Estudo de Caso de Gerenciamento de IaaS em Multi-nuvem com Terraform

Alexandre Canalle[1], **Ariel Frozza**[1]

[1]Pós-Graduação em Computação em Nuvem
Pontifícia Universidade Católica do Paraná
Curitiba – PR – Brazil

ariel.frozza@pucpr.edu.br, canalle.alexandre@pucpr.edu.br

Abstract. *Multi-cloud is increasingly being used in corporate environments to meet the needs of users, who require fast, available and affordable services. However, there are still few solutions that can manage resources across different cloud providers simultaneously. This article explores the concepts of Multi-Cloud and Infrastructure as Code. The difficulties in interoperability among multiple cloud providers are described and possible solutions to the difficulties encountered are discussed. Finally, a case study of the implementation of a WEB service in two distinct Clouds (AWS and Openstack) will be presented, in order to verify the effectiveness of an agnostic solution for Multi-cloud management. The Terraform tool was identified as the one that best serves this purpose.*

Resumo. *Multi-nuvem está sendo cada vez mais utilizada em ambientes corporativos para atender as necessidades dos usuários, que exigem serviços rápidos, disponíveis e baratos. Porém, há ainda poucas soluções capazes de gerenciar recursos em diferentes provedores de Nuvem de forma simultânea. Este artigo explora os conceitos de Multi-nuvem e Infraestrutura como Código. São descritas as dificuldades na interoperabilidade entre múltiplos provedores de Nuvem e discutidas possíveis soluções para as dificuldades elencadas. Por fim, será apresentado um estudo de caso de implementação de um serviço WEB em duas Nuvens distintas (AWS e Openstack), com intuito de verificar a efetividade de uma solução agnóstica para gerenciamento de Multi-nuvem. Foi identificado a ferramenta Terraform como a que melhor atende a este propósito.*

1. Introdução

Atualmente, o uso simultâneo de serviços e recursos de múltiplos provedores de Nuvem é justificado pela necessidade de seus consumidores, expressas em requerimentos tais como qualidade de serviço e custo. As organizações que estão migrando para a Nuvem, seja para estender ou substituir sua infraestrutura *on-premises*, buscam soluções confiáveis, seguras e, se possível, sem aprisionamento à fornecedores ou soluções fechadas (*vendor lock in*).

Este estudo de caso discute princípios, conceitos e desafios e apresenta algumas ferramentas envolvidas no uso simultâneo de recursos de múltiplos provedores de Nuvens (públicas ou privadas). Também serão levantados alguns dos principais pontos de atenção na utilização das atuais soluções, que se propõem a gerenciar recursos de Infraestrutura como Serviço (IaaS) ou Plataforma como Serviço (PaaS) de múltiplas Nuvens. O cenário ideal seria utilizar uma solução (software) que pudesse gerenciar múltiplas Nuvens de

forma centralizada, onde uma única instrução de código pudesse criar recursos de IaaS ou Paas em diferentes provedores. Há poucas soluções agnósticas (Open Source ou proprietárias) disponíveis no mercado, já que, as APIs das principais Nuvens diferem muito entre si, dificultando o aproveitamento de código.

A definição e os benefícios do uso de um ambiente em Multi-nuvem serão apresentados primeiramente, bem como o conceito de Infraestrutura (IaC) como Código, igualmente importante para a discussão do tema. Em seguida, um modelo padrão de arquitetura para Multi-nuvem será apresentado. A partir dessa arquitetura de referência, algumas soluções de *software* que tornam possíveis a IaC em Multi-nuvem são apresentadas e discutidas. Por fim, será feita uma experimentação destacando os processos, requisitos e desafios na implementação de IaaS em Multi-nuvem usando uma ferramenta *Open Source*, o Terraform. O estudo será realizado à luz dos princípios da IaC, validando as condições em que os códigos podem ser reaproveitados, e também, se há vantagens no uso desta solução, frente às soluções proprietárias de cada provedor de Nuvem. A experimentação está disponível a partir do GitHub em https://github.com/arielfrozza/multi-cloud.

2. Definição e Necessidade de Multi-nuvens

Multi-nuvem é o uso simultâneo e combinado de recursos de diferentes provedores de Nuvens públicas ou privadas para atender às necessidades de negócios.

O tipo mais comum de Multi-nuvem é o híbrido, que envolve duas ou mais Nuvens, por exemplo, uma Privada e uma Pública. Comumente, esse modelo híbrido é usado para transbordo de capacidade (*cloud bursting*) onde os recursos são expandidos para a Nuvem Pública quando os recursos da Nuvem Privada chegam nos níveis máximos predefinidos [Ferrer et al. 2012]. A migração de uma Nuvem para outra, mesmo que apenas uma vez, é outro exemplo de uso de Multi-nuvem.

