de infraestrutura dinâmica disponíveis. Os exemplos mais conhecidos são os serviços de nuvem IaaS públicos, como Azure e AWS e IaaS privados, como Openstack (OPENSTACK, 2020). A infraestrutura também pode ser gerenciada usando sistemas de virtualização, como VMware vSphere, que não são executados na nuvem. Além disso, algumas organizações usam ferramentas como Cobbler e Foreman para gerenciar uma infraestrutura inteiramente local (FOREMAN, 2020). A plataforma de infraestrutura dinâmica não é afetada se é executada na nuvem, ou em sistemas de virtualização local, no entanto, é essencial ser programável, sob demanda e auto escaláveis. Programável refere-se aos arquivos de definição mencionados anteriormente e implica que a plataforma dinâmica deve suportar configuração e gerenciamento por meio desses arquivos. O termo sob demanda implica que a plataforma fornece aos usuários a capacidade de criar e

destruir recursos instantaneamente em questão de minutos ou mesmo segundos. Finalmente, uma plataforma de auto escalável não só oferece implantação rápida de recursos, mas também suporta a capacidade de criar, alterar e personalizar recursos com base nos requisitos pré-estabelecidos pelo usuário.

- Ferramentas de automação: Existem duas categorias diferentes de ferramentas de automação para configurar a infraestrutura: ferramentas de provisionamento e ferramentas de configuração.
    - Ferramentas de provisionamento são usadas para especificar e alocar os recursos desejados. Essas ferramentas usam a plataforma de infraestrutura dinâmica para implementar a alocação. Exemplos dessas ferramentas são Terraform, Openstack Heat e Cloud Formation da Amazon.
    - Ferramentas de configuração são usadas para configurar e gerenciar os recursos já provisionados com as dependências e configurações necessárias. Tem muitas ferramentas disponíveis nesta categoria, mas as três mais populares são Puppet, Chef e Ansible.
- Interfaces de programação de aplicativos (API): APIs são geralmente usados para definir as interfaces de acesso de um aplicativo de software ou plataforma baseado na web para que ele possa ser usado por outros componentes de software, promovendo assim a comunicação e a integração com a plataforma subjacente (MEDJAOUI; et al., 2019). As APIs são oferecidas pelas ferramentas de automação para provisionar e configurar os recursos da infraestrutura da forma descrita nos arquivos de definição. Mesmo ao usar as ferramentas disponíveis no mercado, as equipes de engenharia devem ocasionalmente escrever seus próprios scripts e extensões personalizadas para programar as ferramentas em sua API, portanto, as ferramentas usadas devem oferecer suporte a uma ampla variedade de linguagens de programação com as quais a equipe tem experiência. As APIs baseadas em REST são as mais utilizadas por oferecerem acesso remoto, facilidade de uso e alta flexibilidade. Muitas ferramentas e plataformas dinâmicas oferecem suporte a algumas bibliotecas de linguagens de programação específicas, incluindo classes e estruturas úteis que podem ser usadas para

implementar facilmente os componentes da infraestrutura e aplicar operações nela. A ideia é que o usuário crie os arquivos de definição que descrevem a infraestrutura, e esses arquivos sejam inseridos em uma ferramenta de automação que por meio de uma API instrui uma plataforma dinâmica a criar e gerenciar os recursos da infraestrutura.
A Figura 3 ilustra uma visão geral dos elementos e as interconexões entre eles.

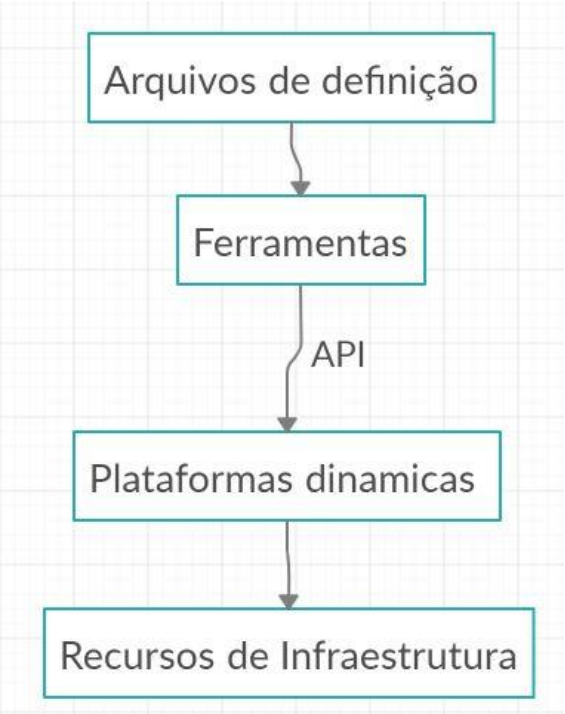

Figura 3 – Principais elementos da infraestrutura definida por software.

3. FERRAMENTAS IAC

Este capítulo apresenta as ferramentas mais utilizadas no mundo da infraestrutura como código (IaC) e faz um comparativo entre elas.

3.1 Ferramentas de provisionamento

O provisionamento é o primeiro passo para construir uma infraestrutura concreta e funcional. As ferramentas de provisionamento visam a criação dos componentes fundamentais da infraestrutura. A maioria das ferramentas de provisionamento só oferece suporte a um fornecedor de infraestrutura, como Amazon ou Google, mas também existem ferramentas que suportam vários fornecedores. As ferramentas de provisionamento específicas de fornecedores mais populares são o CloudFormation para AWS (AMAZON, 2020), o Cloud Deployment Manager da Google Cloud Platform (GOOGLE, 2020) e o Azure Resource Manager for Microsoft Azure Clouds (MICROSOFT, 2020). As ferramentas de provisionamento de código aberto mais populares são:

3.1.1 Terraform.

O Terraform é uma ferramenta de automação de infraestrutura desenvolvida pela HashiCorp há quatro anos e está escrita na linguagem de programação Go. É a primeira ferramenta que permite ao usuário automatizar e configurar elementos de infraestrutura de vários fornecedores de nuvem simultaneamente, bem como soluções personalizadas internas (HARSHICORP, 2020).

O Terraform descreve a infraestrutura através dos arquivos de configuração que são escritos em sua própria linguagem específica de domínio desenvolvida chamada Hashicorp Configuration Language (HCL). Esses arquivos são compatíveis com o JSON e são usados

para implantar os recursos solicitados. Esses arquivos podem ser facilmente compartilhados e reutilizados para criar o mesmo ambiente em outros lugares.

O Terraform também prevê planos de execução, que descrevem o procedimento que é seguido para chegar ao estado desejado da infraestrutura. O plano de execução primeiro dá uma visão geral da infraestrutura e das mudanças que serão executadas, em seguida, o Terraform realmente configura a infraestrutura executando esse plano, é possível observar esses estágios para o provisionamento na figura 4. Além disso, o Terraform é capaz de criar um gráfico dos recursos de infraestrutura, paralelizando a criação e modificação de qualquer recurso não dependente.

O uso do plano de execução combinado com o gráfico de recursos produzido proporciona mais automação para mudanças com menos envolvimento humano, já que o usuário tem mais insights sobre a funcionalidade do Terraform, evitando possíveis erros humanos.

O Terraform armazena o estado da infraestrutura gerenciada em um arquivo local chamado terraform.tfstate. Este arquivo também pode ser armazenado remotamente, o que é útil ao trabalhar em uma equipe remotamente distribuída. Este estado local é usado para criar os planos de execução e fazer as mudanças de infraestrutura necessárias. Após cada operação realizada, a Terraform atualiza o estado para corresponder à infraestrutura em tempo real.

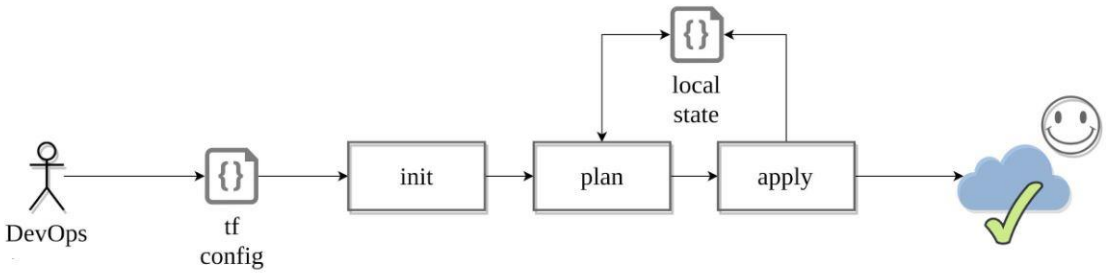

Figura 4 – Estágios de provisionamento Terraform.

3.1.2 Openstack Heat:

Heat é um produto da Fundação Openstack e é a principal ferramenta do programa Openstack Orchestration. Como as ferramentas anteriores, o Heat usa arquivos de modelo na forma de arquivos de texto para implantar e configurar vários recursos em nuvem da infraestrutura de TI desejada (OPENSTACK, 2020).

Os recursos de infraestrutura que podem ser usados incluem: servidores, volumes etc. A Telemetria Openstack também é suportada pelo Heat para que o usuário possa incluir no arquivo de modelo um grupo de escalonamento automático como um recurso possível (OPENSTACK, 2020). Além disso, o usuário pode declarar as conexões e as dependências entre os recursos. O Heat usa essas relações para chamar as APIs Openstack que são responsáveis pela criação da infraestrutura, conforme prescrito pelo usuário. O usuário pode facilmente alterar a infraestrutura simplesmente modificando o arquivo de modelo e, em seguida, o Heat toma todas as medidas necessárias para ajustar a infraestrutura ao estado desejado. O Heat também exclui todos os recursos não usados após o aplicativo terminar sua execução.

O Heat suporta uma API Rest nativa do Openstack, bem como uma API de consulta compatível com a Cloud Formation e os arquivos de modelo do Heat são altamente integrados com ferramentas de gerenciamento de configuração de software populares, como Puppet e Chef. A equipe do Heat está atualmente tornando o formato do modelo Heat compatível com o formato do modelo AWS Cloud Formation, para que muitos modelos de formação de nuvem existentes também possam ser executados no Openstack.

3.1.3 Comparativo das ferramentas de provisionamento.

A comparação das ferramentas de provisionamento descritas é ilustrada na Tabela 1. As ferramentas são comparadas com base em sua disponibilidade, no suporte de plataformas específicas e na linguagem de configuração usada.

Tabela 1 – Comparação das ferramentas de provisionamento

Métricas	Terraform	Openstack Heat	Cloud Formation	Cloud Deployment Manager	Azure Resource Manager
Disponibilidade	Código aberto	Código aberto	Código fechado	Código fechado	Código fechado
Plataformas suportadas	Múltiplas plataformas	Openstack	AWS	Google Cloud	Azure
Linguagem de configuração	DSL (HCL)	DSL (HOT)	YAML, JSON	YAML	JSON

Terraform e Openstack Heat são software de código aberto, enquanto o resto das ferramentas são soluções proprietárias, e são gratuitas para a criação de um número limitado de recursos. O Terraform é a única ferramenta da lista que suporta múltiplas plataformas dinâmicas, permitindo a combinação de diferentes recursos de uma variedade de fornecedores. As outras soluções estão anexadas a uma plataforma dinâmica específica, como Openstack, AWS, Google Cloud e Azure. Terraform e Openstack Heat usam linguagens específicas de domínio para gerenciar a configuração, que são baseadas em YAML, enquanto as demais ferramentas usam YAML ou JSON para descrever seus arquivos de definição.

3.2 Ferramentas de configuração

Uma vez que os elementos da infraestrutura são provisionados, eles precisam ser configurados. As ferramentas de gerenciamento de configuração são usadas para este fim. Existem muitas ferramentas disponíveis no mercado, e cada uma delas tem suas próprias vantagens e desvantagens. No entanto, todos eles servem ao mesmo objetivo: configurar os recursos implantados de acordo com as definições de configuração desejadas. As ferramentas de configuração de código aberto mais populares com base no número de commits e estrelas no GitHub, são descritas na seção a seguir.

3.2.1 Chef

Chef é uma ferramenta de gerenciamento de configuração que ajuda a automatizar a infraestrutura de TI. Chef pode gerenciar infraestruturas na nuvem, físicas, bem como em um ambiente híbrido. Chef é uma ferramenta cloud agnóstica que trabalha com muitos provedores de serviços de nuvem populares, como Microsoft Azure, AWS, Openstack e Cloud Platform (SABHARWAL; WADHWA, 2014).

