Desenvolvimento de aplicações utilizando máquinas virtuais: comparativo entre desenvolvimento local e remoto

Alexandre Zigiotto Zeni, Fabrício Tonetto Londero
Curso de Ciência da Computação
UFN - Universidade Franciscana
Santa Maria - RS
alexandrezzeni@gmail.com, fabriciotonettolondero@gmail.com

***Resumo*—Em uma empresa observou-se uma necessidade de melhorar o desempenho e reduzir a utilização das máquinas de programadores devido aos serviços e tecnologias utilizadas no desenvolvimento de aplicações. Com base nisso, o presente trabalho realizou a criação e configuração de máquinas virtuais, em ambientes locais e remotos e então realizou comparações entre eles. Para a metodologia foram analisados os requisitos de *hardware* e *software* necessários pelos programadores; foi escolhido o ambiente (local ou nuvem); foram criados os arquivos de configuração do *terraform* e *ansible*; foram criadas as máquinas no ambiente escolhido e foi posteriormente realizado testes para validar as configurações e analisar o desempenho. O trabalho apresentou resultados positivos conforme o esperado, mostrando que as máquinas virtuais (locais e remotas) apresentam desempenho igual ou superior às máquinas dos programadores, apesar de possuírem um hardware inferior. Também foi observado que, com a utilização dessas máquinas, ocorreu uma redução na utilização de recursos nas máquinas dos programadores e analisando os custos foi concluído que para esse tipo de aplicação o uso de máquinas locais apresentam um melhor custo-benefício.**

***Palavras-chave* : Virtualização; Terraform; Ansible;**

I. Introdução

Na atualidade, as formas de desenvolvimento de aplicações estão em constante mudança, principalmente com a adoção da computação em nuvem e da arquitetura de microsserviços. O desenvolvimento de *software* não se concentra em apenas uma aplicação, mas sim em diversas aplicações (ou serviços) que operam em conjunto para formar um sistema completo. Deste modo, as empresas de tecnologias estão sempre buscando novas formas e tecnologias para o desenvolvimento de *software*, gerando uma grande diversidade nas ferramentas e ambientes utilizados no desenvolvimento[1].

Para o desenvolvimento de uma aplicação desse tipo é necessário que a máquina do programador consiga executar todas as aplicações que fazem parte do sistema completo, o que requer grande poder computacional. Junto a isso, o uso de diversas linguagens de programação e ambientes de desenvolvimento acabam dificultando o desenvolvimento.

O intuito deste trabalho é apresentar a possibilidade do programador desenvolver *software* de forma remota, sem que seja necessário instalar e executar o *software* na sua própria máquina, permitindo maior flexibilidade, maior segurança no desenvolvimento de aplicações e com menor custo computacional para a sua máquina. Com a utilização do processo de *Infrastructure as Code* (IaC[1]) é possível subir os ambientes de forma rápida e automatizada, reduzindo a chances de erros humanos.

A. Justificativa

Em uma empresa foi observada a necessidade de encontrar soluções para aumentar o desempenho e reduzir a carga nas máquinas dos programadores. Devido às tecnologias e serviços utilizados no desenvolvimento de aplicações nesta empresa se faz necessário alto número de recursos computacionais (processador e memória), principalmente no desenvolvimento de microsserviços que utilizem containers como ambiente de desenvolvimento.

As vantagens esperadas são uma redução da carga nas máquinas dos programadores, um aumento na velocidade para execução dos projetos e um aumento na segurança, pois os projetos e quaisquer outros dados serão mantidos nas máquinas virtuais dentro dos servidores e não nas máquinas dos programadores, com a possibilidade de realizar backups frequentes e a maior facilidade para restaurá-los.

B. Objetivos

O objetivo deste trabalho é analisar as vantagens e desvantagens, junto com realizar uma comparação entre a utilização de máquinas físicas (do programador) e máquinas virtuais (servidores físicos e/ou em nuvem) para o desenvolvimento de aplicações e *sotfware*. E utilizar ferramentas de automação para criação, configuração e gerenciamento dos ambientes e máquinas virtuais, no caso o *ansible* e *terraform*.

Para isso será avaliado o desempenho bruto das máquinas utilizando ferramentas voltadas para isso e também será avaliado o desempenho específicos das máquinas para o desenvolvimento de *software*, junto com uma análise dos

[1]IaC ou Infraestrutura como código se refere ao provisionamento e gerenciamento de infraestrutura utilizando código, ou seja, utilizar arquivos de configuração para especificar as configurações da infraestrutura para facilitar edição e a distribuição das suas configurações.

custos para cada ambiente. Então será realizada uma comparação entre os ambientes para determinar as vantagens e desvantagens de cada ambiente e suas viabilidades.

II. Virtualização

Virtualização é uma tecnologia que permite criar serviços de TI utilizando recursos tradicionalmente vinculados a um determinado *hardware*. Com ela é possível utilizar a capacidade total de uma máquina física distribuindo seus recursos entre vários usuários ou ambientes[2].

A virtualização utiliza *software* para criar uma camada de abstração sobre o *hardware* do computador, permitindo que esse *hardware* (processador, memória, armazenamento, etc.) seja dividido em múltiplos computadores virtuais chamados de máquinas virtuais (VMs) e cada máquina virtual executa seu próprio sistema operacional e se comporta como um computador independente. Ela permite um uso mais eficiente do *hardware* do computador e atualmente é uma prática comum nas empresas e é a base da computação em nuvem, trazendo diversos benefícios aos operadores de *datacenters*[2] e provedores de serviços, entre eles, eficiência na utilização de recursos, facilidade no gerenciamento, provisionamento mais rápido e tempo mínimo de *downtime*[3].

Um dos requisitos necessários para a virtualização é um *software* chamado de *hypervisor*. O *hypervisor* ou VMM (*Virtual machine monitor*) é uma camada de *software* que permite diversos sistemas operacionais (SOs) executarem simultaneamente, compartilhando recursos da máquina física. Ele gerencia e separa as VMs umas das outras de forma lógica, garantindo que elas sejam executadas independentes. Existem duas grandes categorias de *hypervisor*: o tipo 1 também chamado de *bare-metal* (Figura 1) que executa diretamente na máquina física e ocupa o lugar do sistema operacional do *host*, e é o mais comum de ser encontrado em *datacenters* e servidores; e o tipo 2 (Figura 2) que se comporta como uma aplicação e é executada sobre um sistema operacional no *host*, é mais comum em computadores individuais e para uso pessoal[4].