Muitos termos são encontrados na literatura para designar o uso simultâneo de duas ou mais Nuvens. A citar os mais recorrentes [Ferrer et al. 2012]; *Multi-cloud, Cloud-federation, Inter-cloud, Hybryd Cloud, Cloud of Clouds, Sky Computing, Aggregated Clouds, Fog Computing* e *Distributed Clouds*. Sendo assim, torna-se útil delimitar o conceito de Multi-nuvem abordado neste artigo de outros modelos de uso combinado de Nuvens.

De acordo com [Ferrer et al. 2012], existem dois modelos de entrega de serviços em múltiplas Nuvens; Nuvem Federada e Multi-nuvem. A diferença entre estes modelos seria o grau de colaboração entre os provedores de Nuvem e pelo modo com o qual os usuários interagem com as Nuvens. No primeiro modelo há o acordo de uso compartilhado dos recursos entre os provedores. Os usuários de uma Nuvem federada não ficam sabendo, na maioria dos casos, de qual dos provedores um determinado recurso está sendo consumido. No caso do Multi-nuvem, o usuário está ciente e é responsável pela alocação de recursos em um ou outro provedor e não há anuência dos provedores para uso compartilhado de recursos entre os provedores.

Sky Computing, Aggregated Clouds, Multi-tier Clouds ou *Cross-Cloud* são casos particulares de Nuvens Federadas e portanto não serão abordadas neste artigo.

A interoperabilidade entre os diversos serviços ofertados e orquestração de re-

cursos em múltiplas Nuvens usando IaC[Morris 2016], por exemplo, ainda está distante da maturidade. De acordo com [Fisher 2018], todos os provedores de Nuvem experimentam, eventualmente, períodos de indisponibilidade que podem causar impactos a negócios. Também pode ser um problema a performance variável quando os provedores de Nuvem fazem a oversubscrição (*overprovision*) da sua infraestrutura virtualizada e isto resulta em degradação de performance e qualidade de serviço [CloudSpectator 2017]. Outro ponto importante é que a maioria dos provedores de Nuvem (inclusive os líderes de mercado) disponibilizam Software e APIs proprietárias, que não são aderentes à padrões de *Cloud API* como propostos por OASIS TOSCA [TOSCA 2019] ou OASIS CAMP [CAMP 2019].

Os motivos que justificam o uso de múltiplas Nuvens são inúmeros e dependem da natureza do negócio de cada usuário, mesmo assim é possível citar algumas vantagens do uso simultâneo de dois ou mais provedores de Nuvem:

Tolerância à falhas (*fault-tolerance*): o uso de mais de um provedor de Nuvem simultaneamente permite movimentações de contingência no caso de indisponibilidade de um dos provedores.

Neutralidade de fornecedor: a implementação de IaaS e/ou PaaS nos principais provedores de Nuvens usando soluções diversas (em especial *open-source*) evita o aprisionamento à um fornecedor (*vendor lock-in*).

Performance: possibilita ajustes nos níveis de carga para outros provedores de Nuvem em caso de degradação de performance ou de qualidade de serviço em um dos provedores.

Segurança: possibilita a fácil migração de serviços para provedores mais aderentes aos parões de segurança exigidos pela natureza do serviço ofertado.

Eficiência de custos: possibilita a otimização dos custos de recursos computacionais e serviços em um ambiente Nulti-nuvem.

Os custos de operação dos provedores de Nuvem variam em função dos requisitos de uso e da época de contratação dos serviços. Desta forma, os custos de trocar de provedor podem ficar muito altos caso a infraestrutura e/ou aplicações tenham que ser refeitas inteiramente. Por exemplo, um dos modelos de compra de VMs da Amazon, o AWS Spot Instance oferece descontos para certos casos de uso, que podem chegar a 90%, quando sua infraestrutura encontra-se ociosa. Outro exemplo é a redução dos custos operacionais dos provedores de Nuvem ao longo do tempo. De 2011 para 2013, o custo por hora do AWS EC2 baixou cerca de 30% [Golden 2013]. Com o uso de padrões *vendor neutral*, o objetivo seria possibilitara troca de provedor ou priorizar a execução *on-premises* de acordo com a conveniência, sem ter que alterar o *software stack*.

3. Infraestrutura como Código

Infraestrutura como código – IaC (*Infrastructure as Code*) é o processo de gerenciamento e provisionamento de recursos de infraestrutura através de código e arquivos de configuração que descrevem o estado desejado para os recursos de infraestrutura. IaC usa definições declarativas ao invés de processos manuais ou procedurais. Como se tratam de arquivos de código, as definições podem ser armazenadas em um sistema de controle de versões, tal como o Git.