A primeira versão de Chef foi desenvolvida em Ruby, no entanto a versão mais recente é parcialmente escrita em Erlang e Ruby. Chef pode suportar infraestruturas de até 10.000 nós.

Os principais componentes que formam Chef estão descritos na figura 5 e são eles:

- Chef Workstation: Sistema que é usado pelo usuário para interagir com chef e o usuário é capaz de desenvolver cookbooks e recipes, gerenciar os nódulos da infraestrutura, sincronizar o repositório chef e enviar cookbooks e outros arquivos para o servidor chef. O usuário pode interagir com o servidor chef usando o Knife, que é uma ferramenta de linha de comando. O repositório chef armazena tudo o que está relacionado com o servidor chef e os nós de servidores. Chef suporta vários Workstations em um único servidor chef.
- Chef Client: Uma máquina virtual ou física que é gerenciada pelo Chef. Chef também pode gerenciar nós localizados na nuvem. Cada nó tem que incluir um agente conhecido como Chef Client, a fim de interagir com o servidor chef. A configuração de um nó é realizada através de uma ferramenta incorporada chamada Ohai, que é usada para descrever os atributos do nó ao Chef Client.
- Chef Server: um sistema que contém tudo o que é essencial para a configuração dos nós. O servidor armazena os cookbooks, as políticas usadas nos nós e alguns metadados que descrevem os nós que são gerenciados pelo Chef. O cliente Chef que

está instalado em cada nó pede os detalhes de configuração, como recipes e modelos do servidor, e, em seguida, aplica a configuração ao nó especificado.

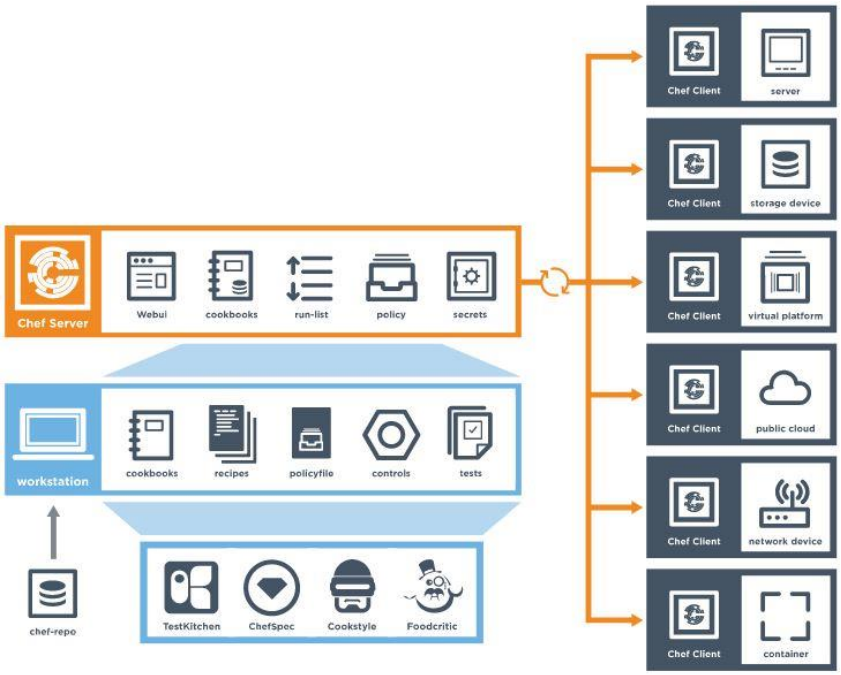

Figura 5 – Estrutura básica do Chef
Fonte: https://docs.chef.io/images/chef_overview.svg

Chef transforma a infraestrutura em código usando arquivos de texto chamados cookbooks, que são a unidade fundamental para configurar e distribuir políticas no Chef. Os cookbooks definem cenários completos e incluem tudo o que é essencial para executar esse cenário. Cookbooks são usados para agrupar e organizar recipes. Recipes são basicamente scripts escritos em Ruby que especificam os recursos necessários e a ordem de sua aplicação (KATSAROS; et al., 2014). Ruby é escolhida como a língua de referência para a criação de Cookbooks, com o apoio de uma DSL estendido para recursos especializados. O cliente chef é equipado com uma variedade de recursos para suportar os cenários de infraestrutura mais comuns, no entanto, o DSL sempre pode ser estendido sempre que são necessários mais recursos.

Chef vem em duas versões: comercial e código aberto. A versão comercial se chama Enterprise Chef e oferece suporte à implantação de alta disponibilidade. Ele é equipado com alguns recursos adicionais em relação à segurança e a relatórios. A versão de código aberto tem quase todos os recursos da versão comercial, exceto para a segurança extra e relatórios. Além disso, o Open Source Chef não suporta a instalação de componentes em vários servidores.

3.2.2 Puppet

Puppet é outra ferramenta popular de gerenciamento de configuração que ajuda a organizar e configurar servidores. Puppet executa os planos de configuração através de uma camada de abstração que descreve os elementos de configuração como objetos genéricos (LOOPE, 2011).

O usuário tem que declarar os recursos e seus atributos e que são dados ao Puppet como entradas para configurar adequadamente os recursos. Puppet recebe o catálogo dos recursos descritos e compara o estado existente dos recursos com o descrito. Em seguida, decide quais ações precisam ser tomadas para chegar a um acordo entre o Estado solicitado e o estado atual dos recursos. Essa abordagem é declarativa (ENDRES; et al., 2017), uma vez que o usuário declara como a configuração deve ser e, em seguida, Puppet toma todas as ações necessárias para alcançar essa configuração pretendida. Essa é a principal diferença com o Chef, que segue uma abordagem processual na qual o usuário tem que descrever os passos necessários para chegar ao estado desejado.

No Puppet, os arquivos de definição de recursos são chamados de manifests e são escritos em um DSL, bastante semelhante ao Ruby. No entanto, o usuário não pode simplesmente escrever código Ruby nos manifests e executá-lo. Os manifests podem ser executados repetidamente, resultando nos mesmos resultados que sempre correspondem ao estado descrito.

Puppet tem dois componentes principais para configurar servidores: puppet agent e o puppet master. O Puppet em si é chamado de puppet agent quando é executado em um modo client no servidor. O puppet master é executado no servidor mestre do cluster e define quais configurações se aplicariam a qual servidor e também armazena todas as informações de configuração em um local central. O puppet agent pede as configurações do puppet master em intervalos de tempo específicos, e quando há necessidade de mudança o puppet agent realmente implementa a mudança. A comunicação entre esses dois componentes é realizada em um canal criptografado seguro usando o protocolo SSL.

3.2.3 Ansible

Ansible é outra poderosa ferramenta de gerenciamento de configuração. A singularidade do Ansible em comparação com outras ferramentas de gestão é que ela também é usada para implantação e orquestração. Ansible é especialmente desenvolvido para ser simples, seguro, confiável e fácil de aprender. Ele oferece uma variedade de recursos para um usuário especializado, mas é igualmente acessível a usuários menos qualificados (REDHAT, 2020).

O Ansible não usa agentes, e nenhum software adicional deve ser instalado nos servidores remotos para gerenciá-los. A Ansible gerencia as máquinas remotas usando as estruturas de gerenciamento remoto que já existem nativamente no SISTEMA OPERACIONAL, por exemplo, SSH para máquinas Linux e UNIX e WinRM para máquinas Windows. A ausência de agentes resulta em menor consumo de recursos nas máquinas gerenciadas quando o Ansible não está operando nelas. Ansible também melhora a segurança funcionando em um modelo baseado em push onde as máquinas remotas recebem apenas as partes necessárias do código (chamados módulos), e as máquinas remotas não podem interagir ou interferir na configuração das outras máquinas. Esses recursos tornaram o Ansible adequado para sistemas de alta segurança e alto desempenho.

No Ansible, os arquivos de definição para configurar, automatizar e gerenciar a infraestrutura de TI são chamados de Playbooks. Esses arquivos são escritos em formato YAML e descrevem como executar uma operação, indicando claramente o que deve ser feito por cada componente da infraestrutura. Cada Playbooks consiste em uma lista de peças que descrevem o processo de automação para um conjunto de hosts, chamado inventário. Cada peça inclui várias tarefas que se referem a um único host ou a um grupo de hosts no inventário. Cada tarefa chama um módulo, que é um pequeno pedaço de código que executa um trabalho específico. As tarefas variam de trabalhos simples a operações complexas. O Ansible também pode agrupar as tarefas do Playbook em unidades conhecidas como Roles. O Ansible usa as Roles para aplicar configurações comumente usadas em vários cenários de forma rápida e fácil.

O Ansible foi desenvolvido de forma a facilitar a extensibilidade. O usuário sempre tem a possibilidade de estender os módulos nativos que são mais de 450 escrevendo seus próprios módulos. Os módulos incorporados são escritos em Python e PowerShell, mas o usuário pode usar qualquer linguagem de programação para desenvolver novos, com a única restrição de que eles têm que ter o JSON como formato de entrada e produzir JSON como formato de saída. Além disso, o Ansible pode ser estendido para suportar o inventário dinâmico, que permite que os Playbooks sejam executados em um grupo de máquinas e infraestrutura que não são constantes e estaticamente definidas, mas podem executar um provedor de nuvem público ou privado que suporte à criação dinâmica e exclusão dos recursos. Ansible suporta a maioria dos provedores de nuvem conhecidos e sempre pode ser estendido para apoiar novos provedores simplesmente escrevendo um programa personalizado (em qualquer linguagem de programação) que dá uma definição de inventário JSON como saída.

O Ansible é um projeto de código aberto promovido pela Red Hat. A versão comercial paga da Ansible é chamada Red Hat Ansible Tower, e oferece gerenciamento para implantações complexas de vários níveis adicionando controle e suporte técnico aos sistemas suportados pela Ansible.

3.2.4 Saltstack

Saltstack é uma ferramenta de gerenciamento de configuração que também é usada para orquestrar a infraestrutura. Ele configura, altera e atualiza a infraestrutura de TI por meio de um repositório central. Pode operar em servidores físicos, virtuais e em nuvem (MYERS, 2016).

Como as ferramentas IaC anteriores, o Saltstack visa automatizar as tarefas administrativas e de implantação de código e reduzir a chance de erro humano, removendo os processos manuais tanto quanto possível. Para isso, o Saltstack usa os métodos push e pull para configurar os servidores. Ele extrai arquivos de configuração e código de um repositório central como o Github e, em seguida, envia esses arquivos para os servidores remotamente.

Saltstack tem dois componentes principais: o Salt master e o Salt minion. O master é o servidor central e todos os minion estão conectados a ele para obter instruções. A conexão entre o master e os minions é criptografada com base em hashes criptográficos. Os minions podem ser comandados pelo master após usar a autenticação de chave pública. Os minions podem ser executados sem um master, mas todo o potencial do Saltstack é aproveitado em uma rede de minions x master. O usuário pode enviar atualizações e arquivos de configuração através do master para os minions ou agendar os minions para verificar o master em intervalos de tempo específicos e obter atualizações e configurações disponíveis. A arquitetura de gerenciamento do Saltstack é altamente orientada a eventos e oferece autodependência e auto recupeção para o sistema, pois aproveita os métodos push e pull para atualizar e se recuperar de erros.

Saltstack inclui alguns outros recursos importantes, como os Salt reactors, agents, minions, grains e pillars. Salt reactors são responsáveis por escutar novos eventos nos minion, enquanto os Salt agents usam shell seguro para executar comandos nos nós de destino. Os minions são os próprios agentes instalados nos servidores remotos para enviar comandos. Os Salt grains fornecem informações valiosas sobre os servidores gerenciados e os Salt pillars são os arquivos de configuração.

O Saltstack é diferente das outras ferramentas de gerenciamento de configuração porque é rápido e pode operar de forma multithread, de forma que pode executar várias tarefas simultaneamente. Além disso, o Saltstack é desenvolvido em Python e o usa para escrever os scripts de configuração. No entanto, ele também pode renderizar scripts desenvolvidos em outras linguagens, como YAML e JSON. Isso torna o Saltstack uma ferramenta independente de linguagem que é facilmente acessível a um amplo público de desenvolvedores de software.

Saltstack é uma estrutura de código aberto que é operada a partir da Interface de linha de comando n (CLI). A edição paga é chamada Saltstack Enterprise e vem com alguns recursos adicionais, como suporte para GUI e suporte para servidores Windows, macOS e Solaris. A versão Enterprise também pode armazenar os eventos em um banco de dados, oferecendo ao usuário um histórico auditável dos eventos.

