Pós-Graduação em Ciência da Computação

Marcus Rafael Xavier Laurentino

Migração de Máquinas Virtuais em Ambientes Multi-Nuvem

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
https://www3.cin.ufpe.br/br/pos-graduacao

Recife
2020

Marcus Rafael Xavier Laurentino

Migração de Máquinas Virtuais em Ambientes Multi-Nuvem

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de Concentração: Sistemas Distribuídos
Orientador: Carlos André Guimarães Ferraz
Co-orientador: Ioram Schechtman Sette

Recife
2020

Catalogação na fonte
Bibliotecária Arabelly Ascoli CRB4-2068

L383m Laurentino, Marcus Rafael Xavier
Migração de máquinas virtuais em ambientes multi-nuvem / Marcus Rafael Xavier Laurentino. – 2020.
80 f.: il. fig., tab.

Orientador: Carlos André Guimarães Ferraz
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação. Recife, 2020.
Inclui referências.

1. Computação na nuvem. 2. Ambientes multi-nuvem. 3. Infraestrutura como serviço. 4. Migração de máquina virtual. I. Ferraz, Carlos André Guimarães (orientador). II. Título.

004.6 CDD (22. ed.) UFPE-CCEN 2020-93

Marcus Rafael Xavier Laurentino

Migração de Máquinas Virtuais em Ambientes Multi-Nuvem

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 14/02/2020.

Orientador: Carlos André Guimarães Ferraz

BANCA EXAMINADORA

Prof. Dr. Vinicius Cardoso Garcia
Centro de Informática / UFPE

Prof. Dr. Andrey Elísio Monteiro Brito
Centro de Engenharia Elétrica e Informática / UFCG

Prof. Dr. Ioram Schechtman Sette
CESAR School / CESAR

Dedico este trabalho à minha família e minha namorada, que foram essenciais diante das dificuldades que passei durante este percurso.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus por minha saúde, e por cuidar de mim nestas idas e vindas entre João Pessoa e Recife.

Agradeço à minha mãe, que sempre me motivou e me mostrou que a única maneira para o sucesso é através da educação, e agradeço à toda minha família, pelo carinho e apoio que me fizeram continuar nestas idas e vindas.

Agradeço à minha namorada por toda a motivação, paciência e confiança na distância que precisamos superar enquanto eu trabalhava na minha pesquisa.

Aos meus amigos de infância, obrigado por entenderem minha ausência nas reuniões aos finais de semana, por eu estar trabalhando em minhas atividades do mestrado.

Gostaria também de agradecer ao meu orientador, Carlinhos, também conhecido por Carlos Ferraz, e ao meu co-orientador Ioram Sette, por todas as excelentes reuniões, todas as conversas e dúvidas tiradas, tanto sobre a pesquisa, quanto à carreira e à vida.

Agradeço ao professor Andrey Brito, por todas as oportunidades que tive quando estava no Laboratório de Sistemas Distribuídos (LSD) da UFCG, que me fizeram ter total interesse por sistemas distribuídos e computação na nuvem, e que certamente me levaram à pós-graduação.

Ao professor Vinicius Garcia, muito obrigado por aceitar, juntamente com Andrey, examinar este trabalho de dissertação, oferecendo suas sugestões de melhorias.

Agradeço também aos meus amigos de apartamento (o qual apelidamos carinhosamente de cativeiro), Alex e Ozonias, por terem deixado a caminhada no mestrado muito mais agradável durante estes anos.

Aos meus colegas de laboratório, pelos cafés e principalmente pelas dicas de pesquisa, vida e investimentos, meu muito obrigado.

Agradeço aos meus professores e aos meus amigos da graduação em computação da Universidade Federal da Paraíba.

Aos colegas do projeto do Observatório de Defesa Cibernética do Exército Brasileiro, obrigado pelas valiosas e descontraídas reuniões.

Agradeço também aos funcionários do CIn-UFPE, por todo o suporte que precisei ao longo de minha pesquisa.

Por fim, agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e também ao Ministério da Defesa pelo apoio financeiro.

Computadores fazem arte
Artistas fazem dinheiro
Computadores avançam
Artistas pegam carona
Cientistas criam o novo
Artistas levam a fama
(Chico Science & Nação Zumbi, 1994)

RESUMO

O *vendor lock-in* é uma situação em que os clientes se tornam dependentes dos serviços e recursos de um provedor de nuvem. Como solução para esse problema, uma estratégia é que esses clientes distribuam seus dados e aplicações entre nuvens diferentes, no entanto neste caso, estes precisarão lidar com a heterogeneidade das nuvens, usando as diferentes ferramentas fornecidas por cada um desses provedores, para gerenciar, muitas vezes manualmente, toda essa infraestrutura distribuída. Outra estratégia é usar um serviço multi-nuvem, capaz de lidar com toda a heterogeneidade entre nuvens de maneira transparente para o cliente, fornecendo uma interface unificada capaz de gerenciar as diferentes nuvens simultaneamente, como no caso de migrações de máquinas virtuais entre nuvens. Este trabalho de dissertação propõe uma arquitetura chamada Kumo para simplificar a migração de máquinas virtuais entre várias nuvens, que quando implementada, simplifica o processo de migração dessas máquinas entre nuvens para o operador em nuvem. Para isso, neste trabalho, os dois componentes projetados na arquitetura são implementados, a interface da linha de comandos e o serviço multi-nuvem de migração de máquinas virtuais. Além disso, foram desenvolvidas integrações com as três nuvens públicas mais usadas atualmente, Amazon Web Services, Microsoft Azure e Google Cloud Platform, o que tornou a migração homogênea e automatizada entre essas nuvens da perspectiva do operador. Para avaliar a solução, foram utilizados os data centers dessas três nuvens nos estados americanos da Virgínia e Califórnia. Com o objetivo de definir um cenário real de uso de Kumo, foram realizadas migrações da costa leste para a costa oeste dos Estados Unidos. Como a migração realizada com a VM em execução (*live*) ainda não é tecnicamente viável, as migrações realizadas neste trabalho foram feitas com a VM desativada (*non-live*). A métrica usada foi o Tempo Total de Migração (TTM), que é o tempo desde que a migração é iniciada na nuvem de origem até a máquina virtual ser criada na nuvem de destino. Essa métrica é um fator importante na decisão de quando e para onde migrar, ou mesmo, se o tempo de migração apresentado pela métrica for muito significativo, recriar todo o ambiente no destino, transferindo os dados e reinstalando as aplicações apropriadas. Como resultado, entre os cenários homogêneos, aqueles em que a nuvem de origem e destino são do mesmo provedor, mas em diferentes data centers, o melhor resultado ocorreu nas migrações entre nuvens da Azure, com um TTM mediana de 45 minutos e 59 segundos, enquanto nos cenários heterogêneos, o melhor resultado ocorreu nas migrações da Google Cloud para a Amazon, com TTM mediana de 45 minutos e 56 segundos.

Palavras-chave: Computação na Nuvem. Ambientes Multi-Nuvem. Infraestrutura como Serviço. Migração de Máquina Virtual.

ABSTRACT

The vendor lock-in is a situation where customers become dependent on the services and resources of a cloud provider. As a solution to this problem, a strategy is for these customers to distribute their data and applications among different clouds, however in this case, these customers will need to deal with the heterogeneity of clouds, using different tools provided by each of these providers, in order to manage, often manually, all of this distributed infrastructure. Another strategy is to use a multi-cloud service, which is able to handle all the heterogeneity among clouds in a transparent way for the client, providing a unified interface capable of managing the different clouds simultaneously, as in the case of machine migrations between clouds. This dissertation work proposes an architecture called Kumo to simplify the migration of virtual machines between multi-clouds, which when implemented, simplifies the process of migrating these machines between clouds for the cloud operator. For this, in this work the two components designed in the architecture are implemented, the command line interface and the multi-cloud virtual machine migration service. Also, integrations were developed with the three most used public clouds nowadays, Amazon Web Services, Microsoft Azure and Google Cloud Platform, which made the migration homogeneous and automated between these clouds from the operator's perspective. To evaluate the solution, the data centers of these three clouds in the American states of Virginia and California were used. With the objective of defining a real scenario of use of Kumo, migrations were performed from the east coast to the west coast of the United States. Because the migration conducted with the VM running (live) is not yet technically feasible, the migrations performed in this work were conducted with the VM turned off (non-live). The metric used was the Total Migration Time (TMT), which is the time since the migration is initiated in the source cloud until the virtual machine is created in the destination cloud. This metric is an important factor in deciding when and where to migrate, or even, if the migration time presented by the metric is very significant, recreate the entire environment at the destination, transferring the data and reinstalling the appropriate applications. As a result, among the homogeneous scenarios, which are those scenarios where the source and destination cloud are from the same provider but in different data centers, the best result occurred in migrations between Azure clouds, with a median TMT of 45 minutes and 59 seconds, while among heterogeneous scenarios, the best result occurred in migrations from Google Cloud to Amazon, with a median TMT of 45 minutes and 56 seconds.

Keywords: Cloud Computing. Multi-Cloud Environments. Infrastructure as a Service. Virtual Machine Migration.

LISTA DE FIGURAS

LISTA DE CÓDIGOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABS	Azure Blob Storage
AC	Azure Compute
AIE	AWS Import/Export
API	Interface de Programação de Aplicações
AVN	Azure Virtual Network
AWS	Amazon Web Services
AZ	Microsoft Azure
CLI	Interface de Linha de Comando
CRUD	Create, Read, Update e Delete
CS	CloudStack
DRBD	Distributed Replicated Block Device
EC2	Elastic Compute Cloud
FaaS	Function as a Service
GB	Gigabyte
GCB	Google Cloud Build
GCE	Google Compute Engine
GCP	Google Cloud Platform
GCS	Google Cloud Storage
GiB	Gibibyte
GUI	Interface Gráfica do Usuário
HQEMU	Hybrid Quick EMUlator
HTTPS	Hyper Text Transfer Protocol Secure
IaaS	Infraestrutura como Serviço
IAM	Identity and Access Management
JSON	JavaScript Object Notation
KVM	Kernel-based Virtual Machine
MB	Megabyte
MiB	Mebibyte
NFS	Network File System
OS	OpenStack
OSC	OpenStack Client
PIP	PIP Installs Packages
QEMU	Quick EMUlator
QoS	Qualidade de Serviço
RAW	RAW Format Hard Drive

S3	Simple Storage Service
SDK	Software Development Kit
SO	Sistema Operacional
TIC	Tecnologia da Informação e Comunicação
TTM	Tempo Total de Migração
VHD	Virtual Hard Disk
VM	Máquina Virtual
YAML	YAML Ain't Markup Language

SUMÁRIO

1 INTRODUÇÃO

A pesquisa anual realizada por FLEXERA (2019) apontou que 94% dos profissionais de Tecnologia da Informação e Comunicação (TIC) entrevistados utilizam serviços de computação na nuvem em suas empresas. 84% das empresas destes profissionais utilizam alguma estratégia de uso de múltiplas nuvens, sendo que 58% adotam a estratégia de nuvem híbrida e 17% adotam a estratégia de utilização de múltiplas nuvens públicas para implementar ambientes multi-nuvens. Entre os desafios na adoção multi-nuvem, o gerenciamento é preocupação para 76% dos entrevistados, e para 73% um outro desafio é a migração entre nuvens de diferentes provedores.

A estratégia multi-nuvem é uma solução para o clássico problema de *vendor lock-in*, que é a situação onde o cliente se torna dependente de serviços e recursos fornecidos por provedores de nuvens (TOOSI; CALHEIROS; BUYYA, 2014). Por outro lado, a heterogeneidade das nuvens se torna um problema por dificultar o uso e o gerenciamento das mesmas. Esta heterogeneidade existe porque cada nuvem adota interfaces proprietárias para seus serviços, sem haver uma padronização.

Para mitigar o problema da heterogeneidade das nuvens, Toosi, Calheiros e Buyya (2014) propõem uma solução capaz de alcançar a interoperabilidade através de um **serviço multi-nuvem**. Tal serviço trata a heterogeneidade de cada provedor, implementando uma camada de adaptação para as diversas Interfaces de Programação de Aplicações (APIs) de cada uma das nuvens, oferecendo uma interface única também através de uma API ou meta-API, para os clientes se comunicarem e utilizarem recursos das várias nuvens (CHAUHAN et al., 2019; ELHABBASH et al., 2019).

No contexto de nuvens de Infraestrutura como Serviço (IaaS), a migração de Máquinas Virtuais (VMs) entre nuvens homogêneas e heterogêneas é um desafio, pois o processo é realizado de maneira parcialmente manual ou automatizada mas não integrada, exigindo assim a utilização de ferramentas disponibilizadas pelos provedores, cada uma com comandos e parâmetros diversos. Diante da dificuldade envolvida na migração de VMs atualmente, percebe-se a necessidade de mecanismos que realizem todo o processo de maneira unificada, tratando a heterogeneidade e ainda de maneira integralmente automatizada.

Para isto, este trabalho apresenta uma solução chamada Kumo, que é um serviço para migração de VMs entre multi-nuvens, que torna o processo inteiramente automatizado através de uma ferramenta que trata a heterogeneidade entre as nuvens para facilitar a migração. Entre as contribuições deste trabalho, estão o serviço multi-nuvem capaz de realizar migrações de VMs entre nuvens, reduzindo o esforço necessário pelo operador de nuvem; a identificação dos passos necessários para a migração de VMs entre nuvens; bem como a análise dos tempos de execução do passo a passo de migrações, tanto homogêneas

quanto heterogêneas. A pesquisa levou em consideração as nuvens Amazon Web Services (AWS), Microsoft Azure (AZ) e Google Cloud Platform (GCP), que são as nuvens públicas comerciais mais utilizadas na atualidade (FLEXERA, 2019).

1.1 PROBLEMA

Diante da diversidade de provedores de nuvem que fornecem serviços de IaaS, realizar migração de recursos como VMs entre nuvens é um desafio (ZHANG et al., 2017; NARANTUYA; ZANG; LIM, 2018; MANSOUR; BOUCHACHIA; COOPER, 2017; YADAV; KRISHNA, 2019; KARGATZIS; SOTIRIADIS; PETRAKIS, 2017). Isto acontece devido a falta de automatização do processo de migração, que torna a migração lenta e também por causa da heterogeneidade das diversas tecnologias, ferramentas e formatos de dados que são adotados e disponibilizados por cada provedor de nuvem.

Clientes, por diversos motivos, podem ter contas em diversos provedores de nuvem, seja em provedores de nuvens diferentes ou ainda várias contas em nuvens de um mesmo provedor. Deste modo, os operadores de nuvem, que são os responsáveis por gerenciar a infraestrutura virtual das nuvens, têm a necessidade de utilizar diversas credenciais e diversas ferramentas que são disponibilizadas pelos provedores para gerenciar suas nuvens.

Uma operação muito comum entre infraestruturas virtuais é a de migração de VMs. No contexto de multi-nuvem, para realizar esta operação o operador de nuvem precisa utilizar as credenciais para as contas da nuvem de origem e de destino, e também as ferramentas de gerenciamento disponibilizadas por cada uma das nuvens de origem e de destino.

O problema do modo manual de migração é que o operador precisa lidar manualmente com a heterogeneidade na autenticação e no uso de cada ferramenta para cada uma das nuvens envolvidas na migração. Diante da dificuldade no gerenciamento manual da infraestrutura de nuvem, ferramentas de automatização surgiram para facilitar a execução destas tarefas.

Entretanto, como as ferramentas de automatização são de propósito geral, ou seja, não são focadas na migração de VMs (ex. Ansible), acontecem situações onde tais ferramentas não são capazes de realizar alguns passos necessários para migrar uma VM da nuvem de origem para a de destino.

Por estes motivos, ter um mecanismo capaz de realizar a migração de VMs entre nuvens de maneira homogênea e completamente automatizada é uma necessidade atual. Acredita-se que este mecanismo, quando implementado, é capaz de reduzir a complexidade encontrada pelos operadores no processo de migração. Logo, o problema que este trabalho trata foi estabelecido da seguinte forma:

> **Como tornar homogêneo e automatizado o processo de migração de máquinas virtuais em ambientes multi-nuvem?**

Assim, este trabalho tem como objetivo mitigar a complexidade envolvida no processo de migração de VMs entre contas em múltiplas nuvens. Para isto, neste trabalho é proposta, implementada e avaliado o desempenho de uma solução que automatiza todo este processo de migração, tendo como propósito fornecer para os operadores de multi-nuvem uma experiência homogênea e automatizada de migração, de modo que a solução trate a complexidade, realizando migrações com o mínimo de esforço possível para o operador.

Conforme a arquitetura apresentada na seção 3.1, a solução proposta consiste em dois componentes principais, uma Interface de Linha de Comando (CLI) (seção 3.1.1), que permite que sejam configuradas e iniciadas as migrações; e o Serviço de Migração (seção 3.1.2), que é um serviço web responsável por implementar as integrações com cada uma das nuvem e realizar as migrações de maneira automatizada, tratando as especificidades de cada nuvem. Com estes dois componentes configurados, o operador utiliza a CLI para solicitar ao Serviço de Migração que execute as migrações de VMs, de uma conta em uma nuvem de origem para outra de destino.

1.2 OBJETIVOS

O objetivo geral deste trabalho é **permitir que máquinas virtuais possam ser migradas de maneira automatizada e homogênea para quaisquer nuvens envolvidas na migração**. Logo, para isto foram definidos três objetivos específicos que são listados a seguir:

1. **Tornar homogêneo o processo de migração de máquinas virtuais em ambientes multi-nuvem**
   Com o objetivo de tornar o processo de migração, que é essencialmente heterogêneo, em homogêneo, é necessário garantir que embora as nuvens sejam heterogêneas, cada passo da migração, quando necessário, sejam executado de maneira equivalente em cada nuvem, respeitando a forma com que a nuvem implementa este passo. A solução deve considerar os seguintes requisitos:

   a) **Exigir esforço mínimo do operador de nuvem**
      A solução deve ser única para quaisquer nuvens envolvidas na migração, de maneira que o operador de nuvem apenas informe qual a VM que deve ser migrada, qual é a nuvem de origem e qual a nuvem de destino para a VM, sendo isto o suficiente para realizar uma migração.

   b) **A migração deve ser agnóstica quanto às nuvens**
      Do ponto de vista do operador de nuvem, detalhes de como a solução implementa as integrações com cada um das nuvens devem ser transparentes. O operador deve apenas se preocupar com a VM e as nuvens envolvidas na migração,

não sendo necessário conhecer como cada passo da migração é internamente realizado.

c) **Deve permitir integração com novas nuvens**
A solução deve fornecer um modelo e mecanismos para que um operador a integre com outras novas tecnologias de nuvem. Com isso, é esperado que este novo mecanismo seja capaz de importar VMs de quaisquer nuvens já integradas para si, e consequentemente exportar de si para qualquer outra nuvem entre as previamente integradas à solução.

2. **Automatizar o processo de migração de máquinas virtuais em ambientes multi-nuvem**
Buscando facilitar o processo de migração, o segundo objetivo específico da solução é quanto à automatização de todo o processo de migração de VMs. Este objetivo específico torna a solução menos propensa a erros, pois previne que o operador de nuvem execute algum passo da migração ou de maneira errada, ou na ordem ou momento errados, podendo assim comprometer a migração como um todo.

3. **Avaliar a execução do processo de migração de máquinas virtuais em ambientes multi-nuvem**
Por fim, com o propósito de mensurar o desempenho do processo de migração de modo homogêneo e automatizado, faz-se necessário avaliar o fator desempenho para as migrações de VMs realizadas usando a solução proposta. Para isto, foram definidos 3 cenários de migração homogêneos e 6 cenários de migração heterogêneas, envolvendo as nuvens AWS, AZ e GCP, para em seguida realizar análises dos resultados obtidos para a métrica de Tempo Total de Migração (TTM) sugerida por Zhang et al. (2018), e assim identificar os melhores e os piores cenários, que serão determinantes para decidir se vale a pena migrar para outra nuvem, e se valer, para qual das nuvens é mais rápido migrar a partir de uma nuvem de origem.

1.3 DESAFIOS DA PESQUISA

Nesta seção são discutidos os desafios técnicos quanto à heterogeneidade e também quanto à falta de automatização que precisam ser superados, a fim de permitir que uma proposta de solução seja concebida. Portanto, a seguir são listados os desafios encontrados na pesquisa realizada nas nuvens AWS, AZ e GCP, que segundo FLEXERA (2019) são as três nuvens públicas mais utilizadas da atualidade.

Quanto à heterogeneidade, um primeiro desafio é quanto às **múltiplas credenciais**, uma vez que para realizar qualquer passo da migração de VMs, o operador de nuvem precisa utilizar as credenciais de cada uma das nuvens envolvidas no processo de migração. Porém, cada uma das nuvens investigadas utiliza um arquivo de credenciais em formatos

distintos, que por sua vez utilizam atributos diferentes para autenticar um usuário em sua nuvem. Vale lembrar que o operador que gerencia múltiplas nuvens precisará gerenciar portanto múltiplas contas, podendo estas serem contas em um mesmo provedor de nuvem ou em provedores diferentes. Logo, é um desafio gerenciar tantas credenciais de múltiplas nuvens, com informações distintas, de modo a realizar migrações de VMs entre contas destas nuvens.

Outro desafio quanto à heterogeneidade, é a **variedade de ferramentas, seus comandos, parâmetros e valores**. Estas ferramentas, que são geralmente são disponibilizadas pelos provedores de nuvens como CLIs, são bastante diversificadas quanto sintaxe dos comandos e quanto aos parâmetros e valores esperados. Assim, o operador das multinuvens precisa ser habilidoso o suficiente para saber como executar cada comando necessário para realizar migração de maneira equivalente, tanto na nuvem de origem quanto na de destino, sem cometer maiores falhas humanas durante o processo de migração.

Ainda quanto à heterogeneidade, a **compatibilidade de formatos e tamanho dos discos** também é um desafio. Ter esta compatibilidade é um fator determinante para a realização das migrações de VMs entre nuvens, uma vez que se existe esta compatibilidade, será possível realizar a migração do disco sem realizar nenhuma intervenção neste, mas caso contrário, será preciso adaptar o disco tornando-o compatível com o esperado pela nuvem de destino. Como exemplo de intervenções que podem ser necessárias para tornar o disco compatível com a nuvem de destino, estão a realização da conversão de seu formato e o ajuste de seu tamanho.

Como o último desafio no que se diz respeito à heterogeneidade, a **integração com novas nuvens** tem como objetivo permitir a integração de novos provedores à solução, sendo assim necessário definir algum mecanismo de padronização para que novas integrações com outras nuvens sejam simplificadas. Como exemplo, uma forma de implementar isto pode ser através da definição de uma interface padronizada, determinando quais os comportamentos que precisam ser implementados, para que assim outras nuvens sejam capazes de interoperar com as demais nuvens, tonando-se capaz de tanto exportar VMs de si pra outras nuvens, quanto importar VMs de outras nuvens para si.

Quanto à **automatização da migração**, o desafio está na própria automatização. Assim, uma vez solucionados todos os desafios referentes à heterogeneidade, será necessário, de acordo com a proposta feita neste trabalho, automatizar a obtenção das credenciais para acesso às nuvens; fornecer uma ferramenta única que realize todos os passos envolvidos na migração e VMs de maneira automatizada; garantir que seja realizada de maneira automatizada a conversão de formato e ajuste do tamanho do disco da VM, de acordo com os requisitos da nuvem de destino; e também que seja possível automatizar, de maneira agnóstica quanto às nuvens, os passos envolvidos no processo de migração.

1.4 SOLUÇÃO PROPOSTA

A solução proposta neste trabalho, chamada **Kumo**, atende aos desafios apresentados na seção 1.3, sendo assim capaz de oferecer, para os operadores de nuvens, uma experiência de migração de VMs entre multi-nuvens facilitada através da homogeneização e automatização dos passos que compõem o processo de migração. A solução é composta por dois componentes principais, uma ferramenta tipo Interface de Linha de Comando (CLI) e um Serviço de Migração de VMs capaz de realizar migrações de VMs de modo *non-live* entre nuvens, realizando para isso o desligamento da VM antes de migrá-la.

A CLI de Kumo é a ferramenta projetada para solucionar o problema das diversas credenciais dispersas em vários locais, arquivos e formatos, e das várias ferramentas necessárias para realizar a migração de VM. Com a CLI é possível centralizar as múltiplas credenciais de todas as contas existentes nas várias nuvens, em um arquivo único de configuração, onde as credenciais são armazenadas e são nomeadas de modo a facilitar a identificação a qual nuvem aquela credencial pertence.