Algumas vantagens da utilização de IaC incluem a eliminação de tarefas repetitivas possibilitando a fácil replicação de implementações, quando o bloco de código pode ser executado repetidas vezes, fazendo poucas modificações. Por exemplo, para criar ou modificar recursos em diferentes ambientes (Produção, Teste, Desenvolvimento) Nuvens (GCP, Azure ou AWS). Uma vez que a infraestrutura está codificada,geralmente em módulos, o código pode ser reaproveitado. O fato do código que define a infraestrutura estar versionado em ferramentas como Git, possibilita maior controle nas aplicações de mudança nos ambientes facilitando, inclusive, a recuperação de desastres. Esta característica também possibilita a simplificação da documentação do código, bem como a sua manutenção e suporte. Além disso, com IaC, fica fácil utilizar-se das mesmas técnicas de planejamento e testes automatizados há muito tempo utilizadas por desenvolvedores.

4. Arquitetura de Referência para Multi-nuvem

Conforme ilustrado na Figura 1, será explorado um padrão de implementação de Multi-nuvem, envolvendo duas Nuvens, uma Privada e outra Pública, que propõe ajudar a atender alguns casos comuns de uso.

Transbordo para Nuvem Pública (*Bursting Pattern*): em algumas situações, a capacidade *on-premises* de uma empresa pode se exaurir e a expansão para uma Nuvem pública pode suprir uma determinada demanda.

Alta Disponibilidade: eventualmente, um provedor de Nuvem pode ficar parcial ou totalmente indisponível ou com degradação na qualidade dos serviços por diversos motivos. A migração de um provedor para outro evita impactar o usuário final. Por exemplo, em caso de falha ou manutenção do *Data Center* que hospeda a Nuvem privada, migram-se os serviços para uma Nuvem pública, mesmo que temporariamente.

Eficiência de Custos: em alguns casos, provedores oferecem descontos ou modificam seus preços em função da oferta e demanda, possibilitando transferência de recursos e serviços de uma Nuvem para outra sem que o usuário final seja impactado.

A Figura 1 ilustra as atividades e processos envolvidos para atingir os objetivos descritos acima. Outros benefícios, tais como Tolerância à Falhas, Segurança, Validação experimental, etc, também podem ser alcançados usando este mesmo padrão ou após serem feitas pequenas modificações. Esses não serão diretamente abordados neste artigo, mas podem ser entendidos de forma mais completa em [Fisher 2018].

Está numerada na Figura 1 a ordem do processo de provisionamento de recursos em Multi-nuvem, que inicia em **Multi-cloud Telemetry (1)** com a coleta e agregação de dados de capacidade das Nuvens integrantes da infraestrutura. Na Figura 1 estão representadas uma Nuvem Privada e outra Pública, mas é possível expandir essa mesma arquitetura para um conjunto maior de diferentes provedores. As informações coletadas são repassadas para o **SLA Enforcer Rules Engine**, que valida a condição para a migração de recursos de uma Nuvem para outra. Por exemplo, se a capacidade de uma das Nuvens chegar a 80%, faz-se *deploy* de novos recursos apenas na outra Nuvem. O **SLA Enforcer Rules Engine** avalia um conjunto de condições que se faça adequado para cada negócio ou situação, validando condições de custo ou tempo de resposta, além da capacidade. O resultado deste processo dispara o gatilho para iniciar as novas implementações através do **Multi-nuvem Deployer/Orquestrator**. Além disso, o **Multi-nuvem Telemetry & Log**

Aggregator também gera subsídio para a bilhetagem através de métricas de utilização que são contabilizadas e disponibilizadas para o usuário através do processo **Multi-cloud Billing**. Além disso, os alertas de erros são tratados e processados pelo **Sistema de Alertas e Notificações**.