3.2.5 Comparativo das ferramentas de configuração.

A Tabela 2 representa uma comparação entre as ferramentas de configuração mencionadas anteriormente. Há uma grande variedade de fatores, que podem ser usados para comparar esse tipo de ferramenta, mas para efeito desta pesquisa foram selecionados as cinco métricas mais adequadas.

Tabela 2 – Comparação das ferramentas de configuração

Métricas	Chef	Puppet	Ansible	Saltstack
Linguagem de configuração	Ruby	DSL (Ruby-based)	YAML	YAML, Python, JSON
Versão Corporativa	Sim	Sim	Sim	Sim
Arquitetura	Master x Client	Master x Client	Apenas Master	Master x Client
Método de configuração	Pull	Pull	Push	Pull e Push
Sistemas operacionais suportados	Master (Linux) Agents (Linux & Windows)	Master (Linux) Agents (Linux & Windows)	Master (Linux) Agents (Linux & Windows)	Master (Linux) Agents (Linux & Windows)

Todas as ferramentas descritas fornecem altos níveis de escalabilidade. Essas ferramentas são capazes de gerenciar grandes infraestruturas com mais de 10.000 nós e o usuário só deve declarar o IP ou o nome dos nós que requerem configuração, e as ferramentas executam a configuração desejada. Além disso, todas as ferramentas fornecem uma versão suportada por empresas, que vem com preços diferentes dependendo do tamanho da infraestrutura.

Chef, Puppet e Saltstack seguem uma arquitetura de mestre x cliente, enquanto Ansible requer apenas a instalação do mestre na máquina do servidor de controle e nenhum software adicional é instalado nas máquinas clientes. Chef e Puppet seguem o método de configuração de pull, enquanto Ansible usa o método push. Saltstack usa métodos de push e pull para configurar os servidores. Ele solicita os arquivos de configuração e código de um repositório central e, em seguida, envia esses arquivos para os servidores remotamente.

Chef e Puppet usam linguagens de configuração baseadas em Ruby para gerenciar seus arquivos de definição, o que requer um entendimento básico sobre a programação, a fim de usá-los de forma eficiente. Por outro lado, Ansible e Saltstack usam YAML para configuração, que é uma linguagem de configuração fácil de usar. Saltstack também oferece suporte para Python e JSON.

4. PREPARAÇÃO DO PROJETO.

O objetivo deste capítulo é apresentar o projeto, definiremos um software alvo para automatização, a plataforma de infraestrutura dinâmica e as ferramentas de provisionamento e configuração que serão utilizadas.

Tabela 3 – Uma visão geral das soluções selecionadas.

Escopo	Solução	Raciocínio
Aplicação	webBudget	• Código aberto • Interface Web. • Interface responsiva (Computadores e dispositivos móveis). • Banco de dados. • Controle de acesso.
Plataforma de infraestrutura dinâmica	AWS	• Solução em nuvem pública. • Suporte extensivo do Terraform. • Apoio da comunidade e farta documentação na internet. • Solução mais utilizada no mercado atualmente.
Provisionamento	Terraform	• Código aberto • Suporta múltiplas plataformas dinâmicas. • As opções alternativas são soluções específicas para fornecedores.
Configuração	Ansible	• Código aberto. • Implantação sem agente. • Rápido e confiável. • Seguro, pois usa SSH para empurrar módulos. • Suporta inventários dinâmicos.

4.1Escolha da aplicação.

Para uma demonstração pratica dos conceitos aprendidos nesta pesquisa será selecionada uma aplicação de código aberto que possua as mesmas tecnologias de sistemas que estão sendo utilizados atualmente pelo mercado como banco de dados, interface web, controle de acesso e acesso tanto pelo computador tradicional como por dispositivos móveis. Seguindo esses requisitos escolhemos o software webBudget.

Segundo GREGÓRIO (2020, online), autor e desenvolvedor da aplicação, o webBudget é:

> Um sistema grátis e de código aberto para controle financeiro pessoal ou de pequenas empresas. Com ele você poderá fazer a organização de receitas e despesas dentro de um período de tempo, classificar a movimentação através de centros de custos e tipos de entrada e saída, controlar o uso de seus cartões de crédito e ainda acompanhar tudo isso em tempo real na dashboard de posição financeira.

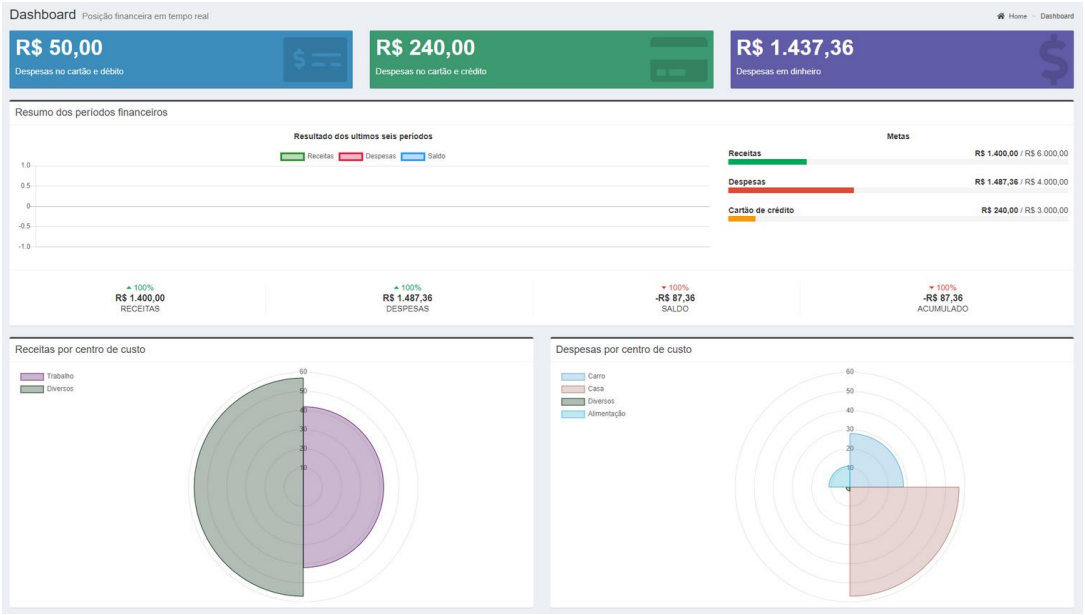

Figura 6 – Tela principal webBudget
Fonte: https://webbudget.com.br/assets/images/prints/3_1516.PNG

O objetivo desta pesquisa não se encontra no código, funcionabilidades e nem nas tecnologias utilizadas para desenvolvimento do WebBudget, mas sim em como será usado os conceitos de infraestrutura como código na organização da sua infraestrutura , automação do provisionamento e da configuração do seu ambiente computacional, afim de reproduzir este ambiente quantas vezes forem necessário e disponibilizá-lo rapidamente aos usuários finais através da web.

4.2 Escolha da plataforma de infraestrutura dinâmica.

A plataforma dinâmica oferece os recursos programáveis que são utilizados pela ferramenta de provisionamento para configurar a infraestrutura de TI. A plataforma mais adequada para este caso de uso específico é a AWS.

Plataformas de nuvem privada como Openstack, não podem atender aos requisitos deste caso de uso específico, pois não possuímos um datacenter físico para elaboração deste projeto e nossa intenção é disponibilizar o sistema web na internet e plataformas de nuvem publica oferecem maior facilidade para isso.

Dentre as plataformas de nuvem publica foi escolhida a AWS, pois tem respaldo de várias grandes empresas, tem um forte apoio da comunidade e diversas documentações e manuais de utilização na internet.

Atualmente é a plataforma lider no setor com cerca de 33% do mercado de nuvem segundo pesquisa da Synergy Research Group demostrado na figura 7 (SYNERGY, 2020).

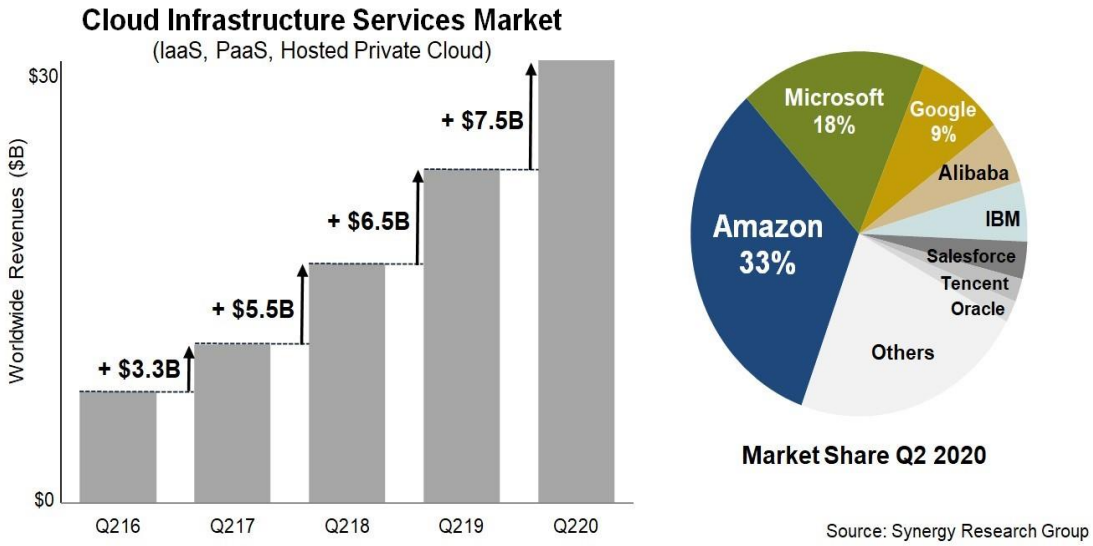

Figura 7 – Participação de mercado dos provedores segundo trimestre de 2020.
Fonte: https://cdn.statcdn.com/Infographic/images/normal/18819.jpeg

Além de oferecer a maioria dos elementos necessários para criação da nossa infraestrutura de forma gratuita em um período de avaliação viável a elaboração desta pesquisa.

4.3 Ferramenta de provisionamento.

O provisionamento é o passo mais importante para a construção de uma infraestrutura de TI totalmente funcional. Portanto, a escolha da ferramenta de provisionamento é fundamental para todo o sistema definido por software e deve se concentrar em oferecer escalabilidade e manutenção ao sistema.

Uma comparação das ferramentas de provisionamento disponíveis está disponível na Tabela 1 na Seção 3.1.3. O Terraform é a solução mais adequada da lista, pois é uma solução de código aberto e declarativa que usa os arquivos HCL para especificar o estado desejado da infraestrutura e, em seguida, usa APIs para chegar a esse estado. Além disso, o Terraform suporta recursos de vários provedores de nuvem simultaneamente e é capaz de suportar várias

soluções personalizadas. Como resultado, o sistema tem suporte para múltiplas plataformas dinâmicas (por exemplo, Openstack, AWS, Azure, Google Cloud Platform) dentro do mesmo arquivo HCL que oferece uma ampla gama de recursos de infraestrutura para construir, levando a sistemas mais dinâmicos e escaláveis. Outras ferramentas de provisionamento são de fontes fechadas ou especificas para um provedor de serviço por exemplo, CloudFormation, Cloud Deployment Manager, Azure Manager Openstack Heat.

4.4 Ferramentas de configuração

As ferramentas de configuração são usadas para configurar e gerenciar os recursos já provisionados com as dependências e configurações necessárias. Uma comparação das ferramentas de configuração mais utilizadas está disponível na Tabela 2 na Seção 3.2.5, e o Ansible é o software de configuração preferido para este caso de uso específico.

O Ansible foi selecionado entre Chef, Puppet e Saltstack, pois oferece implantação sem agente que adiciona velocidade e confiabilidade ao sistema, em comparação com o modelo de implantação com agente utilizado pelo Chef e Puppet. Além disso, o modelo sem agente elimina pontos de falha e vários problemas de desempenho no sistema, e o uso de SSH para a comunicação entre os nós melhora a segurança geral da infraestrutura. Além disso, O Ansible oferece uma instalação rápida e fácil e no geral é uma solução bastante fácil para novos usuários. Ansible também pode suportar a natureza dinâmica, onde novos hosts estão constantemente girando e desligando em resposta às demandas dos sistemas. O Ansible pode atender a essa demanda com o uso de inventários dinâmicos, que podem ser adicionados ao Terraform como template de saída para acompanhar a infraestrutura.