Também na CLI é tratado o desafio da necessidade da utilização de diversas ferramentas, comandos, parâmetros e valores de cada nuvem para realizar a migração de VMs. A ferramenta permite que, com apenas um comando (`migrate`), o operador consiga solicitar ao Serviço de Migração de Kumo um pedido de migração de uma VM que esteja em qualquer nuvem de origem para qualquer nuvem de destino, bastando apenas informar em um arquivo YAML Ain't Markup Language (YAML) qual o nome da VM, e ainda qual o nome da conta de origem e da conta de destino, nomes estes que foram os usados no arquivo de configuração para identificar as credenciais destas contas.

O outro componente de Kumo é o Serviço de Migração de VMs, que é o responsável por tratar, primeiramente, as incompatibilidades entre os formatos de disco e outras configurações da origem em relação ao esperado pela nuvem de destino. Nesta situação, automaticamente, o Serviço de Migração realiza a conversão do formato de disco antes de enviá-lo para a nuvem de destino, com o objetivo de permitir a interoperabilidade do disco, dispensando qualquer intervenção do operador de nuvem (ELHABBASH et al., 2019). Por último, mas não menos importante, o Serviço de Migração também é o responsável por automatizar os 9 passos da migração identificados na pesquisa realizada nas nuvens AWS, AZ e GCP.

Quanto à avaliação da solução, foi realizada uma avaliação de desempenho através da execução de migrações de VMs em cenários homogêneos e heterogêneos. O tamanho do disco migrado entre as nuvens foi fixado como 32 Gigabytes (GB) e a métrica utilizada para avaliar quais foram os melhores cenários de migração foi a de TTM, que é definida como a soma dos tempos para cada um dos 9 passos realizados durante uma migração.

1.5 ORGANIZAÇÃO DO TRABALHO

Este trabalho de dissertação está organizado em 5 capítulos, esta **Introdução** e mais quatro. O **Capítulo 2** apresenta o problema de *vendor lock-in*, e mostra que para resolver este problema existem quatro formas documentadas na literatura. Entre estas soluções apresentadas está a de Serviço Multi-Nuvem, que é a solução que este trabalho adota. Ainda neste capítulo é apresentada a motivação para desenvolver um serviço multi-nuvem focado em migração de VMs, apresentando para isto formas de como conduzir uma migração de VMs. Os problemas encontrados com as formas atuais de migração de VMs são apresentados, e também são especificados quais são os passos gerais para realização de uma migração de VMs entre as nuvens da AWS, AZ e GCP. Neste capítulo também são discutidos trabalhos relacionados.

No **Capítulo 3** é apresentada a arquitetura de Kumo, que quando implementada permite que a migração de VMs entre nuvens seja possível. A arquitetura é formada por dois componentes, a CLI e o Serviço de Migração de VMs, sendo neste mesmo capítulo descrito como foram realizadas suas implementações. A interface que define a assinatura dos métodos que precisam ser implementados é também apresentada. Quando implementados, estes métodos são capazes de integrar nuvens à solução proposta por Kumo, tornando estas nuvens, portanto, interoperáveis umas com as outras.

A avaliação da solução é realizada no **Capítulo 4**, que mostra o plano de experimentação das migrações de VMs e de suas configurações, tanto aquelas que servirão para executar Kumo, quanto as que serão de fato migradas entre as nuvens de origem e de destino. Neste capítulo é apresentada a métrica usada para comparar as migrações, que é a métrica chamada Tempo Total de Migração (TTM). Também é documentado no capítulo como foi realizada a fase de experimentação e são apresentas instruções de como reproduzir estes experimentos. Por fim, os resultados para as migrações homogêneas e heterogêneas são apresentados, sendo realizadas em seguida as análises dos resultados da avaliação de desempenho.

Por último, no **Capítulo 5** são apresentadas as conclusões deste trabalho, evidenciando os resultados mais importantes obtidos. Também são apresentadas as contribuições obtidas com o desenvolvimento deste trabalho de dissertação, e também quais são suas limitações e trabalhos futuros que podem ser desenvolvidos, com o objetivo de evoluir o estado da arte no que diz respeito a migrações de VMs entre multi-nuvens.

2 MIGRAÇÃO DE VMS ENTRE MÚLTIPLAS NUVENS

Segundo Toosi, Calheiros e Buyya (2014), *vendor lock-in* é a situação em que um cliente se torna dependente de um provedor de nuvem, de modo que não seja possível ou viável a migração de seus serviços para outro provedor, seja devido ao custo ou ainda por causa dos esforços técnicos consideráveis que são necessários para a realização da migração.

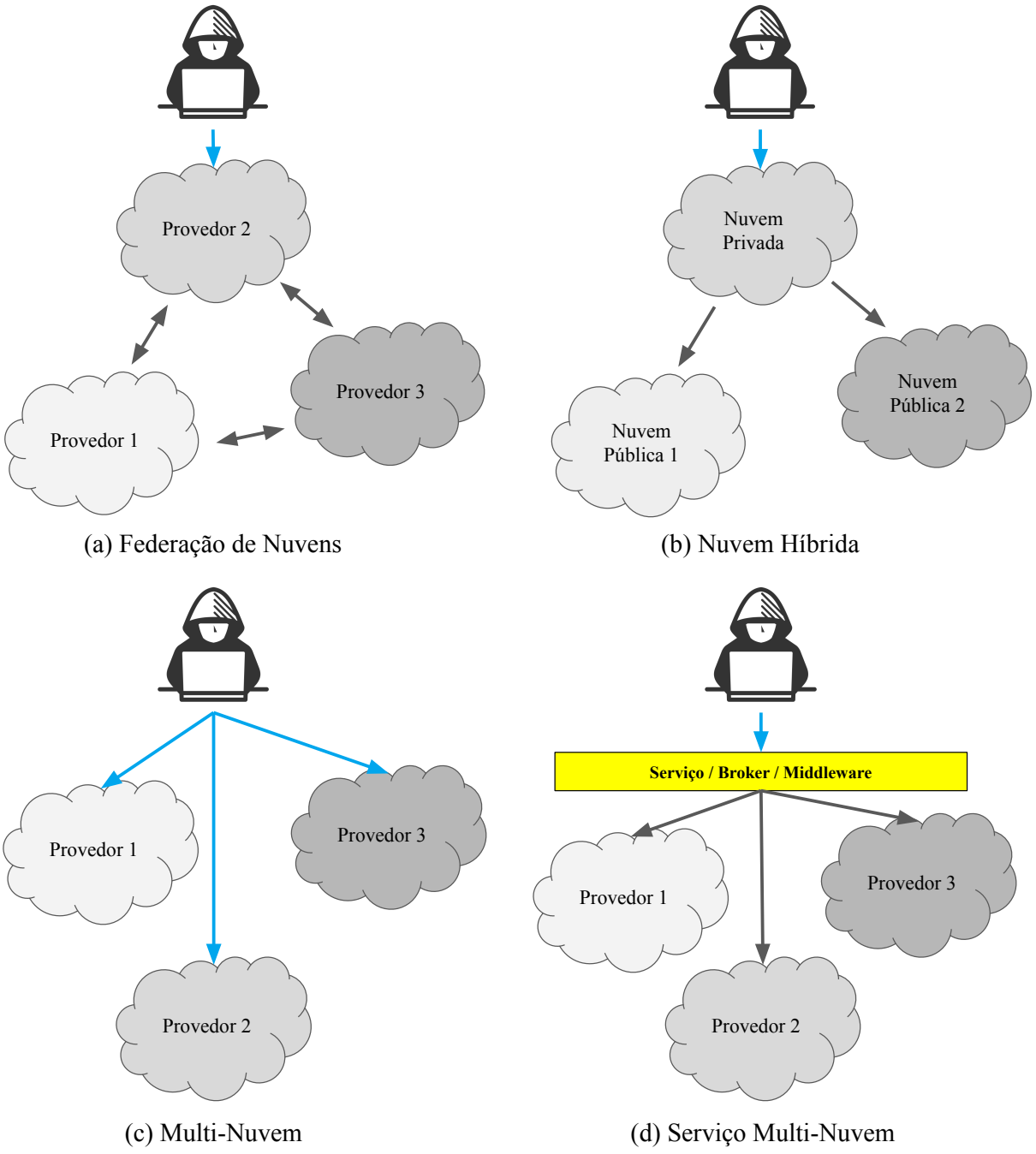
Esta situação acontece, segundo Petcu (2014), porque as nuvens são projetadas para atender aos interesses de seus provedores, não se preocupando com a interoperabilidade destas nuvens com as de outros provedores. Evitar o *vendor lock-in* é uma grande motivação dos clientes de nuvens, enquanto que para os provedores o *vendor lock-in* pode ser atraente, pois auxilia na retenção de clientes (TOOSI; CALHEIROS; BUYYA, 2014).

Então, com o objetivo de evitar o *vendor lock-in*, a utilização de várias nuvens começa a ser adotada, permitindo assim uma série de vantagens para os clientes de nuvens, como as citadas por Petcu (2014) refentes à otimização dos custos de utilização das nuvens; busca de melhorias na Qualidade de Serviço (QoS) das aplicações; garantia de uma melhor resiliência durante casos de desastres ou durante manutenções programadas; alta disponibilidade no fornecimento de aplicações durante picos de requisições; ou ainda a possibilidade de consumir serviços de várias nuvens de acordo com suas particularidades.

Por estes motivos, Toosi, Calheiros e Buyya (2014) apresentam quatro estratégias para a adoção do uso de múltiplas nuvens: Federação de Nuvens, Nuvem Híbrida, Multi-Nuvem e Serviços Multi-Nuvem (também apresentado na literatura como *middleware* ou *broker* multi-nuvem), que são classificadas em estratégias centradas nos provedores de nuvem e estratégias centradas nos clientes.

Nas estratégias centradas nos provedores de nuvem, que são as estratégias de Federação de Nuvens e de Nuvem Híbrida, são os provedores de nuvem que tornam as nuvens interoperáveis, ou seja, os próprios provedores se encarregam de adotar interfaces, protocolos, formatos ou componentes na sua arquitetura de modo a facilitarem a colaboração entre as nuvens. Enquanto isto, nas estratégias para adoção de múltiplas nuvens centradas no cliente, que são a Multi-Nuvem e o Serviço Multi-Nuvem, pelo fato dos provedores não implementarem qualquer facilidade para integração das nuvens, resta ao cliente a responsabilidade de implementar toda esta integração entre os provedores (TOOSI; CALHEIROS; BUYYA, 2014).

Na Figura 1 são apresentadas as arquiteturas das estratégias que buscam facilitar a interoperabilidade no cenário de utilização de várias nuvens. Na estratégia de Federação de Nuvens, representada pela Figura 1 (a), os 3 provedores de nuvem representados na figura precisam aceitar um acordo que define quais recursos serão compartilhados e sob quais circunstâncias. Como a estratégia é implementada pelos provedores, fica transparente ao usuário que a nuvem que este está usando é membro de uma federação.



(a) Federação de Nuvens (b) Nuvem Híbrida

(c) Multi-Nuvem (d) Serviço Multi-Nuvem

Figura 1 – Soluções para o *vendor lock-in*, segundo Toosi, Calheiros e Buyya (2014)

No caso da estratégia de Nuvem Híbrida, representada na Figura 1 (b), uma empresa move parte de sua infraestrutura de nuvem privada para outras duas nuvens públicas, em caso de situações onde a infraestrutura de sua nuvem local não é suficiente para atender a sua demanda atual. Novamente, como se trata de uma estratégia que precisa ser implementada pelos provedores de nuvem, deve ficar transparente aos usuários que estão utilizando recursos de uma nuvem pública em vez dos recursos de sua nuvem privada.

Ainda segundo Toosi, Calheiros e Buyya (2014), mesmo que a interoperabilidade na nuvem não seja suportada pelos provedores, os clientes ainda poderão se beneficiar da

interoperabilidade centrada no cliente, seja utilizando individualmente cada uma das nuvens do cenário Multi-Nuvem, ou ainda utilizar um Serviço Multi-Nuvem. No cenário Multi-Nuvem, representado pela Figura 1 (c), o cliente precisa utilizar individualmente cada uma das nuvens para implementar seus *workflows*, o que pode ser facilitado por meio do uso de bibliotecas para Multi-Cloud, como por exemplo a LibCloud[1], que busca oferecer uma forma homogênea para gerenciar mais de 50 tecnologias de nuvem, através da linguagem de programação Python[2]. Embora que para Petcu (2014), o uso de bibliotecas como LibCloud faz com que as particularidades das nuvens não sejam preservadas em nome da homogeneização.

Por fim, a Figura 1 (d) que se refere ao Serviço Multi-Nuvem, agrega os serviços de vários provedores de nuvem, oferecendo um serviço integrado aos clientes, abstraindo toda a comunicação com cada uma das nuvens dos provedores, e tendo ainda como outro benefício o de fornecer um único ponto de entrada que permite acessar qualquer nuvem integrada ao serviço.

Para Petcu (2014) e Carvalho et al. (2018), as estratégias que exigem acordo entre os provedores de nuvens, como Federação de Nuvens e Nuvem Híbrida, estão mais alinhadas com a academia, onde os acordos para compartilhamento de recursos são mais fáceis de serem estabelecidos do que para a indústria, onde os provedores de nuvem são concorrentes. Enquanto isto, as estratégias que não exigem acordo entre os provedores, como no caso da Multi-Nuvem ou do Serviço Multi-Nuvem, estão melhor alinhadas com a indústria e com os usuários finais, por serem estratégias não invasivas, dispensando o acesso direto à infraestrutura de mais baixo nível das nuvens, e por gerarem valor para organizações que estiverem dispostas à agregar os serviços das várias nuvens.

2.1 MIGRAÇÃO DE MÁQUINAS VIRTUAIS

De acordo com Petcu (2014), a característica mais importante que um Serviço Multi-Nuvem precisa suportar é a portabilidade. Porém, para Carvalho et al. (2018), embora o uso de várias nuvens resolva o problema do *vendor lock-in*, a portabilidade entre estas nuvens ainda é um desafio, tanto na perspectiva do usuário quanto na perspectiva do provedor, pois mesmo com as soluções de padronização propostas pela comunidade acadêmica e pela indústria, nenhuma destas soluções é amplamente adotada, devido à sua falta de flexibilidade.

Um desafio relacionado à portabilidade de infraestrutura é a migração de Máquinas Virtuais (VMs) entre múltiplas nuvens (PETCU, 2014; KOSTOSKA; GUSEV; RISTOV, 2015). Quanto à migração de VMs, Toosi, Calheiros e Buyya (2014) argumentam que esta é mais limitada do que quando a migração é realizada entre *hypervisors* em dois servidores locais. Na migração entre dois *hypervisors* de servidores locais, exitem armazenamento e

[1] https://libcloud.apache.org
[2] https://www.python.org

rede compartilhados entre os dois servidores, sendo assim possível realizar a migração no modo *live*, sem que haja uma interrupção perceptível da VM durante a migração, desde que os *hypervisors* sejam compatíveis com este modo, como por exemplo no caso do XEN[3] e do KVM[4], como sugerido por Toosi, Calheiros e Buyya (2014).

No entanto, na migração de VMs entre nuvens, como são geralmente nuvens concorrentes, não é uma possibilidade o compartilhamento de suas infraestruturas de rede e armazenamento independentes, sendo ainda isolados por seus respectivos *firewalls*, de modo que a única possibilidade para a migração entre estas nuvens, é realizar o desligamento da VM para que seu disco virtual possa ser exportado da nuvem de origem, enviado para a nuvem de destino, e importado também na nuvem de destino. Por causa deste desligamento da VM, que causa uma interrupção durante o processo de migração, este modo de migração é chamado de *non-live*.

Outro problema da migração de VMs entre nuvens está, segundo Kargatzis, Sotiriadis e Petrakis (2017), na heterogeneidade das multi-nuvens, onde uma vez que uma VM precise ser migrada, é preciso garantir que esta VM seja compatível com as tecnologias, padrões e ferramentas da nuvem de destino. Para Petcu (2014), um exemplo da heterogeneidade na migração de VMs é encontrada nas tecnologias envolvidas no processo, como nos *hypervisors* e nos formatos de disco virtual utilizados pelos provedores. Com isto, é uma contribuição que um Serviço Multi-Nuvem seja capaz de tratar e ocultar toda esta heterogeneidade. Para Toosi, Calheiros e Buyya (2014), tratar a heterogeneidade é um problema crucial, que acontece devido aos provedores de nuvem implementarem suas soluções usando interfaces e soluções proprietárias.

Para Carvalho et al. (2018), o gerenciamento de recursos entre as multi-nuvens, como no caso da migração de VMs, pode ser realizado de duas formas: de forma interativa, que necessita de intervenção de um operador de nuvem; e de forma automática, onde a gerência é realizada sem nenhuma intervenção do operador. E mais, o gerenciamento pode acontecer de quatro formas: a manual, que necessita total interação do operador de nuvem; a semi-automatizada, onde alguns passos intermediários são executados de maneira manual e outros de maneira automatizada; a forma automatizada, onde após o comando do operador todos os passos são executados sem intervenção; e a forma autônoma, onde existe uma inteligência que percebe as oportunidades de realizar melhorias nos sistemas, o que pode resultar em migrações de VMs, sem qualquer intervenção humana.

2.2 REQUISITOS PARA UM SERVIÇO MULTI-NUVEM

Conforme proposto por Petcu (2014), existem diversos requisitos que podem ser implementados em serviços do tipo Multi-Nuvem. Dentre aqueles que fazem sentido para a concepção de um serviço Multi-Nuvem de migração de VMs, foram selecionados:

[3] https://xenproject.org
[4] https://www.linux-kvm.org

1. Definir uma interface abstrata que controle os serviços das nuvens;
2. Preservar as particularidades de cada nuvem;
3. Desenvolvimento de uma interface para gerenciamento das migrações;
4. O serviço precisa ser agnóstico quanto às nuvens envolvidas na migração;
5. Ser um serviço agregador e que combine os serviços de diversas nuvens;
6. Automatizar a migração entre as nuvens;
7. Criação de um repositório de credenciais;
8. Permitir a integração dos provedores de nuvem mais relevantes;
9. Não impor restrições às nuvens;
10. Introduzir o mínimo de sobrecarga na migração;
11. Padronizar as Interfaces de Programação de Aplicações (APIs) do serviço e as mensagens trocadas com o serviço;
12. Suportar múltiplas tecnologias de *hypervisor*.

2.3 POR QUE MIGRAR?

Um dos motivos mais fortes para realizar a migração de VMs entre nuvens é a redução de custos com infraestrutura da nuvem de origem (NARANTUYA; ZANG; LIM, 2018), mas existem diversos outros motivos para realizar uma migração de VM entre duas nuvens. Em Zhang et al. (2018) são apresentados alguns destes motivos que justificam a migração. O primeiro é quando uma manutenção no servidor que está executando VMs é necessária, e por isso as VMs precisam ser migradas para outro servidor para que aquele servidor possa passar pela manutenção exigida. Um segundo motivo é para balancear a carga entre servidores, que quando sobrecarregados podem ter algumas de suas VMs migradas para outros servidores que estejam menos sobrecarregados. Um terceiro motivo para migrar uma VM é para que esta fique mais próxima de outras VMs com as quais têm mais afinidade. Outra justificativa para a migração (frequente) de uma VM é para que esta se adeque à mobilidade do usuário e assim possa fornecer um serviço com menor latência (ex., um cenário de *fog computing* (BITTENCOURT et al., 2018) ou *follow the sun* (SHEN et al., 2016)). O resfriamento dos servidores do *data center* pode também motivar a migração de VMs para servidores em *data centers* localizados em lugares mais frios ou para onde é noite (ex., *follow the moon*).

Ainda em Zhang et al. (2018) são apresentados outros motivos para migrar VMs, como em casos onde nuvem híbrida é usada e as VMs que precisem de mais recursos podem

ser migradas da nuvem privada para a nuvem pública, ou no caso de federação de nuvens onde podem ser migradas para a infraestrutura do membro com mais recursos disponíveis. Por fim, o problema mais clássico que pode motivar um cliente a migrar suas VMs entre provedores distintos, é para evitar o *vendor lock-in*, que é a situação onde o cliente da nuvem cria dependência por serviços e recursos do provedor (TOOSI; CALHEIROS; BUYYA, 2014; KOSTOSKA; GUSEV; RISTOV, 2015).

2.4 TIPOS DE MIGRAÇÃO

Na pesquisa realizada foram identificadas 3 classificações para as migrações de VMs entre nuvens: a primeira se refere ao modelo de implantação da nuvem, a segunda é quanto à arquitetura das nuvens, e a última classificação é quanto ao modo de migração.

Quanto ao modelo de implantação, as migrações de VMs entre nuvens podem ser classificadas como **migrações entre nuvens privadas**, como entre nuvens OpenStack (OS)[5] e o CloudStack (CS)[6]; **migrações entre nuvens públicas**, como entre as nuvens Amazon Web Services (AWS)[7] e a Microsoft Azure (AZ)[8]; ou ainda **entre nuvens híbridas**, como no caso de uma nuvem privada como o OpenStack utilizando recursos da nuvem pública AWS.

Na classificação quanto à arquitetura, as migrações de VMs podem ser realizadas **entre nuvens homogêneas**, como por exemplo numa migração de uma VM entre duas nuvens de mesma tecnologia, como de OpenStack para OpenStack ou de AWS para AWS, ou ainda podem ser classificadas como **migrações entre nuvens heterogêneas**, como por exemplo em uma migração de VM feita de uma nuvem OpenStack para a Google Cloud Platform (GCP)[9].

Por último, na classificação quanto ao modo de migração de VMs entre nuvens, existem dois modos, cada um com diferentes características e propósitos. Como estudado por Celesti et al. (2010), o modo de **migração *live*** é um modo usado quando é necessário que a VM continue em execução no *hypervisor* da nuvem de origem, enquanto o seu disco e sua memória são migrados para o destino. Por outro lado, no modo de **migração *non-live***, a VM tem sua execução interrompida para que seu disco possa ser migrado para a nuvem de destino.

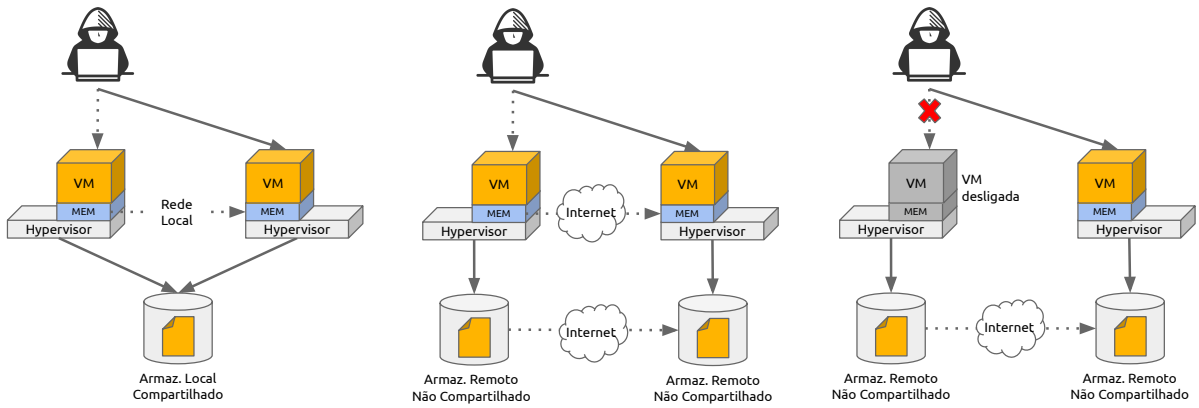
Como observado por Narantuya, Zang e Lim (2018), como a migração realizada no modo *live* envolve a transferência de dados da memória da VM da origem para a memória da VM do destino, este modo se torna mais lento do que o modo *non-live*. Complementando esta opinião, Shirvani, Rahmani e Sahafi (2018) afirma que embora o modo *non-live* seja mais rápido por não realizar transferência de memória da origem para o destino, este é

[5] https://www.openstack.org
[6] https://cloudstack.apache.org
[7] https://aws.amazon.com
[8] https://azure.microsoft.com
[9] https://cloud.google.com

um modo que tem sua aplicação restrita a cenários onde as VMs possam ter sua execução interrompida, como na migração planejada de um provedor de nuvem para outro.