[2]Centro de processamento de dados (CPD) é um local onde estão concentrados os servidores de uma empresa ou organizção

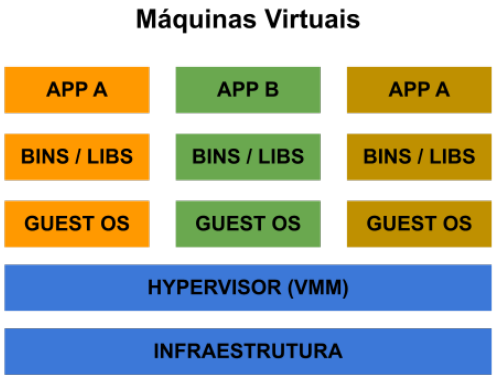

Figura 1. Arquitetura de Máquinas Virtuais (*Hypervisor* tipo 1). Fonte: Autor.

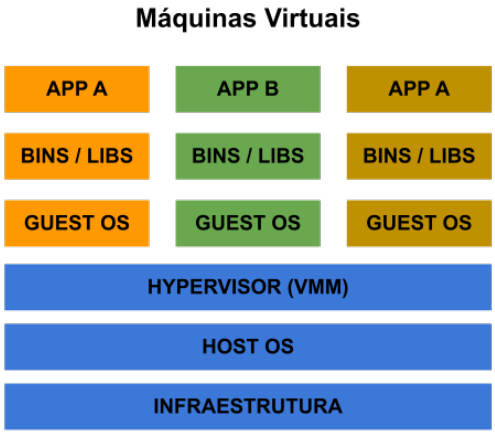

Figura 2. Arquitetura de Máquinas Virtuais (*Hypervisor* tipo 2). Fonte: Autor.

III. Máquinas Virtuais

A máquina virtual é uma representação virtual ou emulação de um computador físico, ou seja, ela possui todos os recursos, libs[3], *bins*[4] e funcionalidades de uma máquina física. Essas máquinas virtuais são normalmente chamadas de *guest* enquanto o computador físico executando elas é chamado de *host*[5].

Como citado na secção anterior, as máquinas virtuais são executadas sobre um *hypervisor* sendo executado no *host*. O *hypervisor* trata os recursos de computação como um pool[5] (conjunto de recursos) que pode ser realocado com facilidade

[3]As libs (ou *libaries*) são bibliotecas que contém uma coleção de subprogramas utilizados no desenvolvimento de software, elas contêm códigos, dados e funções para auxiliar no desenvolvimento

[4]Os *bins* são arquivos de imagem para a instalação de sistemas operacionais, programas de softwares, driver de dispositivo, entre outros, de modo a ocupar menos espaço em disco

[5]É uma coleção de recursos agregados, no caso CPU, memória e armazenamento

entre os *guests* existentes ou para novas máquinas virtuais. O uso de máquinas virtuais permite a utilização de sistemas operacionais distintos sendo executados simultaneamente em um único computador de forma isolada, e cada sistema operacional é executado da mesma forma que em uma máquina física, portanto a experiência do usuário final é emulada dentro da VM de modo similar a utilização de uma máquina física[6].

IV. Containers

Containers são similares às máquinas virtuais, porém eles são unidades leves e portáteis de *software* que inclui tudo o que é necessário para executar um aplicativo, incluindo o código, as bibliotecas e as dependências, de maneira comum para poder ser executado em qualquer lugar independente do *hardware* e *software* presente na máquina *host*. Eles são criados a partir de imagens, que são pacotes de *software* pré-configurados e que contém todas as instruções e dependências necessárias para executar um aplicativo específico. Containers são pequenos, rápidos e portáteis porque ao contrário de uma VM, que emula todo o sistema operacional e serviços, os containers utilizam funcionalidades e recursos do sistema operacional sendo executado no *host*. Como pode ser observado na Figura 3, cada container contém apenas o aplicativo a ser executado e as suas dependências, não contendo um sistema operacional inteiro[7].

As principais vantagens de utilizar containers é que eles são leves por utilizarem o *kernel* do sistema operacional do *host*, portáteis e independem de plataforma por carregarem todas as suas dependências, suportam arquiteturas modernas de desenvolvimento por serem leves e portáteis e possuem melhor uso de recursos do *host* por permitirem uma arquitetura de microsserviços e fácil escalabilidade[7].

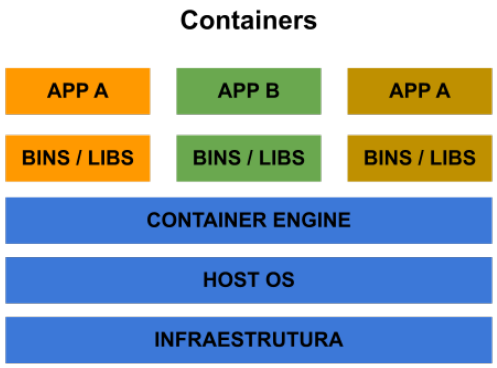

Figura 3. Arquitetura de Containers. Fonte: Autor.

V. Gerência de estruturas

Os orquestradores ou ferramentas de orquestração são ferramentas que permitem ao programador ou administrador de sistemas criar, configurar e gerenciar de forma automatizada serviços, aplicações e recursos de *hardware*. Alguns exemplos são ferramentas como o *ansible*, *terraform*, *kubernetes*, entre outras. Essas ferramentas são comumente encontradas no contexto de arquiteturas orientadas a serviço, virtualização e provisionamento de recursos na computação em nuvem[8].

Enquanto o *terraform* e o *ansible* são ferramentas utilizadas para criar e configurar estruturas em nuvem, o Kubernetes (também chamado de k8s) é uma plataforma de orquestração de containers *open source* que automatiza grande parte dos processos manuais necessários para criar, gerenciar e escalar aplicações em containers. Um cluster Kubernetes pode abranger diversos tipos de *hosts*, sejam eles, físicos ou em nuvem e é a forma mais utilizada para hospedar serviços na nuvem devido a sua alta disponibilidade e escalabilidade[9].