5. IMPLEMENTAÇÃO DO PROJETO

A Implementação do sistema definido por software é um processo desafiador, a distinção deste processo em etapas (preparação da aplicação, provisionamento da infraestrutura e configuração do ambiente) facilita e simplifica a implantação do projeto, uma vez que a seleção da ferramenta tem sido realizada para cumprir os requisitos de cada uma dessas etapas.

5.1 Preparação e empacotamento da aplicação

Nesta etapa a aplicação será conteinerizada. O objetivo é empacotar toda aplicação suas dependências e seu banco de dados facilitando a distribuição. Através arquivos de definição chamados Dockerfile e Docker-compose é possível garantir que determinado padrão seja seguido, proporcionando maior previsibilidade e aumentando a confiança na replicação, tornando viável escalar a aplicação rapidamente.

É possível configurar parâmetros nesses arquivos de definição que permitem à aplicação conteinerizada se comportar de formas diferentes entre distintos ambientes, essa caraterística se torna fundamental para este projeto, pois serão através desses parâmetros que será informado a aplicação onde encontrar seus componentes, pois na nuvem a cada implantação as informações de localização como endereço de rede mudam devido à natureza dinâmica da nuvem .

É possível construir a imagem do contêiner através do arquivo Dockerfile ou se disponível baixá-las diretamente do Docker Hub.

Docker Hub é o maior repositório online de imagens de contêiner, onde é possível encontrar diversas imagens inclusive algumas homologadas pelos fornecedores oficiais (Docker Inc, 2020).

Na figura 8 é possível observar a estrutura do diretório da aplicação, na pasta files se encontram os arquivos essenciais da aplicação:

- module.xml – Arquivo para instalação do modulo do banco de dados (Postgres) no servidor Web Windfly.
- postgresql-42.2.5.jar – Driver do banco de dados (Postgres) para aplicações java.
- Standalone.xml – Arquivo de configuração geral do servidor web Wildfly.
- web-budget-3.0.1-RELEASE.war – Arquivo java compactado contendo todo o código da aplicação web.

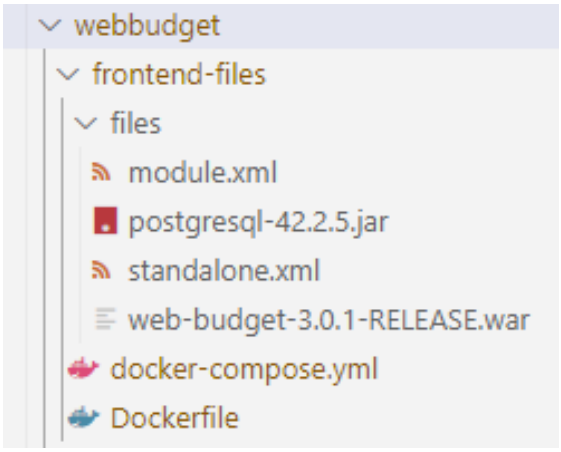

Figura 8 – Estrutura de diretórios da aplicação.

Abaixo é possível analisar o código dos arquivos de definição do contêiner da aplicação e os comentários explicativos em cada linha:

Dockerfile:

```
# baixando imagem docker do servidor web wildfly
FROM jboss/wildfly:16.0.0.Final

# variáveis de ambiente padrão para o servidor web wildfly
ENV WF_ADMIN_USER webbudget
ENV WF_ADMIN_PASS webbudget

# variáveis de ambiente padrão para o banco de dados
```

```
ENV DB_HOST postgres
ENV DB_PORT 5432
ENV DB_NAME webbudget
ENV DB_USER sa_webbudget
ENV DB_PASS sa_webbudget

# definindo o usuário para executar comandos abaixo
USER root

# criando a pasta onde o driver do banco de dados (postgres) será colocado
RUN mkdir -p /opt/jboss/wildfly/modules/system/layers/base/org/postgresql/main

# copiando driver do banco de dados (postgres)
COPY files/module.xml /opt/jboss/wildfly/modules/system/layers/base/org/postgresql/main
COPY files/postgresql-42.2.5.jar /opt/jboss/wildfly/modules/system/layers/base/org/postgresql/main/

# copiando configurações do servidor web wildfly
COPY files/standalone.xml /opt/jboss/wildfly/standalone/configuration/standalone.xml

# copiando arquivo de implantação(war) do aplicativo para a pasta
COPY files/web-budget-3.0.1-RELEASE.war /opt/jboss/wildfly/standalone/deployments

# adicionando usuário administrador do servidor wildfly
RUN /opt/jboss/wildfly/bin/add-user.sh ${WF_ADMIN_USER} ${WF_ADMIN_PASS} --silent

# expondo a porta https para aplicação e gerenciamento do servidor web
EXPOSE 443 9993 80

# iniciando servidor web
CMD ["/opt/jboss/wildfly/bin/standalone.sh", "-b", "0.0.0.0", "-bmanagement", "0.0.0.0"]
```

Implementação do arquivo Dockerfile da aplicação (Linguagem própria declarativa).

docker-compose.yml:

```
version: '3.5' # Versão do arquivo.

services: # Nesta parte declaramos os conteineres a serem iniciados e suas carateristicas.
  webbudget: # Declaração do nome do serviço.
    build: ./ #Indica a necessidade da construção local da imagem através do arquivo dockerfile e també
m sua localização no disco.
    image: webbudget:1.0 # Nome da imagem docker a ser gerada.
```

```
    container_name: webbudget_app # Nome do contêiner.
    ports: # Portas a serem expostas no contêiner.
      - "9993:9993"
      - "443:443"
      - "80:80"
    extra_hosts: # Apontamento do endereço de rede do banco de dados.
      -
 "postgres:10.0.1.150" # Este endereço de rede e informado pela integração com a ferramenta de provison
amento (Terraform).
    networks: # Declaração do grupo de rede do contêiner.
      - webbudget_network

networks: # Nesta parte criamos os grupos de rede e suas caracteristicas.
  webbudget_network: # Criação do grupo de rede.
    name: webbudget # Nome do grupo de rede.
    driver: bridge # Tipo de conectividade da rede.
```

Implementação do arquivo docker-compose da aplicação (Linguagem yaml).

Próximo passo é o código do arquivo de definição do banco de dados que também será conteinerizado, para o banco de dados o processo é mais simples, pois já existe uma imagem pronta e homologada e pelo fornecedor oficial (Postgres) no DockerHub e não será preciso construí-la como foi feito na aplicação, logo só será necessário apenas o arquivo docker-compose .

docker-compose.yml:

```
version: '3.5' # Versão do arquivo.

services: # Nesta parte declaramos os conteineres a serem iniciados e suas carateristicas.
  postgres: # Declaração do nome do serviço.
    image: postgres:11.2-alpine # Indica o nome da imagem docker a ser baixada na nuvem do docker.
    container_name: webbudget_db # Nome do contêiner.
    ports: # Portas a serem expostas no contêiner.
      - "5432:5432"
    environment: # Informa as variáveis de ambiente do sistema.
      - POSTGRES_DB=webbudget
      - POSTGRES_USER=sa_webbudget
```

```
      - POSTGRES_PASSWORD=sa_webbudget
      - POSTGRES_PORT=5432
    networks: # Declaração do grupo de rede do contêiner.
      - webbudget_network
networks: # Nesta parte criamos os grupos de rede e suas caracteristicas.
  webbudget_network: # Criação do grupo de rede.
    name: webbudget  # Nome do grupo de rede.
    driver: bridge # Tipo de conectividade da rede.
```

Implementação do arquivo docker-compose do banco de dados (Linguagem yaml).

Com os códigos desenvolvidos acima será possível iniciar os contêineres, basta acessar o diretório dos arquivos e executar o comando a seguir no terminal: docker-compose up. Esse comando será efetuado pela ferramenta de configuração Ansible posteriormente.

5.2 Provisionamento da infraestrutura

A etapa inicial para o desenvolvimento de uma infraestrutura de TI é o provisionamento dos componentes e recursos necessários, como máquinas virtuais e redes virtuais. Terraform é a ferramenta de provisionamento selecionada, pois pode construir uma grande variedade de componentes de infraestrutura. Terraform provisiona os recursos de uma plataforma de infraestrutura dinâmica, através de um provedor Terraform que é usado para interagir com as APIs e expor os recursos da plataforma dinâmica correspondente. O Terraform permite que seus usuários declarem recursos de diferentes provedores no mesmo ou em diferentes arquivos HashiCorp Config Language (HCL). Neste caso de uso específico, o provedor AWS será utilizado para criar toda nossa infraestrutura.

O Terraform precisa da instalação da Interface de linha de comando AWS (AWS CLI) para se comunicar com a API da AWS, o processo de instalação e configuração e informado a seguir (Amazon,2020):

```
# Em ambiente Linux
    curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
    unzip awscliv2.zip
    sudo ./aws/install

# No Windows
    Download em:  https://awscli.amazonaws.com/AWSCLIV2.msi

# No macOs:
    Download em:   https://awscli.amazonaws.com/AWSCLIV2.pkg
```

Comandos de instalação do AWS CLI

Agora é necessário configurar o AWS CLI com as credenciais da AWS e o formato de saída JSON, formato suportado pelo Terraform:

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

Comandos de configuração do AWS CLI

Próximo passo é a instalação do Terraform que pode ser feita usando os comandos abaixo (Terrafrom,2020):

```
# Em ambinte linux
    curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
    sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -
cs) main"
    sudo apt-get update && sudo apt-get install terraform

# No windows
    choco install terraform

# No macOs:
    brew tap hashicorp/tap
    brew install hashicorp/tap/terraform
```

Comandos de instalação do Terraform

Para melhor organização do projeto, o código foi separado em diferentes arquivos de acordo com tipo de recurso a ser provisionado, esses arquivos de definição os comentários explicativos em cada linha de código serão analisados a seguir:

instances.tf – Esta parte do código é responsável pela escolha do provedor e pelo provisionamento das máquinas virtuais.

```
provider "aws" {        # Primeira parte do codigo onde é selecionado o provedor de serviços na nuvem a
 verão e a região.
  version = "~> 3.0"
  region  = var.region
}


## Maquinas Virtuais ##
resource "aws_instance" "db-machine" {  # Provisionando a maquina virtual db-machine
  subnet_id =  aws_subnet.Public_subnet.id # Informa a sub rede qua a maquina deve receber
  ami = var.amis["sa-east-1"] #Recebe o id da imagem do sistema operacional que a maquina deve receber
  instance_type = "t2.micro" # Informa o tipo de servidor a ser provisionado
  key_name = var.keyname # Informa a chave publica a ser instalada na maquina.
  tags = { # Informa o nome da máquina para gerencia futura
    Name = "db-machine"
    }
  vpc_security_group_ids = [aws_security_group.blocked-all.id, aws_security_group.internal-
access.id] # Informa os grupos de segurança que a máquina deve pertencer
}

resource "aws_instance" "web-machine" { # Provisionando a maquina virtual web-machine
  subnet_id = aws_subnet.Public_subnet.id
  ami = var.amis["sa-east-1"]
  instance_type = "t2.micro"
  key_name = var.keyname
  tags = {
    Name = "web-machine"
    }
  vpc_security_group_ids = [aws_security_group.acesso-web.id, aws_security_group.internal-access.id]
}
```

```
resource "aws_instance" "web-machine2" { # Provisionando a maquina virtual web-machine2
  subnet_id = aws_subnet.Public_subnet2.id
  ami = var.amis["sa-east-1"]
  instance_type = "t2.micro"
  key_name = var.keyname
  tags = {
    Name = "web-machine2"
    }
  vpc_security_group_ids = [aws_security_group.acesso-web.id, aws_security_group.internal-access.id]
}
```