Para Ahmad et al. (2015), as migrações realizadas através da Internet, como as migrações entre nuvens públicas, enfrentam vários desafios devido à largura de banda limitada, maiores latências, comportamento imprevisível da rede, *design* heterogêneo da arquitetura de rede, distâncias maiores de comunicação e maior taxa de perda de pacotes do que nas migrações realizadas entre dois servidores locais.

Em adição, Zhang et al. (2018) introduz estes mesmos dois modos de migração, *live* e *non-live*, apresentados por Celesti et al. (2010) e caracterizadas por Ahmad et al. (2015), porém apresenta a migração de modo *live* sob duas perspectivas, a migração *live* local e *live* remota, como pode ser visto na Figura 2. Na migração *live* local, representada pela Figura 2 (a), são requisitos, tanto o armazenamento compartilhado para que o disco virtual possa ser compartilhado entre os *hypervisors*, quanto a rede compartilhada para a transmissão do conteúdo da memória da VM da origem para o destino. Enquanto isto, na migração *live* remota, como visto na Figura 2 (b), apenas a rede compartilhada é necessária, tanto para transmitir a memória quanto o disco da VM para o destino. Por fim, comparando estas duas perspectivas do modo *live* com o modo *non-live*, representado pela Figura 2 (c), percebe-se que no contexto de nuvens públicas como a AWS, a AZ e a GCP, onde não é possível ter acesso aos servidores e seus *hypervisors* para configurar o compartilhamento de armazenamento e/ou rede, o modo *live*, tanto local quanto remoto, é inviável para migração entre nuvens públicas, sendo assim a única abordagem praticável para a migração de VMs, entre nuvens públicas, o modo *non-live*.



(a) Migração *live* local (b) Migração *live* remota (c) Migração *non-live* remota

Figura 2 – Classificação dos modos de migração, segundo Zhang et al. (2018)

2.5 COMO MIGRAR?

Segundo Ebert et al. (2016), a cultura DevOps busca a colaboração entre times de desenvolvedores e de operadores de infraestrutura. Com o propósito de integrar estes dois

times, na cultura DevOps é incentivado o uso de ferramentas que facilitem a colaboração e automatização de tarefas. Com isso, é esperado que juntos estes times consigam realizar entregas contínuas, permitindo lançamentos de versões cada vez mais rápidas. Entretanto, os provedores de nuvens atuais fornecem para seus clientes ferramentas para gerenciamento de infraestrutura que não favorecem esta colaboração, sendo muito mais familiares para operadores de nuvens do que para desenvolvedores.

A alternativa atual fornecida pelos provedores de nuvem se dá por meio de diversas ferramentas que precisam ser instaladas, configuradas de maneiras diferentes e utilizadas uma a uma. Além dos comandos utilizados para realizar uma migração serem diferentes em cada uma das ferramentas, o processo de migração é realizado de maneira manual, sem integração. O operador de nuvem se autentica em cada nuvem, e precisa saber quais comandos e quais parâmetros são necessários para exportar o disco virtual da nuvem de origem, baixá-lo, convertê-lo usando uma ferramenta de terceiros (ex. QEMU Disk Image Utility[10]), enviar o disco para o destino, para em seguida importá-lo no destino, e assim poder criar uma nova VM, tudo isso utilizando comandos diferentes para cada uma das nuvens envolvidas na migração.

Para avaliar como o processo de migração poderia ser automatizado, foi utilizada uma ferramenta que, segundo a pesquisa realizada por FLEXERA (2019), trata-se da ferramenta mais utilizada pelas empresas para gerenciar recursos em ambientes multi-nuvens, que é a ferramenta de automatização Ansible[11]. A versão 2.8 do Ansible foi avaliada e percebeu-se que esta não disponibilizava alguns módulos necessários para realizar algumas das ações específicas referentes à migração de VMs em algumas nuvens. Por exemplo, para exportar o disco das VMs na GCP e na AWS.

Como o Ansible é uma ferramenta de gerenciamento automatizado de infraestrutura de propósito geral, ou seja, não é uma ferramenta focada na migração de VMs entre nuvens, esta ferramenta precisaria de algum esforço de implementação para que dentre suas funcionalidades, estivesse a capacidade de também realizar migrações de VMs entre nuvens de forma automatizada. Uma segunda dificuldade de Ansible quanto à migração de VMs, o que também acontece na forma manual com as ferramentas fornecidas pelas nuvens, é a heterogeneidade dos parâmetros que precisam ser passados para a ferramenta, o que torna o processo, além de incapaz de realizar todos os passos da migração de maneira nativa (sem implementação de nenhuma adaptação), complexo devido a quantidade de parâmetros que precisam ser passados.

2.5.1 Migração Manual: Ferramentas Proprietárias

Os provedores de nuvem atuais fornecem geralmente 3 formas de acesso a seus serviços, as Interfaces Gráficas do Usuário (GUIs), as Interfaces de Linha de Comando (CLIs) e as

[10] https://linux.die.net/man/1/qemu-img
[11] https://www.ansible.com

bibliotecas para as linguagens de programação. Geralmente, entre estas 3 formas, a forma mais utilizada pelos operadores de nuvem é a CLI, que permite um acesso mais privilegiado do que na GUI, sendo consideravelmente mais ágil, por exemplo, do que desenvolver um código para implementar seus *workflows* utilizando bibliotecas ou Software Development Kits (SDKs) para a comunicação com as nuvens.

Entre as CLIs, para a AWS o provedor disponibiliza a AWS CLI[12], compatível com os Sistemas Operacionais (SOs) Windows e Linux. Para a AZ, é disponibilizada a Azure CLI[13] com versões distintas para os SOs Windows (utilizando a Linguagem PowerShell[14]) e Linux. E para a GCP são disponibilizadas a GSUtil CLI[15] para o serviço de armazenamento, e a GCloud CLI[16] para os demais serviços, que assim como a AWS CLI possuem compatibilidade com os dois SOs, Windows e Linux.

Para realizar uma migração de maneira manual é preciso executar até 9 **passos** (descritos na seção 3.2). Vale ressaltar que, para a execução destas migrações de maneira manual, é necessário sempre utilizar ao menos duas ferramentas do tipo CLI (geralmente uma da nuvem de origem – ex. AWS CLI, e duas da nuvem de destino – ex. GCloud CLI e GSUtil CLI) com seus respectivos comandos, parâmetros e valores esperados, e ainda caso o disco exportado da nuvem de origem não esteja de acordo com o tamanho ou formato esperados pela nuvem de destino, será ainda necessária mais uma ferramenta para realizar uma adequação deste disco, por meio do redimensionamento ou conversão do formato para um novo formato compatível com a nuvem de destino. A Tabela 1 mostra todos os comandos necessários para realizar cada um dos 9 passos de maneira manual, usando para isso as ferramentas CLI disponibilizadas pelos provedores AWS, AZ e GCP.

[12] https://docs.aws.amazon.com/cli/latest/reference/index.html
[13] https://docs.microsoft.com/en-us/cli/azure
[14] https://docs.microsoft.com/en-us/powershell/scripting/overview
[15] https://cloud.google.com/storage/docs/gsutil
[16] https://cloud.google.com/sdk/gcloud

Tabela 1 – Comandos para realizar os passos das migrações manualmente

Passo \ Nuvem	AWS	AZ	GCP

1	`aws s3 mb;`
	`aws iam create-role;`
	`aws iam put-role-policy`
		`az storage container create`	`gsutil mb`

2	`aws s3 mb;`
	`aws iam create-role;`
	`aws iam put-role-policy`
		`az storage container create`	`gsutil mb`

3	`aws ec2 stop-instances`	`az vm stop`	`gcloud compute instances stop`
4	`aws ec2 export-image`	`az snapshot create`	`gcloud compute images export`
5	`aws s3 cp`	`az snapshot grant-access; wget`	`gsutil cp`

6	-	`qemu-img convert;`
		`qemu-img resize;`
		`qemu-img convert`
			-

7	`aws s3 cp`	`az storage blob upload`	`gsutil cp`
8	`aws ec2 import-image`	`az image create`	`gcloud compute images import`
9	`aws ec2 run-instances`	`az vm create`	`gcloud compute instances create`

O passo 6 apenas é executado pela AZ, que tem como requisito que o disco tenha seu tamanho alinhado à 1 Mebibyte (MiB), o que é realizado por meio dos comandos `"convert"` e `"resize"` da ferramenta QEMU Disk Image Utility.
Observação: Vale lembrar que a repetição dos comandos nos passos 1 e 2 é proposital e acontece devido ao sentido da migração. Logo, uma vez que uma migração é realizada, por exemplo, da origem AWS para o destino GCP, será executado no passo 1 os comandos da AWS e no passo 2 os da GCP. Entretanto, caso o sentido seja invertido (GCP → AWS), será executado no passo 1 os comandos da GCP e no passo 2 os da AWS.

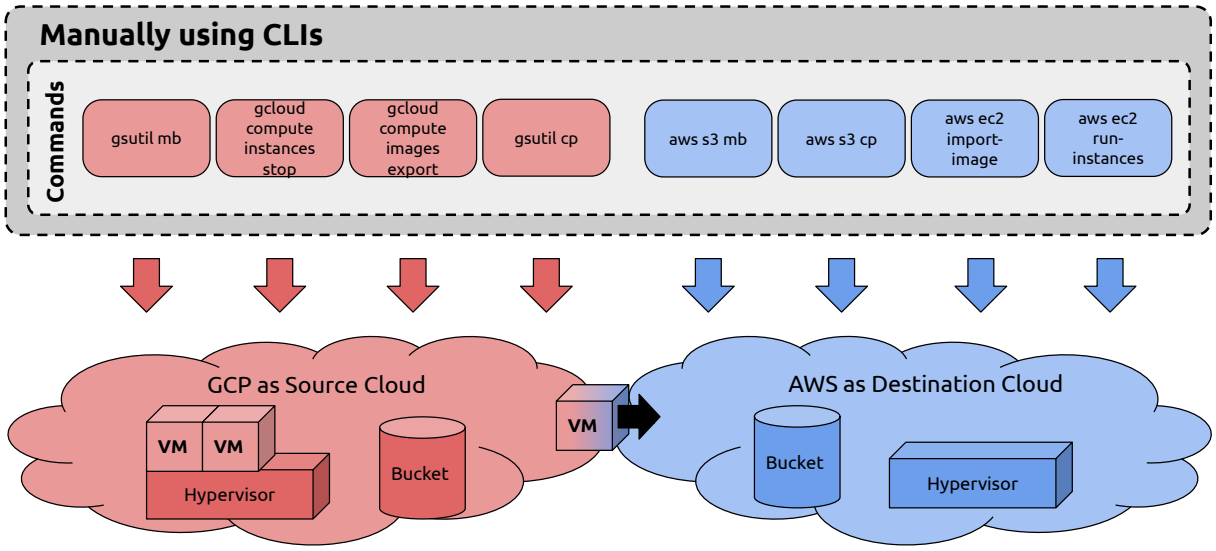


Figura 3 – Representação da forma manual de migração

2.5.2 Migração Automatizada: Ansible

Ansible[17] é uma ferramenta que permite a automatização da gerência de configuração de infraestrutura de Tecnologia da Informação e Comunicação (TIC). Quando utilizada no contexto de nuvem, Ansible pode ser usado, por exemplo, na configuração de recursos e infraestrutura virtual de diversos serviços para vários provedores de nuvem. Como mostrado em Thiyagarajan (2020) por meio de seus estudos de caso realizados no contexto de migração devido à *Disaster Recovery*, Ansible é uma ferramenta com suporte à multi-nuvem e que permite que infraestrutura possa ser gerenciada por meio de código.

Segundo Ebert et al. (2016), dentre outras opções de ferramentas para gerência de configuração, como por exemplo Puppet[18] e Chef[19], Ansible é que mais se destaca por sua facilidade de uso, pois dispensa a instalação de agentes como dependência para execução das tarefas remotas. Atualmente existem para o Ansible integrações com as nuvens AWS, AZ e GCP, por exemplo, que são implementadas por meio dos chamados **módulos**. O Código 1 mostra um exemplo do uso de um módulo chamado `s3_bucket`, responsável por automatizar as ações referentes ao serviço de armazenamento Simple Storage Service (S3) da nuvem da AWS.

A Figura 4 mostra como é realizada a execução de um *playbook* do Ansible, que é

[17] https://docs.ansible.com/ansible/latest/index.html
[18] https://puppet.com
[19] https://www.chef.io

```
- name: Cria o bucket "kumo-cin-ufpe" para armazenar arquivos na AWS.
- s3_bucket:
    name: kumo-cin-ufpe
    policy: "{{ lookup('file','policy.json') }}"
    versioning: yes
    tags:
      example: kumo
```

Código 1 – Exemplo do uso do módulo `s3_bucket` do Ansible

um arquivo YAML Ain't Markup Language (YAML) onde são declarados conjuntos de diversas *tasks* que precisam ser executadas sequencialmente em sistemas. Por exemplo, para realizar migrações de VMs, os *playbooks* declaram quais os módulos que são necessários para executar os passos envolvidos na migração, para as nuvens de origem e destino. Quando a CLI do Ansible é executada recebendo o *playbook* como parâmetro, este é interpretado para detectar quais módulos estão sendo utilizados naquele *playbook*, e assim, aquelas *tasks* declaradas no *playbook* são executadas através destes módulos. Entretanto, durante tentativas de execução de migrações com Ansible versão 2.8 para a AWS, a AZ e para a GCP, percebeu-se que faltam alguns módulos para que a migração possa ser realizada, sendo estes os módulos referente à exportar discos virtuais das nuvens, e outro módulo que seja capaz de realizar a conversão do formato do disco, assim também como seu redimensionamento para atender à exigência do processo de importação das nuvens. Observe na Figura 4 os módulos que são disponibilizados por Ansible para a migração de GCP para AWS, por exemplo, quais sejam:

1. `gcp_storage_bucket`: Cria um diretório para armazenamento de arquivos na GCP;
2. `s3_bucket, iam_role, iam_policy`: Cria um diretório para armazenamento de arquivos na AWS, com o devido *role* e *policy*, autorizando o Elastic Compute Cloud (EC2) à acessar o Simple Storage Service (S3) e criar uma imagem baseada no disco lá armazenado;
3. `gcp_compute_instance`: Desliga uma VM na GCP;
4. `gcp_storage_object`: Baixa o arquivo de disco da GCP;
5. `aws_s3`: Faz o envio de um arquivo de disco para a AWS;
6. `ec2_ami`: Cria uma imagem de disco com base no disco enviado;
7. `ec2_instance`: Cria uma instância de VM na AWS.

Na Tabela 2 podem ser vistos os módulos do Ansible que são necessários para realizar migrações automatizadas. Vale observar que para que isto seja possível é necessário automatizar a execução de comandos que não possuem módulos equivalentes no Ansible. Para estes casos, uma possibilidade é a utilização do módulo `command` do Ansible, que é capaz

de executar, de maneira automatizada, comandos de terminal que foram projetados e desenvolvidos para serem executados manualmente. Outra observação importante quanto ao Ansible na realização migrações de VMs entre nuvens, é que vários passos da migração são realizados de maneira assíncrona, ou seja, o Ansible precisaria ser capaz de manter o estado de qual momento da migração está sendo executado, para que uma vez que um passo termine o próximo possa ser iniciado. Entretanto, isto pode ser um problema, inclusive em passos que são implementados usando o módulo `command`, uma vez que não existem os mecanismos necessários para esperar até que o passo anterior seja concluído com sucesso, para assim iniciar o passo seguinte. Com isso, uma solução é contribuir com o projeto Ansible[20], que é *open source* e permite que usuários desenvolvam novos módulos que, além de executarem os comandos necessários para a realização do passo, tratem de forma apropriada os comportamentos necessários para tarefas que são executadas de maneira assíncrona, bem como tratamento de erro e exceções, e ainda comportamentos de tolerância à falhas, tudo de acordo com a padronização especificada pela comunidade.

[20] https://docs.ansible.com/ansible/latest/community/how_can_I_help.html

Tabela 2 – Módulos para realizar os passos das migrações com Ansible

Passo \ Nuvem	AWS	AZ	GCP
1	`s3_bucket`, `iam_role`, `iam_policy`	`azure_rm_storageblob`	`gcp_storage_bucket`
2	`s3_bucket`, `iam_role`, `iam_policy`	`azure_rm_storageblob`	`gcp_storage_bucket`
3	`ec2_instance`	`azure_rm_virtualmachine`	`gcp_compute_instance`
4	–	`azure_rm_snapshot`	–
5	`aws_s3`	`command: az snapshot grant-access; wget`	`gcp_storage_object`
6	–	`command: qemu-img convert`; `qemu-img resize`; `qemu-img convert`	–
7	`aws_s3`	`azure_rm_storageblob`	`gcp_storage_object`
8	`ec2_ami`	`azure_rm_image`	`gcp_compute_image`
9	`ec2_instance`	`azure_rm_virtualmachine`	`gcp_compute_instance`

Como não existe nenhum módulo no Ansible com o propósito de baixar o disco exportado, ou ainda de converter e redimensionar discos virtuais, o módulo `command` do Ansible pode ser usado para automatizar a execução de comandos, como no caso do `"az snapshot grant-access"` e `"wget"` no **passo 5**, e no **passo 6** dos comandos `"qemu-convert"` e `"qemu-resize"` da ferramenta QEMU Disk Image Utility.

Observação: Vale lembrar que a repetição dos módulos nos passos 1 e 2 é proposital e acontece devido ao sentido da migração. Logo, uma vez que uma migração é realizada, por exemplo, da origem AWS para o destino GCP, será executado no passo 1 os comandos da AWS e no passo 2 os da GCP. Entretanto, caso o sentido seja invertido (GCP → AWS), será executado no passo 1 os comandos da GCP e no passo 2 os da AWS.

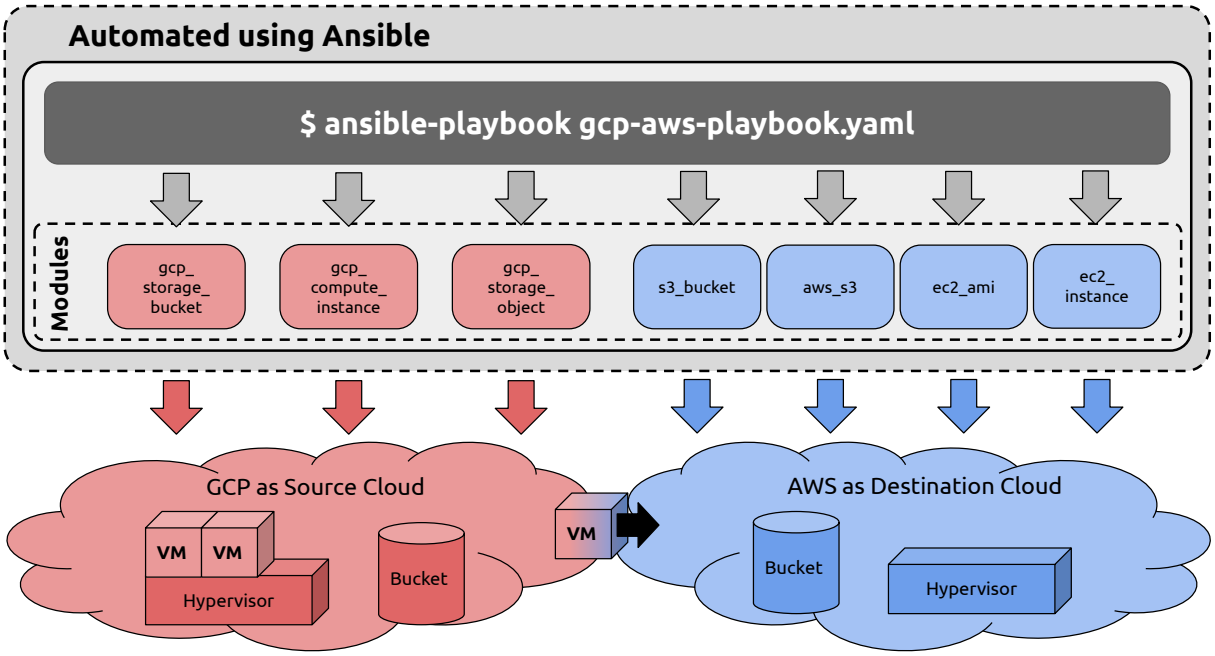


Figura 4 – Representação da forma automatizada de migração

2.6 QUANTIDADE DE PASSOS PARA MIGRAÇÃO

Em geral, são necessários diversos passos para realizar a migração de VMs entre nuvens, sejam elas homogêneas ou heterogêneas. Como mostrado na Tabela 3, nos estudos feitos na AWS, AZ e GCP, observou-se que a migração se dá em 8 passos quando é feita da AWS para AWS. Quando realizada da AWS para a AZ são necessários 9 passos.

Tabela 3 – Quantidade de passos para a migração de VMs

Origem \ Destino	AWS	AZ	GCP
AWS	8	9	8
AZ	7	7	7
GCP	8	9	8

Para os cenários onde a AZ é a origem da migração são dados 7 passos para qualquer nuvem de destino. Por fim, quando a nuvem de origem é a GCP e a nuvem de destino se trata da AWS, a migração é feita em apenas 8 passos. Quando realizada da GCP para a AZ, a migração precisa executar um total de 9 passos. E por fim, no cenário homogêneo entre nuvens da GCP, também são necessários apenas a execução de 8 passos.

Logo, de maneira geral, este trabalho considera um total de 9 passos para a migração de VMs entre nuvens (detalhados na seção 3.2), e possui enfoque no modo de migração

non-live, apresentando resultados de migrações realizadas entre nuvens públicas homogêneas e heterogêneas (na seção 4). O processo de migração foi automatizado através do desenvolvimento de um serviço Multi-Nuvem chamado Kumo, apresentado no capítulo 3.

2.7 TRABALHOS RELACIONADOS

A migração de VMs continua sendo um problema atual, que foi atualizado do modelo de virtualização para o modelo de Computação na Nuvem, e sendo novamente atualizado mais recentemente, tendo agora como foco o uso de múltiplas nuvens, o que faz com que a migração de VMs ganhe ainda mais importância e volte a chamar atenção de pesquisadores, que buscam livrar os usuários de múltiplas nuvens dos problemas relacionados à esta estratégia, como a heterogeneidade (MANSOUR et al., 2016). Quando este problema é solucionado, seja entre ambientes de nuvens privadas, públicas ou híbridas, é esperado que as VMs possam ser migradas mais facilmente, eliminando a necessidade de reinstalar todo um sistema em uma nova VM na nuvem de destino (NARANTUYA; ZANG; LIM, 2018).

Os autores Zhang et al. (2017) propõem uma nova forma de realizar migração de VMs através da criação de um repositório central para armazenamento de imagens de SOs externo às nuvens, mas no experimento foi utilizada uma instalação KVM emulando o comportamento de uma nuvem. Nesta solução, quando um usuário da nuvem solicita a criação de uma VM, é necessário fornecer o identificador para uma imagem armazenada no repositório central, onde aquela imagem escolhida deve ser baixada pela nuvem onde a VM será criada, para que em seguida possa ser usada como base para a criação do disco daquela VM. Quando o usuário decide que precisa ser realizada uma migração, o mesmo identificador utilizado para baixar a imagem na nuvem de origem, é enviado para a nuvem de destino que acessa o mesmo repositório de imagens, baixando assim a mesma imagem de disco que foi utilizada pela nuvem de origem, porém agora lançando a VM com aquela imagem na nuvem de destino. O próximo passo é então criar um armazenamento compartilhado, utilizando uma tecnologia chamada Distributed Replicated Block Device (DRBD) (REISNER, 2002), entre a VM de origem e a VM recém lançada no destino, para que os dados que estão no disco da VM de origem possam ser transferidos para a VM de destino.

Em Narantuya, Zang e Lim (2018), o foco do trabalho está na possibilidade de realizar migrações de múltiplas VMs no menor tempo possível, onde estas VMs têm dependência entre si, formando juntas uma única aplicação. Assim, os autores utilizam captura de tráfego de rede para definir quando existe uma dependência entre as VMs, para que em seguida seja realizada uma análise dos valores obtidos nesta captura de tráfego, e assim determinar quais VMs devem ser migradas primeiro. As migrações são então realizadas entre duas nuvens OpenStack, através de um agente na origem que envia a VM para outro agente na nuvem de destino. Nesta solução, as migrações realizadas utilizam o modo *non-live*, que segundo os autores é mais indicado para casos como este de migração de múltiplas

VMs, pois é considerado que a aplicação só poderá ser executada novamente no destino depois que todas as VMs forem migradas. Se forem migradas de forma *live* (o que pode ser feito quando as duas nuvens OpenStack têm acesso direto ao *hypervisor* uma da outra), além deste modo de migração demorar mais do que o *non-live*, de toda forma a aplicação precisaria esperar que todas as VMs fossem migradas para poder iniciar.