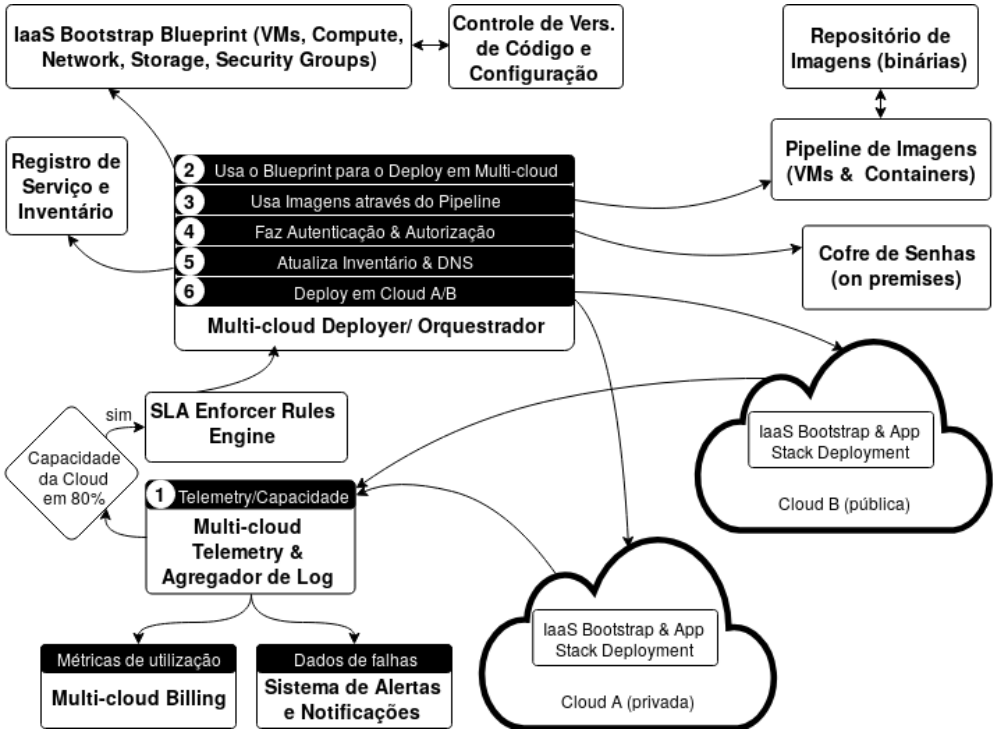

Figura 1. Diagrama de relacionamentos de processos em Multi-nuvem.

As etapas **(2)** e **(3)** do processo **Multi-nuvem Deployer/Orquestrator** utilizam as definições de implementações armazenadas como código em controle de versão (Git, por exemplo). Estas definições de implementações são aplicadas sobre as imagens selecionadas do **Pipeline de Imagens** que podem ser VMs (em formato OVA, VMDK, VHD ou RAW) ou Containers (Docker, por exemplo).

Toda informação sensível, em especial senhas ou chaves, é obtida do **Cofre de Senhas (4)**, que por motivos de segurança e de facilidade de acesso, geralmente está *on-premises* e não em uma das Nuvens que fazem parte da Multi-nuvem.

Os serviços e itens de configuração ativos são registrados através do **Registro de Serviços e Inventário (5)**, assim como o DNS, que é atualizado para refletir os novos registros e rotas.

Alguns processos do **Cloud Deployer/Orquestrator (6)** criam os componentes, definidos nas etapas anteriores, em cada uma das Nuvens especificadas usando as APIs de cada fornecedor. Os *softwares* usados nesta etapa do Cloud Deployer devem possuir uma camada de abstração das diversas APIs para que a implementação seja o mais transparente possível e que os códigos envolvidos possam ser reaproveitados e não sejam complexos, ou difíceis de manter e documentar. Alguns softwares serão apresentados e demonstrados na seção a seguir.

O cenário descrito acima inicia em **Telemetry (1)**, assim possibilitando situações automatizadas de orquestração dos recursos na Multi-nuvem, entretanto, este processo também poderia iniciar manualmente, com instruções sendo passadas diretamente ao **Multi-cloud Deployer/Orquestrator**.

Os processos descritos acima e esquematizados na Figura 1 podem ser bastante complexos, envolver outros subprocessos e também um conjunto diversificado de ferramentas e softwares. Como exemplo, será explorado em mais detalhes o **Pipeline de Imagens**.