Arquivo de provisionamento das máquinas virtuais - Terraform

network.tf – Esta é a parte do código responsável pela criação dos elementos da rede virtual.

```
resource "aws_vpc" "VPC_WEBBUCKET" {  # Provisionando a rede virtual privada (VPC)
  cidr_block           = var.vpcCIDRblock
  instance_tenancy     = var.instanceTenancy
  enable_dns_support   = var.dnsSupport
  enable_dns_hostnames = var.dnsHostNames
tags = {
    Name = "VPC_WEBBUCKET"
}
}

#Subnets

resource "aws_subnet" "Public_subnet" {   # Provisionando a primeira subnet
  vpc_id                  = aws_vpc.VPC_WEBBUCKET.id
  cidr_block              = var.publicsCIDRblock["1"]
  map_public_ip_on_launch = var.mapPublicIP
  availability_zone       = var.availabilityZone["a"]
tags = {
   Name = "Public subnet"
}
}

resource "aws_subnet" "Public_subnet2" { # Provisionando a segunda subnet
  vpc_id                  = aws_vpc.VPC_WEBBUCKET.id
```

```
  cidr_block              = var.publicsCIDRblock["2"]
  map_public_ip_on_launch = var.mapPublicIP
  availability_zone       = var.availabilityZone["c"]
tags = {
   Name = "Public subnet2"
}
}

resource "aws_network_acl" "Public_NACL" { # # Provisionando a lista de controle de acesso da VPC
  vpc_id = aws_vpc.VPC_WEBBUCKET.id
  subnet_ids = [ aws_subnet.Public_subnet.id, aws_subnet.Public_subnet2.id ]

  ingress {
    protocol   = "tcp"
    rule_no    = 100
    action     = "allow"
    cidr_block = var.publicdestCIDRblock
    from_port  = 22
    to_port    = 22
  }
  ingress {
    protocol   = "tcp"
    rule_no    = 200
    action     = "allow"
    cidr_block = var.publicdestCIDRblock
    from_port  = 443
    to_port    = 443
  }
  ingress {
    protocol   = "tcp"
    rule_no    = 300
    action     = "allow"
    cidr_block = var.publicdestCIDRblock
    from_port  = 80
    to_port    = 80
  }
  ingress {
    protocol   = "tcp"
    rule_no    = 400
    action     = "allow"
    cidr_block = var.publicdestCIDRblock
```

```
    from_port  = 9993
    to_port    = 9993
  }
  # Porta efêmera
  ingress {
    protocol   = "tcp"
    rule_no    = 500
    action     = "allow"
    cidr_block = var.publicdestCIDRblock
    from_port  = 1024
    to_port    = 65535
  }

  egress {
    protocol   = -1
    rule_no    = 100
    action     = "allow"
    cidr_block = var.publicdestCIDRblock
    from_port  = 0
    to_port    = 0
  }

tags = {
    Name = "Public NACL"
}
}

resource "aws_internet_gateway" "IGW" { #Provisionando o gateway para acesso a internet
 vpc_id = aws_vpc.VPC_WEBBUCKET.id
 tags = {
        Name = "Internet gateway"
}
}

resource "aws_route_table" "Public_RT" { # Criando uma tabela de roteamento
 vpc_id = aws_vpc.VPC_WEBBUCKET.id
 tags = {
        Name = "Public Route table"
}
}
```

```
resource "aws_route" "internet_access" { # Anexando a tabela de roteamento ao intenet gateway
  route_table_id         = aws_route_table.Public_RT.id
  destination_cidr_block = var.publicdestCIDRblock
  gateway_id             = aws_internet_gateway.IGW.id
}

resource "aws_route_table_association" "Public_association" { #Associando a tabela de roteamento a prim
eira subnet
  subnet_id      = aws_subnet.Public_subnet.id
  route_table_id = aws_route_table.Public_RT.id
}

resource "aws_route_table_association" "Public_association2" { #Associando a tabela de roteamento a seg
unda subnet
  subnet_id      = aws_subnet.Public_subnet2.id
  route_table_id = aws_route_table.Public_RT.id
}
```

Arquivo de provisionamento dos componentes da rede virtual - Terraform

security_group.tf – Esta parte do código é responsável pelo provisionamento de regras de acesso a rede das máquinas virtuais, funciona semelhante a um firewall virtual

```
resource "aws_security_group" "acesso-web" { # Provisionando Grupo de segurança
  name        = "acesso-web"
  description = "acesso-web"
  vpc_id      = aws_vpc.VPC_WEBBUCKET.id

# Porta de conexão web segura https
  ingress { # Criando regra de entrada
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = var.cidrsRemoteAcess
  }

# Porta de conexão web http
  ingress { # Criando regra de entrada
    from_port   = 80
    to_port     = 80
```

```
    protocol    = "tcp"
    cidr_blocks = var.cidrsRemoteAcess
  }

# Porta de gerência do servidor web windfly
  ingress { # Criando regra de entrada
    from_port   = 9993
    to_port     = 9993
    protocol    = "tcp"
    cidr_blocks = var.cidrsRemoteAcess
  }

  # liberação da comunicação de saida
    egress { # Criando regra de saida
    protocol   = -1 # Todos os protocolos
    cidr_blocks = var.cidrsRemoteAcess
    from_port  = 0 #Libera todas as portas
    to_port    = 0
  }
  tags = {
      Name = "acesso-web"
  }
}

# BLOQUEIO TOTAL DE ENTRADA

resource "aws_security_group" "blocked-all" {
  name        = "blocked-all"
  description = "blocked-all"
  vpc_id      = aws_vpc.VPC_WEBBUCKET.id

  ingress {
    from_port   = 0
    to_port     = 0
    protocol    = "tcp"
    cidr_blocks = var.cidrsRemoteAcess
  }

  egress {
    protocol   = -1
    cidr_blocks = var.cidrsRemoteAcess
```

```
    from_port   = 0
    to_port     = 0
  }

  tags = {
    Name = "blocked-all"
  }
}


# ACESSO LIBERADO ENTRE AS INSTANCIAS

resource "aws_security_group" "internal-access" {
  name        = "internal-access"
  description = "internal-access"
  vpc_id      = aws_vpc.VPC_WEBBUCKET.id

  ingress {
    from_port   = 0
    to_port     = 0
    protocol    = -1
    cidr_blocks = [aws_vpc.VPC_WEBBUCKET.cidr_block,var.cidrsControler]
  }
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = -1
    cidr_blocks = [aws_vpc.VPC_WEBBUCKET.cidr_block,var.cidrsControler]
  }

  tags = {
    Name = "internal-access"
  }
}
```

Arquivo de provisionamento dos grupos de segurança - Terraform

certificate.tf – Nesta parte é provisionada o certificado digital e o balanceador de carga que irá recebê-lo.

```
resource "aws_acm_certificate" "cariocatcc" { # Criando um certificado para o dominio.
  domain_name       = "*.cariocatcc.com"

  validation_method = "DNS"
}

data "aws_route53_zone" "selected" { # Selecionando o nome do endereço do domínio previamente criado.
  name = var.domainName
}


resource "aws_route53_record" "validation" { # Primeiro processo de validação do certificado digital co
m o dominio selecionado.
  for_each = {
    for dvo in aws_acm_certificate.cariocatcc.domain_validation_options : dvo.domain_name => {
      name   = dvo.resource_record_name
      record = dvo.resource_record_value
      type   = dvo.resource_record_type
    }
  }

  allow_overwrite = true
  name            = each.value.name
  records         = [each.value.record]
  ttl             = 60
  type            = each.value.type
  zone_id         = data.aws_route53_zone.selected.zone_id
}

resource "aws_acm_certificate_validation" "validation" { # Segundo processo de validação do certificado
 digital com o dominio selecionado.
  certificate_arn         = aws_acm_certificate.cariocatcc.arn
  validation_record_fqdns = [for record in aws_route53_record.validation : record.fqdn]
}

resource "aws_elb" "lb" { # Provisionando um load balancer .
  name = "web-elb"
  security_groups = [aws_security_group.acesso-
web.id] # Associando o load balancer a um grupo de segurança .
  subnets =  [aws_subnet.Public_subnet.id, aws_subnet.Public_subnet2.id] # Associando o load balancer a
 duas subnets.
```

```
  listener { # Configurando a porta 80 como porta de escuta e porta de saida do load balancer.
    instance_port     = 80
    instance_protocol = "http"
    lb_port           = 80
    lb_protocol       = "http"
  }

  listener { # Configurando a porta 443 como porta de escuta e porta de saida do load balancer.
    instance_port      = 443
    instance_protocol  = "https"
    lb_port            = 443
    lb_protocol        = "https"
    ssl_certificate_id = aws_acm_certificate_validation.validation.certificate_arn # Associando o cerif
icado digital a porta 443 do load balancer.
 }
  health_check { # Verificando se a porta 443 está respondendo nos servidores de destino.
    healthy_threshold   = 2
    unhealthy_threshold = 2
    timeout             = 3
    target              = "HTTPS:443/"
    interval            = 30
  }
  instances                   = [aws_instance.web-machine.id, aws_instance.web-
machine2.id] # Associando os 2 servidores de aplicação como destino do load balancer.
  cross_zone_load_balancing   = false
  idle_timeout                = 400
  connection_draining         = true
  connection_draining_timeout = 400

  tags = {
    Name = "Webbudget elb"
  }
}

resource "aws_route53_record" "www" { # Associando o endereço do dominio ao load balancer.
  zone_id = data.aws_route53_zone.selected.zone_id
  name    = var.prefixName
  type    = "A"

  alias {
    name                   = aws_elb.lb.dns_name
```

```
    zone_id                = aws_elb.lb.zone_id
    evaluate_target_health = true
  }
}
```

Arquivo de provisionamento do certificado digital e do load balancer - Terraform

Output.tf – Nesta parte são provisionados os recursos locais que permitem a integração com o Ansible, ele enviar informações cruciais ao Ansible para localização das máquinas virtuais e outros endereços necessários.

```
### Recursos para exportação de informações para o Ansible
resource "local_file" "AnsibleInventory" { # Gera um arquivo de inventário do Ansible a partir de um te
mplate com todos os endereços publicos atuais dos servidores provisionados
 content = templatefile("${path.module}/templates/inventory.tmpl",
 {
  db-machine-ip = aws_instance.db-machine.public_ip,
  web-machine-ip = aws_instance.web-machine.public_ip,
  web-machine2-ip = aws_instance.web-machine2.public_ip }
 )
 filename = "${path.module}/../ansible-
webbudget/inventory/hosts.yml" # Endereço de saida do arquivo de configuração provisionado
}
resource "local_file" "Dns_webserver" { # Gera um arquivo de variaveis do Ansible contendo informações
necessarias para a integração entre as ferramentas
 content = templatefile("${path.module}/templates/all.tmpl",
 {
  web-machine-dns = aws_route53_record.www.fqdn, # Endereço web final do aplicativo provisionado
  db-machine-ip = aws_instance.db-
machine.private_ip, # Endereço ip privado da maquina de banco de dados, para configuração na aplicação
 })
 filename = "${path.module}/../ansible-
webbudget/group_vars/all.yml"}# Endereço de saida do arquivo de configuração provisionado
```

Arquivo que promove a integração entre o Terraform e Ansible - Terraform

variaveis.tf – Este é o arquivo de variáveis do Terraform nele é possível informar as possíveis variações nos recursos a serem provisionados.

```
#CONFIGURAÇÃO DE ACESSO SSH
## COLOQUE ABAIXO O IP DA MÁQUINA CONTROLADORA DO ANSIBLE ##
variable "cidrsControler" {
    default = "179.158.178.134/32"
}
##--------------------------------------##-----------------------------------------##

##CONFIGURAÇÕES DA WEB

#Nome do domínio do site
variable "domainName" {
    default = "cariocatcc.com"
}

#Nome do prefixo do site
variable "prefixName" {
    default = "www"
}

#RANGE DE IP DA VPC
variable "vpcCIDRblock" {
    default = "10.0.0.0/16"
}

#RANGE DE IP DA SUBNET

variable "publicsCIDRblock" {
    type = map(string)
    description = "publicsCIDRblock"
    default = {
        "1" = "10.0.1.0/24"
        "2" = "10.0.2.0/24"
    }
}



#RANGE DE IP DO CLIENTE DO SOFTWARE -
 CASO SEJA NECESSÁRIO RESTRINGIR O ACESSO A APLICAÇÃO A REDE DO CLIENTE
variable "cidrsRemoteAcess" {
    type = list
```

```
    default = ["0.0.0.0/0"]
}

##-------------------------------------##-----------------------------------------##

##VARIÁVEIS DAS MÁQUINAS VIRTUAIS##

variable "amis" {
    type = map(string)
    description = "AMIS"
    default = {
        sa-east-1 = "ami-083aa2af86ff2bd11"
    }
}
variable "region" {
     default = "sa-east-1"
}

variable "keyname" {
     default = "webbudget"
}

##-------------------------------------##-----------------------------------------##

##VARIÁVES DE REDE##

#VARIÁVEIS DA REDE VIRTUAL PRIVADA

variable "instanceTenancy" {
    default = "default"
}

variable "dnsSupport" {
    default = true
}

variable "dnsHostNames" {
    default = true
}

#VARIÁVEIS DA SUBNET
```

```
variable "mapPublicIP" {
    default = true
}

variable "availabilityZone" {
    type = map(string)
    description = "availabilityZone"
    default = {
        "a" = "sa-east-1a"
        "c" = "sa-east-1c"
    }
}

#VARIÁVEIS DA LISTA DE LISTA DE CONTROLE DE ACESSO.

variable "publicdestCIDRblock" {
    default = "0.0.0.0/0"
}
```

Arquivo de variáveis do Terraform – Terraform

O próximo passo é realizar a implantação dos arquivos de definição do Terraform na nuvem da AWS, acessando a pasta do projeto e executando os comandos abaixo no terminal.

```
$ terraform init #comando é usado para inicializar um diretório de trabalho contendo arquivos de config
uração do Terraform
$ terrafom plan # comando é usado para criar um plano de execução. O Terraform realiza uma atualização,
 e então determina quais ações são necessárias para atingir o estado desejado especificado nos arquivos
 de configuração.
$ terrafom apply # comando é usado para aplicar as mudanças necessárias para atingir o estado desejado
da configuração, ou o conjunto pré-determinado de ações gerado por um terraform planplano de execução.
```

Comandos de implantação do Terraform

Ao provisionar os recursos com o comando terraform apply e exibido no terminal um plano de ação que informa as modificações que serão realizadas solicitando a confirmação da ação, como podemos observar na figura 9.