VI. Computação em Nuvens

A computação em nuvem é um serviço de recursos computacionais sob demanda acessado pela internet, sejam eles, aplicações, servidores (físicos e/ou virtuais), armazenamento, ferramentas de desenvolvimento, recursos de rede, entre outros. Esses recursos são hospedados em um *datacenter* remoto, sendo controlados por um provedor de serviços em nuvem (Amazon Web Services, Microsoft Azure, Google Cloud, entre outros) e disponibilizados por uma assinatura mensal. Comparado ao modelo tradicional de servidores locais, esse modelo traz diversas melhorias, entre elas, redução de custos com TI, maior agilidade, maior custo-benefício e maior facilidade para escalar aplicações e serviços. A virtualização é uma funcionalidade fundamental para a computação em nuvem, pois permite maximizar o uso dos recursos de servidores em *datacenters*[10].

A. Modelos de serviços

A maioria dos modelos de computação em nuvem são divididos nas seguintes categorias: IaaS, PaaS, SaaS e FaaS (que serão explicados na sequência). Essa divisão dos modelos de serviços permite uma maior flexibilidade e facilidade na configuração de serviços em nuvem, pois cada categoria/-modelo possuí características que podem facilitar a criação e manutenção de sistemas com características específicas[11].

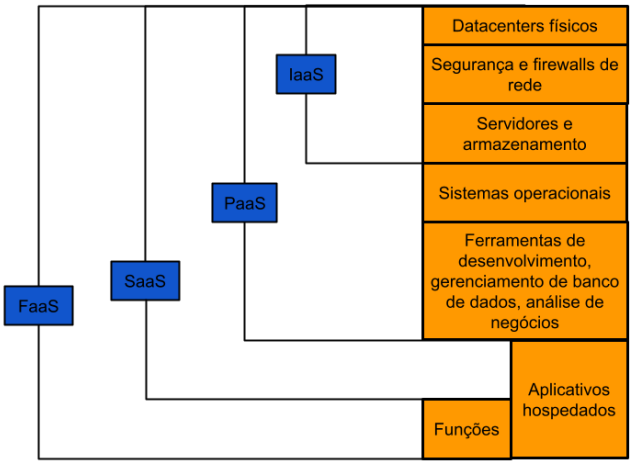

Figura 4. Modelos de Serviço. Fonte: Autor.

1) IaaS (Infrastructure-as-a-Service): Também conhecido como Infraestrutura como Serviço (Figura 4), é um modelo de computação em nuvem que fornece ao consumidor acesso fundamental e sob demanda a recursos computacionais como servidores físicos, servidores virtuais, rede, armazenamento e recursos computacionais. Com o IaaS não é necessário que o cliente adquira servidores físicos, pois o provedor do IaaS é responsável por gerenciar e manter a infraestrutura em nuvem, o cliente precisa apenas instalar, configurar e gerenciar os recursos que desejar, junto a isso o usuário paga apenas pelo que utilizar, ao contrário do modelo de servidores locais onde há um grande custo inicial para adquirir os servidores. Em comparação ao SaaS e Paas, o Iaas fornece ao usuário o maior nível de controle sobre os recursos computacionais na nuvem. Iaas era o modelo mais popular quando foi criado em 2010, mas perdeu espaço para os modelos de Paas e Saas[12].

2) PaaS (Platform-as-a-Service): Também conhecido como Plataforma como Serviço (Figura 4), é um modelo de computação em nuvem que fornece ao consumidor uma plataforma em nuvem completa, com *hardware*, *software* e infraestrutura para desenvolvimento e para executar e gerenciar aplicações sem as desvantagens de servidores físicos. Assim como o IaaS, o PaaS inclui infraestrutura, servidores, armazenamento e rede, além de diversos outros serviços. Ele é criado para dar suporte ao ciclo de vida de um aplicativo Web completo: compilação, teste, implantação, gerenciamento e atualização. O provedor do PaaS hospeda os servidores, as estruturas de rede, o armazenamento, SO, *software*, banco de dados e ferramentas de desenvolvimento em seus *datacenters* em que cliente apenas paga para usar esses recursos para desenvolver, testar e executar suas aplicações. Alguns benefícios do PaaS são: menores custos, maior flexibilidade para os times de desenvolvimento, facilidade para escalar a aplicação, escalabilidade com custo-benefício e permite testar ambientes com um risco menor[13].

3) SaaS (Software-as-a-Service): Também conhecido como aplicação em nuvem ou Software como Serviço (Figura 4), é uma aplicação que está hospedada na nuvem acessada via um navegador web, computador dedicado ou uma API[6] que integra essa aplicação com um computador ou aplicativo móvel, e o provedor da aplicação é responsável por gerenciar e manter a aplicação e a infraestrutura na nuvem funcionando. O provedor do SaaS gerencia tudo necessário para a aplicação funcionar e o cliente apenas cuida da aplicação em si, ou seja, o provedor gerencia os componentes de *hardware* (servidores, rede, armazenamento), SO, requisitos da aplicação, entre outros. É o serviço mais comum na computação em nuvem e o modelo dominante de entrega de *software*. Alguns benefícios do SaaS são: redução de custos, escalabilidade, atualizações automáticas do serviço e proteção contra perda de dados[14].

4) FaaS (Function-as-a-Service): Também conhecido como Função como Serviço, é um modelo de computação em nuvem que funciona como um serviço, ele permite que clientes executem código em resposta a eventos, sem a necessidade de gerenciar uma infraestrutura complexa, normalmente associada com a criação e execução de aplicações e microsserviços. Nesse modelo o *hardware*, máquina virtual e o SO são gerenciados pelo provedor da nuvem, permitindo que os desenvolvedores foquem totalmente no desenvolvimento da aplicação. Alguns benefícios do FaaS são: maior foco no código e não na infraestrutura, pagar apenas pelo que foi utilizado, escalabilidade automática e diversos benefícios de uma estrutura de nuvem robusta[15].

VII. Trabalhos correlatos

Nesta seção encontram-se alguns trabalhos relacionados ao tema escolhido.