Em Mansour, Bouchachia e Cooper (2017), a proposta é realizar migração de VMs entre um *hypervisor* local e outro *hypervisor* instalado em uma VM na AWS. Esta contribuição é um primeiro passo para realização de migrações entre quaisquer infraestruturas de virtualização, inclusive nuvens, de modo *live*. Para a realização dos experimentos foi realizada uma instalação do virtualizador Quick EMUlator (QEMU) localmente, enquanto que como destino da migração foi usada uma instância do EC2, onde foi instalada uma versão customizada do QEMU, o Hybrid Quick EMUlator (HQEMU), que permite a instalação do *hypervisor* em máquinas onde a tecnologia de virtualização dos processadores (como por exemplo na Intel, a tecnologia VT-x) foi desabilitada, justamente para impedir a instalação de outro *hypervisor* nas VMs do EC2. Também foi implementado um túnel IPsec entre as duas máquinas. Foram avaliadas duas abordagens diferentes: na primeira abordagem, a migração foi implementada mantendo os discos virtuais na origem e foram lançadas duas VMs dentro da VM do EC2 utilizando os dois discos da origem como discos destas VMs, resultando em uma migração bem sucedida (disco com Windows XP) e uma mal sucedida (disco com Ubuntu 14.04); no segundo cenário ou abordagem, os discos foram de fato migrados (enviados) para o destino, e as migrações foram bem sucedidas para ambas as VMs.

Na pesquisa realizada por Yadav e Krishna (2019), é proposta uma solução baseada em um diretório compartilhado entre uma instalação de Network File System (NFS) no servidor de origem e outra no servidor de destino. Segundos os autores, tradicionalmente a migração de VMs é implementada através do envio do disco virtual da VM, com seu SO e seus dados, do servidor de origem para o servidor de destino, causando um alto tráfego na rede entre os servidores. A solução proposta é que apenas as páginas de memória da VM sejam migradas, enquanto o disco com todos os outros arquivos fique armazenado no diretório compartilhado no NFS do servidor de origem e compartilhado com o NFS do servidor de destino, que utilizará este disco virtual para criar a nova VM no destino, permitindo que em seguida as páginas de memória sejam migradas para o destino, de maneira *live*, através do próprio Kernel-based Virtual Machine (KVM).

No trabalho de Kargatzis, Sotiriadis e Petrakis (2017), é mostrada uma forma de fazer migração de VMs entre duas nuvens, uma OpenStack e como nuvem de destino os autores utilizam uma instalação de um *hypervisor* VMWare[21], ambos instalados localmente. Para realizar estas migrações, os autores utilizam os próprios serviços das nuvens para retirar uma VM da nuvem OpenStack e enviá-la para o VMWare, e utilizam o modo *non-live* de

[21] https://www.vmware.com

migração, realizando assim o desligamento da VM na nuvem de origem antes de enviá-la para a nuvem de destino. Vale notar que não há migrações sendo realizadas no sentido VMWare, como origem, para o OpenStack, como destino.

Nestes trabalhos relacionados foram observados alguns aspectos importantes, como na proposta de Zhang et al. (2017), que propõe o uso de um repositório de armazenamento central, mas que necessitaria que as nuvens aceitassem e confiassem utilizar estas imagens. Outro aspecto relevante proposto por Narantuya, Zang e Lim (2018), com a possibilidade de migrar múltiplas VMs de uma única vez, VMs estas que juntas compõem um único serviço, o que poderia ser um trabalho de investigação futuro para evolução desta pesquisa. O trabalho de Mansour, Bouchachia e Cooper (2017) realiza migração *live* entre uma nuvem local e a AWS, mas precisa para isto instalar um *hypervisor* modificado para executar uma VM dentro de outra na AWS, o que limita a aplicação da solução. Em Yadav e Krishna (2019) é proposta a utilização de NFS para compartilhar o disco entre os *hypervisors* da origem e de destino, necessitando se concentrar na migração de páginas de memória. O trabalho de Kargatzis, Sotiriadis e Petrakis (2017) é o mais próximo deste, trazendo pontos relevantes, como a migração sendo conduzida de maneira não invasiva, sem a necessidade de se ter acesso direto às infraestruturas das nuvens de origem e destino, e de modo *non-live*, que são aspectos alvos de investigação no trabalho aqui proposto.

Um diferencial deste trabalho em relação aos demais trabalhos, é que este utiliza nuvens públicas reais e concorrentes (por sinal, as mais utilizadas na atualidade), tanto homogêneas quanto heterogêneas, para realizar as migrações e coletar seus tempos da execução, para assim poder avaliar o desempenho destas migrações (visto no Capítulo 4). Logo, este trabalho está mais alinhado com a realidade Multi-Nuvem adotada pela indústria, que utiliza em sua maioria nuvens públicas (FLEXERA, 2019), diferentemente de outros trabalhos, que utilizam instalações de nuvens ou *hypervisors* instalados localmente, ou ainda simuladores para avaliar o comportamento das migrações entre as múltiplas nuvens.

2.8 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Neste capítulo foi apresentado o problema do *vendor lock-in*, que segundo Toosi, Calheiros e Buyya (2014) é a situação em que um cliente se torna dependente de um provedor de nuvem, de modo que não seja possível ou viável a migração de seus serviços para outro provedor, seja devido ao custo ou ainda por causa dos esforços técnicos consideráveis que são necessários para a realização da migração.

Entre as soluções para o *vendor lock-in*, Toosi, Calheiros e Buyya (2014) sugerem o uso de um Serviço Multi-Nuvem. Na solução do Serviço Multi-Nuvem, os serviços de vários provedores de nuvem são agregados e fornecidos de maneira integrada aos clientes, abstraindo toda a comunicação com cada uma das nuvens dos provedores, e tendo ainda

como outro benefício fornecer um único ponto de entrada que permite acessar qualquer nuvem integrada ao serviço.

Em seguida o problema da migração de VMs é apresentado, mostrando as dificuldades de sua implementação devido à heterogeneidade das nuvens e de *hypervisors*, e à presença ou não de redes e/ou armazenamentos compartilhados entre os servidores das nuvens de origem e de destino. É realizada uma distinção entre os modos de migração *live*, que realiza a migração da VM sem realizar seu desligamento, mas sendo tecnicamente inviável na migração entre nuvens públicas, e o modo *non-live* que realiza o desligamento da VM, tornando assim este modo factível no contexto de nuvens públicas, tanto entre nuvens homogêneas quanto heterogêneas, e sendo por este motivo o modo investigado neste trabalho.

Foram apresentados requisitos que, segundo Petcu (2014), devem ser implementados em serviços multi-nuvem, como: ter uma interface abstrata que controle os serviços das nuvens; ser agnóstico quanto às nuvens envolvidas na migração; e automatizar a migração entre as nuvens. Foram apresentadas também justificativas para uma dúvida fundamental quando se pesquisa sobre migrações de VMs, o porquê de realizar a migração, sendo a redução de custos e evitar o *vendor lock-in* as justificativa mais clássicas.

Ainda no capítulo são classificados os tipos de migração: quanto ao modelo de implantação, as possibilidades são a migração entre nuvens privadas, públicas ou híbridas; quanto à sua arquitetura, homogêneas quando origem e destino possuem a mesma tecnologia, e heterogêneas quando as tecnologias são distintas; e a última classificação é quanto ao modo, sendo *live* o modo em que a VM permanece ligada durante a migração, e *non-live*, quando é necessário realizar seu desligamento.

Foram apresentadas formas de como migrar, que são as formas: manual, usando as ferramentas disponibilizadas pelos provedores de nuvens; a automatizada usando ferramentas de automatização como Ansible, que para ser completamente automatizada, necessita que o operador configure-a de modo que esta se torne capaz de realizar migrações, o que no caso de Ansible é uma funcionalidade para qual este não foi projetado para realizar; e por fim, a forma completamente automatizada, usando Kumo, por exemplo, que necessita que o operador configure credenciais para cada uma de suas contas e parâmetros específicos da migração, sendo assim uma alternativa à ferramenta Ansible. A quantidade de passos necessários para fazer a migração entre quaisquer combinações de nuvens de origem e de destino foram mostradas, seguindo-se dos trabalhos relacionados.

Quanto aos trabalhos relacionados avaliados, percebe-se que, com o uso de múltiplas nuvens, a migração de VMs volta a ser um problema de pesquisa importante, ficando evidente o desafio em realizar migrações entre nuvens, inclusive entre as públicas, que são as nuvens objetos de estudo deste trabalho. Os trabalhos relacionados permitiram auxiliar na tomada de decisões importantes na determinação do escopo da pesquisa e no projeto da arquitetura da solução de migração proposta, como a utilização do modo

de migração *non-live*, uma vez que o modo *live* é impraticável, pelo menos em nuvens públicas, por exigir acesso aos *hypervisors* dos servidores; da necessidade de baixar o disco do armazenamento da nuvem de origem e enviá-lo para o armazenamento da nuvem de destino, o que é a única solução praticável, atualmente, no cenário de nuvens públicas, onde não há compartilhamento nem de armazenamento e nem de rede entre as mesmas; e também na importância de avaliar tanto migrações realizadas entre tecnologias iguais, como das nuvens homogêneas, quanto migrações feitas entre tecnologias distintas, como as das nuvens heterogêneas.

A seguir, no Capítulo 3 é apresentada e implementada a solução proposta neste trabalho, que utiliza os conceitos que foram apresentados neste capítulo.

3 SIMPLIFICANDO A MIGRAÇÃO DE VMS ENTRE NUVENS

Este capítulo apresenta a solução proposta por este trabalho, apresentando o projeto e o desenvolvimento dos componentes que juntos facilitam o processo de migração de Máquinas Virtuais (VMs) entre nuvens. No projeto foram considerados os objetivos de homogeneização e automatização do processo de migração definidos na seção 1.2, assim como os desafios da pesquisa referentes ao gerenciamento de múltiplas credenciais (tanto para nuvens homogêneas quanto heterogêneas), utilização de múltiplas ferramentas (nuvens heterogêneas) para realização da migração e a variedade dos formatos de disco compatíveis com as nuvens (homogêneas ou heterogêneas), que foram apresentados na seção 1.3.

Considerando os desafios da pesquisa apresentados na seção 1.3, foi projetada uma arquitetura para uma solução que diminui o esforço necessário pelo operador de nuvem na realização de migrações de VMs entre duas ou mais nuvens, sejam elas homogêneas ou heterogêneas, que atende aos desafios e, por consequência, cumpre com os objetivos que foram estabelecidos e apresentados na seção 1.2. O nome **Kumo** foi dado à solução, que na língua japonesa significa **nuvem** e é representada pelo *kanji* 雲.

3.1 ARQUITETURA

Na Figura 5 é mostrada a arquitetura de Kumo, que é formada por dois componentes principais. O primeiro componente é a Interface de Linha de Comando (CLI), apresentada na seção 3.1.1, que unifica para o operador de nuvem o processo de solicitar que migrações de VMs entre quaisquer nuvens sejam realizadas. O segundo componente é o Serviço de Migração, apresentado na seção 3.1.2, que é um serviço multi-nuvem capaz de automatizar cada um dos passos que uma migração de VM pode realizar.

3.1.1 Interface de Linha de Comando (CLI)

A Interface de Linha de Comando (CLI) tem como objetivo ser uma ferramenta única e que unifica a migração de VMs para o operador de nuvem, de modo que a forma usada para solicitar uma migração de VM entre nuvens aconteça da mesma forma para quaisquer nuvens envolvidas na migração. Para isto se tornar realidade, foi preciso projetar duas funcionalidades para que esta ferramenta pudesse atingir este objetivo.

Atualmente, utilizando as ferramentas disponibilizadas pelos provedores de nuvem, o operador de nuvem precisa gerenciar manualmente as credenciais de cada provedor, para cada usuário, em cada ferramenta. Estas credenciais são bastante heterogêneas quanto a seus formatos, atributos e valores; e o uso da estratégia de multi-nuvem torna o gerenciamento destas credenciais ainda mais complexo, pois faz com que exista uma grande quantidade de credenciais que precisam ser gerenciadas pelo operador.

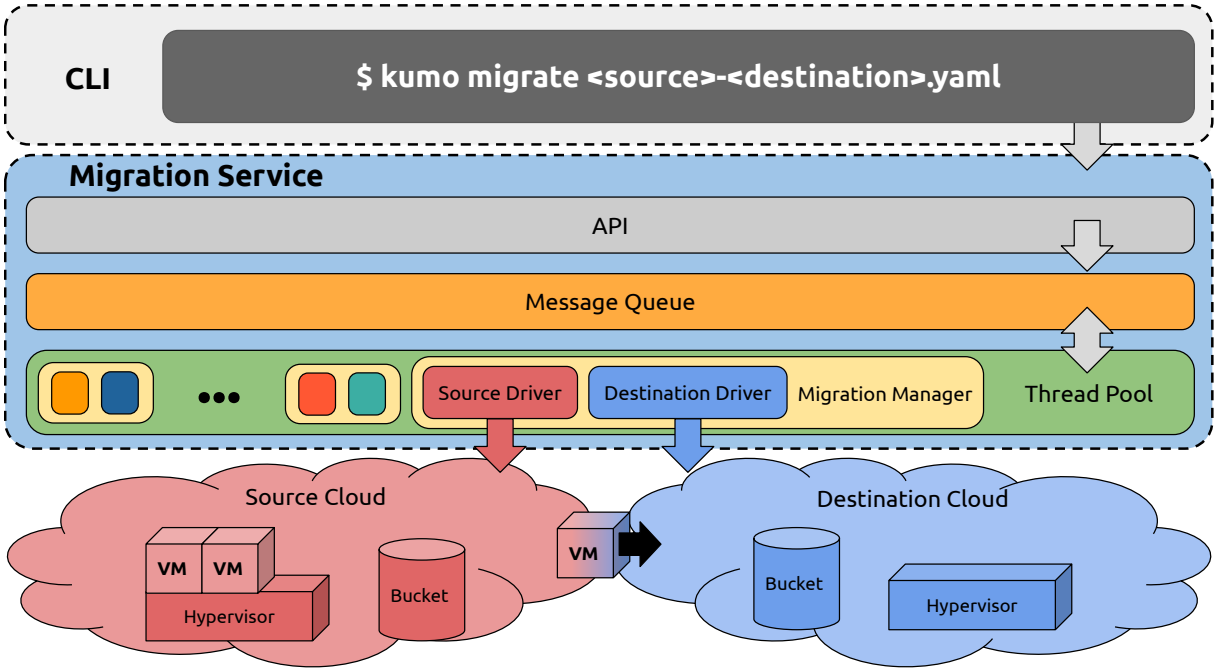


Figura 5 – Arquitetura do serviço multi-nuvem Kumo

Assim, a primeira funcionalidade da CLI é de centralizar estas credenciais na própria ferramenta, de modo que o operador cadastre todas as credenciais que poderão ser usadas para realizar migrações de VMs, permitindo que estas credenciais sejam nomeadas de modo a identificar qual o seu propósito e a qual nuvem pertencem, resolvendo assim o primeiro desafio da pesquisa definido na seção 1.3.

Outro problema encontrado nas ferramentas disponibilizadas pelos provedores, é que no cenário de multi-nuvem se torna muito complexo usar cada uma destas várias ferramentas, com seus comandos, parâmetros e valores específicos, para realizar cada um dos passos envolvidos na migração de VMs. Por este motivo, também foi adotada a estratégia de centralização, mas agora focando em centralizar e armazenar os valores relacionados à uma conta na nuvem juntamente com as credenciais daquela conta na nuvem, resolvendo assim o segundo desafio da pesquisa que é definido na seção 1.3.

Para as duas funcionalidades, de centralização de credenciais e de unificar os comandos necessários para realizar migração de VM, foi projetado um arquivo de configuração para a ferramenta chamado `kumo.conf`, como pode ser ser visto no Código 2, Este arquivo serve para armazenar informações específicas de cada uma das contas das nuvens, como credenciais e configurações específicas. Seu preenchimento necessário apenas uma vez após a instalação da CLI, onde para cada uma das contas cadastradas neste arquivo devem ser cadastradas tanto as credenciais pertencentes àquela conta, quanto valores referentes às configurações da nuvem, sendo ambas usadas durante o processo de migração das VMs.

Com todas as contas devidamente configuradas com suas credenciais e valores para os parâmetros relativos à migração, o operador precisa apenas criar um arquivo de con-

figuração da migração com base no *template* disponibilizado pela CLI de Kumo em seu repositório, como pode ser visto na Figura 6 (arquivo `gcp-aws.yaml`) ou ainda pelo Código 3. O arquivo de configuração serve para apontar tanto o sentido da migração quanto a VM a ser migrada. Para isto, é necessário indicar a conta da nuvem de origem, que é a conta na nuvem onde a VM está sendo executada (ex. Google Cloud Platform (GCP)); a conta da nuvem de destino, que é a conta para onde a VM deve passar a ser executada (ex. Amazon Web Services (AWS)); e também o identificador da VM que será migrada.

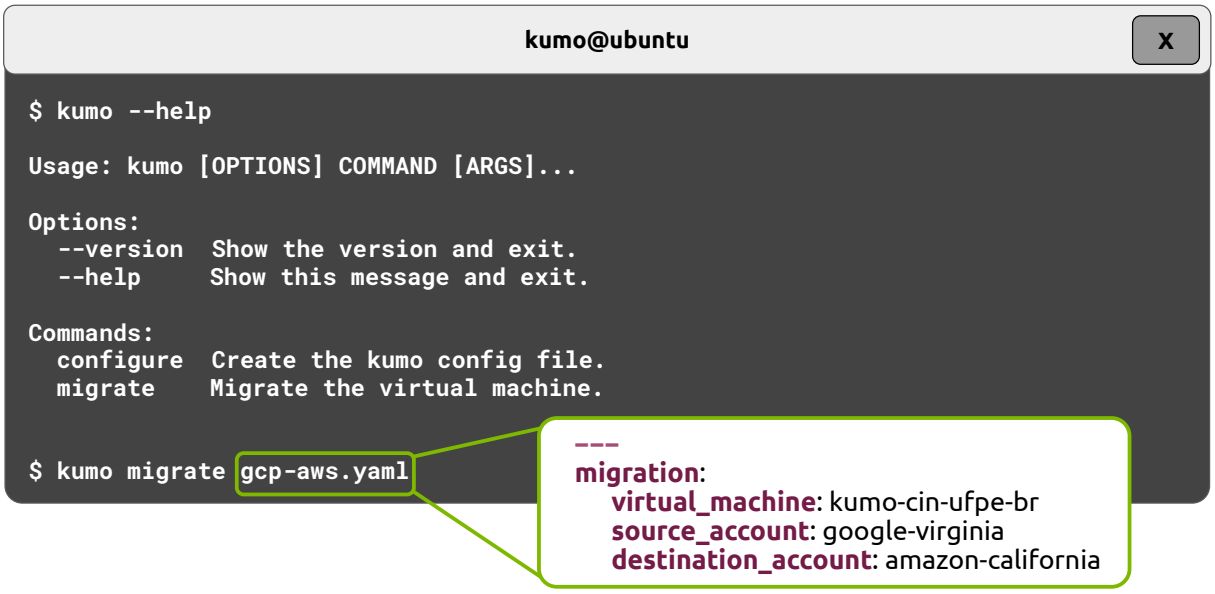


Figura 6 – Interface de Linha de Comando de Kumo

Quanto à implementação, a CLI pode ser implementada, por exemplo, utilizando a linguagem de programação Python[1] e a biblioteca Click (Command Line Interface Creation Kit)[2], biblioteca esta que tem como propósito exatamente o desenvolvimento de aplicações do tipo CLI. Para o arquivo de configuração `kumo.conf` (Código 2), que é criado pelo comando `configure`, pode ser usado o formato chave/valor para a declaração das configurações, por exemplo, como é amplamente utilizado em arquivos de configuração de sistemas UNIX. Para o arquivo de configuração da migração (como pode ser visto no Código 3), uma opção é o uso do formato YAML Ain't Markup Language (YAML), enquanto que para o *payload* (mostrado no Código 4) que é enviado da CLI para o Serviço de Migração quando o comando `migrate` é executado, uma opção de acordo é o uso do formato JavaScript Object Notation (JSON).

[1] https://www.python.org
[2] https://click.palletsprojects.com

3.1.2 Serviço de Migração (SM)

O Serviço de Migração tem como objetivo ser um serviço único e unificado, capaz de migrar uma VM de uma nuvem de origem para uma outra nuvem de destino qualquer. Para isto, conforme definido nos desafios da pesquisa na seção 1.3, é necessário que o serviço seja capaz de tratar a heterogeneidade entre os formatos de disco utilizados pelos diferentes provedores de nuvem, e também de realizar todo o processo de migração de VMs de maneira completamente automatizada.

Neste trabalho foram identificadas duas formas de realizar migração de VMs, a primeira é realizada de maneira manual, utilizando as ferramentas disponibilizadas pelos provedores de nuvem (ex. AWS CLI, AZ CLI, GCloud CLI e GSUtil CLI), enquanto que a outra maneira é a automatizada utilizando Ansible[3], que deveria ser uma forma completamente automatizada, entretanto não possui uma implementação suficiente (nativamente), ao menos na versão 2.8 que foi avaliada, para as funções e exportar discos de ou para as nuvens investigadas, necessitando que estes passos sejam realizados ou de forma manual, ou ainda modificando o código do Ansible para implementar um novo módulo com estas funcionalidades que faltam. Neste sentido, se isto fosse feito, este trabalho estaria contribuindo para propor uma extensão de Ansible.

Porém, tanto na forma manual quanto na forma automatizada via Ansible, é preciso tratar manualmente a heterogeneidade das soluções, que

- no caso manual tem como fator dificultante a heterogeneidade entre as ferramentas, parâmetros e valores envolvidos na migração, e
- no caso automatizado com Ansible, a heterogeneidade se refere a utilizar um dos diversos *playbooks*, que são pedaços de códigos que tentam padronizar a execução de algumas funcionalidades em sistemas como nuvens, que também possuem seus diversos parâmetros e valores.

Como um exemplo de heterogeneidade que afeta o processo de migração de VMs, pode ser considerada também a variedade de formatos de disco que são aceitos por algumas nuvens e por outras não. Por este motivo, a primeira funcionalidade projetada para o Serviço de Migração foi de realizar conversão automática do formato de disco, sempre que este formato não for compatível com a nuvem de destino, solucionando o terceiro desafio da pesquisa definido na seção 1.3.

Outro problema encontrado tanto na forma manual quanto na automatizada com Ansible, é que se faz necessário um esforço considerável do operador, para usar corretamente cada uma destas ferramentas com seus comandos, parâmetros e valores diferentes, em cada uma das diferentes nuvens envolvidas na migração. E ainda, a documentação disponibilizada pelos provedores de nuvem aparece como outro fator dificultante, pois os

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

provedores de nuvem detalham geralmente bem o processo de trazer uma VM de outra nuvem para sua infraestrutura, mas não documentam da mesma forma o processo de levar uma VM de sua infraestrutura para a de outro provedor.

Por este motivo, o Serviço de Migração de Kumo implementa seu próprio mecanismo de automatização, utilizando as bibliotecas para desenvolvimento de aplicações disponibilizadas pelos próprios provedores de nuvens, para todo o processo de migração de VMs no Serviço de Migração. Para tanto, define um *driver*, que é uma interface que, quando corretamente implementada, é capaz de realizar a migração de VMs de maneira automatizada entre quaisquer nuvens de Infraestrutura como Serviço (IaaS), atacando o quarto e último desafio da pesquisa definido na seção 1.3.

Quanto à implementação do Serviço de Migração conforme a Figura 5, a Interface de Programação de Aplicações (API), que desempenha a função de receber as requisições de migração realizadas pela CLI, pode ser implementada utilizando um *framework* web, como por exemplo o Flask[4]. Para a implementação da fila de mensagens (*Message Queue*), onde as requisições realizadas são armazenadas, pode ser utilizado um serviço de fila de mensagens como o RabbitMQ[5]. O *Thread Pool*, que disponibiliza *threads* para executar as migrações, é possível ser implementado via Celery[6]. Enquanto isto, o *Migration Manager*, componente responsável por orquestrar a sequência de passos da migração, não necessita de qualquer tecnologia externa à linguagem de programação. Por fim, os *Drivers* necessitam apenas utilizar as bibliotecas disponibilizadas pelos provedores de nuvem para serem implementados.