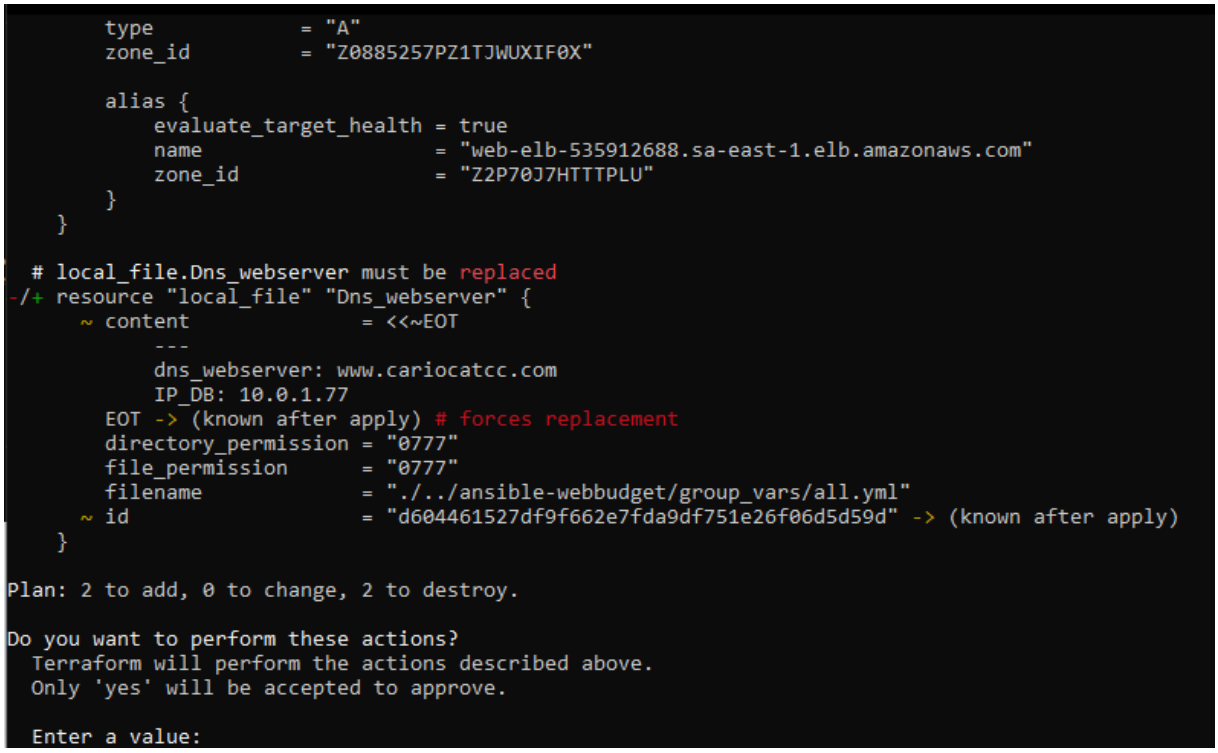

Figura 9 – Painel de provisionamento do Terraform.

- Criando: recursos com um sinal de mais (+) serão criados (em verde)
- Modificando: recursos com um sinal de til (~) serão modificados (em âmbar)
- Destruindo: os recursos com um sinal de menos (-) serão excluídos (em vermelho)

Após confirmar as modificações na infraestrutura, em caso de êxito é exibida uma mensagem, como no exemplo a seguir: Apply complete! Resources: 2 added, 0 changed, 2 destroyed.

5.3 Configuração do ambiente

A configuração dos recursos provisionados é a próxima etapa para o desenvolvimento de um sistema definido por software funcional. A automação do procedimento de configuração é realizada com o Ansible, que é instalado em uma máquina de controle central

é através desta máquina que é possível configurar e instalar software nas máquinas provisionadas instalando pequenos programas, chamados módulos. Uma vez que os módulos são instalados, Ansible usa SSH para executá-los e os remove automaticamente após sua execução.

Atualmente, o Ansible oferece suporte para máquinas de controle com sistemas operacionais Unix. Isso inclui Red Hat, Debian, CentOS, macOS e assim por diante, o Windows não tem suporte para máquina de controle.

É possível utilizar os comandos abaixo para instalar o Ansible dependendo da versão de sistema operacional (Ansible,2020).

```
# No Fedora:
$ sudo dnf install ansible

# No RHEL e CentOS:
$ sudo yum install ansible


# No Ubuntu:
$ sudo apt update
$ sudo apt install software-properties-common
$ sudo apt-add-repository --yes --update ppa:ansible/ansible
$ sudo apt install ansible

# No macOS:
$ sudo brew install ansible
```

Comandos de instalação do Ansible

Primeiro passo é modularizar a estrutura dos diretórios para trabalhar com roles, as roles seguem uma organização de diretórios estabelecida; uma role é denominada pelo diretório de nível superior. Os subdiretórios nomeados de tasks contêm arquivos YAML, nomeados main.yml que possuem ações serrem configuradas nos servidores remotos como por exemplo: Copiar um arquivo, instalar uma dependência, iniciar um contêiner etc. É possível também criar subdiretórios contendo arquivos referenciados pelos arquivos YAML.

Na figura 10 é possível observar a estrutura de pastas do projeto:

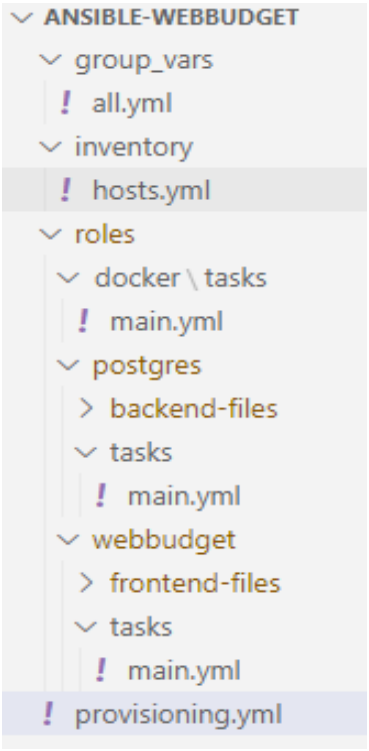

Figura 10 – Estrutura de diretórios Ansible.

A seguir serão analisados os códigos e os comentários explicativos de cada arquivo de definição do Ansible:

- group_vars\all.yml – Este é o arquivo de declaração de variáveis, com ele é possível representar as variações entre diferentes ambientes, neste projeto utilizamos ele para realizar a integração com a ferramenta de provisionamento Terraform.

```
--- # Arquivo de variáveis de grupo do Ansible.
dns_webserver: www.cariocatcc.com # Variável informada pela integração com terraform
IP_DB: 10.0.1.80 # Variável informada pela integração com terraform
```

Arquivo de declaração de variáveis - Ansible.

- inventory\hosts.yml – Este é o arquivo de inventário que é uma lista organizada de servidores que serão gerenciados pelo Ansible, é possível organizar esses servidores em grupos ou de forma unitária. Em neste projeto este arquivo é fornecido via integração com o Terraform que provisiona os recursos e informa os endereços dos servidores ao Ansible.

```
all: # Grupo contendo todos os subgrupos e servidores do projeto.
    children: # Modulo que permite a criação de subgrupos.
        backend: # Nome dado aos servidores de banco de dados.
            hosts: # Modulo que permite declarar os servidores em um grupo.
                db-machine: # Nome dado ao servidor de banco de dados.
                    ansible_host: XXX.XXX.XXX.XXX # Endereço de ip do servidor remoto que
será fornecido pela integração com terraform.
                    ansible_user: ubuntu # Usuario local do servidor remoto.
                    ansible_ssh_private_key_file: ./inventory/webbudget.pem # Chave privad
a para acesso SSH ao servidor remoto.
        frontend:  # Nome dado aos servidores da aplicação web.
            hosts: # Modulo que permite declarar os servidores em um grupo.
                web-machine-1: # Nome dado ao servidor primeiro servidor da aplicação web.
                    ansible_host: XXX.XXX.XXX.XXX # Endereço de ip do servidor remoto que
será fornecido pela integração com terraform.
                    ansible_user: ubuntu # Usuario local do servidor remoto.
                    ansible_ssh_private_key_file: ./inventory/webbudget.pem # Chave privad
a para acesso SSH ao servidor remoto.

                web-machine-2: # Nome dado ao servidor segundo servidor da aplicação web.
                    ansible_host: XXX.XXX.XXX.XXX # Endereço de ip do servidor remoto que
será fornecido pela integração com terraform.
                    ansible_user: ubuntu # Usuario local do servidor remoto.
                    ansible_ssh_private_key_file: ./inventory/webbudget.pem # Chave privad
a para acesso SSH ao servidor remoto.
```

Arquivo de inventario - Ansible.

- provisioning.yml – Este é o arquivo Playbook Ansible, ele é usado para gerenciar configurações e implantações em máquinas remotas, definindo as tasks ou roles a serem executas em um determinado servidor ou grupo de servidores. Neste projeto

optamos em trabalhar com o conceito de roles que é uma forma de reaproveitar um código entre vários Playbooks.

```
---
  - hosts: all # Declara a qual o grupo ou máquina do inventário a role ou task
será executada.
    roles: # Indica a lista de roles a ser executada no host ou grupo de destino.
      - docker # Nome da role a ser executada.

  - hosts: backend # Declara a qual o grupo ou máquina do inventário a role ou task
será executada.
    roles: # Indica a lista de roles a ser executada no host ou grupo de destino.
      - postgres # Nome da role a ser executada.

  - hosts: frontend # Declara a qual o grupo ou máquina do inventário a role ou task
será executada.
    roles: # Indica a lista de roles a ser executada no host ou grupo de destino.
      - webbudget # Nome da role a ser executada.
```

Arquivo Playbook - Ansible

- roles\docker\tasks\main.yml – Este é o arquivo de tarefas da role Docker nele existem tarefas programas com os passos necessários para configuração de um ambiente Docker em uma máquina Ubuntu, como por exemplo a instalação de seus pacotes e dependências.

```
---
- name: Instalando aptitude.
  apt:  # Módulo de instalação de pacotes e dependências apt.
    name: aptitude
    state: latest
    update_cache: yes
    force_apt_get: yes
  become: yes

- name: Instalando pacotes requeridos pelo sistema.
  apt: # Módulo de instalação de pacotes e dependências apt.
    name: ['apt-transport-https', 'ca-certificates', 'curl', 'software-properties-
common', 'python3-pip', 'virtualenv',
'python3-setuptools', 'docker-compose']
    state: latest
```

```
    update_cache: yes
  become: yes

- name: Instalando chave do repositório do docker.
  apt_key: # Módulo de instalação de chaves de repositórios apt.
    url: https://download.docker.com/linux/ubuntu/gpg
    state: present
  become: yes

- name: Adicionando repositório docker.
  apt_repository: # Módulo para adição de repositórios apt.
    repo: deb https://download.docker.com/linux/ubuntu bionic stable
    state: present
  become: yes

- name: Atualizando repositórios e instalando docker-ce
  apt:  # Módulo de instalação de pacotes e dependências apt.
    name: docker-ce
    state: latest
    update_cache: yes
  become: yes


- name: Instalando o módulo Docker para Python.
  pip: # Módulo de instalação de pacotes e dependências Python.
    name: docker
  become: yes

- name: Adicionando usuário ao grupo "docker".
  user: # Módulo de configuração de usuário e grupos do linux.
    name: ubuntu
    group: docker
    append: yes
  become: yes
```