A. Automação Do Provisionamento De Infraestrutura Em Nuvem Para Implantação De Sistemas

O trabalho apresenta uma implementação para a automação do provisionamento de infraestrutura na nuvem voltada para implantação de sistemas, utilizando a ferramenta *terraform* e seu conceito de infraestrutura como código (IaC) para a criação da infraestrutura na nuvem da AWS (Amazon Web Services). Foi desenvolvido e implementado um sistema de gerenciamento de ponto eletrônico disponibilizado por meio de uma API HTTP executado em container utilizando o *terraform* na nuvem da AWS[16]. O trabalho apresentou um resultado positivo conforme o que foi proposto, pois o projeto mostrou-se eficiente no provisionamento, com escalabilidade, seguro e altamente disponível.

Esse trabalho difere do presente, pois o *terraform* será utilizado para criação de uma estrutura em servidores locais

[6]*Application Programming Interface* ou Interface de programação de aplicações

e em nuvem para os programadores utilizarem para desenvolver suas aplicações e não para a execução de aplicações em produção.

B. Gestão De Controle De Mudanças Em Ti: Automatização De Configurações

O trabalho apresenta um estudo, análise e validação, comparando as principais ferramentas *opensource* de gerenciamento de infraestrutura, entre elas o *ansible*, *pupet*, *saltstack* e utilizando o conceito de infraestrutura como código (IaC). Foi observado que em torno de todas essas ferramentas existem uma grande comunidade para auxiliar o usuário, e foi possível obter bons resultados referentes ao que foi proposto no trabalho, mas em algumas situações foi necessária a intervenção manual quando a automação falha. O trabalho também mostra que apesar dessas ferramentas terem como principal aplicação servidores virtuais, é possível utilizá-las em ambientes físicos com um grande nível de automação[17].

Esse trabalho difere do presente, pois é realizada uma comparação entre diversas ferramentas para automatização de configurações tanto em nuvem como local. No presente trabalho será utilizado o *ansible* para a gestão das configurações e não será realizada uma comparação.

C. Infraestrutura Por Código uma Alternativa Para Gerenciamento De Infraestrutura De Redes De Computadores

O trabalho demonstra que, usando a automatização de infraestrutura como código, pode diminuir o tempo gasto para gerenciar uma infraestrutura de redes em relação ao modelo tradicional (manual). Ele analisa vantagens e desvantagens dessa abordagem em relação ao modelo tradicional e traz um exemplo de automatização do processo, através do desenvolvimento de uma aplicação WEB. Também é exposta a importância da disponibilidade do *software*, e como falhas humanas e a demora no processo podem interferir no funcionamento do *software*. O trabalho apresentou resultados positivos conforme o que foi proposto, expondo os pontos positivos dessa automatização, entre eles, a redução no tempo de criação e configuração do ambiente e também a redução de falhas humanas[18].

Esse trabalho difere do presente, pois é realizada a criação e configuração de um ambiente utilizando apenas o *terraform* que foi criado para executar uma aplicação em produção. No presente trabalho será utilizado o *terraform* em conjunto com o *ansible* para a criação do ambiente, e este será criado e configurado para o desenvolvimento de aplicações.

VIII. Metodologia

Devido ao modelo de desenvolvimento de *software* da empresa, aplicações executadas em containers (um container executando a aplicação) e posteriormente utilizando a arquitetura de microsserviços (vários containers para executar a aplicação), as máquinas utilizadas pelos programadores, que são Windows ou Mac, apresentam dificuldades para executar todas as aplicações necessárias para desenvolvimento, principalmente o Docker (*software* para executar os containers) porque em sistemas Windows e Mac, o Docker necessita criar uma máquina virtual Linux para executar os containers, o que consome recursos das máquinas e causa lentidões e travamentos. Embora as novas versões do Windows possuam a possibilidade de executar máquinas linux utilizando a WSL[7], o que ajuda a reduzir o problema de desempenho, ainda é necessário emular o linux no Windows, enquanto em máquinas linux o Docker e outros serviços para executar containers executam de forma nativa.

Foram então utilizadas ferramentas como *ansible*, *terraform* e *bash scripts* para criar e configurar as máquinas virtuais em ambientes locais (servidores físicos na empresa executando *proxmox*) ou em ambientes de nuvem (AWS), utilizando o modelo de serviço IaaS. Para a conexão entre as máquinas virtuais e a máquina dos programadores foi utilizado o protocolo SSH (*secure shell*).

A. Materiais e métodos

Nesta seção foi realizada uma descrição do que são e como funcionam as ferramentas que foram utilizadas neste trabalho, entre elas: Amazon Web Services (AWS), *proxmox*, *terraform*, *ansible*.

1) Amazon Web Services: Também chamada de AWS, ela é uma plataforma de serviços de computação em nuvem oferecidos pela Amazon.com. São oferecidos diversos serviços, entre eles o *Amazon Elastic Compute Cloud* (serviço de computação em nuvem), o Amazon S3 (serviço de armazenamento de arquivos) e o *Amazon Relational Database Service* (serviço de banco de dados), distribuídos entre vários *datacenters* ao redor do mundo, entre eles a região de São Paulo (sa-east-1), e é composta por 3 zonas. É possível encontrar mais informações sobre a AWS e seus serviços na documentação e no site oficial[8].

2) Proxmox: O *proxmox* também chamado de Proxmox VE ou PVE (*Proxmox Virtual Environment*) é um *software* de código aberto para virtualização, ele é um *hypervisor* tipo 1 que permite executar diversos sistemas operacionais de diversas arquiteturas, baseado em *Debian* com um *Kernel* modificado que permite a execução de máquinas virtuais utilizando o KVM (*Kernel-based Virtual Machine*) e QEMU (*Quick Emulator*), assim como containers (LXC[9]). O *proxmox* também possuí uma interface Web para facilitar o gerenciamento do sistema e uma REST API para monitoramento e gerenciamento das VMs e containers. É

[7]Também chamado de Windows Subsystem For Linux, ela é um recurso opcional disponível no Windows que permite executar binários e scripts em Linux diretamente no Windows, traduzindo as instruções enviadas para o sistema para uma instrução válida para o kernel do Windows

[8]https://aws.amazon.com/pt/

[9]LXC ou Linux Containers é um método de virtualização ao nível de SO para executar sistemas Linux isolados do *host* usando o mesmo *Kernel*

possível encontrar mais informações sobre o *proxmox* na documentação e site oficial[10].