3.2 PASSOS DA MIGRAÇÃO

Com o objetivo de projetar o Serviço de Migração, precisou ser realizado um estudo nas nuvens Amazon Web Services (AWS), Microsoft Azure (AZ) e Google Cloud Platform (GCP) com o objetivo de descobrir quais eram as dependências necessárias para desenvolver uma versão inicial do serviço ainda sem automatização. Assim, como resultado da investigação e através de alguns testes preliminares, foi descoberto que seriam necessários a execução de até 9 passos para realizar uma migração de VMs entre estas nuvens.

Os passos identificados nestas execuções preliminares de Kumo podem ser vistos a seguir:

1. **Criar Bucket na Origem**: Este primeiro passo é responsável por criar um *bucket* no serviço de armazenamento da nuvem de origem para receber o disco exportado pelo Passo 5 – um *bucket* pode ser entendido como um espaço para armazenamento de objetos estruturados hierarquicamente na forma de arquivos e diretórios, como em um sistema de arquivos.

[4] https://flask.palletsprojects.com
[5] https://www.rabbitmq.com
[6] http://www.celeryproject.org

2. **Criar Bucket no Destino**: De forma semelhante ao Passo 1, este passo também cria um *bucket* no serviço de armazenamento da nuvem de destino, porém para receber o disco que será enviado para a nuvem de destino no Passo 7.

3. **Desligar VM na Origem**: Neste passo a VM tem sua execução interrompida na nuvem de origem, e quando completamente desligada, permite a execução do Passo 4 – lembrando que este trabalho foca em migrações *non-live*.

4. **Exportar Disco da Origem**: Exportar o disco é o passo onde a nuvem de origem cria uma arquivo do disco da VM e o envia para o *bucket* criado pelo Passo 1.

5. **Baixar Disco da Origem**: Com o disco exportado e armazenado no *bucket* da nuvem de origem, o disco precisa ser baixado para algum armazenamento temporário para que em seguida, se necessário, seja convertido e redimensionado durante a execução do Passo 6.

6. **Preparar Disco da Origem**: Caso o formato do disco não seja compatível com nenhum dos formatos de disco aceitos pela nuvem de destino, ou seu tamanho não esteja de acordo com o esperado pela nuvem de destino, o disco precisa ser ajustado neste passo.

7. **Enviar Disco para o Destino**: Neste passo é feito o envio do disco baixado da nuvem de origem para o *bucket* da nuvem de destino que foi criado no Passo 2.

8. **Importar Disco no Destino**: Este passo envolve carregar o disco e criar uma imagem na nuvem de destino, usando como base o disco enviado no Passo 7.

9. **Criar VM no Destino**: Por último, é criada uma VM na nuvem de destino usando a imagem criada como resultado do processo de importação do disco realizado no Passo 8.

Vale salientar que os passos identificados podem conter passos menores que servem para implementar os passos maiores, por exemplo, na AWS para importar ou exportar um disco virtual de um *bucket* ou para um *bucket*, é necessário criar uma política de autorização permitindo que o AWS Import/Export (AIE), que é o serviço da nuvem responsável por esta tarefa, esteja autorizado a acessar este *bucket*. Assim, os passos registrados neste trabalho são compreendidos como macro-passos, podendo assim incluir passos menores que juntos implementam o passo maior.

Tanto a ordem dos passos quanto as nuvens que recebem a requisição para a execução de cada passo podem ser vistos na Figura 7. A figura mostra que, no **Passo 1** e no **Passo 2** Kumo solicita que os *buckets*, tanto da nuvem de origem quanto da nuvem de destino, sejam criados. Depois, no **Passo 3**, Kumo solicita para a nuvem de origem que desligue a VM para que no **Passo 4** o disco seja exportado para o *bucket* da nuvem de

origem. No **Passo 5** Kumo baixa para si o disco que foi exportado da nuvem de origem, e se necessário, no **Passo 6** ele realiza a conversão do formato de disco ou ainda algum redimensionamento necessário. Com o disco atendendo os requisitos da nuvem de destino, no **Passo 7** o disco é enviado para o *bucket* da nuvem de destino. No **Passo 8** o disco é importado na nuvem de destino, tornando-se uma imagem de disco que pode ser usada como base para a criação de novos discos que serão criados e associados às novas VMs criadas. Por fim, no **Passo 9**, Kumo solicita à nuvem de destino que crie uma nova VM usando aquela imagem criada no passo anterior como base para a criação de um novo disco para esta VM.

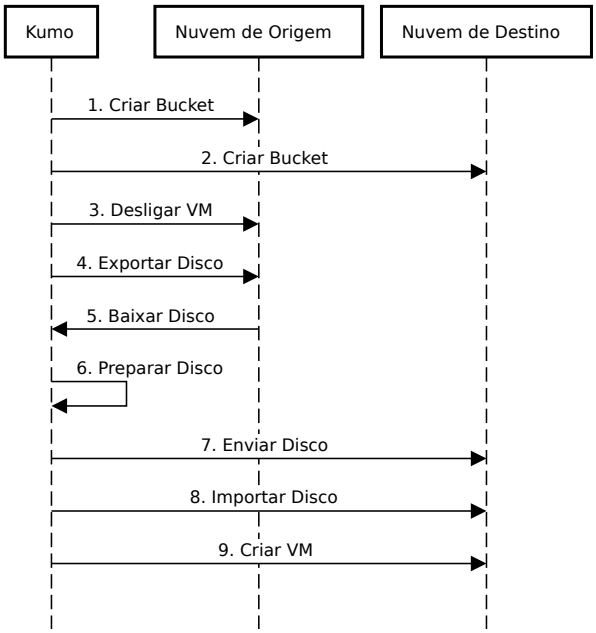


Figura 7 – Passo a passo da migração de VMs utilizando Kumo

Também durante a realização dos testes preliminares, foi percebido que nem todas as nuvens necessitavam realizar todos os passos necessários no processo de migração. Como pode ser visto na Tabela 4, dos 9 passos que foram identificados, nem todos são executados em todas as migrações. Como exemplo, observa-se que sempre que a nuvem de origem da migração é a Microsoft Azure (AZ), a execução do passo 1 não é realizada. Isto acontece pois o processo de exportação do disco da AZ, que é realizado no passo 4, dispensa o uso de um *bucket*, precisando apenas, para exportar o disco, realizar a criação de um link para que aquele disco seja baixado, dispensando então o uso do *bucket*.

Outra observação referente à Tabela 4 é que o passo 6 é executado apenas quando a nuvem de destino é a AZ. Isto acontece devido ao requisito de importação da AZ, que quando executado no passo 7, exige que o disco que será importado tenha um tamanho total que seja múltiplo de 1 Mebibyte (MiB). Logo, em casos como este, é necessário realizar um redimensionamento no disco durante a execução do passo 6, para que o disco possa ser corretamente importado quando o passo 7 for executado na nuvem de destino.

Tabela 4 – Relação entre as migrações e os passos nelas executados

Migração \ Passo	1	2	3	4	5	6	7	8	9
AWS - AWS	✓	✓	✓	✓	✓	✗	✓	✓	✓
AWS - AZ	✓	✓	✓	✓	✓	✓	✓	✓	✓
AWS - GCP	✓	✓	✓	✓	✓	✗	✓	✓	✓
AZ - AWS	✗	✓	✓	✓	✓	✗	✓	✓	✓
AZ - AZ	✗	✓	✓	✓	✓	✗	✓	✓	✓
AZ - GCP	✗	✓	✓	✓	✓	✗	✓	✓	✓
GCP - AWS	✓	✓	✓	✓	✓	✗	✓	✓	✓
GCP - AZ	✓	✓	✓	✓	✓	✓	✓	✓	✓
GCP - GCP	✓	✓	✓	✓	✓	✗	✓	✓	✓

3.3 NUVENS PESQUISADAS

Com o objetivo de avaliar a solução proposta neste trabalho, foi necessário realizar a implementação dos componentes que juntos compõem a solução. Como diferencial deste trabalho em relação a outros trabalhos que tratam de migração de VMs, que geralmente utilizam simuladores, neste trabalho foram utilizados provedores de nuvem públicas reais e comerciais. O critério utilizado para escolher quais seriam as nuvens mais indicadas para serem integradas à solução, foi que deveriam estar entre as nuvens dos provedores de nuvem pública mais utilizados da atualidade.

Conforme mostrado na Figura 8, segundo a pesquisa anual realizada por FLEXERA (2019) referente às nuvens públicas mais adotadas por empresas de Tecnologia da Informação e Comunicação (TIC), a nuvem pública mais adotada atualmente é a AWS com 61%, seguida pela AZ com um total de 52% de adoção, enquanto que a GCP aparece na terceira posição com 19% de adoção. Também na figura, observa-se um crescimento generalizado na adoção da computação em nuvem, na comparação entre 2019 e 2018, com uma pequena queda apenas em relação à AWS.

3.3.1 Serviços necessários para a migração de VMs

Para implementar os passos definidos na seção 3.2, na nuvem Amazon Web Services (AWS) foi realizada uma pesquisa para descobrir quais serviços oferecidos pela nuvem precisavam ser utilizados para implementar estes passos. Assim, por este trabalho se tratar de migração de VMs, o serviço mais fundamental para ser utilizado é o serviço de

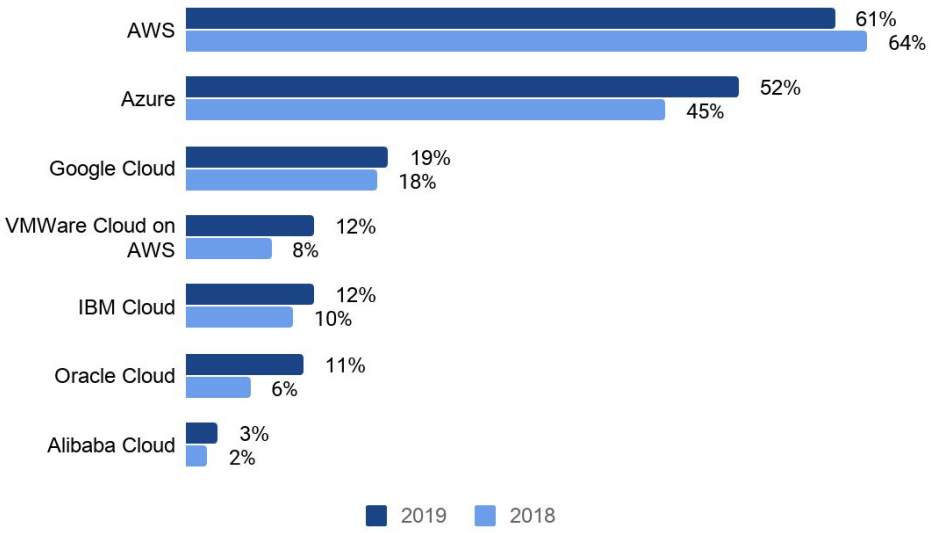


Figura 8 – Nuvens públicas mais usadas na atualidade, segundo FLEXERA (2019)

computação, Elastic Compute Cloud (EC2). O EC2 é um serviço web que oferece serviços de provisionamento de VMs e também gerencia os discos virtuais utilizados pelas VMs, enquanto que para a AZ o serviço correspondente é o Azure Compute (AC) e para a GCP é o serviço Google Compute Engine (GCE).

Outro tipo de serviço identificado que está diretamente associado à migração de VMs, são os serviços de armazenamento de arquivos. Na migração de VMs, o comportamento mais comum para quando o disco de uma VM é exportado ou importado de uma nuvem, é que o disco seja enviado para um *bucket* para ser armazenado em um serviço de armazenamento, como por exemplo, na AWS o serviço Simple Storage Service (S3), na AZ o Azure Blob Storage (ABS) e na GCP o Google Cloud Storage (GCS).

Outros serviços que precisaram ser utilizados na AWS foram o serviço AWS Import/Export (AIE) e o serviço Identity and Access Management (IAM). O serviço AIE é o serviço capaz de importar um arquivo de disco contido em um *bucket* do serviço de armazenamento S3 e transformá-lo em um imagem de disco que pode ser utilizada como base para criação de novos discos virtuais durante o lançamento de novas instâncias de VMs, ou ainda é capaz de realizar a operação contrária, de exportar um disco que está sendo usado em uma VM e transformá-lo em um arquivo de disco em um dado *bucket* no S3. Mas para que o serviço AIE tenha permissão de acessar o *bucket* de uma conta de usuário na AWS, é necessário que seja criada uma política de autorização que estabeleça uma relação de confiança entre o serviço AIE e o S3. Para isto, o serviço necessário para estabelecer esta confiança é o serviço IAM.

Para a AZ, além dos serviços citados, apenas mais um outro serviço é utilizado para realizar migrações de VMs, que é o serviço Azure Virtual Network (AVN), sendo necessário para criar uma interface de rede virtual para permitir conectividade entre a VM

instanciada no passo 9 da migração e outras VMs na conta do usuário. Já na GCP, também é utilizado mais um serviço para permitir que VMs sejam migradas, tanto da GCP, quanto para a GCP, que é o serviço Google Cloud Build (GCB), que de forma similar ao que o serviço AIE da AWS faz, realiza tanto a exportação de um disco da nuvem para um *bucket*, quanto importa um disco para a nuvem a partir de um arquivo de disco armazenado em um *bucket*.

Em resumo, todos os serviços utilizados por cada uma das 3 nuvens, para cada um dos 9 passos da migração, estão reunidos na Tabela 5, onde é possível observar que para o passo 6 da AWS e da GCP, e para o passo 1 da AZ não existem serviços associados, pois estes passos não são executados. Entretanto, para que as migrações para a AZ sejam possíveis, Kumo assume a responsabilidade de executar o passo 6 e assim tornar a AZ interoperável com as demais nuvens, no sentido de ser capaz de receber uma VM de uma outra nuvem origem.

Tabela 5 – Serviços responsáveis por realizar cada passo das migrações

Passo / Nuvem	1	2	3	4	5	6	7	8	9
AWS	S3; IAM	S3; IAM	EC2	EC2; AIE	S3	✗	S3	EC2; AIE	EC2
AZ	✗	ABS	AC	AC	ABS	✗*	ABS	AC	AC; AVN
GCP	GCS	GCS	GCE	GCE; GCB	GCS	✗	GCS	GCE; GCB	GCE

(*) Passo implementado por Kumo através dos comandos `"convert"` e `"resize"` da ferramenta QEMU Disk Image Utility.

3.4 IMPLEMENTAÇÃO

Aqui são discutidas as implementações da CLI e do Serviço de Migração (SM) para, respectivamente, tratar das credenciais e das várias ferramentas para realizar cada um dos passos envolvidos na migração de VMs, e da heterogeneidade dos formatos de disco utilizados pelos diferentes provedores de nuvem. Os artefatos gerados nesta seção estão disponíveis em seus respectivos repositórios no GitHub, sendo para o Kumo CLI o link `"https://github.com/marcusrafael/kumo-cli"`, e para o Serviço de Migração o link `"https://github.com/marcusrafael/kumo"`. Ambos artefatos são *open source* e estão sobre a licença MIT, que permite uso comercial, distribuição, modificação e uso privado, desde que mantida a licença e a nota de *copyright* dos arquivos.

3.4.1 Implementando a Interface de Linha de Comando (CLI)

A implementação da CLI se deu através da linguagem de programação Python e da biblioteca Click, que tem como objetivo facilitar o desenvolvimento deste tipo de aplicações do tipo CLI na linguagem Python. No código Python foram definidas duas funções chamadas `configure` e `migrate`, onde o `configure` é a função responsável por realizar a criação do arquivo de configuração `kumo.conf` da ferramenta, e o `migrate` que é a função responsável por ler informações do arquivo `kumo.conf` e a partir destas informações ler o arquivo de configuração da migração `gcp-aws.yaml` (representado pelo Código 3) para obter as informações necessárias para solicitar ao Serviço de Migração que realize a migração.

A função da biblioteca Click é utilizar sua capacidade de realizar reflexão computacional (MAES, 1987) para interpretar o próprio código das funções `configure` e `migrate`, e assim identificar quais nomes os comandos e os parâmetros devem ter quando o operador instalar a CLI de Kumo.

Para implementar a função `configure`, que cria o `kumo.conf`, que é o arquivo de configuração para centralização das credenciais e outras configurações específicas de cada nuvem, foi utilizada a biblioteca ConfigParser do Python. Assim, quando o operador instalar a CLI de Kumo e executar o comando `configure`, será criado um modelo para ajudar este operador a cadastrar credenciais de acesso e configurações específicas para cada uma das nuvens que deseja realizar migração de VMs. Um recorte de um arquivo `kumo.conf` criado pelo comando `configure` pode ser visto no Código 2.

Outro arquivo necessário para realizar a migração de VMs entre nuvens é o arquivo de configuração da migração (Código 3). Este arquivo serve para indicar dentre as diversas credenciais cadastradas no arquivo de configuração e outras configurações, quais credenciais serão usadas para as contas de nuvem e de destino, e ainda qual o nome da VM que deverá ser migrada entre estas duas contas. No Código 3, pode ser visto um exemplo de acordo com as configurações definidas anteriormente no `kumo.conf`, representando no arquivo de configuração da migração na linguagem YAML que pode ser escrito por um operador de nuvem, informando que a VM que será migrada tem o nome de `kumo-cin-ufpe-br`, que a conta da nuvem de origem da migração é `google-virginia` e a conta de destino da migração é `amazon-california`, conforme declarado no arquivo de configuração do Código 2.

Para implementar o comando `migrate`, assim como no `configure`, foi necessário utilizar a biblioteca ConfigParser e também a biblioteca do YAML para Python. Assim, quando o operador executa o comando (ex. `prompt> kumo migrate gcp-aws.yaml`), o arquivo de configuração `kumo.conf` é lido através da biblioteca ConfigParser, enquanto o arquivo de configuração da migração é lido pela biblioteca YAML. Com as informações

```
[kumo]
url = https://kumo.com.br:5000

[google-virginia] # Origem
cloud = google
bucket = kumo-google
zone = us-east4-a  # Virginia
system = ubuntu-1804
machine_type = n1-standard-1
type = service_account
project_id = tough-timing-248914
private_key_id = ce80236c9e9f1fb7813ae00f9f3c79bfdb54e65c
client_email = mrxl@tough-timing-248914.iam.gserviceaccount.com
client_id = 117804550085589546156
auth_uri = https://accounts.google.com/o/oauth2/auth
token_uri = https://oauth2.googleapis.com/token
auth_provider_x509_cert_url = https://www.googleapis.com/oauth2/v1/certs
client_x509_cert_url = https://www.googleapis.com/robot/v1/metadata/x509/
    mrxl40tough-timing-248914.iam.gserviceaccount.com
private_key = -----BEGIN PRIVATE KEY-----
ABCDEFGHIJKLMNOPQRSTUVWXYZ
-----END PRIVATE KEY-----

[amazon-california] # Destino
cloud = amazon
bucket = kumo-amazon
region = us-west-1
availability_zone = us-west-1a  # California
instance_type = t2.small
aws_access_key_id = ABCDEFGHIJKLMNOPQRSTUVWXYZ
aws_secret_access_key = ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Código 2 – Exemplo de um arquivo `kumo.conf`

```
---
migration:
  virtual_machine: kumo-cin-ufpe-br
  source_account: google-virginia
  destination_account: amazon-california
```

Código 3 – Instância de um arquivo de configuração da migração – `gcp-aws.yaml`

obtidas nestes arquivos, é gerado um *payload* JSON que é enviado via Hyper Text Transfer Protocol Secure (HTTPS) para o Serviço de Migração, com todas as informações necessárias para realizar a migração entre as duas nuvens. Um exemplo deste *payload* JSON que é gerado pode ser visto no Código 4.

Vale ressaltar que para cada migração de VM que precise ser realizada, é necessária a execução do comando `migrate`, uma vez por migração. Logo, para casos onde uma mesma VM precise ser migrada para mais de uma nuvem, é necessário executar o comando de migração uma vez para cada migração que precise ser realizada, ou seja, no momento este trabalho se limita a permitir migrações de VMs de 1 nuvem de origem para 1 nuvem de destino $(1:1)$, e não para N nuvens ao mesmo tempo $(1:N)$.

```
{
  "source_account":{
    "cloud":"google",
    "bucket":"kumo-google",
    "zone":"us-east4-a",
    "system":"ubuntu-1804",
    "machine_type":"n1-standard-1",
    "type":"service_account",
    "project_id":"tough-timing-248914",
    "private_key_id":"ce80236c9e9f1fb7813ae00f9f3c79bfdb54e65c",
    "client_email":"mrxl@tough-timing-248914.iam.gserviceaccount.com",
    "client_id":"117804550085589546156",
    "auth_uri":"https://accounts.google.com/o/oauth2/auth",
    "token_uri":"https://oauth2.googleapis.com/token",
    "auth_provider_x509_cert_url":"https://www.googleapis.com/oauth2/v1/certs",
    "client_x509_cert_url":"https://www.googleapis.com/robot/v1/metadata/x509/
    mrxl40tough-timing-248914.iam.gserviceaccount.com",
    "private_key":"-----BEGIN PRIVATE KEY-----\nABCDEFGHIJKLMNOPQRSTUVWXYZ\n
    -----END PRIVATE KEY-----"
  },
  "destination_account":{
    "cloud":"amazon",
    "bucket":"kumo-amazon",
    "region":"us-west-1",
    "availability_zone":"us-west-1a",
    "instance_type":"t2.small",
    "aws_access_key_id":"ABCDEFGHIJKLMNOPQRSTUVWXYZ",
    "aws_secret_access_key":"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
  },
  "virtual_machine":"kumo-cin-ufpe-br"
}
```

Código 4 – Exemplo de *payload* JSON enviado para o Serviço de Migração

3.4.2 Implementando o Serviço de Migração (SM)

Para ser capaz de receber o *payload* JSON que é enviado pela CLI na execução do comando `migrate`, e como pode ser visto na Figura 5 (arquitetura – seção 3.1), quem recebe esta requisição feita pela CLI no Serviço de Migração (*Migration Service*) é a API. Esta API, assim como a CLI, é implementada em Python e usa o *framework* web Flask. Para realizar a migração, é esperada uma requisição web via o método POST, que quando recebida pela API tem o *payload* JSON recebido cadastrado na fila de mensagem (*Message Queue*) do serviço RabbitMQ. A comunicação entre o Kumo CLI e o Serviço de Migração é feita de modo assíncrono com base no modelo *publish/subscribe*.

O RabbitMQ recebe a requisição com o *payload* e notifica o *worker* do Celery que é um inscrito (modelo *publish/subscribe*) desta fila de mensagem. Quando este *worker* consome o *payload*, aloca uma *thread* de seu *Thread Pool* e instancia o *Migration Manager* nesta *thread*. Em seguida, o *Migration Manager* utiliza as informações recebidas para saber quais *drivers* de origem e de destino deverão ser carregados, e assim a migração é iniciada, utilizando em cada um dos passos as informações necessárias, que foram enviadas pelo operador através da CLI, para conectar e executar as devidas ações nas nuvens envolvidas na migração.

Porém, para que Kumo consiga realizar os passos da migração em cada uma das nuvens, faz-se necessária a definição de uma interface padronizada, para que os *drivers*

possam implementar, e assim serem capazes de realizar, os 9 passos definidos na seção 3.2, quando necessário. Vale ressaltar que em cada migração dois *drivers* são usados, e por este motivo para executar os 9 passos de uma migração com Kumo, são invocados 4 métodos do *driver* de origem e 5 métodos do *driver* de destino para realizar a migração. Os Códigos 5, 6 e 7 mostram as assinaturas dos 8 métodos que um *driver* precisa implementar para integrar uma nova nuvem a Kumo.

Analisando o Código 5, é possível ver que o primeiro método é o `create_bucket`, responsável por implementar ou o passo 1 ou o passo 2 referentes à criação de um *bucket*, sendo o fator que determinará se o método implementa o passo 1 ou 2 é se este *driver* está sendo usado como *driver* de origem ou *driver* de destino da migração, onde se for o *driver* de origem este método é executado como sendo passo 1, e se for o *driver* de destino é executado como passo 2.