Arquivo de tasks da role Docker - Ansible

- roles\postgres\tasks\main.yml - Este é o arquivo de tarefas da role postgres, ele é responsável pelas tarefas executadas para levantar o contêiner do banco de dados deste projeto.

```
---
    - name: Copiando arquivos do back-end e definindo proprietário e permissões dos arquivos.
      copy: # modulo de cópia de arquivos.
        src: ./backend-files
        dest: /home/ubuntu
        owner: ubuntu
        group: docker
        mode: '0664'
      become: yes

    - name: Iniciando o contêiner do banco de dados.
      docker_compose: # Modulo de inicialização do docker-compose.
        project_src: /home/ubuntu/backend-files/
        build: no
      register: output
      become: yes
```

Arquivo de tasks da role Postgres - Ansible

- roles\webbudget\tasks\main.yml - Este é o arquivo de tarefas da role webbudget, nela estão descritas as tarefas e passos necessários para construir e iniciar a imagem do contêiner da aplicação.

```
---
  - name: Copiando arquivos do front-end e definindo propretário e permissões dos arquivos.
    copy: # Modulo de copia de arquivos.
      src: ./frontend-files
      dest: /home/ubuntu/
      owner: ubuntu
      group: docker
      mode: '0664'
    become: yes

  - name: Configurando ip privado atual do banco de dados no docker-compose.
    replace: # Módulo que substitui todas as instâncias de um padrão em um arquivo.
      path: /home/ubuntu/frontend-files/docker-compose.yml
      regexp: 'IP_DB'
      replace: "{{ IP_DB }}" # Recebe esse parametro em uma variável informada pela integração
 com terraform.
    become: yes
```

```
 - name: Iniciando o conteiner da aplicação.
   docker_compose: # Modulo de inicialização do docker-compose.
     project_src: /home/ubuntu/frontend-files/
     build: yes
   register: output
   become: yes

 -
name: Informando endereço do site. # Mensangem amigavél exibida no final das configurações.
   debug: # Modulo de que exibe informações da tela de controle do Ansible.
     msg:
       -
'Seu portal esta pronto no endereço: {{ dns_webserver }}' # Recebe esse parametro em uma vari
ável informada pela integração com terraform.
       - 'Observações:'
       - '- Pode ser necessário até 15 min para propagação do endereço no DNS Global'
```

Arquivo de tasks da role Webbudget - Ansible

O próximo é realizar a implantação dos arquivos de definição do Ansible no ambiente computacional já provisionado pelo Terraform, acessando a pasta do projeto e executando o comando abaixo no terminal.

```
$ ansible-playbook -l all -i ./inventory/hosts.yml provisioning.yml
```

Comando de implantação do Ansible

É possível descrever as partes deste comando da seguinte forma:

- ansible-playbook - Executa os Playbooks do Ansible, executando as tarefas definidas nos hosts de destino.
- -l - Especifica qual grupo será executado o playbook.
- -i - Especifica o caminho do arquivo de inventário.

Na figura 11 é demostrado a execução do comando de implantação, é possível observar as tarefas sendo executadas nos servidores remotos.

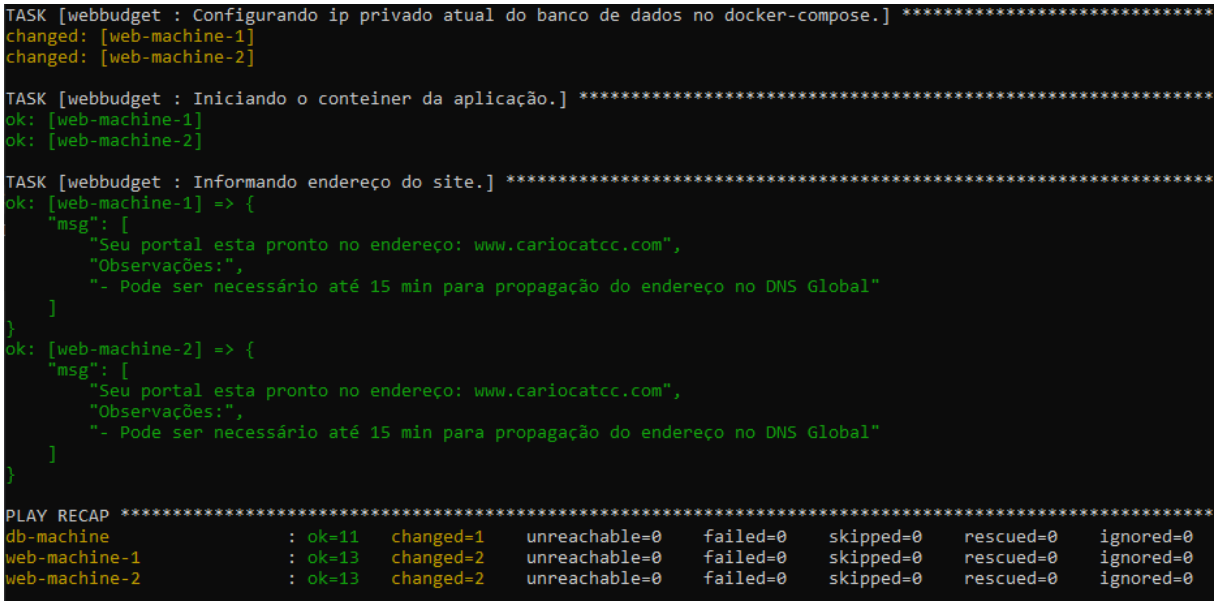

Figura 11 – Painel de execução das tarefas - Ansible.

A cada execução de uma tarefa é esperado um retorno que pode ser:

- Ok – O Ansible verifica que o servidor já se encontra no estado desejado pela tarefa, logo não será necessário realizar alterações.
- Changed – A tarefa é executada com sucesso, configurando o servidor remoto e mudando o status daquela tarefa para concluído.
- Unreachable – O Ansible não consegue se conectar o servidor remoto para executar as tarefas.
- Failed – A tarefa não pode ser executada retornado um erro.
- Skipped – O Ansible informa que uma tarefa foi pulada.
- Rescued – Um bloco de resgate do código foi executado.
- Ignored – Quando um erro de no código foi ignorado para critérios de teste.

Caso todo o processo de configuração ocorra com êxito o Ansible está programado para informar o endereço do site provisionado através de uma mensagem no console, como é possível observar na figura 11.

5.4 Interoperabilidade entre as ferramentas

A implantação de um sistema de forma automatizada pode ser um desafio, pois envolve várias novas tecnologias que são altamente dependentes umas das outras para produzir um sistema totalmente funcional.

A integração e a interoperabilidade entre essas tecnologias e ferramentas se torna crucial, e são realizadas através de API’s e módulos das ferramentas e do provedor de serviço em nuvem.

Na figura 12 é possível observar um diagrama ilustrando essas integrações.

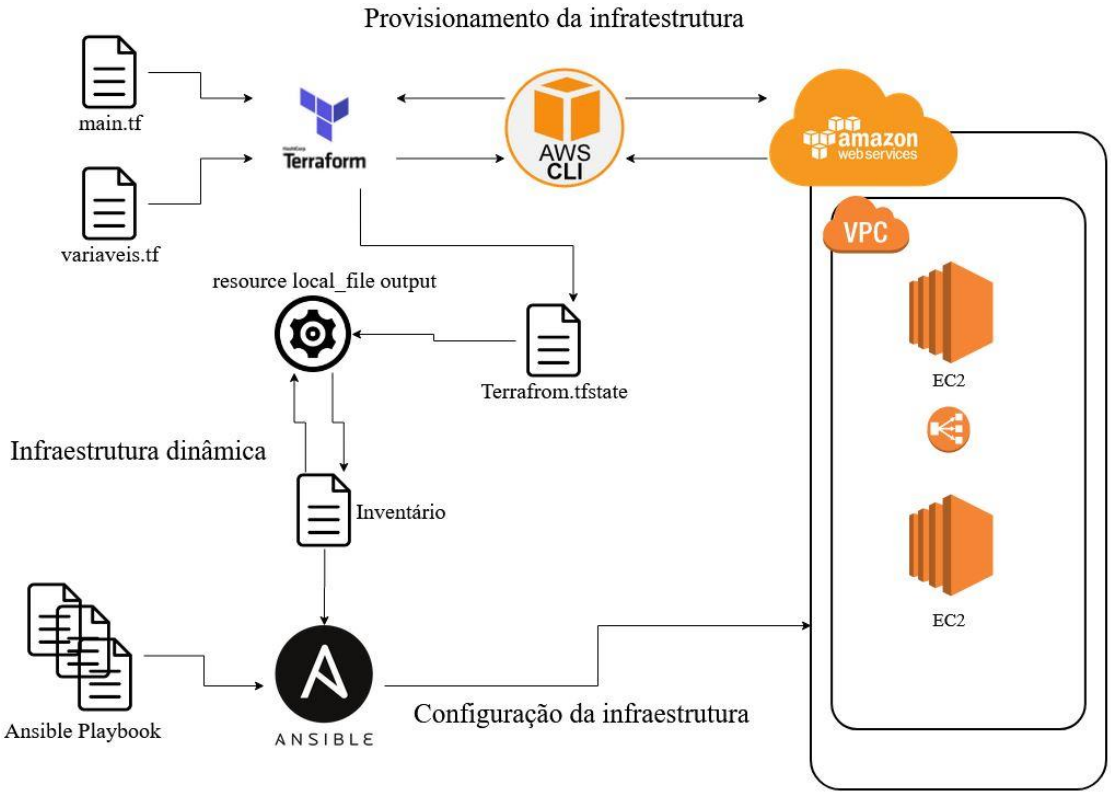

Figura 12 – Diagrama de integração das ferramentas.

O Terraform faz a leitura de seus arquivos de definição e planeja a adequação da infraestrutura para o estado desejado, ao realizar o provisionamento da infraestrutura planejada ele converte os arquivos de definição para scripts que são interpretados pelo AWS CLI que por sua vez se comunica com a API da AWS na nuvem e provisiona os recursos. A API da AWS retorna para o AWS CLI com as informações dos recursos provisionados, essas informações são repassadas ao Terraform e armazenadas no arquivo Terraform.tfstate.

Através de um módulo do Terraform chamado local_file alguns dados do arquivo Terraform.tfstate são repassados para o arquivo de inventário do Ansible, que por sua vez agora com as informações do inventário se conecta aos servidores provisionados por SSH e logo executa as configurações como programado em seu Playbook.

5.5 Arquitetura final do projeto

Neste capítulo, a arquitetura final do projeto proposto é descrita. Na figura 13 é possível observar todos os componentes provisionados e configurados pelas ferramentas de infraestrutura como código.