3) Terraform: O *terraform* é uma ferramenta de orquestração de código aberto utilizada para criar e configurar recursos em nuvem e em outros ambientes, seguindo o modelo de infraestrutura como código. Ele utiliza *providers* para interagir com diversas outras plataformas e serviços que possuem uma API, entre elas a AWS e o Proxmox. Esses *providers* podem ser oficiais (criados pela HashiCorp[11]) ou não-oficiais, criados pela comunidade, e todos eles estão disponíveis no Terraform Registry[12]. Com isso o *terraform* permite gerenciar a infraestrutura de forma centralizada, permite monitorar alterações, permite automatizar alterações e permite padronizar configurações.

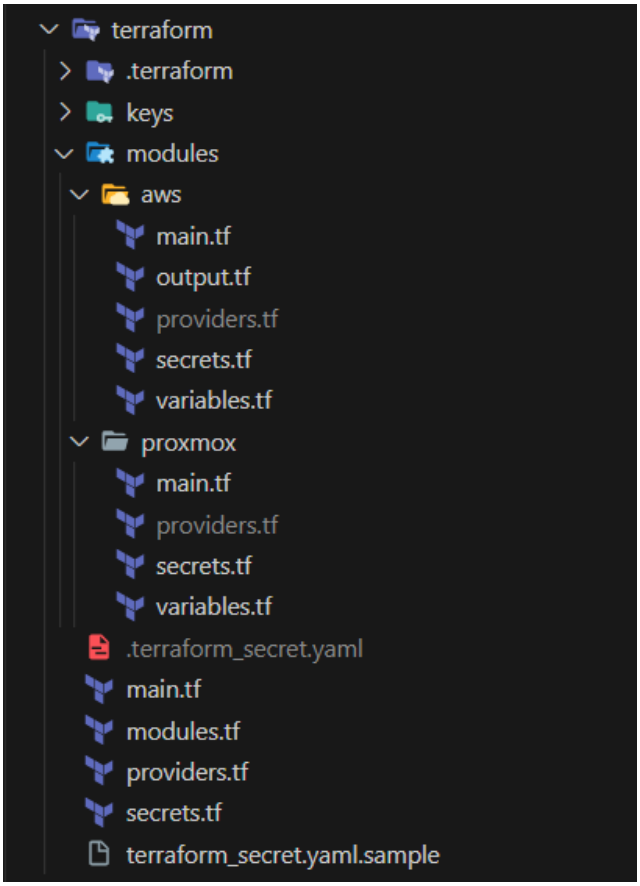

Figura 5. Estrutura de arquivos Terraform. Fonte: Autor.

Conforme a Figura 5, o *terraform* possuí uma pasta principal, a pasta *modules*, que é utilizada para armazenar os módulos ou grupos de recursos que o *terraform* irá gerenciar e uma pasta secundária contendo quaisquer outros arquivos necessários para gerenciar recursos.

Dentro de cada módulo, o *terraform* possuí os mesmo arquivos que na pasta raiz com a adição de mais um arquivo. O arquivo main.tf contém as configurações dos recursos que serão gerenciados, o arquivo secrects.tf inclui o mapeamento de chaves e valores secretos que serão utilizados, o arquivo providers.tf abrange as configurações dos *plugins* ou *providers* que serão utilizados naquele módulo e o arquivo variables.tf engloba todas as variáveis referentes ao gerenciamento dos recursos do módulo.

Na pasta raiz, o arquivo main.tf é usado para armazenar configurações do *terraform* e de *providers*, o arquivo modules.tf contém as configurações de todas as pastas dentro da pasta modules, o arquivo providers.tf abrange as configurações dos *plugins* ou *providers* que serão no *terraform* e o arquivo secrects.tf inclui o mapeamento de chaves e valores secretos que serão utilizados. É possível encontrar mais informações sobre o *terraform* na documentação e site oficial[13]

4) Ansible: O *Ansible* é uma ferramenta de orquestração utilizada para configurar e gerenciar diversas máquinas, sejam elas virtuais ou físicas. É uma ferramenta de código aberto para gerenciar, automatizar e configurar servidores (Unix e Windows) e implementar aplicativos a partir de um sistema central (normalmente Unix). Ele utiliza uma linguagem declarativa chamada de YAML e não possuí um agente para ser instalado nas máquinas remotas, ao invés, ele se conecta à máquina temporariamente através do protocolo SSH (*secure shell*) ou WRM (*Windows Remote Management*) para executar as tarefas e o único pré-requisito para executar é ter o python instalado.

[10]https://www.proxmox.com/en/

[11]A HashiCorp é uma empresa de desenvolvimento de *software* responsável pelo desenvolvimento do *terraform*

[12]O Terraform Registry contém uma lista de todos os *plugins* disponíveis para o *terraform*, com a documentação deles

[13]https://www.terraform.io/

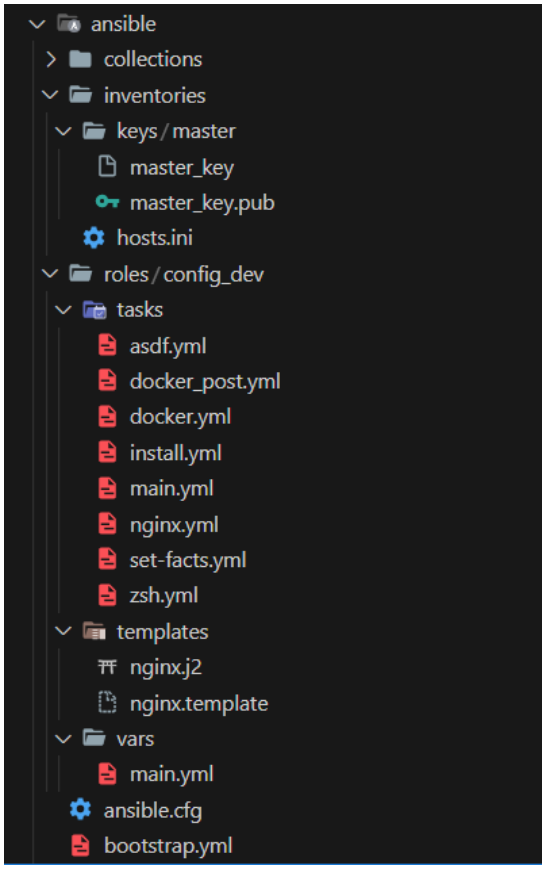

Figura 6. Estrutura de arquivos Ansible. Fonte: Autor.