```
1  import abc
2
3
4  class BaseDriver(metaclass=abc.ABCMeta):
5      """A base driver class for implementing Kumo drivers."""
6
7      @abc.abstractmethod
8      def create_bucket(self):
9          """Creates a bucket on source or destination clouds.
10
11         This method creates a bucket on source cloud if the
12         driver is being used as source driver. Otherwise creates
13         the bucket on destination cloud.
14
15         This method is often executed synchronously in API
16         requests to the cloud.
17         """
```

Código 5 – Interface base para os *drivers* – Método: `create_bucket`

O Código 6, o método `stop_server` é responsável por implementar o passo 3 da migração, referente à desligar a VM para que possa ser migrada. O método `export_disk` implementa o passo 4 relativo à exportar o disco da nuvem de origem, enquanto que o passo 5 é implementado pelo método `download_disk`, que baixa o disco da nuvem de origem para Kumo. O passo 6 é implementado pelo método `prepare_disk` e, dentre os 3 *drivers* implementados neste trabalho, foi apenas implementado no *driver* da AZ, que como mostrado na Tabela 5, foi implementando usando uma solução própria implementada em Kumo, que implementou as instruções passadas pela documentação da AZ, no sentido de converter o disco do formato Virtual Hard Disk (VHD), especificado por Microsoft (2006), para o formato RAW Format Hard Drive (RAW), redimensioná-lo para atender o requisito de ter um tamanho total que seja múltiplo de 1 MiB, e em seguida converter o disco de volta para VHD.

```
    @abc.abstractmethod
    def stop_server(self):
        """Stops a given virtual machine on the source cloud.

        This method stops a virtual machine on source cloud
        before starting its exporting process.

        This method is often executed synchronously in API
        requests to the cloud.
        """

    @abc.abstractmethod
    def export_disk(self):
        """Exports a virtual machine's disk from source cloud.

        This method performs the process of disk exporting
        from the source cloud. Commonly, exporting the
        disk to the bucket created by the 'create_bucket'
        method.

        This method is often executed synchronously in API
        requests to the cloud.
        """

    @abc.abstractmethod
    def download_disk(self):
        """Downloads the disk from source cloud to Kumo.

        This method downloads the disk that was exported
        by method 'export_disk' and stores it in Kumo.

        This method is often executed synchronously in API
        requests to the cloud.
        """

    @abc.abstractmethod
    def prepare_disk(self):
        """Prepares the disk to destination cloud.

        This method, if needed, performs changes on the disk
        like resizing it or even convert the disk to a new
        format file. Currently, this method is only used by
        Microsoft Azure driver for Kumo.

        This method is always executed synchronously locally
        in Kumo.
        """
```

Código 6 – Interface base para os *drivers* – Métodos: `stop_server`, `export_disk`, `download_disk`, `prepare_disk`

No Código 7, o método `upload_disk` implementa o passo 7 referente ao envio do disco preparado por Kumo para a nuvem de destino, enquanto que os métodos `import_disk` e `create_server` são responsáveis, respectivamente, por importar o disco enviado para o *bucket* da nuvem de destino (passo 8) e instanciar uma nova VM no destino (passo 9).

```
    @abc.abstractmethod
    def upload_disk(self):
        """Uploads the disk to destination cloud.

        This method uploads the disk from Kumo to the
        destination cloud. At this point, the disk
        should meet all the requirements from the
        destination cloud.

        This method is often executed synchronously in API
        requests to the cloud.
        """

    @abc.abstractmethod
    def import_disk(self):
        """Imports a virtual machine's disk to destination cloud.

        This method performs the process of disk importing
        on the destination cloud. Commonly, importing the
        disk from the bucket created by the 'create_bucket'
        method.

        This method is often executed synchronously in API
        requests to the cloud.
        """

    @abc.abstractmethod
    def create_server(self):
        """Creates a new virtual machine using the disk imported.

        This method instantiates a new virtual machine using,
        as a disk, the disk image created on the destination
        cloud by the import disk process.

        This method is often executed synchronously in API
        requests to the cloud.
        """
```

Código 7 – Interface base para os *drivers* – Métodos: `upload_disk`, `import_disk`, `create_server`

Implementados os *drivers* para AWS, AZ e GCP, foi considerado o uso de uma forma de instalação para o Serviço de Migração de modo que permitisse que os vários componentes de sua arquitetura fossem rapidamente iniciados de maneira automatizada, o que seria proveitoso durante o processo de avaliação da solução, que precisou que Kumo fosse instalado em cada uma das nuvens de origem durante a execução de cada migração. Para isto, como pode ser visto na Figura 9 que representa a topologia de Kumo, foi utilizado o software para provisionamento de *containers* Docker[7], que permite a criação de uma imagem de *containers* com todas as dependências e todos os componentes do Serviço de Migração instalados e configurados.

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

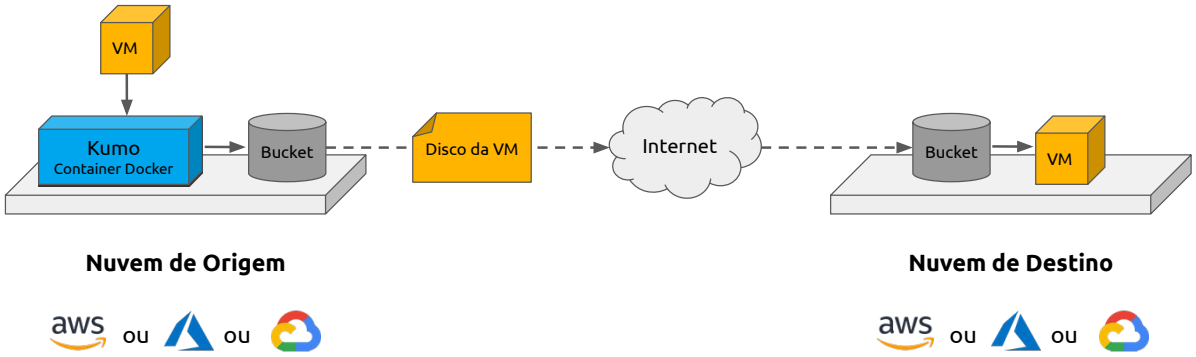


Figura 9 – Topologia de Kumo

O Código 8 mostra o Dockerfile responsável por realizar a instalação das dependências de Kumo, tanto no que se diz respeito às dependências do serviço (linhas 5-13), quanto às dependências dos *drivers* (linhas 16-27), e ainda à inicialização de todos os subcomponentes do serviço (linhas 36-38).

```
1  FROM ubuntu:latest
2  MAINTAINER marcusrafael mrxl@cin.ufpe.br
3
4  # Kumo Migration Service dependencies
5  RUN apt update && \
6      apt install -y curl && \
7      apt install -y python3-dev && \
8      apt install -y python3-pip && \
9      apt install -y qemu-utils && \
10     apt install -y rabbitmq-server && \
11     pip3 install celery && \
12     pip3 install flask && \
13     pip3 install gunicorn
14
15 # AWS driver dependency
16 RUN pip3 install boto3
17
18 # AZ driver dependencies
19 RUN pip3 install azure-mgmt-compute && \
20     pip3 install azure-mgmt-network && \
21     pip3 install azure-storage
22
23 # GCP driver dependencies
24 RUN curl -sSL https://sdk.cloud.google.com | bash && \
25     pip3 install google-api-python-client && \
26     pip3 install google-cloud-storage && \
27     pip3 install oauth2client
28
29 ENV PATH $PATH:/root/google-cloud-sdk/bin
30 ENV PYTHONUNBUFFERED=1
31
32 COPY . /kumo
33 WORKDIR /kumo
34
35 # Start RabbitMQ, Celery Thread Pool and Kumo API
36 CMD service rabbitmq-server start && \
37     celery -A kumo.conductor.conductor worker --loglevel=info & \
38     gunicorn -b 0.0.0.0:5000 kumo.api.api:app
```

Código 8 – Arquivo Dockerfile do Serviço de Migração

3.5 INSTRUMENTAÇÃO

Com o objetivo de **avaliar a execução do processo de migração de máquinas virtuais em ambientes multi-nuvem**, conforme definido na seção 1.2, foi realizada uma instrumentação no código para definir quanto tempo Kumo leva para executar cada um dos passos da migração, que são implementados pelos métodos definidos na seção 3.4.2 e com isto poder chegar aos valores resultantes para a métrica de Tempo Total de Migração (TTM), ou *Total Migration Time*, proposta em Zhang et al. (2018), referente à execução de cada uma das migrações, sejam nos cenários homogêneos ou ainda nos cenários heterogêneos, de modo a concluir quais foram as melhores combinações entre as nuvens de origem e de destino.

No Código 9 é possível ver o *decorator* (como são chamadas as *annotations* na linguagem Python) que é usado para capturar o tempo total de execução. Na linha 4 é capturado o tempo inicial, que se refere ao tempo antes da execução de um método, como por exemplo o `create_server`. Na linha 9, o método é de fato invocado, enquanto que na linha 10 o tempo é novamente capturado. Para obter o tempo total de execução deste método, na linha 13 é calculada a diferença dos tempos para que o resultado possa ser impresso nos *logs* de Kumo no formato de segundos, o que é feito pelas linhas 14 e 15.

```
1  def audit(func):
2      @wraps(func)
3      def wrapper(*args, **kwargs):
4          initial_time = datetime.datetime.now()
5          class_name = args[0].__class__.__name__
6          function_name = func.__name__
7          print("{}:{}:{}:started".format(
8              class_name, function_name, initial_time))
9          result = func(*args, **kwargs) # Decorated method call.
10         final_time = datetime.datetime.now()
11         print("{}:{}:{}:ended".format(
12             class_name, function_name, final_time))
13         total_time = final_time - initial_time
14         print("{}:{}:{}:total".format(
15             class_name, function_name, total_time.total_seconds()))
16         return result
17     return wrapper
```

Código 9 – Decorator para captura dos tempos de execução dos métodos

3.6 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Neste capítulo foi apresentada uma solução chamada Kumo proposta por este trabalho, para realizar migração de VMs entre multi-nuvens homogêneas ou heterogêneas de maneira uniforme e automatizada. A arquitetura da solução foi apresentada com seus dois principais componentes, primeiramente a Interface de Linha de Comando (CLI), que é

responsável por centralizar as credenciais e os valores que são usados para a migração, oferecendo uma forma única e unificada de realizar migração entre duas nuvens de maneira facilitada. Em seguida, o outro componente, chamado de Serviço de Migração de VMs, foi apresentado, focando na conversão de formatos e redimensionamento de disco, e na automatização de todo o processo de migração.

Também neste capítulo foram apresentados os 9 passos descobertos para o processo de migração de VMs, ficando evidente que, dependendo da combinação de nuvem de origem e de destino, podem ser necessários realizar apenas 7 ou 8 destes 9 passos. Em seguida, foi descrito de maneira geral como podem ser implementados os *drivers* para migração, considerando as nuvens mais adotadas por empresas que utilizam computação na nuvem, a AWS, a AZ e a GCP, sendo detalhados quais os serviços que precisam ser usados para implementar os passos da migração nestas nuvens.

Foi apresentada a interface base com os métodos que devem ser implementados por um *driver*, para que qualquer nuvem de IaaS possa ser integrada a Kumo, e assim permitir que VMs possam ser migradas de uma nuvem qualquer para outra nuvem, ou desta nova nuvem para qualquer outra nuvem previamente integrada a Kumo. Em resumo, Kumo surge como uma solução que permite a integração de novas nuvens, sendo agnóstico quanto às nuvens envolvidas no processo de migração e que exige um esforço reduzido para realizar migrações de VMs quando comparado a mecanismos manuais de migração, como as ferramentas disponibilizadas pelos provedores de nuvem, ou ainda automatizados, como no caso de Ansible.

Por fim, a instrumentação que é usada para medir o tempo de execução de cada um dos métodos dos *drivers* é mostrada. O *decorator* apresentado no Código 9 mostra como é realizada a captura do tempo antes e depois da execução dos métodos da interface `BaseDriver` (que podem ser vistos nos Códigos 5, 6 e 7), por fim a diferença destes dois tempos é utilizada para determinar o tempo de execução de cada passo, e consequentemente o TTM, que é obtido através da soma dos tempos de execução de cada um dos passos, conforme apresentado na Equação 4.1.

4 AVALIAÇÃO

Como o objetivo de verificar o desempenho das migrações para cada combinação das variáveis, nuvem de origem e nuvem de destino, esta seção apresenta a avaliação dos cenários executados, realizando análises sobre os seguintes aspectos:

- A análise das migrações por inteiro, que determina quais são os melhores cenários para as migrações, entre nuvens homogêneas e heterogêneas, através da manipulação das variáveis nuvem de origem e de destino, para as 3 nuvens pesquisadas;
- A análise do passo a passo, que permite realizar um comparativo entre os tempos de execução dos passos, para cada cenário definido na seção 4.1, identificando quais passos são executados, e quais destes passos mais têm influência nos resultados obtidos na análise das migrações por inteiro;
- A conclusão das análises, que possibilita atestar quais foram os melhores e piores cenários entre as migrações homogêneas e heterogêneas, bem como quais passos foram determinantes para chegar nestes resultados, apresentando também uma porcentagem de quão significativos foram os passos na migração.

Para avaliar a solução implementada na seção 3.4, foi preciso definir diversos cenários de migração para avaliar o tempo necessário para realizar migrações de Máquinas Virtuais (VMs) utilizando Kumo, lembrando que para tanto foi adotada a métrica de Tempo Total de Migração (TTM), ou *Total Migration Time*, proposta em Zhang et al. (2018). A métrica TTM reflete para a comunidade e indústria que o tempo para se realizar uma migração é um fator determinante para a escolha da nuvem de destino, que caso o usuário opte por migrar, deverá admiti-lo, e caso não o admita, uma opção será realizar a recriação do ambiente da VM na nuvem de destino, o que significa criar uma nova VM na nuvem de destino, migrar os dados da antiga VM para a nova, e ainda reinstalar as aplicações.

Para a execução de migrações de VMs, foram implementados a Interface de Linha de Comando (CLI) e o Serviço de Migração, e nesta seção são avaliadas tanto as migrações homogêneas quanto as heterogêneas, através da realização de migrações de VMs entre as nuvens públicas apontadas na pesquisa feita por FLEXERA (2019) como as nuvens públicas mais utilizadas na atualidade: Amazon Web Services (AWS), Microsoft Azure (AZ) e Google Cloud Platform (GCP), como visto na seção 3.3.

4.1 PLANEJAMENTO DAS MIGRAÇÕES

No planejamento das migrações foram definidos 9 cenários de migração de VMs entre nuvens, onde 3 destes cenários são homogêneos e outros 6 são heterogêneos, referentes às nuvens. Para as **migrações homogêneas** os cenários definidos foram:

1. de AWS para AWS,
2. de AZ para AZ e
3. de GCP para GCP.

Para as **migrações heterogêneas**, os demais 6 cenários de migração de VMs entre nuvens foram definidos da seguinte maneira:

4. de AWS para AZ,
5. de AWS para GCP,
6. de AZ para AWS,
7. de AZ para GCP,
8. de GCP para AWS e
9. de GCP para AZ.

Definidos os cenários de migração, foi necessário também definir qual seria a **configuração utilizada pelas VMs onde Kumo seria instalado** em cada uma das nuvens para realizar as migrações, e ainda a configuração das VMs que seriam criadas em cada nuvem e migradas entre a origem e o destino. A VM utilizada para a instalação de Kumo usou o Sistema Operacional (SO) Ubuntu 18.04 LTS com a seguinte configuração: 1 vCPU, 2 Gigabytes (GB) de memória e 100 Gibibytes (GiB) de disco para AWS e AZ, ou 107 GB para a GCP – vale lembrar que 1 GiB equivale a 1,07 GB. Enquanto isto, para as **VMs que seriam migradas entre as nuvens**, foi utilizado o sistema Ubuntu 18.04 LTS e a seguinte configuração: 1 vCPU, 2 GB de memória e 30 GiB de disco para AWS e AZ, ou 32 GB para GCP. A escolha do tamanho do disco foi feita considerando os tamanhos recomendados por cada um dos três provedores de nuvem, que foram de 8 GiB para a AWS, 30 GiB para a AZ e 10 GB para a GCP, optando-se assim por nivelá-los pelo maior tamanho recomendado (30 GiB $\approx$ 32 GB).

A Figura 10 mostra que existem *data centers* dos provedores de serviços de nuvens AWS, AZ e GCP nos mesmos estados, tanto na região leste dos Estados Unidos (Virgínia), quanto na região oeste (Califórnia). Também na figura pode ser visto que neste trabalho, para os experimentos, **a origem da migração foi fixada na Virgínia**, enquanto **o destino foi definido como sendo a Califórnia**. Este sentido (Virgínia $\rightarrow$ Califórnia) foi escolhido considerando um cenário de **expansão de operações**, onde uma empresa percebe que, pela demanda de acesso dos usuários na região oeste dos Estados Unidos, existe a necessidade de expandir suas operações, que atualmente se concentram na Virgínia, também para a Califórnia. Assim, visando minimizar possíveis atrasos no acesso à

uma aplicação concentrada na Virgínia, uma opção é realizar a migração seletiva de VMs para a nuvem na Califórnia (BITTENCOURT et al., 2015).

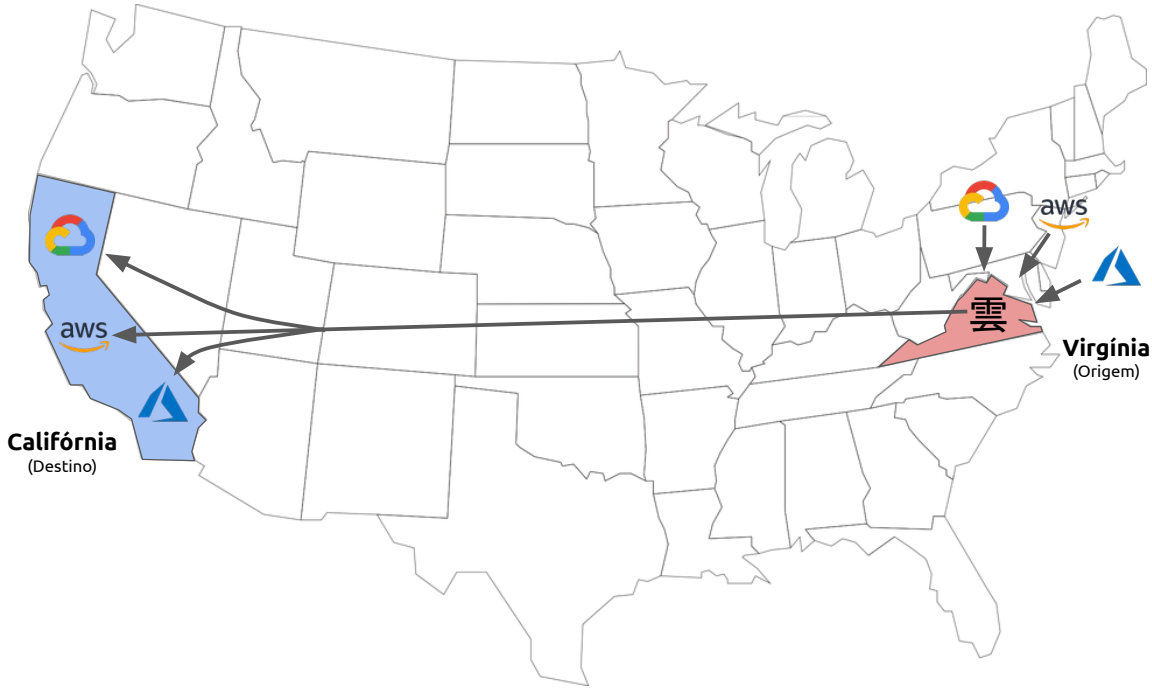


Figura 10 – Mapa das migrações realizadas da Virgínia para a Califórnia

Diante do cenário de que seriam avaliadas as migrações realizadas no sentido Virgínia → Califórnia, Kumo foi instalado sempre nas nuvens de origem. Embora a **instalação de Kumo na origem** tenha sido adotada neste trabalho, uma outra opção praticável seria a instalação de Kumo no destino, de forma semelhante. Por este motivo Kumo ficou fixo nas nuvens de origem na Virgínia, como pode ser visto na Figura 10.

Outra decisão relevante que precisou ser tomada na fase de planejamento das migrações, foi a de qual a **métrica** que deveria ser utilizada para avaliar as migrações das VMs. Na literatura são apontadas geralmente duas métricas referentes ao processo de migração de VMs, a de Tempo de Inatividade e a de TTM, ou *Total Migration Time*. O Tempo de Inatividade é, segundo Zhang et al. (2017), a duração em que a VM migrada fica indisponível durante a migração. Enquanto isto, o TTM é o tempo desde o momento em que a migração começa na nuvem de origem (primeiro passo) até o momento em que a VM migrada é iniciada no servidor de destino (neste estudo, passo 9 da seção 3.2). Porém, como argumentado por Zhang et al. (2017), no caso da migração em modo *non-live*, que é o modo como as migrações são realizadas por Kumo, o Tempo de Inatividade e o TTM são iguais. Assim, a métrica TTM foi escolhida e aferida em cada execução do experimento.

Por fim, por causa do altos valores de TTM descobertos nas execuções das migrações preliminares, descobriu-se que as migrações poderiam alcançar valores de TTM acima de 3 horas, e que os custos das execuções das migrações, por serem feitas em nuvens públi-

cas, mostraram-se significativos. Assim a quantidade de experimentos executados ficou definida como **5 execuções para cada um dos 9 cenários definidos**, 3 homogêneos e 6 heterogêneos, somando assim um **total de 45 execuções de migrações realizadas** pelo experimento. Vale mencionar que cada um dos 9 cenários foi executado serialmente, ou seja, o próximo cenário só pôde ser iniciado após o término do anterior, o que também ocorreu para cada uma das 5 execuções de cada cenário, onde para que uma migração pudesse ser iniciada a migração anterior precisava ser concluída.

A Equação 4.1 mostra como calcular o TTM em Kumo, o que é feito através da soma dos valores obtidos pelo *decorator* mostrado no Código 9 que são exibidos pelos *logs* do Serviço de Migração.

$$TTM = \sum_{i=1}^{9}(Tf_i - Ti_i) \tag{4.1}$$

Onde:

i = Passo da Migração

Tf_i = Tempo de Final da Execução do Passo i

Ti_i = Tempo de Inicial da Execução do Passo i

4.1.1 Descobertas iniciais

Durante execuções preliminares no ambiente descrito nesta seção, foi descoberta uma diferença entre os tamanhos dos discos que estavam sendo baixados por Kumo. Enquanto o tamanho do disco baixado da AZ tinha exatos 32 GiB, na AWS e na GCP os discos tinham tamanhos em torno de 2 GB. Então, descobriu-se que tanto a AWS quanto a GCP usam tamanhos de disco chamados de **tamanhos dinâmicos**, o que significa que logo que a VM é instanciada, o disco não tem o tamanho especificado no momento da criação, e à medida que o usuário preenche o disco com dados, o tamanho máximo é expandido até que chegue ao limite especificado. No caso da AZ, desde o momento da criação da VM o disco tem o tamanho especificado, o que é classificado como um disco de **tamanho fixo**. Logo, **para garantir o isolamento da variável `tamanho do disco`**, por causa da existência dos discos de tamanho fixo e de tamanho dinâmico, resolveu-se que os discos deveriam ter seu tamanho fixado no tamanho máximo, e por este motivo optou-se por realizar o preenchimento completo dos discos com um arquivo de conteúdo aleatório gerado através do comando `dd if=<input_file> of=<output_file> bs=<size>` (ex. `dd if=/dev/urandom of=kumo.txt bs=32000000000`).

Outra descoberta durante as execuções preliminares foi feita, referente à **unidade para representação da informação**. Como mencionado, a AZ utiliza para o tamanho de seus discos unidade binária, que possui tamanho fixo (32 GiB desde o momento da criação da VM) para o

tamanho do disco da VM. Assim, quando um disco é exportado da AZ, este disco tem um tamanho exato de 32 GiB. Porém, na documentação da AZ (vide https://docs.microsoft.com/en-us/azure/virtual-machines/linux/create-upload-generic) é mencionado que para um disco ser importado para uma nuvem AZ, o tamanho do disco precisa ser múltiplo de 1 Megabyte (MB), quando na verdade o correto é ser múltiplo de 1 Mebibyte (MiB). Se a informação da documentação estivesse correta, um disco que fosse criado na AZ não poderia ser importado por esta mesma nuvem, dado que seu tamanho de 32 GiB não é múltiplo de 1 MB. Na Equação 4.2 é mostrado como deve ser calculado o valor para o redimensionado do disco, para casos onde um disco criado na AWS ou na GCP e que precise ser importado na AZ não esteja alinhado a 1 MiB.

$$ALIGN = \left\lfloor \frac{(T_d + Mi_B - 1)}{Mi_B} \right\rfloor \times Mi_B \tag{4.2}$$

Onde:

T_d = Tamanho do Disco Antes do Alinhamento

$Mi_B = 1$ MiB $= 1024 \times 1024 = 1048576$ Bytes

4.2 EXECUÇÃO DAS MIGRAÇÕES

Com o intuito de documentar como replicar este estudo, este capítulo mostra como instalar, configurar e usar os dois componentes de Kumo, o Serviço de Migração e a CLI. Para executar os comandos aqui documentados é necessário que o usuário disponha de um ambiente Linux, como a distribuição Ubuntu 18.04 LTS, que foi usada para a execução do experimento realizado neste trabalho; também é necessário ter ao menos duas contas criadas, sejam elas criadas em um mesmo provedor ou em provedores diferentes, mas sendo indispensável serem contas nas nuvens avaliadas neste trabalho, a AWS, AZ ou GCP; e por fim ter uma VM lançada em uma das contas, conta esta que será a origem da migração, o que põe automaticamente a segunda conta como destino da migração.

4.2.1 Executando o Serviço de Migração

No Código 10 é mostrado como realizar a execução do Serviço de Migração de Kumo. As linhas 1–4 são referentes à instalação do Docker. A linha 5 mostra como baixar o código do Serviço de Migração a partir de seu repositório no GitHub[1]. As linhas 6–7 mostram como gerar a imagem base para a criação do *container* do Serviço de Migração, *container* este criado pelos comandos das linhas 8–10. A linha 11 mostra como obter os logs do Serviço de Migração, que devem mostrar, se o *container* foi iniciado de acordo, exatamente o que é mostrado na Figura 11, ou seja, que o Serviço de Migração foi iniciado com sucesso e está pronto para receber as solicitações das migrações realizadas por meio da CLI de Kumo.

[1] https://github.com/marcusrafael/kumo