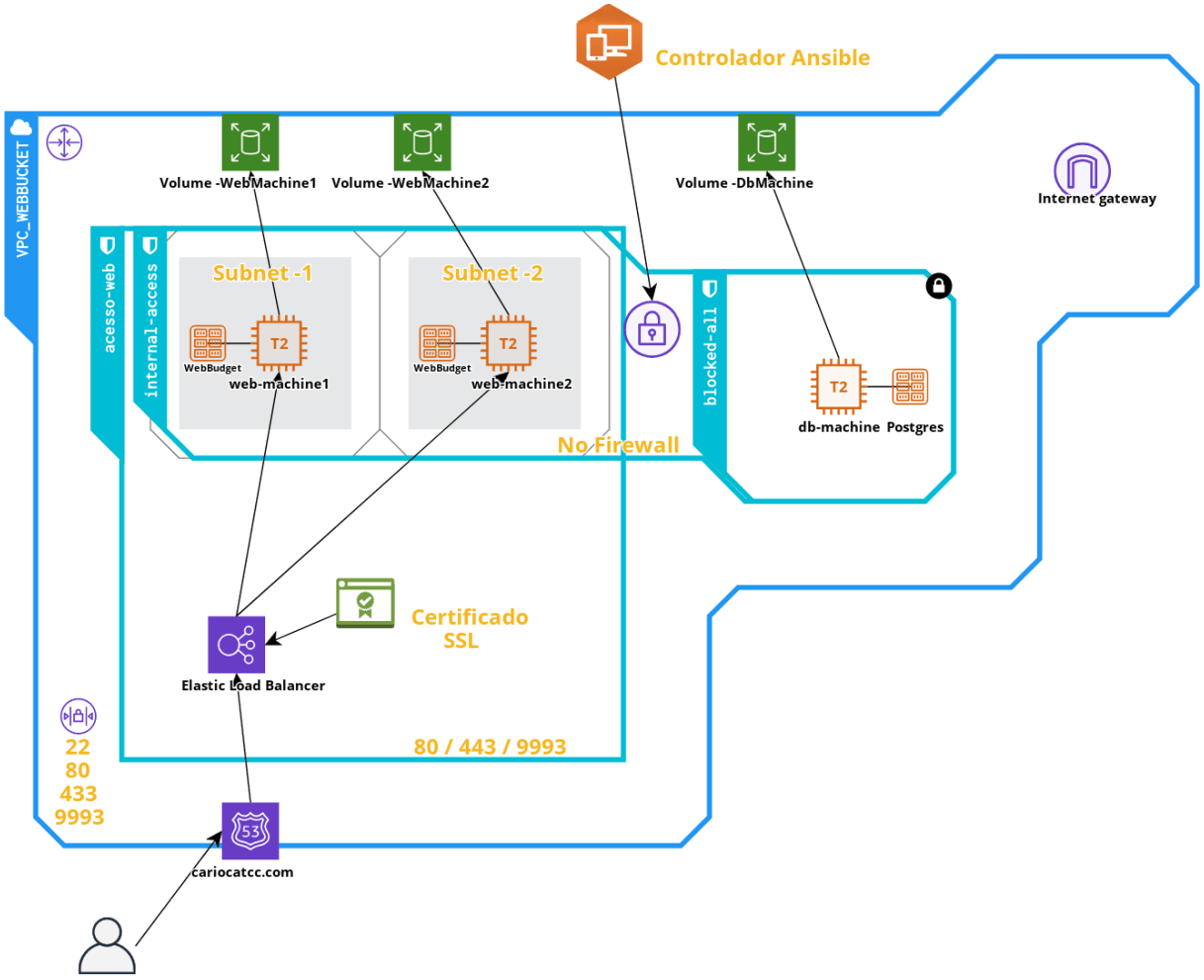

Figura 13 – Arquitetura do ambiente provisionado.

Os principais componentes são as máquinas virtuais seus discos e contêineres, foram provisionadas uma máquina para o banco de dados e duas máquinas para aplicação.

O load balancer é responsável por distribuir as cargas entre essas duas máquinas da aplicação, neste load balancer foi provisionado um certificado digital para acesso seguro via protocolo HTTPS.

Existe um servidor DNS que redireciona todo o tráfico web do endereço https://www.cariocatcc.com para este load balancer.

Uma rede virtual privada (VPC) foi criada, junto com ela foi provisionado uma lista de controle de acesso (ACL) que funciona como firewall permitindo apenas conexão nas portas TCP 22, 80,443 e 9993, um Internet Gateway também foi provisionado para permitir acesso da VPC para internet.

Alguns grupos de segurança foram provisionados, eles funcionam como uma camada adicional de segurança, semelhante a um firewall controlando o tráfego de entrada e saída, é possível observar os grupos de segurança e suas regras abaixo.

- Acesso-web: Permite acesso as portas 80, 443 e 9993, apenas as máquinas da aplicação se encontram nesta regra, ela que garante o acesso do usuário final a aplicação.
- Blocked-all: Esta regra bloqueia todo acesso externo liberando apenas conexões de saída, nesta regra se encontra a máquina do banco de dados.
- Internal-access: Esta regra garante o acesso completo entre as máquinas da aplicação e do banco de dados e permite também liberar o acesso para máquina de controle do Ansible para que seja possível configurar os servidores.

6. CONCLUSÃO

O estudo destacou como a infraestrutura de TI era provisionada é gerenciada antes da virtualização e da nuvem, e como conceitos e ferramentas como metodologia ágil, DevOps e IaC afetam os processos de gestão da Infraestrutura de TI.

Uma visão geral sobre DevOps e infraestrutura como código foi realizada, abordando conceitos, benefícios, desafios, práticas e ferramentas.

Um estudo sobre as oportunidades e os desafios para adoção da infraestrutura como código em uma organização foi considerado importante, então um estudo de caso foi realizado para obter uma visão sobre como e quais são as suas implicações práticas.

Como resultado é possível reconhecer o papel da metodologia ágil, da cultura DevOps e da infraestrutura como código para melhoria dos processos de provisionamento e configuração da infraestrutura. As descobertas sugerem que as empresas devem adotar a cultura DevOps e a infraestrutura como código, para poder provisionar e configurar a infraestrutura de maneira mais rápida, eficiente e eficaz, acompanhando assim o ciclo de vida cada vez mais rápido dos lançamentos e atualizações de softwares exigido pelo mercado.

Com o avanço contínuo das tecnologias em nuvem melhorias sempre podem ser realizadas, como por exemplo, a implantação em recursos especializados da nuvem, que são auto redimensionáveis e auto escaláveis.

REFERÊNCIAS BIBLIOGRÁFICAS

AMAZON. **Auto Scaling Groups.** Disponível em <https://docs.aws.amazon.com/autoscaling/ec2/userguide/AutoScalingGroup.html>. Acesso em: 18 de outubro de 2020.

AMAZON. **AWS Cloud Formation.** Disponível em <https://aws.amazon.com/cloudformation/>. Acesso em: 18 de outubro de 2020.

AMAZON. **Installing, updating, and uninstalling the AWS CLI version 2.** Disponível em <https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html>. Acesso em: 29 de outubro de 2020.

ANSIBLE. **Installing Ansible.** Disponível em < https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html>. Acesso em: 29 de outubro de 2020.

BANG, S.K.; CHUNG, S.; CHOH, Y; DUPUIS, M. **A Grounded Theory Analysis of Modern Web Applications - Knowledge, Skills, and Abilities for DevOps**. In Proceedings of the 2nd annual conference on Research in information technology. New York. 2013. 61–62 p.

BERNSTEIN, David. **Containers and Cloud: From LXC to Docker to Kubernetes**. IEEE Cloud Computing. 1. 2014. 81-84 p.

BRIKMAN, Yevgeniy. **Terraform: Up and Running: Writing Infrastructure as Code**. 1 ed, O'Reilly Media Inc. 2017.

DOCKER. **O serviço líder mundial para localizar e compartilhar imagens de contêineres.** Disponível em <https://www.docker.com/products/docker-hub>. Acesso em: 23 de outubro de 2020.

EBERT, C.; GALLARDO, G.; HERNANTES, J.; SERRANO, N. **DevOps**, IEEE Software 33, 3. Ieee Software. Maio,2016. 94–100 p.

ENDRES, Christian; BREITENBUCHER, Uwe; FALKENTHAL, Michael; KOPP, Oliver; LEYMANN, Frank; WETTINGER, Johannes. **Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications.** 9th International Conference of Pervasive Patterns and Applications. 2017. 22-27 p.

FOREMAN. **Foreman is a complete lifecycle management tool for physical and virtual servers.** Disponível em <https://theforeman.org/>. Acesso em: 18 de outubro de 2020.

GREGÓRIO, Arthur. **WebBudget um sistema de controle financeiro grátis e de código aberto.** Disponível em <https://webbudget.com.br/>. Acesso em: 18 de outubro de 2020.

HARSHICORP. **HarshiCorp Terraform.** Disponível em <https://www.terraform.io/>. Acesso em: 18 de outubro de 2020.

HOSONO, Shigeru. **A DevOps framework to shorten delivery time for cloud applications.** International Journal of Computational Science and Engineering. 7. 2012. 329-344 p.

HUMMER, W.; ROSENBERG, F.; OLIVEIRA, F.;EILAM, T. **Testing idempotence for infrastructure as code.** In ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing. Berlin, 2013. 368-388 p.

KATSAROS, Gregory; MENZEL, Michael; LENK, Alexander; RAKE-REVELANT, Jannis; SKIPP, Ryan; EBERHARDT, Jacob. **Cloud Application Portability with TOSCA, Chef and Openstack.** In Proceedings of the 2014 IEEE International Conference on Cloud Engineering. IEEE Computer Society. 2014. 295-302 p.

LOOPE, James. **Managing Infrastructure with Puppe**t. 1 ed. O'Reilly Media Inc. 2011.

MEDJAOUI, Mehdi; WILDE, Erik; MITRA, Ronnie; AMUNDSEN, Mike. **Continuous API Management: Making the Right Decisions in an Evolving Landscape.** 1 ed, O'Reilly Media Inc. 2019.

MICROSOFT. **Azure Resource Manager Overview**. Disponível em <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-overview>. Acesso em: 18 de outubro de 2020.

MORRIS, Kief. **Infrastructure as Code: Managing Servers in the Cloud.** 1 ed, O'Reilly Media Inc. 2016. 5-40 p.

MYERS, Colton. **Learning SaltStack: Learn how to manage your infrastructure by utilizing the power of SaltStack.** 2 ed. Packt Publishing Ltd. 2016.

MELL, Peter; GRANCE, Timothy. **The NIST Definition of Cloud Computing.** NIST Special Publication 800-145. NIST. 2011.

OPENSTACK. **Adding Speed and Agility to Virtualized Infrastructure with Openstack.** Disponível em <https://www.openstack.org/assets/pdf-downloads/virtualization-Integration-whitepaper-2015.pdf> Acesso em: 18 de outubro de 2020.

OPENSTACK. **Openstack Heat.** Disponível em <https://wiki.openstack.org/wiki/Heat>. Acesso em: 18 de outubro de 2020.

OPENSTACK. **Telemetry.** Disponível em <https://wiki.openstack.org/wiki/Telemetry>. Acesso em: 18 de outubro de 2020.

REDHAT. **Ansible in Depth.** Disponível em <http://www.cmsdistribution.com/wp-content/uploads/2016/09/Ansible-in-Depth-Whitepaper.pdf>. Acesso em: 18 de outubro de 2020.

SABHARWAL, Navin; WADHWA, Manak. **Automation through Chef Opscode: A Hands-on Approach to Chef.** 1 ed. Apress. 2014. 5 p.

SDXCENTRAL. **What is Software Defined Compute?** Disponível em <https://www.sdxcentral.com/networking/sdn/definitions/what-is-software-defined-compute/>. Acesso em: 18 de outubro de 2020.

SYNERGY. **Quarterly Cloud Spending Blows Past $30B; Incremental Growth Continues to Rise.** Disponível em <https://www.srgresearch.com/articles/quarterly-cloud-spending-blows-past-30b-incremental-growth-continues-rise> Acesso em: 20 de outubro de 2020.

TAYLOR, Kelsey. **10 Best Open Source Hypervisor.** Disponível em < https://www.hitechnectar.com/blogs/open-source-hypervisor/>. Acesso em: 18 de outubro de 2020.

USTINOVA, Tatiana; JAMSHIDI, Pooyan. **Modelling Multi-tier Enterprise Applications Behaviour with Design of Experiments Technique.** In Proceedings of the 1st International Workshop on Quality-Aware DevOp. New York. 2015 13–18 p.

WEI. **Five Game-Changing Advantages of Software Defined Computing.** Disponível em <https://cdn2.hubspot.net/hubfs/1774733/Tech_Briefs/Tech_Brief_-_5_Game-changing_Advantages_of_Software_Defined_Computing.pdf> Acesso em: 18 de outubro de 2020.

GLOSSÁRIO

Commit: É uma operação que envia as últimas alterações do código-fonte para o repositório.

Firewall: Equipamento ou software responsável em filtrar, regrar e limitar o acesso, entrante e sainte, de dados de uma rede de computador interna e uma rede externa, com o propósito de segurança.

Git: É um Sistema de Controle de Versão Distribuída.

GitHub: É uma empresa com fins lucrativos que oferece serviços de hospedagem de repositório Git baseado em nuvem.

Hipervisor: Um hipervisor é um processo que separa o sistema operacional e os aplicativos de um computador do hardware físico subjacente.

Load Balancer: É um balanceador de carga que distribui o tráfego de rede ou aplicativo em vários servidores.

Virtualização: Se refere ao ato de criar uma versão virtual de algo, incluindo plataformas de hardware de computador virtual, dispositivos de armazenamento e recursos de rede de computador.

Push: Em sistemas de controle de versão e um comando para enviar o conteúdo do repositório local para um repositório remoto

Pull: Em sistemas de controle de versão e um comando para puxar o conteúdo de um repositório remoto para um repositório local.