Conforme a Figura 6 o *ansible* possuí duas pastas principais, a pasta *roles* que é usada para armazenar os conjuntos de tarefas que serão executadas pelo *ansible* na máquina remota e a pasta *inventories* ou *inventory* usada para armazenar os arquivos que contém os *hosts* (máquinas remotas) e qualquer outro arquivo comum aos *roles* ou arquivo geral. Com as duas pastas principais, o *ansible* também inclui alguns arquivos de configuração, o arquivo ansible.cfg engloba configurações do *ansible*, o arquivo bootstrap.yml que abrange as declarações de *roles* e o arquivo requirements.yml que inclui os *roles* que devem ser baixados ou importados (oficiais ou da comunidade).

A pasta *defaults* é usada para armazenar os valores padrão de variáveis utilizadas no *role*, isso permite que os valores padrão sejam gerenciados de forma centralizada, pois eles podem variar de acordo com políticas ou usuários. A pasta *files* é usada para armazenar arquivos extra necessários para executar o *role*, esses arquivos geralmente são estáticos sendo copiados para a máquina remota. A pasta *handlers* é usada para armazenar comandos para gerenciar serviços e aplicar configurações alteradas por outras tarefas.

A pasta *tasks* é a pasta principal do *role* e é onde ficam armazenadas todas as tarefas executadas pelo *role*. A pasta *templates* é usada para armazenar arquivos que servem de modelo para criar arquivos de configuração na máquina remota e são no formato *Jinja2*[14]. A pasta *vars* é usada para armazenar os valores de variáveis usados nas tarefas, e é normalmente utilizado para variáveis permanentes que não são alteradas entre ambientes. É possível encontrar mais informações sobre o *ansible* na documentação e site oficial[15].

B. Fluxo de Desenvolvimento

Nesta seção será explicada as etapas para desenvolvimento e criação dos arquivos e configurações necessárias para execução dos ambientes com as máquinas virtuais, conforme a Figura 7.

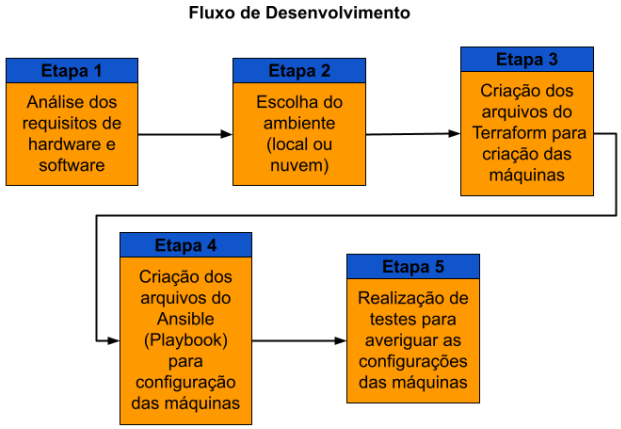

Figura 7. Fluxo de desenvolvimento. Fonte: Autor.

1) Etapa 1: Na primeira etapa foi realizada uma análise junto ao desenvolvedor para verificar quais programas foram necessários para desenvolvimento e também foi analisado quais foram os recursos de *hardware* (processador, memória e armazenamento) necessários para o desenvolvimento.

Após uma análise na empresa foi constatado que é necessário ter os seguintes aplicativos instalados: Node.JS, npm (Node Package Manager), asdf, Docker, git, yarn e zsh. E os seguintes requisitos de *hardware*: processador com 4 ou mais núcleos, 8GB de memória RAM ou mais, 75GB de armazenamento ou mais e rede de 100mbps.

2) Etapa 2: Na segunda etapa foi realizada a escolha do ambiente para criação das máquinas virtuais. Para isso foi realizada uma análise do cenário da empresa para verificar qual das opções (local ou em nuvem) se enquadra melhor para a empresa, considerando os custos e possível disponibilidade de servidores físicos na empresa. Para este trabalho foi realizada a criação em ambos os ambientes a fim de

[14] *Jinja2* é um mecanismo de template da web para a linguagem de programação Python, semelhante ao Django

[15] https://www.ansible.com/

comparação, mas para a situação na empresa foi escolhido apenas um dos ambientes.

Para a escolha do ambiente foi considerado o fato da empresa já possuir os servidores e a estrutura necessária para operá-los, devido a isso foi escolhido o ambiente local para implementação, e como não haverá o alto custo inicial, foi decidido realizar o upgrade dos servidores com a implementação.

3) Etapa 3: Na terceira etapa foram criados os arquivos do *Terraform* para criação das máquinas virtuais, seguindo os requisitos de *hardware* definidos na etapa 1 e a documentação oficial do *Terraform*.

4) Etapa 4: Na quarta etapa foram criados os arquivos do *Ansible* (*Playbook*) para configuração e instalação de *software* nas máquinas virtuais, seguindo os requisitos de *software* definidos na etapa 1 e a documentação oficial do *Ansible*.

5) Etapa 5: Na quinta etapa foi realizada a criação das máquinas virtuais com os arquivos criados na etapa 3 e a configuração delas com os arquivos criados na etapa 4 e foram posteriormente realizados testes para validar as configurações das máquinas virtuais conforme a análise realizada na etapa 1.

C. Produção

Seguindo o fluxo de desenvolvimento definido na subseção anterior, foi realizada uma análise dos requisitos de *hardware* e *software* necessários para desenvolvimento e também uma análise para determinar qual será o ambiente utilizado (local ou nuvem). Depois, com base nos dados coletados na análise anterior, foi realizada a criação dos arquivos do *Terraform* e *Ansible* para criar e configurar as máquinas virtuais no ambiente escolhido. Porem, para que as comparações entre os ambientes pudessem ser realizadas, as máquinas foram criadas em servidores locais e em nuvem, mas em um caso normal apenas um dos ambientes seria escolhido. Posteriormente foram realizados os testes para verificar o funcionamento das máquinas e validação das configurações. Conforme a Figura 8 pode-se observar as especificações das 3 máquinas utilizadas no trabalho, máquina física (do programador), máquina virtual remota (AWS) e máquina virtual local (*Proxmox*).