```
1  sudo apt update
2  sudo apt install -y docker.io
3  sudo usermod -aG docker ubuntu
4  newgrp docker
5  git clone https://github.com/marcusrafael/kumo
6  cd kumo
7  docker build -t kumo .
8  docker run -d -p 5000:5000 \
9      -v kumo-volume:/home/ubuntu/volume \
10     --name kumo kumo
11 docker logs -f kumo
```

Código 10 – Comandos para a execução do Serviço de Migração

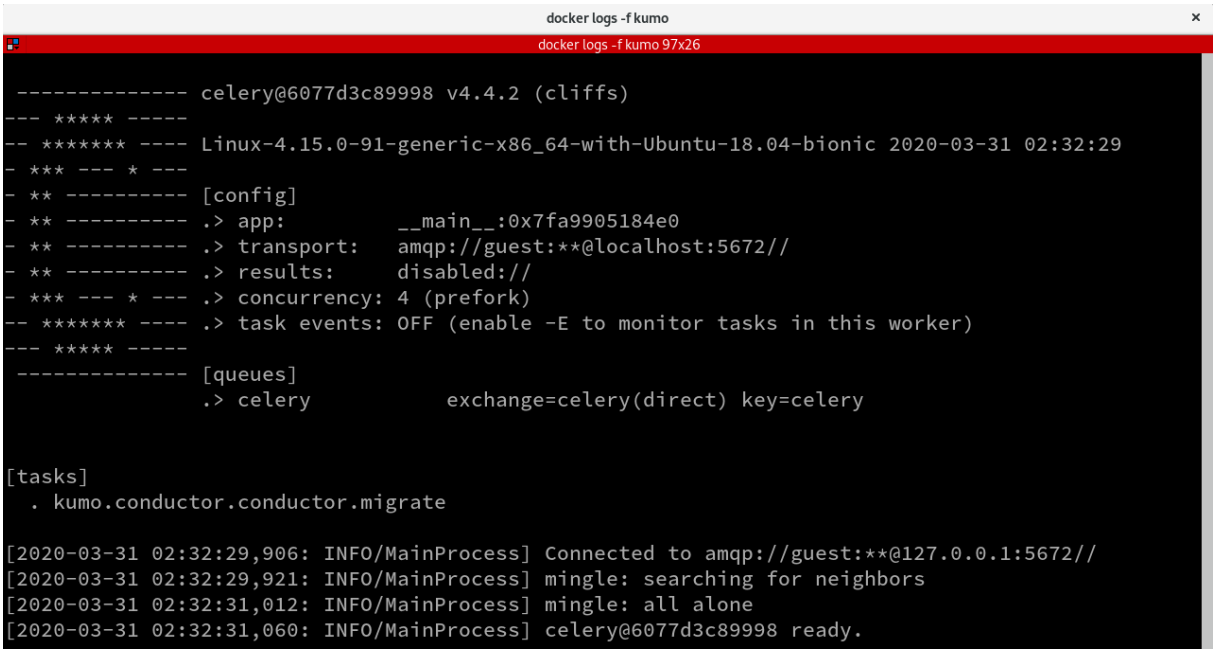


Figura 11 – Logs do Serviço de Migração após iniciado

4.2.2 Executando a CLI

Com o Serviço de Migração iniciado, o próximo passo é executar a CLI de acordo com o passo a passo mostrado pelo Código 11. Este Código 11 mostra nas linhas 1–5 como instalar o PIP Installs Packages (PIP)[2], que é uma ferramenta necessária para poder instalar pacotes de software desenvolvidos para a linguagem Python, para que em seguida através do PIP instalar o VirtualEnv[3], que permite que a instalação da CLI e de suas dependências sejam realizadas isoladamente, sem causar conflitos com outras dependências do SO. A linha 6 mostra como baixar o código da CLI de Kumo a partir de seu repositório no GitHub[4]. As linhas 7–9 mostram como é realizada a instalação da CLI, enquanto a linha 10 mostra como deve ser iniciada a configuração da ferramenta, criando o arquivo de configuração `kumo.conf` no diretório *home* do usuário, como mostrado na linha 11, que deve ter seus campos preenchidos de acordo com

[2] https://pip.pypa.io/en/latest
[3] https://virtualenv.pypa.io/en/latest
[4] https://github.com/marcusrafael/kumo-cli

as configurações das contas de cada nuvem. Na linha 12 devem ser cadastradas no arquivo `"templates/template.yaml"` qual VM será migrada e quais contas cadastradas no `kumo.conf` serão usadas como origem e destino da migração. Por fim, na linha 13 a CLI usa as informações do arquivo `"templates/template.yaml"` para recuperar as contas do arquivo `kumo.conf` que serão usadas na migração, e em seguida as envia, juntamente com o nome da VM, para o Serviço de Migração realizar a migração.