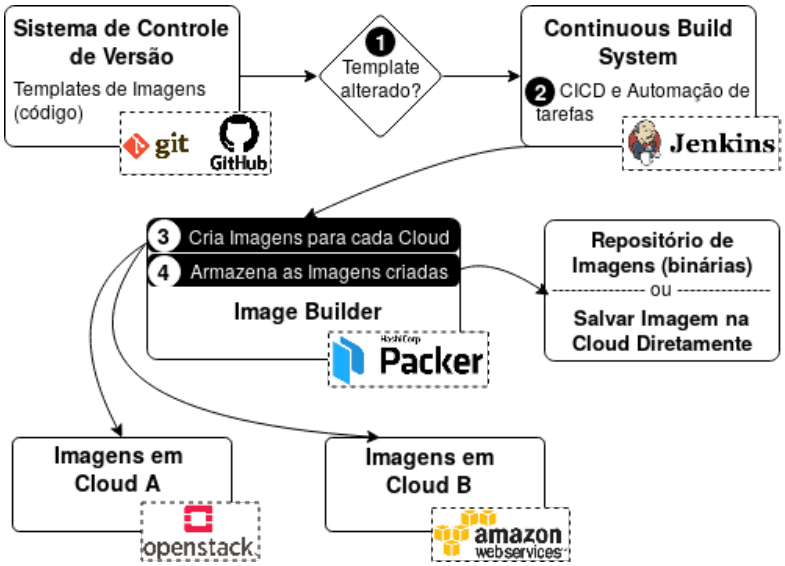

Figura 2. Diagrama do processo de Pipeline de Imagens.

O objetivo do processo da Figura 2 é prover imagens binárias padronizadas de servidores que estão dentro dos padrões para rodar em qualquer Nuvem IaaS que fazem parte da Multi-nuvem. Para alcançar seus objetivos, este processo utiliza produtos *open-source*. Os referenciados na Figura 2 são; **Packer** (https://www.packer.io) que é um framework capaz de produzir múltiplas imagens em formatos diferentes, apropriadas para a maioria dor provedores de Nuvem (Públicas e Privadas). Packer pode ser usado com uma variedade grande de *build systems*, dentre os quais destacamos o **Jenkins** (https://jenkins.io), que é uma plataforma para entrega contínua (*Continuous Delivery*) de *releases* de software em ambiente distribuído.

O processo ilustrado na Figura 2 é iniciado a partir de um *commit* no **Sistema de Controle de Versão** em alguma *branch* (Master por exemplo) que é monitorada pelo **Continuous Build System (1)**. Sempre que uma alteração é feita, o Jenkins executa as rotinas **(2)** que são requisitos para que o Packer (principal software do **Image Builder**) possa gerar as imagens de acordo com os *blueprints* de cada Nuvem **(3)**. O Packer pode armazenar essas imagens para uso futuro em um repositório centralizado (*on-premises* geralmente) ou pode também gravar cada imagem na respectiva Nuvem **(4)**. Essa última alternativa pode ser mais rápida na hora de criar uma instância, pois não precisa fazer a transferência (via *Internet*) da Imagem armazenada *on-premises* para a Nuvem.

5. Validação Experimental - *Deployment* de aplicação em Multi-nuvem

Uma das maneiras de experimentar e explorar o processo de *deploy* de IaaS ou PaaS em Multi-nuvem pode ser através do uso de alguns softwares que automatizam a criação de recursos em Nuvem. O uso das consoles WEB ou outros mecanismos disponibilizados pelos principais provedores de Nuvem, propiciam maior chance de erro humano, e requerem que a documentação seja feita adicionalmente nas fases conclusivas dos projetos. Desta forma, não aproveitam os benefícios da IaC, discutidos anteriormente.

Existem diversas ferramentas que trabalham sob o conceito de IaC e que atendem os requisitos para criação de infraestrutura em Multi-nuvem. Uma delas é um serviço nativo do Openstack, o Heat, que interage com a API do Openstack para criação ou modificação de Infraestrutura. O Heat também consegue atuar em AWS, mas alguns recursos de Infraestrutura ainda não estão definidos e, portanto o seu escopo de atuação, para AWS especificamente, ainda é muito restritivo, fazendo com que ele não seja usado em ambientes de produção corporativos.

Outras ferramentas, tais como Chef, Puppet, Ansible e SaltStack, são muito utilizadas, inclusive em ambientes produtivos para gerenciamento de configuração de servidores, também podem ser usados para criação de recursos e infraestrutura em Nuvens públicas ou Privadas. Estes *softwares*, entretanto, dependem de módulos, bibliotecas ou outros *softwares* adicionais para interagirem com as APIs das diversas nuvens suportadas por essas ferramentas. Outro fator limitante é que nuvens com menor expressividade no mercado tendem a ter menor suporte por parte dessas ferramentas.

Há ainda outra forma de interagir com as APIs das Nuvens, via Software Development Kit - SDK, um exemplo disso é a AWS que utiliza boto3, em Python. Cada provedor de Nuvem disponibiliza seu próprio SDK e eles geralmente diferem muito entre si, o que implica que o SDK de uma Nuvem não pode ser usado em outra.