Especificações			
	Máquina Física	Máquina Virtual Remota	Máquina Virtual Local
Sistema Operacional	Windows 11 Pro (22H2) (Bluid 22621.2070)	Ubuntu 22.04.3 (6.2.0-1009-aws)	Ubuntu 22.04.3 (5.15.0-39-generic)
Qantidade de Memória	32GB DDR4	16GB DDR4	12GB DDR4
Número de Núcleos	4/8 (i5-9300H)	4/4 (E5-2686v4)	6/6 (E5-2670v3)
Quantidade Disco	512GB NVME	75GB Virtual (SSD)	75GB Virtual (NVME)

Figura 8. Especificações das Máquinas. Fonte: Autor.

D. Testes e discussão dos resultados

Os testes foram realizados em duas etapas. Foram primeiramente realizados testes para determinar o desempenho bruto das máquinas, tanto as virtuais como as físicas e depois foi realizado testes para determinar o desempenho específico das máquinas para desenvolvimento. Por fim, esses dados foram agrupados em uma tabela e foram gerados gráficos a partir desses dados para comparar o desempenho das máquinas e avaliar a viabilidade do projeto.

1) Desempenho bruto: Para medir o desempenho bruto das máquinas foram utilizadas as ferramentas *geekbench*[16] e *passmark*[17] para medir o desempenho do processador e da memória. Analisando os resultados obtidos nos testes (Figura 9), podemos observar que ambas as máquinas virtuais possuem um desempenho similar (no teste de memória) devido ao seu *hardware* (componentes do servidor físico) e que ambas possuem desempenho inferior à máquina do programador. Porém, como pode ser observado na parte de consumo de memória, grande parte desse desempenho superior fica dedicado a executar as aplicações para desenvolvimento e assim prejudica o uso geral da máquina e gera uma experiência ruim para o programador.

[16]O Geekbench é uma ferramenta que suporta diversas plataformas e é utilizado para realizar testes de desempenho em CPUs e GPUs em computadores, laptops, notebooks, tablets e telefones

[17]O Passmark é uma ferramenta que suporta diversas plataformas e é utilizado para realizar testes de desempenho em CPUs e memórias, utilizando diversos testes para analisar de forma objetiva o desempenho delas

Desempenho Bruto			
	Máquina Física	Máquina Virtual Remota	Máquina Virtual Local
CPU (GeekBench)	Single-core: 1436 Multi-core: 4534	Single-core: 895 Multi-core: 2933	Single-core: 858 Multi-core: 3583
CPU (PassMark)	8472	4761	6677
Memória (Passmark)	2810	1977	1911
Consumo de Memória	Mim Mem: 6.8GB Max Mem: 22.7GB Delta: 15.9GB	Mim Mem: 0.3GB Max Mem: 10GB Delta: 9.7GB	Mim Mem: 0.3GB Max Mem: 9.3GB Delta: 9GB

Figura 9. Desempenho Bruto. Fonte: Autor.

2) Desempenho específico: Para medir o desempenho específico das máquinas foi realizada uma simulação de programação em cada máquina e analisado o tempo para execução dos containers docker, tempo para criação de imagem de um container docker, tempo para criação de imagem de um container com a aplicação Shinobi[18], tempo para realizar build do *frontend* de uma aplicação e uso de recursos para executar essas tarefas. Os resultados desses testes (Figura 10) nos mostram que as 3 máquinas possuem um desempenho similar em alguns testes e desempenho divergentes em outros testes. Nos testes que envolvem a execução do *frontend* as 3 máquinas apresentam um desempenho muito similar, porém a máquina física, possuí uma desvantagem devido ao barulho gerado durante essa execução. Já nos testes que envolvem o docker, podemos observar que as máquinas virtuais possuem um desempenho muito superior e o uso de recursos é inferior à máquina física devido a forma como o docker é executado em Windows e Linux.

Desempenho Específico			
	Máquina Física	Máquina Virtual Remota	Máquina Virtual Local
Tempo de Compilação (Frontend)	211.10s	265.60s	206.30s
Tempo Execução em desenvolvimento (Frontend)	41.12s	43.33s	40.57s
Tempo Execução Docker	197.45s	12.14s	33.23s
Tempo Build Docker	56.43s	208.73s	171.82s
Tempo Build Docker (Shinobi)	293.19s	265.59	153.10s

Figura 10. Desempenho Específico. Fonte: Autor.

3) Discussão dos resultados: Nesta seção foi realizada a análise dos resultados obtidos nos testes e foi realizada a comparação dos dados. Esperava-se que o resultado obtido mostrasse que as máquinas virtuais (locais e na nuvem) possuem o desempenho igual ou superior à máquina do programador e que devido à utilização das máquinas virtuais fosse observada uma redução na utilização de recursos na máquina do programador, permitindo que essas máquinas sejam mais fracas ou que esses recursos sejam utilizados para execução de outras tarefas na máquina física.

Como observado nas subseções anteriores, os testes de desempenho bruto (Figura 9) e os testes de desempenho específico (Figura 10) nos mostram que as máquinas virtuais (locais e remotas) possuem um desempenho similar entre elas e que elas possuem um desempenho geralmente similar à máquina do programador (física) quando comparados cenários de desenvolvimento, porém um desempenho inferior quanto ao desempenho bruto, que é um resultado esperado, visto que a máquina do programador possuí *hardware* muito superior (conforme Figura 8). Isso deixa evidente que o sistema operacional também tem um papel importante no desenvolvimento de aplicações, principalmente no uso de memória, visto que a maioria das máquinas dos programadores possuem 16GB de memória e nos testes foi observado um uso muito mais elevado do que o disponível nessas máquinas usando o Windows, porém quando utilizamos uma máquina virtual executando Linux para realizar o desenvolvimento, o uso de recursos da máquina física é reduzido.