```
1  sudo apt update
2  sudo apt -y install python3-pip
3  sudo pip3 install virtualenv
4  virtualenv -p python3 ~/.venv/kumo
5  source ~/.venv/kumo/bin/activate
6  git clone https://github.com/marcusrafael/kumo-cli
7  cd kumo-cli
8  pip3 install -r requirements.txt
9  pip3 install . --upgrade
10 kumo configure
11 vim ~/.kumo/kumo.conf # Fill with your credentials.
12 vim templates/template.yaml # Fill with your credentials' name.
13 kumo migrate templates/template.yaml
```

Código 11 – Comandos para a execução da Interface de Linha de Comando

A Figura 12 mostra o que acontece no Serviço de Migração quando uma requisição é recebida, nela é possível ver que foram executados os 9 passos de uma migração conforme definido neste trabalho, e também que são capturados os tempos antes da execução de cada passo, o tempo após o fim da execução, e por fim a diferença entre os tempos. Assim, através da soma do tempo de execução dos passos é obtido o TTM, que foi a métrica utilizada para a avaliação deste trabalho. Vale observar que os tempos mostrados na Figura 12 são fictícios e não refletem a realidade, sendo usados na figura apenas com o propósito de ilustrar a aparência dos *logs* que são encontrados pelos usuários de Kumo.

Logo, para replicar os experimento realizado neste trabalho, é necessário criar um total de 6 contas, 2 na AWS, 2 na AZ e 2 na GCP; para cada cenário é também necessário que os pares de contas de origem e destino sejam cadastradas no arquivo `kumo.conf`; e para cada combinação de origem e destino é necessário criar, sempre na nuvem de origem:

- **Uma VM para instalar o Serviço de Migração e a CLI**: Criar uma VM com SO Ubuntu 18.04 LTS e 100 GiB para a AWS e AZ ou 107 GB para a GCP, e em seguida instalar o Serviço de Migração e a CLI;
- **Uma VM para ser migrada por Kumo**: Criar uma VM na origem, com um disco de 30 GiB para AWS e AZ ou 32 GB para a GCP, preencher completamente o disco com dados usando o comando `dd if=<input_file> of=<output_file> bs=<size>` (ex. `dd if=/dev/urandom of=kumo.txt bs=32000000000`).

Por fim, basta configurar o arquivo `"templates/template.yaml"` da CLI com as informações referentes à nuvem de origem, nuvem de destino e o identificador ou nome da VM que será

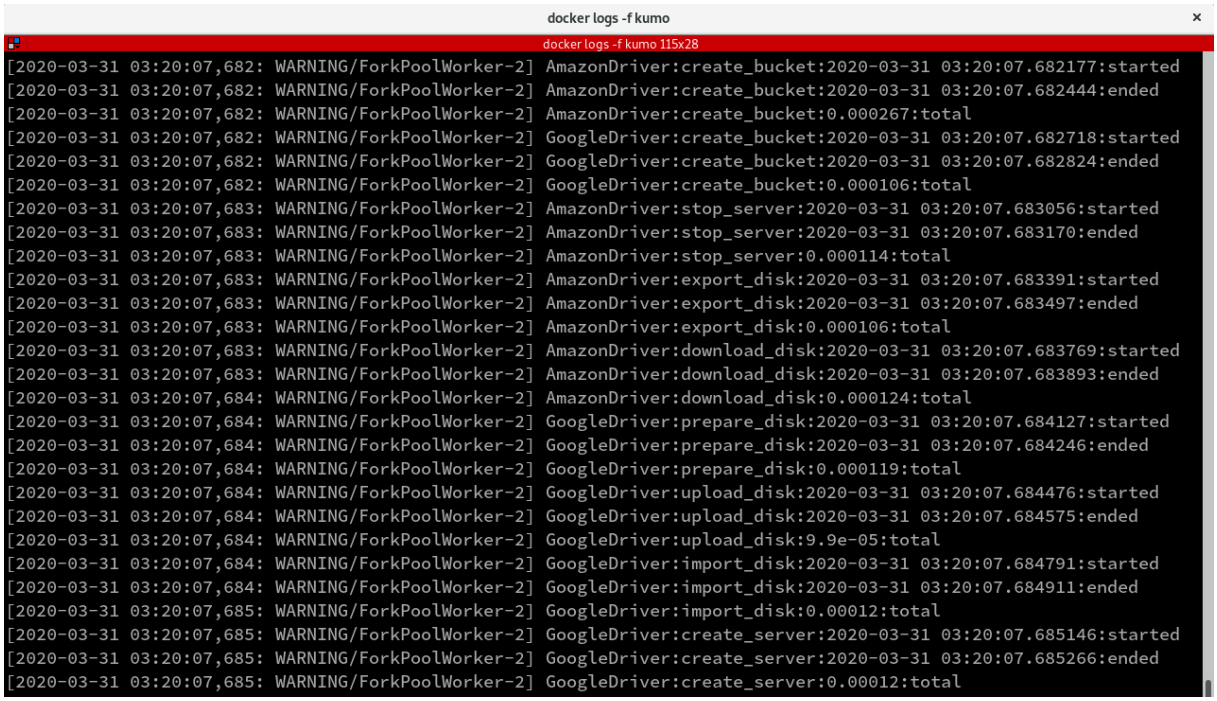


Figura 12 – Logs do Serviço de Migração com os tempos de execução dos passos

migrada, para que assim as migrações sejam iniciadas. Vale lembrar que neste trabalho foram realizadas 5 migrações para cada um dos 9 cenários definidos, resultando em um total de 45 execuções. Outra informação importante está no sentido da migração, onde as migrações sempre foram executadas em um mesmo sentido fixo, a partir do *data center* da nuvem de origem na Virgínia para o *data center* de destino na Califórnia (Virgínia → Califórnia).

4.3 ANÁLISE DAS MIGRAÇÕES

Conforme os cenários definidos na seção 4.1, foram criadas as devidas VMs para instalar Kumo e aquelas VMs que deveriam ser migradas, como mostrado na seção 4.2. Após a configuração do ambiente ser completamente finalizada, foram iniciadas as migrações que teriam seus tempos analisados. Dois tipos inter-relacionados de análise foram realizados neste trabalho: (1) sobre os tempos totais das migrações (TTM) realizadas nos 9 cenários (3 homogêneos e 6 heterogêneos) descritos anteriormente, visando dar suporte ao usuário na análise de opções para a tomada de decisão; e (2) a análise do impacto de cada um dos passos da migração de VMs nos mesmos 9 cenários.

4.3.1 Análise das Migrações por Inteiro

A partir dos intervalos interquartílicos (Q1, Q2/Mediana, Q3) mostrados em cada célula da Tabela 6, observa-se que não houve uma grande dispersão entre os valores da amostra de cada cenário. Na tabela é possível perceber que a melhor opção dentre todos os cenários, tanto homogêneos quanto heterogêneos, foi a migração de VMs da GCP para a AWS, com mediana de 45 minutos e 56 segundos. É importante frisar que este cenário de migração é realizado entre

duas nuvens concorrentes e heterogêneas, e ainda assim se saiu melhor do que realizar migrações entre nuvens homogêneas (AWS-AWS, AZ-AZ e GCP-GCP).

Isto pode ser melhor entendido na análise do passo a passo feita a seguir na seção 4.3.2, onde se nota (Figuras 13 e 14) que o passo 7 (Enviar Disco Para a Nuvem de Destino), dentre todas as migrações, foi o mais rápido, em torno de 320 segundos.

Tabela 6 – Tempos totais das migrações para os 9 cenários de Origem-Destino

Origem \ Destino	AWS	AZ	GCP	Quartil
AWS	2h14m31s	2h55m56s	2h59m01s	Q1
	2h23m18s	**3h09m44s**	**3h05m38s**	Mediana
	2h23m47s	3h13m02s	3h10m08s	Q3
AZ	1h28m12s	0h45m13s	1h41m42s	Q1
	1h29m26s	**0h45m59s**	1h43m13s	Mediana
	1h30m58s	0h48m35s	1h44m22s	Q3
GCP	0h43m24s	2h30m30s	1h25m26s	Q1
	0h45m56s	2h31m35s	1h25m49s	Mediana
	0h58m49s	3h10m33s	1h26m51s	Q3

Também entre as migrações homogêneas e heterogêneas, os dois piores cenários, com os maiores tempos (TTM), aconteceram nas migrações AWS-AZ e AWS-GCP, ambos com medianas acima de 3 horas. Para a migração de AWS para AZ, o passo 4 (Exportar Disco da Origem) e também o passo 7 (Enviar Disco para o Destino) prejudicaram mais significativamente a migração, representando respectivamente 53% e 26% do tempo total da migração (TTM). Enquanto que para o cenário de migração de AWS para GCP, os passos que mais afetaram o TTM foram, mais uma vez, o passo 4, que representa 51% do tempo total da migração, e o passo 8 (Importar Disco na Nuvem de Destino), representando 27% do tempo total.

Analisando especificamente as migrações entre nuvens homogêneas, observa-se que os melhores TTMs ocorreram nas migrações GCP-GCP, enquanto que os piores se deram entre AWS-AWS, lembrando que, para todos os casos, a migração se deu dos respectivos *data centers* da Virgínia para os da Califórnia.

4.3.2 Análise do Passo a Passo

Desta vez, analisando todos os passos da migração entre as nuvens através das Figuras 13 e 14, percebe-se que os **passos 1**, **2**, **3** e **9** têm pouca interferência em todos os cenários de migração. O **passo 1** (Criar Bucket na Origem), como percebido nas figuras, não é executado quando a origem é a AZ. Isto pode ser explicado pelo fato da AZ, diferentemente das demais nuvens,

exportar o disco apenas criando um link para que este seja baixado, dispensando assim que um *bucket* seja criado na nuvem de origem para armazená-lo.

Outro passo que não é executado em todas as migrações é o **passo 6** (Preparar Disco da Origem), sendo apenas executado quando a nuvem de origem é a AWS ou GCP e o destino da migração é a AZ. Isto acontece devido à nuvem AZ, quando destino, exigir que o disco que será importado tenha um tamanho que seja múltiplo de 1 MiB, o que pode não acontecer quando o disco é exportado pela nuvem de origem AWS ou ainda pela GCP, necessitando realizar o redimensionamento. Por exemplo, na AWS um disco de 32 GB quando tem seu tamanho convertido em MiBs, resulta em 30.517,578125 MiBs, o que não é um múltiplo de 1 MiB. Assim, nestes casos é necessária uma intervenção de Kumo, através do passo 6, para alinhar seu tamanho a 1 MiB realizando as devidas conversões necessárias. Vale ressaltar que, como o passo 6 é executado sempre na nuvem de origem e como a conversão do arquivo de disco é uma tarefa que faz uso intensivo do disco (I/O-*bound*) da VM que executa Kumo, é possível concluir que a AWS tem um disco mais rápido do que a GCP. Isso justifica o tempo consideravelmente menor para o passo 6 quando executado no cenário de AWS para a AZ do que da GCP para a AZ.

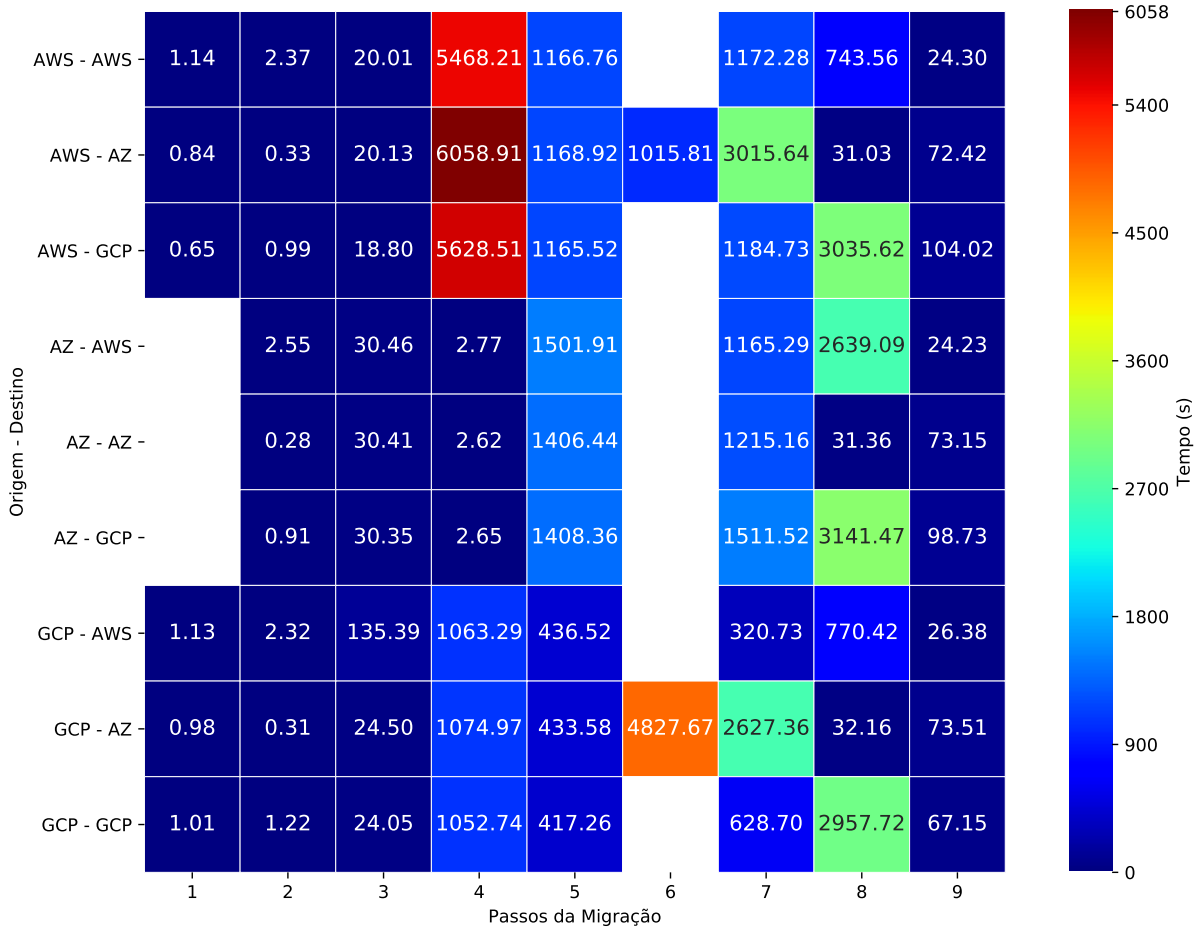


Figura 13 – Mapa de calor da mediana dos passos das migrações

No **passo 4** (Exportar Disco da Origem), quando a nuvem de origem é a AZ, os tempos de execução são muito baixos em relação às demais origens (AWS e GCP). Pela Figura 14, nota-se que quando a nuvem de origem é a AWS, além de terem os valores mais altos de mediana entre as

medianas de todos os passos 4, também é notável a presença de uma variabilidade considerável, representada pelo tamanho das caixas nos *boxplots*. No **passo 5** (Baixar Disco da Origem), destacam-se os três cenários onde a GCP é a nuvem de origem, sendo executados cerca de três vezes mais rápido do que nas outras nuvens, o que pode ser mais facilmente percebido na Figura 13.

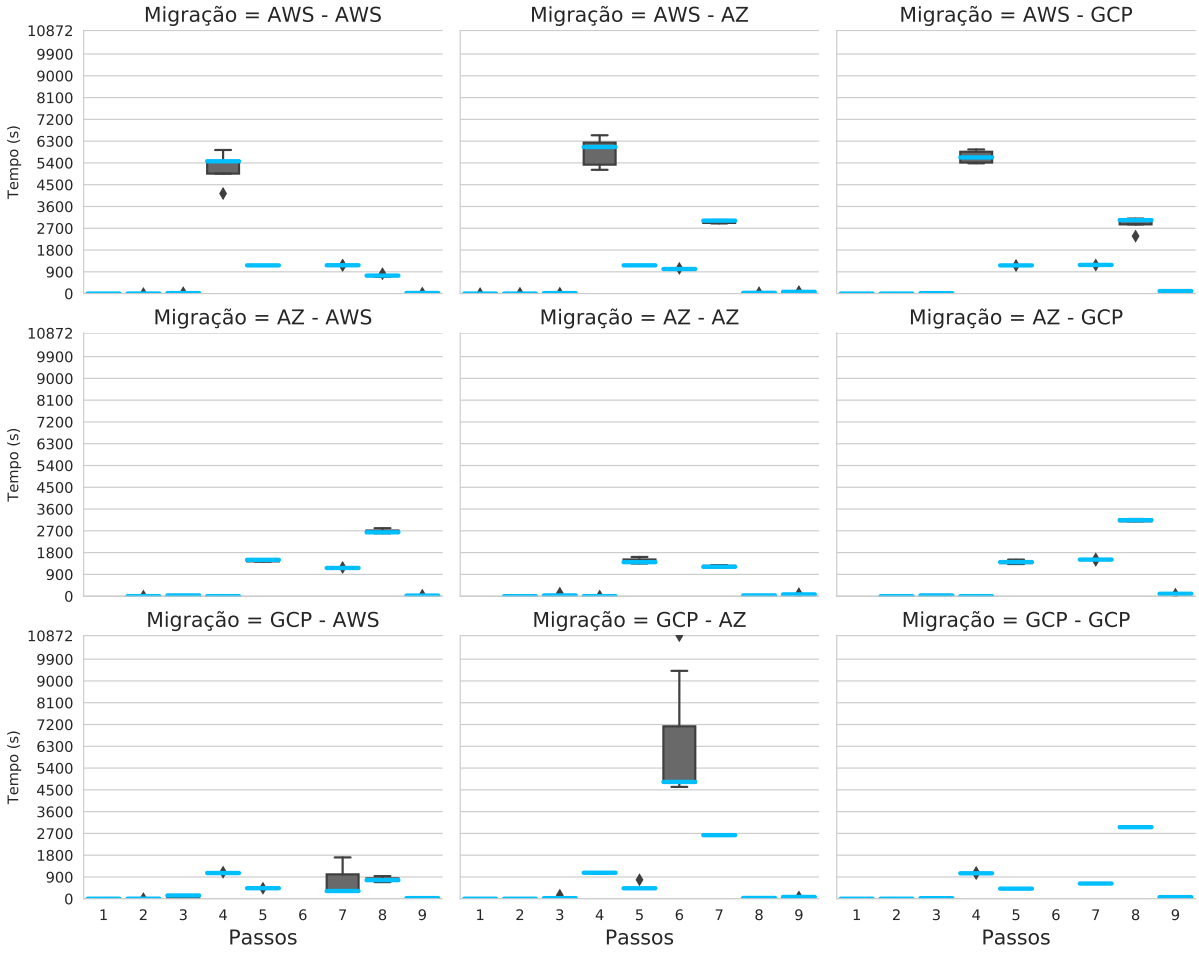


Figura 14 – Comparações entre os passos das migrações realizadas usando Kumo

Para o **passo 6** (Preparar Disco da Origem), percebe-se pela Figura 14 uma alta variabilidade nos tempos quando o passo é executado na nuvem GCP com destino a AZ. Pela Figura 13, fica claro que o pior resultado é no cenário da migração da GCP para a AZ. Este comportamento acontece devido ao processo de conversão do disco do formato Virtual Hard Disk (VHD) para o formato RAW Format Hard Drive (RAW), para em seguida redimensioná-lo alinhando-o a 1 MiB, e por fim convertê-lo de volta para o formato VHD, que é mais lento na GCP do que na AWS, por exemplo.

No **passo 7** (Enviar Disco para o Destino), nota-se que o melhor caso acontece no cenário da migração de GCP para AWS, que apesar de na Figura 14 apresentar alguma variabilidade, na Figura 13 se percebe que este cenário se sai em média melhor do que, por exemplo, na migração homogênea entre nuvens GCP. Outra observação para o passo 7 é que sempre é pior enviar um disco quando a nuvem de destino é AZ. Por fim, no **passo 8** (Importar Disco no Destino)

nota-se que os três piores cenários são aqueles cujo destino é GCP. Percebe-se também que o melhor caso para importação de um disco é quando esta é realizada pela nuvem AZ, pois a nuvem AZ checa apenas se o disco está no formato VHD e se seu tamanho está alinhado a 1 MB, não realizando qualquer outra etapa de inspeção do disco para verificar se este atende a outros requisitos.

4.3.3 Conclusões das Análises Realizadas

De acordo com as avaliações realizadas, percebe-se que Kumo foi capaz de tornar a experiência de migração de VMs entre multi-nuvens, homogênea e automatizada, tanto para nuvens homogêneas quanto para nuvens heterogêneas, conforme os objetivos da pesquisa. Neste sentido, a contribuição de Kumo para a avaliação deste trabalho acontece, principalmente, na automatização da execução dos passos e obtenção dos tempos de execução destes passos, para que assim sejam realizadas as análises feitas na fase de avaliação das migrações.

Os resultados obtidos mostram que, entre os cenários homogêneos, o melhor cenário se dá para um usuário com contas na AZ, onde o TTM é de 45 minutos e 59 segundos (mediana). Os TTM dos outros cenários homogêneos (AWS e AZ) são até maiores que em cenários de migração entre nuvens heterogêneas. Neste caso, o melhor cenário se dá para a migração de uma nuvem GCP para a nuvem AWS. Este caso, inclusive, é o melhor dentre todas as opções, tanto homogêneas quanto heterogêneas, com TTM de 45 minutos e 56 segundos (mediana). Ao final, os 9 passos para a migração automatizada de VMs foram analisados detalhadamente, mostrando que 3 deles (4, 7 e 8) causam os maiores impactos no TTM, com cerca de 75% do total.

5 CONCLUSÕES

Este trabalho apresentou Kumo, um serviço capaz de realizar migrações de Máquinas Virtuais (VMs) entre múltiplas nuvens, tanto homogêneas quanto heterogêneas, de maneira automatizada. Foram estudadas as três plataformas mais conhecidas e usadas de nuvens públicas, atualmente, a Amazon Web Services (AWS), a Microsoft Azure (AZ) e a Google Cloud Platform (GCP), buscando conhecer os passos necessários para a migração de VMs, e foi identificado que, no geral, uma migração se dá em até 9 passos.

Em seguida, para avaliar o desempenho do processo de migração entre as nuvens, Kumo foi implementado contendo uma Interface de Linha de Comando (CLI) e um serviço de migração de VMs. Para lidar com a heterogeneidade em relação às diferentes formas de acesso de cada nuvem, suas ferramentas com seus diversos comandos e parâmetros, bem como os variados formatos de disco, um *driver* para cada nuvem foi implementado contemplando cada um dos 9 passos identificados. Para manter a consistência da avaliação, as nuvens de origem e de destino foram fixadas, respectivamente, nos estados americanos da Virgínia e da Califórnia, onde se encontram *data centers* de todas as plataformas pesquisadas. A métrica usada foi a de Tempo Total de Migração (TTM), que é dada pela soma dos tempos da execução de cada um dos passos. Entre os cenários de migração homogêneos, o melhor cenário se dá para um usuário com conta na AZ, onde o TTM fica em torno de 45 minutos. O melhor cenário dentre todas as opções se dá para a migração de uma nuvem GCP para outra nuvem AWS, com TTM em torno de 45 minutos e alguns segundos mais rápida do que a migração AZ-AZ. Cerca de 75% do TTM é resultado da soma das execuções dos passos 4, 7 e 8, os mais impactantes.

5.1 INTEGRANDO OUTRA NUVEM A KUMO

Como exemplo de como seria integrar uma nova nuvem a Kumo, foi escolhido o OpenStack. O OpenStack é um sistema operacional para nuvem que controla grandes quantidades de recursos de computação, armazenamento e rede em um *data center*, todos gerenciados e provisionados por meio de Interfaces de Programação de Aplicações (APIs) com mecanismos de autenticação comuns (OPENSTACK, 2020).

A Figura 15 mostra quais são os componentes principais do OpenStack e também como estes componentes se relacionam entre si. Primeiramente, para acessar o OpenStack é necessário criar credenciais no serviço de identidade Keystone (*Identity*, na Figura 15), que permite configurar tanto quais usuários poderão acessar a nuvem, como também, uma vez autenticado, define quais são as funcionalidades e recursos que cada um dos usuários está autorizado a acessar.

Quanto à forma de acesso dos usuários e operadores de nuvem ao OpenStack, existem 3 formas de interação destes atores com o OpenStack: o *dashboard*, que é a Interface Gráfica do Usuário (GUI) fornecida pelo serviço Horizon (*Dashboard*, na Figura 15); a ferramenta de Linha de Comando do OpenStack, conhecida como OpenStack Client (OSC); e ainda existe uma forma de utilizar os recursos do OpenStack via código Python, que é através do OpenStack Software Development Kit (SDK). É através do OpenStack SDK que o *driver* OpenStack para Kumo

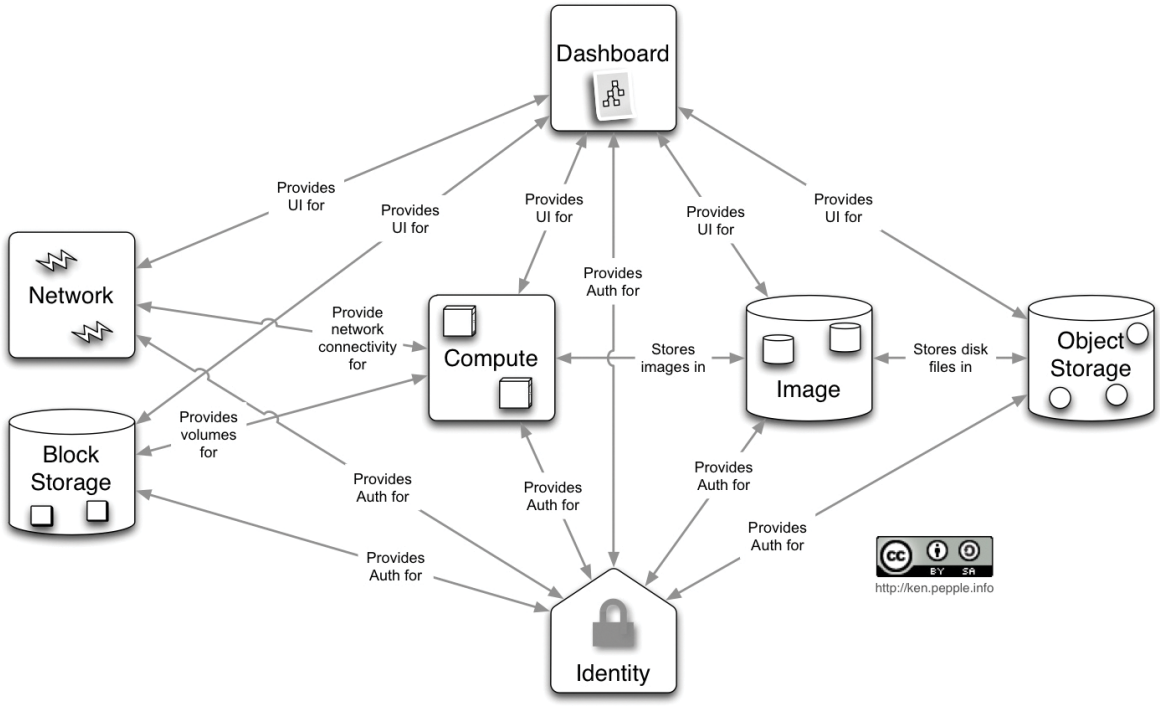


Figura 15 – Serviços principais da arquitetura do OpenStack, segundo Pepple (2013)

terá acesso aos serviços do OpenStack, tornando-o interoperável com as demais nuvens. Para isto, é necessário realizar a implementação de um *driver* de OpenStack para Kumo, que deve ser implementado no Serviço de Migração de Kumo.

Para associar a forma como o OpenStack implementa os passos de migração de VMs com os 9 passos definidos em Kumo, e que precisam ser implementados para desenvolver um *driver* para Kumo, é necessário implementar os 8 métodos definidos pela interface padronizada definida na seção 3.4.2. Para poder, ainda que de maneira superficial, projetar como poderia ser realizada uma integração com o OpenStack, foi preciso descobrir na documentação do mesmo como é realizada a exportação e importação de discos virtuais. Assim, na documentação do OpenStack SDK, referente ao serviço de gerenciamento de imagens de disco, chamado Glance[1] (*Image*, na Figura 15), é mostrado que dentre as operações de Create, Read, Update e Delete (CRUD) existentes, existem também operações para exportação e importação de discos virtuais.

Na pesquisa realizada para idealizar como o OpenStack poderia ter uma integração em Kumo, percebeu-se que nem o **passo 1** (método `create_bucket`, quando o *driver* for usado como origem) nem o **passo 2** (`create_bucket`, quando o *driver* for usado como destino) são necessários ser implementados, pois a forma como o Glance exporta e importa discos virtuais dispensa a criação de *buckets*. O **passo 3** (método `stop_server`) deve ser implementado usando o serviço de computação, chamado Nova (*Compute*, na Figura 15), através do OpenStack SDK para executar a ação de desligar a VM. O **passo 4** (método `export_disk`) também é executado pelo Nova, que deve criar uma nova imagem de disco contendo todo o conteúdo da VM que deverá ser migrada. No **passo 5** (método `download_disk`), a imagem criada no passo 4 deve ser

[1] https://docs.openstack.org/openstacksdk/latest/user/guides/image.html

baixada para Kumo. Para o **passo 6** (método `prepare_disk`), como os *hypervisors* disponíveis no OpenStack possuem compatibilidade com o formato de disco Virtual Hard Disk (VHD)[2], que é o formato utilizado nas implementações dos *drivers* da AWS, AZ e GCP, não deve ser necessário implementar este passo. O **passo 7** (método `upload_disk`) é realizado pelo Glance, que executa o envio do disco para a nuvem OpenStack de destino. O **passo 8** (método `import_disk`) não precisa ser implementado pois durante a execução do passo 7, além da operação de envio ser implementada por este passo, o Glance automaticamente realiza a importação do disco virtual como imagem, executando assim as duas operações em um único passo. Por fim, o **passo 9** (método `create_server`) é realizado pelo Nova através também do OpenStack SDK.

Portanto, desde que sejam implementados apenas 5 métodos (`stop_server, export_disk, download_disk, upload_disk, create_server`) de um total de 8 definidos na interface base para os *drivers* de Kumo, o OpenStack pode ser integrado e assim interoperar com qualquer uma das 3 nuvens anteriormente integradas, que são AWS, AZ e GCP, sendo assim capaz de tanto receber VMs das nuvens, como também de enviar suas VMs para estas outras nuvens.

5.2 CONTRIBUIÇÕES

As maiores contribuições deste trabalho para o estado da arte referente a migrações de máquinas virtuais entre nuvens, são listadas a seguir:

1. **A arquitetura da solução e o serviço de migração** projetados e implementados, permitindo que migrações de VMs sejam realizadas, na perspectiva do operador de nuvem, de maneira agnóstica e não invasiva, sejam quais forem as nuvens envolvidas, de modo a superar o problema do *vendor lock-in*. A arquitetura se beneficia do conceito de *drivers* para permitir que outras nuvens possam ainda ser integradas à solução, bastando para isto implementar os métodos necessários que foram definidos na interface base para a implementação de qualquer *driver*. A arquitetura não faz distinção entre nuvens públicas, privadas, híbridas, homogêneas ou heterogêneas, graças ao modo de migração *non-live*, que não exige interconexão entre a rede ou armazenamento compartilhados entre as nuvens de origem e destino;

2. **A identificação dos passos da migração** é importante tanto para Kumo quanto para qualquer outra ferramenta que venha a ser desenvolvida por outros pesquisadores ou organizações que tenham interesse em realizar migrações de VMs de modo *non-live* entre nuvens, sejam estas nuvens públicas, privadas, híbridas, homogêneas ou heterogêneas. Na pesquisa realizada nas nuvens AWS, AZ e GCP foram identificados no geral 9 passos necessários para realizar migrações de VMs entre nuvens. Nesta pesquisa, os passos identificados possibilitaram a definição de uma interface com as assinaturas referentes aos 8 métodos necessários para realizar migrações utilizando Kumo;

3. **A determinação dos melhores cenários de migração de VMs** permite ajudar usuários de nuvem, operadores e empresas que utilizam ou utilizarão serviços de Computação

[2] https://docs.openstack.org/glance/latest/user/formats.html

na Nuvem no que diz respeito às nuvens de Infraestrutura como Serviço (IaaS). Como resultado, para os cenários executados, é possível indicar que, para o cenário homogêneo, a nuvem da AZ pode ser a opção em que as migrações das VMs serão executadas de maneira mais rápida. Para o cenário heterogêneo, o indicativo é que se o destino for AWS ou AZ, é aconselhado usar inicialmente a nuvem da GCP como origem. Porém, se em algum momento o objetivo for migrar para a GCP, a melhor opção pode ser adotar como origem a nuvem AZ.

5.3 LIMITAÇÕES E TRABALHOS FUTUROS

A seguir são listados as limitações e os trabalhos futuros selecionados para futuros incrementos à solução proposta neste trabalho:

- Uma única aplicação pode ser composta por vários serviços e instalada de modo a utilizar múltiplas VMs, desta forma, seria uma evolução deste trabalho realizar a migração destas várias VMs, respeitando a dependência entre elas e as especificidades de cada uma;
- Como a migração e gerenciamento de recursos, inclusive de VMs, exige que os clientes sejam bastante habilidosos para lidar com a heterogeneidade das nuvens, uma contribuição para este e outros trabalhos seria a concepção de uma GUI que facilite o gerenciamento e migração de recursos de forma mais fácil e intuitiva;
- Investigar como adaptar a solução para livrar clientes do *vendor lock-in*, migrando para outros tipos de recursos como, por exemplo, dados, aplicações, configurações, funções de Function as a Service (FaaS) (YUSSUPOV et al., 2019), ou ainda realizar orquestração multi-nuvem;
- Pesquisar formas de como tornar Kumo capaz de realizar migrações de modo *live* entre nuvens heterogêneas, sejam nuvens públicas, privadas ou híbridas;
- Estudar como seria a criação de um módulo de Kumo para o Ansible ou ainda como seria tornar Kumo, em si, um módulo Ansible, permitindo que clientes de multi-nuvens façam migrações de VMs mais facilmente, se comparado com a alternativa atual oferecida por Ansible;
- Tornar Kumo capaz de realizar migrações de maneira autônoma, identificando assim oportunidades de melhorar a qualidade do serviço ou diminuir o custo de manter a infraestrutura virtual, realizando autonomamente migrações quando julgar favorável.

REFERÊNCIAS

AHMAD, R. W.; GANI, A.; HAMID, S. H. A.; SHIRAZ, M.; XIA, F.; MADANI, S. A. Virtual machine migration in cloud data centers: a review, taxonomy, and open research issues. *The Journal of Supercomputing*, v. 71, n. 7, p. 2473–2515, Jul 2015. ISSN 1573-0484. Disponível em: <https://doi.org/10.1007/s11227-015-1400-5>.

BITTENCOURT, L.; IMMICH, R.; SAKELLARIOU, R.; FONSECA, N.; MADEIRA, E.; CURADO, M.; VILLAS, L.; DASILVA, L.; LEE, C.; RANA, O. The internet of things, fog and cloud continuum: Integration and challenges. *Internet of Things*, v. 3-4, p. 134 – 155, 2018. ISSN 2542-6605. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2542660518300635>.

BITTENCOURT, L. F.; LOPES, M. M.; PETRI, I.; RANA, O. F. Towards virtual machine migration in fog computing. In: *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*. [S.l.: s.n.], 2015. p. 1–8. ISSN null.

CARVALHO, J. O.; TRINTA, F.; VIEIRA, D.; CORTES, O. A. C. Evolutionary solutions for resources management in multiple clouds: State-of-the-art and future directions. *Future Generation Computer Systems*, v. 88, p. 284 – 296, 2018. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X18300025>.

CELESTI, A.; TUSA, F.; VILLARI, M.; PULIAFITO, A. Improving virtual machine migration in federated cloud environments. p. 61–67, Sep. 2010. ISSN 2156-7190.

CHAUHAN, S. S.; PILLI, E. S.; JOSHI, R.; SINGH, G.; GOVIL, M. Brokering in interconnected cloud computing environments: A survey. *Journal of Parallel and Distributed Computing*, v. 133, p. 193 – 209, 2019. ISSN 0743-7315. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0743731518305719>.

Chico Science & Nação Zumbi. *Computadores Fazem Arte*. 1994. Rio de Janeiro: Estúdio Nas Nuvens. Disponível em: <https://www.youtube.com/watch?v=suEq9NqF91E>.

EBERT, C.; GALLARDO, G.; HERNANTES, J.; SERRANO, N. Devops. *IEEE Software*, v. 33, n. 3, p. 94–100, May 2016. ISSN 1937-4194.

ELHABBASH, A.; SAMREEN, F.; HADLEY, J.; ELKHATIB, Y. Cloud brokerage: A systematic survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 51, n. 6, jan. 2019. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3274657>.

FLEXERA. *Cloud Computing Trends: 2019 State of the Cloud Survey*. 2019. https://www.flexera.com/blog/cloud/2019/02/cloud-computing-trends-2019-state-of-the-cloud-survey. Online; Postado em 27/02/2019; Acessado em 02/12/2019.

KARGATZIS, D.; SOTIRIADIS, S.; PETRAKIS, E. G. M. Virtual machine migration in heterogeneous clouds: from openstack to vmware. p. 1–6, Sep. 2017.

KOSTOSKA, M.; GUSEV, M.; RISTOV, S. An overview of cloud portability. Springer International Publishing, Cham, p. 248–254, 2015.

MAES, P. Concepts and experiments in computational reflection. In: *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications*. New York, NY, USA: Association for Computing Machinery, 1987. (OOPSLA '87), p. 147–155. ISBN 0897912470. Disponível em: <https://doi.org/10.1145/38765.38821>.

MANSOUR, I.; SAHANDI, R.; COOPER, K.; WARMAN, A. Interoperability in the heterogeneous cloud environment: A survey of recent user-centric approaches. ACM, New York, NY, USA, p. 62:1–62:7, 2016. Disponível em: <http://doi.acm.org/10.1145/2896387.2896447>.

MANSOUR, I. E. A.; BOUCHACHIA, H.; COOPER, K. Exploring live cloud migration on amazon ec2. p. 366–371, Aug 2017.

MICROSOFT. *Virtual Hard Disk Image Format Specification - Version 1.0*. 2006. https://www.microsoft.com/en-us/download/details.aspx?id=23850. Online; Postado em 11/10/2006; Acessado em 27/01/2020.

NARANTUYA, J.; ZANG, H.; LIM, H. Service-aware cloud-to-cloud migration of multiple virtual machines. *IEEE Access*, v. 6, p. 76663–76672, 2018. ISSN 2169-3536.

OPENSTACK. *What is OpenStack?* 2020. https://www.openstack.org/software. Online; Acessado em 26/01/2020.

PEPPLE, K. *OpenStack Conceptual Architecture*. 2013. https://de.wikipedia.org/wiki/Datei:Openstack-conceptual-arch-folsom.jpg. Online; Postado em 19/04/2013; Acessado em 26/01/2020.

PETCU, D. Consuming resources and services from multiple clouds. *Journal of Grid Computing*, v. 12, n. 2, p. 321–345, Jun 2014. ISSN 1572-9184. Disponível em: <https://doi.org/10.1007/s10723-013-9290-3>.

REISNER, P. Distributed replicated block device. In: *Int. Linux System Tech. Conf.* [S.l.: s.n.], 2002.

SHEN, Z.; JIA, Q.; SELA, G.-E.; RAINERO, B.; SONG, W.; RENESSE, R. van; WEATHERSPOON, H. Follow the sun through the clouds: Application migration for geographically shifting workloads. In: *Proceedings of the Seventh ACM Symposium on Cloud Computing*. New York, NY, USA: Association for Computing Machinery, 2016. (SoCC '16), p. 141–154. ISBN 9781450345255. Disponível em: <https://doi.org/10.1145/2987550.2987561>.

SHIRVANI, M. H.; RAHMANI, A. M.; SAHAFI, A. A survey study on virtual machine migration and server consolidation techniques in dvfs-enabled cloud datacenter: Taxonomy and challenges. *Journal of King Saud University - Computer and Information Sciences*, 2018. ISSN 1319-1578. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1319157818302842>.

THIYAGARAJAN, R. B. *Single and Multi-Cloud Disaster Recovery Management using Terraform and Ansible*. Tese (MSc) — Dublin, National College of Ireland, 2020. Disponível em: <http://trap.ncirl.ie/4144>.

TOOSI, A. N.; CALHEIROS, R. N.; BUYYA, R. Interconnected cloud computing environments: Challenges, taxonomy, and survey. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 47, n. 1, p. 7:1–7:47, maio 2014. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/2593512>.

YADAV, Y.; KRISHNA, C. R. Shared folder based virtual machine migration in cloud computing. In: *Futuristic Trends in Network and Communication Technologies*. Singapore: Springer Singapore, 2019. p. 237–250. ISBN 978-981-13-3804-5.

YUSSUPOV, V.; BREITENBüCHER, U.; LEYMANN, F.; MüLLER, C. Facing the unplanned migration of serverless applications: A study on portability problems, solutions, and dead ends. In: *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. New York, NY, USA: Association for Computing Machinery, 2019. (UCC'19), p. 273–283. ISBN 9781450368940. Disponível em: <https://doi.org/10.1145/3344341.3368813>.

ZHANG, F.; LIU, G.; FU, X.; YAHYAPOUR, R. Cbase: A new paradigm for fast virtual machine migration across data centers. p. 284–293, May 2017.

ZHANG, F.; LIU, G.; FU, X.; YAHYAPOUR, R. A survey on virtual machine migration: Challenges, techniques, and open issues. *IEEE Communications Surveys Tutorials*, v. 20, n. 2, p. 1206–1243, Secondquarter 2018. ISSN 2373-745X.