Das soluções listadas acima, podemos notar algumas características que podem ser limitações para uso em produção em ambientes de Multi-nuvem. Os SDKs da maioria dos provedores de Nuvem atendem apenas as especificações de API do respectivo provedor de Nuvem. Outras ferramentas, como o CloudFormation (que atende somente AWS) não são boas soluções para gerenciar Multi-nuvem com IaC, pois, não favorecem a fácil replicação de implementações, nem o reaproveitamento de código, sua manutenção e suporte. Como descrito anteriormente, a maioria dos provedores de Nuvem não oferecem APIs padronizadas, dificultando muito o desenvolvimento de ferramentas agnósticas. As ferramentas como Chef, Puppet, Ansible e SaltStack têm foco no gerenciamento de configuração de servidores e não oferecem bom suporte e escopo para criação de infra em múltiplos provedores de Nuvem [Morris 2016].

Para gerenciar múltiplas Nuvens simultaneamente, é preciso que o Software possa interagir com as diversas APIs dos provedores de Nuvens (públicas ou privadas) e que conte com suporte o mais amplo possível ao gerenciamento dos recursos das Nuvens. Além disso, é importante que a solução tenha constante manutenção e atualizações, em especial a inclusão rápida das novas funcionalidades ou serviços lançados pelos provedores. Esta característica pode ser medida pela atividade e quantidade de contribuidores do GitHub, e eventualmente usada como critério de comparação com outras ferramentas. De forma complementar, pode-se desejar que a ferramenta tenha uma estrutura de

suporte técnico formal para atender à necessidade eventualmente regulatória de algumas corporações.

Durante a pesquisa para este estudo de caso, identificou-se o Terraform, que é uma solução open-source específica para a criação de infraestrutura em Nuvem (dentro do modelo IaC), como uma promissora possibilidade para atender os principais requisitos descritos anteriormente. Diferentemente de ferramentas como o CloudFormation ou os SDKs, o Terraform é uma solução agnóstica, permitindo-lhe criar infraestrutura em praticamente qualquer ambiente, seja ele em Nuvem, ambiente virtualizado *on-premisses*, Docker, Kubernetes, recursos de infraestrutura de Redes, bancos de dados, dentre outros. Exemplos de provedores suportados são: Amazon AWS, Microsoft Azure, Google GCP, IBM Cloud, Digital Ocean, OpenStack, VMWare, Xen, Virtual Box, F5 BIG-IP, Palo Alto Networks, MySQl ou PostgreSQL (ver www.terraform.io par lista completa). Terraform aind conta com uma sólida e ativa comunidade de contribuidores (https://github.com/hashicorp/terraform). A Hashicorp, empresa que criou o Terraform, oferece suporte, consultorias e treinamento com foco no mercado corporativo.

Na seção a seguir, utiliza-se o Terraform para fazer o provisionamento de um servidor com uma aplicação WEB de teste em duas Nuvens: Openstack e AWS, com o intuito de verificar se o código desenvolvido para os dois provedores atende aos requisitos e características da IaC, conforme listados e discutidos anteriormente.

5.1. Validação Experimental com Terraform

Em https://github.com/arielfrozza/multi-cloud estão disponibilizados os arquivos de código que foram utilizados para a experimentação deste estudo de caso, e que fazem o *deploy* de uma aplicação WEB simples em duas nuvens diferentes; AWS e Openstack. Esse cenário tem o objetivo de representar o uso de um caso particular de Multi-nuvem, que é a Nuvem híbrida. A execução desses códigos pode ser feita mediante instalação do Terraform em *host* com acesso às duas Nuvens, e deve-se ter também as credenciais necessárias para a criação de instâncias. Os detalhes dos códigos e sua implementação não serão descritos neste artigo, mas os detalhes e o passo a passo encontram-se no repositório do GitHub abaixo dos diretórios openstack_tf e aws_tf, para cada uma das duas Nuvens.

O Terraform utiliza como *input* um conjunto de arquivos contendo a definição da infraestrutura, escrita em uma linguagem própria, a ser criada nas duas Nuvens. A arquitetura dessa validação segue o modelo apresentado na Figura 1, mas não de forma integral. O interesse deste estudo é apenas verificar e identificar as vantagens e também as limitações do uso de tal solução à luz dos conceitos de IaC definidos anteriormente. Para tal, é criada uma instância Ubuntu 18.04 em cada uma das Nuvens com subsequente instalação de alguns softwares (Python 2.7 e Flask) e *deploy* de um código HTTP REST simples escrito em Flask.

Dessa forma, não serão abordadas nem discutidas as etapas de Telemetria (e as métricas ou SLAs envolvidos), Pipeline de Imagens, Cofre de Senhas ou Registro de Serviço e Inventário. Estes processos estão indicados pelos números 1, 3, 4, 5 na Figura 1.

A estrutura de código do Terraform (conforme documentada em www.terraform.io/guides) é bem estruturada e pode ser segmentada, permitindo que o código seja dividido em múltiplos arquivos para melhor organização e manutenção.