Para a análise dos custos foi levado em conta um ambiente executando doze máquinas virtuais vinte quatro horas diárias, sete dias por semana no servidor local, que atualmente é a limitação dele antes que seja necessário realizar melhorias, e um ambiente executando doze máquinas virtuais por oito horas diárias, cinco dias por semana na AWS. Conforme a Figura 11 pode-se observar que as máquinas virtuais em ambos os ambientes possuem um desempenho similar (menos de vinte por cento de diferença) entre elas, porém quando é analisado os custos de cada ambiente observa-se uma grande diferença entre eles. O ambiente na AWS possuí um custo inicial zero, pois não é necessário adquirir nenhum equipamento e o ambiente local possuí um custo inicial na casa dos doze mil reais, pois é necessário adquirir o servidor e as peças para melhorá-lo. Já o custo mensal da AWS fica na casa dos quatro mil reais e o custo do ambiente local fica na casa dos trezentos a quinhentos reais, dependendo de custo com manutenção e energia.

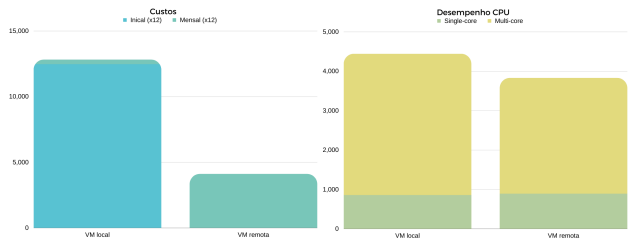

Figura 11. Desempenho VS Custo. Fonte: Autor.

[18]O Shinobi ou ShinobiCCTV é uma aplicação *opensource*, escrita em Node.JS que funciona como um servidor de vigilância por vídeo ou NVR para gravar e monitorar câmeras de segurança. Mais informações disponíveis em: https://shinobi.video/

IX. Conclusão

Neste trabalho foi apresentado uma necessidade de melhorar a forma como as aplicações são desenvolvidas em uma empresa, buscando melhorar o desempenho e reduzir a carga nas máquinas de programadores. Principalmente no desenvolvimento de microsserviços que utilizem o Docker como ambiente de desenvolvimento, devido ao alto número de recursos computacionais (processador e memória) necessários para executar o docker em máquinas com Windows e MacOS.

Como uma solução para esse problema, foi apresentada uma forma de automatizar a criação de máquinas virtuais utilizando ferramentas de automação, como *terraform* e *ansible* em ambientes locais e em nuvem, para que essas possam ser utilizadas para o desenvolvimento de *software* e reduzir a utilização na máquina dos colaboradores. O *terraform*, um automatizador de configurações para estruturas, foi utilizado para criar as máquinas virtuais no ambiente escolhido e o *ansible*, um automatizador de configurações de máquinas, foi utilizado para configurar as máquinas virtuais nos ambientes.

O trabalho apresentou resultados positivos conforme o que era esperado, pois mostrou que a utilização das máquinas virtuais para desenvolvimento apresentam vantagens ao programador, reduzindo a carga na máquina dele, trazendo um aumento na segurança ao concentrar todos os dados nos servidores e diminuindo o tempo para execução das aplicações para desenvolvimento. O trabalho também mostrou que entre os ambientes analisados (AWS e local), existe uma vantagem em utilizar servidores físicos a longo prazo devido ao menor custo mensal. Posteriormente, os resultados deste trabalho podem ser utilizados na tomada de decisão do fluxo de desenvolvimento de uma empresa.

Referências

[1] Redação Cronapp. "Desenvolvimento de software: Tendências do momento e do futuro". Em: (2020). Disponível em <https://blog.cronapp.io/desenvolvimento-de-software-as-tendencias-do-momento-e-do-futuro/>.

[2] RedHat. "O que é virtualização?" Em: (2018). Disponível em <https://www.redhat.com/pt-br/topics/virtualization/what-is-virtualization>.

[3] IBM. "What is virtualization?" Em: (2022). Disponível em <https://www.ibm.com/topics/virtualization>.

[4] IBM. "What is virtualization?" Em: (2022). Disponível em <https://www.ibm.com/topics/hypervisors>.

[5] IBM. "What are virtual machines (VMs)?" Em: (2022). Disponível em <https://www.ibm.com/topics/virtual-machines>.

[6] RedHat. "O que é uma máquina virtual (VM)?" Em: (2022). Disponível em <https://www.redhat.com/pt-br/topics/virtualization/what-is-a-virtual-machine>.

[7] IBM. "What are containers?" Em: (2022). Disponível em <https://www.ibm.com/topics/containers>.

[8] RedHat. "What is orchestration?" Em: (2019). Disponível em <https://www.redhat.com/en/topics/automation/what-is-orchestration>.

[9] RedHat. "O que é o Kubernetes?" Em: (2023). Disponível em <https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes>.

[10] IBM. "What is cloud computing?" Em: (2023). Disponível em <https://www.ibm.com/topics/cloud-computing>.

[11] "What is cloud computing?" Em: (2023). Disponível em <https://azure.microsoft.com/en-in/resources/cloud-computing-dictionary/what-is-cloud-computing>.

[12] Google Cloud. "O que é IaaS?" Em: (2023). Disponível em <https://cloud.google.com/learn/what-is-iaas?hl=pt-br>.

[13] Microsoft. "What is PaaS?" Em: (2023). Disponível em <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-paas/>.

[14] RedHat. "O que é SaaS?" Em: (2023). Disponível em <https://www.redhat.com/pt-br/topics/cloud-computing/what-is-saas>.

[15] IBM. "What is FaaS?" Em: (2023). Disponível em <https://www.ibm.com/topics/faas>.

[16] Alex Treviso. "Automação do provisionamento de infraestrutura em nuvem para implantação de sistemas". Em: (2022).

[17] TALLES TAYSON SALDANHA DE FARIAS. "CURSO DE ENGENHARIA DE TELECOMUNICAÇÕES". Em: ().

[18] RENÃ FREITAS DA CRUZ, Fagner Coin Pereira e João Padilha Moreira. "INFRAESTRUTURA POR CÓDIGO". Em: *PROJETOS E RELATÓRIOS DE ESTÁGIOS* 2.1 (2020).