A Figura 3 lista os arquivos utilizados para este estudo e que estão disponíveis no GitHub. Os arquivos de extensão ".tf"e ".tfvars"contém o código Terraform que interage com as Nuvens para a execução das rotinas. Consta ainda nestes diretórios a aplicação de teste, em Flask (simple_app.py) e o par de chaves (mykey e mykey.pub) para a conexão remota via SSH com as instâncias criadas nas duas Nuvens.

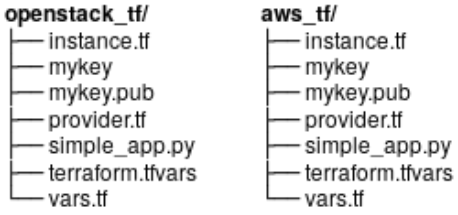

Figura 3. Comparativo da estrutura dos diretórios e listagem de arquivos das duas Nuvens.

É importante, do ponto de vista da IaC, que os dois diretórios ilustrados na Figura 3 tenham a mesma estrutura e relação de arquivos, pois facilita a criação, atualização de documentos e procedimentos operacionais. Além disso, possibilita o reaproveitamento de estruturas inteiras ou de blocos de código, a manutenção e o controle de versões e mudanças através da segmentação em diferentes arquivos com funções melhor definidas.

Openstack

```
provider "openstack" {
  user_name   = "${var.USER_NAME}"
  tenant_name = "${var.TENANT_NAME}"
  password    = "${var.PASSWORD}"
  auth_url    = "${var.AUTH_URL}"
  region      = "${var.REGION}"
  domain_name = "${var.DOMAIN_NAME}"
}
```

AWS

```
provider "aws" {
  access_key = "${var.AWS_ACCESS_KEY}"
  secret_key = "${var.AWS_SECRET_KEY}"
  region = "${var.AWS_REGION}"
}
```

Figura 4. Comparativo de código dos arquivos provider.tf para as duas Nuvens.

Entretanto, ao fazer a comparação do conteúdo de cada arquivo do Terraform (os ".ft"), é possível observar facilmente que os códigos diferem substancialmente, conforme ilustrado nas Figuras 4 e 5. Apesar da estrutura e quantidade de arquivos de código escritos para as duas Nuvens serem iguais, devido ao funcionamento do Terraform, a sintaxe para a criação de recursos pode mudar bastante dependendo do provedor de Nuvem em uso. Por exemplo, os métodos de autenticação são essencialmente diferentes para as duas Nuvens utilizadas neste estudo. O Openstack requer um usuário com senha e também a URL do Keystone, a região e o domínio utilizados, já a AWS requer apenas a chave de acesso e a *secret key* para fazer a autenticação e autorização, conforme ilustrado na Figura 4.

Outro exemplo dessa diferença de sintaxe também pode ser observado na Figura 5 em que apenas um trecho do código que provisiona as VMs é exibido, mostrando claramente que as instruções do Terraform para as duas Nuvens em estudo, diferem muito entre si.

Essas diferenças entre as sintaxes do Terraform para cada provedor de Nuvem, implicam que todos os arquivos sejam parcialmente diferentes no conteúdo, de uma Nuvem para outra, conforme Figura 5. Este fato potencialmente dificulta o controle das mudanças de código e também a reutilização de trechos quando, por exemplo, deseja-se replicar a infraestrutura em questão para um terceiro provedor.

Openstack

```
resource "openstack_compute_instance_v2" "vm01" {
 name          = "vm01"
 image_id      = "fdcecf4e-443a-46e0-84af-af606bb5b08a"
 flavor_id     = "1"
 key_pair      = "keypair_1"
 security_groups = ["default"]
 network {
  uuid = "278a6eb3-a05a-49f4-826f-b978f3833c29"
 }
}
```

AWS

```
resource "aws_instance" "mv01" {
 ami = "${lookup(var.AMIS, var.AWS_REGION)}"
 instance_type = "t2.micro"
}
```

Figura 5. Comparativo de trecho de código dos arquivos instance.tf para as duas Nuvens.

Durante o desenvolvimento dos códigos em Terraform, foi possível obter grandes vantagens com o uso do GitHub para controle de versões no código, pois, cada mudança no código ou incremento, pode ser testado com mais frequência. Também a documentação ficou mas robusta e coerente, pois, a cada novo *commit* feito no GitHub, as anotações de comentário eram obrigatórias e síncronas. O reúso de código também é facilitado quando um versionador de arquivos é usado. No caso do GitHub, um *fork* poderia ser criado para o gerenciamento de infraestrutura, configuração em outra Nuvem ou para a replicação em um ciclo de vida de aplicação; de Desenvolvimento para Homologação, que por fim pode ser promovido para Produção.

No caso de haver interesse na replicação de uma implementação para um terceiro ou vários outros provedor(es) de Nuvem teriam apenas que ser incluídas ou retiradas algumas linhas específicas para aquela Nuvem e seriam alteradas também os valores e nomes das variáveis. Toda a estrutura, relação e quantidade de arquivos permaneceria igual às outras. Portanto, a reutilização de código, mesmo que demandando um certo nível de esforço, ainda pode ser feita.

O Terraform também mostrou-se muito eficaz em abstrair para o usuário, as APIs das duas Nuvens estudadas. Em nenhuma etapa da escrita ou execução dos códigos foi necessário interagir ou escrever código que interagisse diretamente com as APIs das Nuvens em estudo. Isso facilita a escrita e documentação dos códigos.

No caso de uso das SDKs (ou outros serviços como Heat e CloudFormation) das Nuvens, todo o código para interagir com as APIs das Nuvens seriam totalmente diferentes entre si, e nada poderia ser reaproveitado nos casos de extensão de serviços para um provedor de Nuvem adicional (e diferente das demais).

6. Conclusão

Multi-nuvem está sendo cada vez mais utilizada em ambientes corporativos, especialmente no formato de Nuvem híbrida, sendo uma ótima solução para que elas possam ofertar serviços WEB tolerantes a falhas, com ótima performance e com custos eficientes.

A concorrência em alguns setores de serviços WEB e a crescente exigência do usuário por serviços rápidos, disponíveis e baratos também pode impulsionar ainda mais o uso de Multi-nuvem. Porém, ainda há pouca disponibilidade de soluções agnósticas, completas e robustas o bastante no mercado, capazes de gerenciar recursos em diversos provedores de Nuvem Pública ou Privadas simultaneamente.

Os principais provedores de Nuvem parecem se proteger do mercado Multi-nuvem ao criarem suas APIs de forma fechada. A não adesão a padrões de Nuvem API, como as propostas por OASIS [CAMP 2019] (à qual o OpenStak é uma das poucas aderentes) torna muito difícil o desenvolvimento de soluções agnósticas que se propõem a resolver o problema do gerenciamento de recursos e infraestrutura em Multi-nuvem.

Durante a validação experimental deste estudo, identificou-se o Terraform como uma das mais completas e robustas soluções para o caso, por ser uma solução agnóstica, desenvolvida para possibilitar o uso sob os princípios da Infraestrutura como Código, onde o versionamento (em GitHub neste estudo) e o reaproveitamento de código são valores importantes. O Terraform facilita o estabelecimento de processos padronizados e estruturados de desenvolvimento de código, por conseguir abstrair para o usuário, as APIs dos diferentes provedores de Nuvem. Idealmente, o usuário poderia trabalhar e manter apenas um código de provisionamento em diversas Nuvens, entretanto, foi possível identificar que os códigos do Terraform apresentam substanciais diferenças de um provedor para outro, o que dificulta o reaproveitamento de código.

Apesar das limitações com relação ao reaproveitamento de código, o Terraform pode ser mais eficiente e eficaz que o uso de SDKs ou outras soluções, como Heat ou CloudFormation, por serem muito específicas para determinados provedores de Nuvem. Esta característica impossibilita totalmente o reaproveitamento de código, assim como a replicação ou migração de implementações entre provedores de Nuvem diferentes. Em particular, no caso de uso para Multi-nuvem com grande número de provedores participantes, o uso de SDKs ou ferramentas específicas pode gerar muita dificuldade no desenvolvimento, manutenção, testes e implementação dos códigos.

Referências

Bond, J. (2015). *The Enterprise Cloud: Best Practices for Transforming Legacy IT*. O'Reilly Media.

CAMP, O. (2019). Oasis cloud application management for platforms.

CloudSpectator (2017). Top 10 cloud iaas providers benchmark.

Ferrer, A. J., Hernadez, F., et al. (2012). Optimis: A holistic approach to cloud service provisioning. 28:66–77.

Fisher, A. (2018). *Multi-Cloud Patterns, Best Practicies & Framework: Focusing on Deployment Management, Fault Tolerance, Security, Self-Healing, Cost Efficiency and Vendor Neutrality*. OReilly Media.

Golden, B. (2013). *Amazon Web Services for Dummies*. John Wiley & Sons.

Morris, K. (2016). *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media, 15th edition.

TOSCA, O. (2019). Oasis topology and orchestration specification for cloud applications.