Multi-Cloud

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

[1]Instituto de Informática – Universidade Católica do Paraná (PUC-PR)
Caixa Postal 15.064 – 91.501-970 – Curitiba – PR – Brazil

arielfrozza@gmail.com, roxsnd@gmail.com

Abstract. *This meta-paper describes the style to be used in articles and short papers for SBC conferences. For papers in English, you should add just an abstract while for the papers in Portuguese, we also ask for an abstract in Portuguese ("resumo"). In both cases, abstracts should not have more than 10 lines and must be in the first page of the paper.*

Abstract. *Neste trabalho exploraremos os conceitos de Multi-Cloud e Infrastructure as Code, bem como suas possíveis utilizações em ambientes corporativos. Também descreveremos as dificuldades na implementação de Multi-Cloud e elencaremos possíveis soluções para as dificuldades descritas. Por fim, apresentaremos um roteiro de uso das ferramentas OpenSource Terraform e Ansible para demonstrar a implementação de um serviço web em duas Nuvens simultaneamente, uma pública (AWS) e uma privada (Openstack).*

1. Introdução

Atualmente, o uso simultâneo de serviços e recursos de múltiplos provedores de Cloud 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 ficar aprisionados à fornecedores ou soluções fechadas (*vendor lock in*).

Este artigo pretende descrever alguns desafios do uso simultâneo de mais de um provedor de Nuvem e levantar alguns dor principais pontos de atenção na utilização das atuais soluções que buscam atender as necessidades dos usuários de múltiplas Nuvens. Primeiramente, a definição e a necessidade de um ambiente em Multi-cloud é discutida, em seguida, um modelo padrão de arquitetura para Multi-cloud é apresentado. A partir desta arquitetura de referência, algumas soluções (software) que tornam possíveis a Infraestrutura como Código em Multi-cloud são apresentados e discutidos. Por último, ilustraremos, a partir de um exemplo prático (disponível em https://github.com/arielfrozza/multi-cloud) os processos, requisitos e desafios na implementação de IaaS em Multi-cloud usando ferramentas e princípios típicos de Infraestrutura como Código.

Este artigo não apresenta abordagem inovadora, entretanto, ele intenta apontar algumas lacunas e dificuldades a serem sanadas por desenvolvedores e usuários de ambientes em Multi-cloud. Em especial, discutiremos frameworks e padrões de provisionamento em Nuvem (pública, privada ou híbrida) que ofereçam APIs que disponibilizam funcionalidades compatíveis ou equivalentes com os softwares, serviços e APIs ofertados pelos

principais provedores de Nuvem, de modo que o provisionamento, orquestração e uso de vários provedores de Núvens diferentes seja facilitado.

2. Definição e Necessidade de Multi-clouds

Clouds podem ser usadas de forma serial, por exemplo, quando migramos de uma Cloud para outra, ou de forma simultânea, quando usamos recursos de duas ou mais Clouds diferentes ao mesmo tempo. O senário mais comum para o caso de uso simultâneo é a Cloud Híbrida, quando alguns serviços são disponibilizados a partir de uma Nuvem (Privada) e outros seriços estão em uma Nuvem Pública.

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 Cloud:

Tolerância à falhas (fault-tolerance) por envolver uso de mais de um provedor de Nuvem simultaneamente, permitindo movimentações de contingência no caso de indisponibilidade de um dos provedors, por exemplo.

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

Performance como habilidade para ajustar os níveis de carga para outros provedores de Cloud em caso de degradação de performance ou qualidade de serviço em um dos provedores.

Segurança

Eficiencia de custos como abilidade de aproveitar os melhores preços de recursos computacionais e serviços em um ambiente multi-cloud.

Os custos de operação em um determinado provedor Cloud variam em função dos requisitos de uso e da época de contratação dos serviços assim como os custos de trocar de provedor podem ficar muito altos caso a infraestrutura e/ou aplicações tenham que ser refeitos/revistos 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 reflexo da redução dos custos operacionais dos provedores de Nuvem ao longo do tempo. De 2011 para 2013, ocusto por hora do AWS EC2 baixou cerca de 30% [Golden 2013]. Com o uso de padrões *vendor neutral*, o objetivo seria ser capaz de mudar de provedor de cloud ou priorizar a execução on-premises de acordo com a conveniência, sem ter que alterar o software stack.

Muitos termos são encontrados na literatura para designar o uso simultâneo de duas ou mais clouds. 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, Distributed Clouds, etc.

Sendo assim, achamos útil delimitar o conceito de Multi-cloud abordado neste artigo de outros modelos de uso combinado de Clouds.

De acordo com [Ferrer et al. 2012], temos dois modelos de entraga de serviços em

múltiplas clouds; Federated Cloud e Multi-Cloud. A diferença entre estes modelos seria o grau de colaboração entre os Provedores de Cloud e pelo mode pelo qual os usuários interagem com as Clouds. No primero modelo há o acordo de uso compartilhado dos recursos entre os Provedores de Cloud. Os usuários de uma núvem federada não ficam sabendo, na maioria dos casos, de qual dos provedores um determinado recurso está sendo consumido. No caso do Multi-cloud, o usuário está ciente e é restponsável pela alocação de recursos em um ou outro provedor e não há nuência dos provedores para uso compartilhado de recursos entre os provedores.

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

O tipo mais comum de Multi-cloud é a Cloud-Hibrida na qual involve duas ou mais clouds, por exemplo, uma Cloud Privada e uma Pública. Comumente, esse modelo de Cloud Híblrida é usado para "Cloud Bursting"onde os recursos são espandidos para a Cloud Pública quando os recursos da Cloud privada chegam nos níveis máximos prédefinidos [Ferrer et al. 2012]. A migração de uma Cloud para outra, mesmo que apenas uma vez, é outro exemplo de uso de (one-time) Multi-cloud.

Um dos maiores problemas é a interoperabilidade entre diferentes Cloudse a portabilidade de aplicações entre diferentes provedores.

Entretanto, a interoperabilidade entre os diversos serviços ofertados e orquestração de recursos em múltiplas Nuvens (usando Infraestrutura como Código [Morris 2016], por exemplo) ainda está distante da maturidade, já que, por exemplo, todos os provedores de Cloud experimentam, eventualmente, períodos de indisponibilidade que podem causar impactos a negócios [Fisher 2018]. 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].

A seguir descreveremos um framework para o uso de Multi-cloud envolvendo duas Clouds, uma Privada e outra Pública.

3. Arquitetura de Referência para Multi-nuvem

A seguir exploramos um padrão de implementação de Multi-nuvem, conforme ilustrado na Figura1, que propõem ajudar a atender dois casos comuns de uso de multi-cloud:

Trasnbordo para Nuvem Pública (*Busting Pattern*): em algumas situações, a capacidade on premises de uma empresa pode exaurir e pode-se querer usar a capacidade em uma cloud pública para suprir uma determinada demanda.

Alta Disponibilidade: Eventualmente, um provedor de Cloud pode ficar (parcial ou totalmente) indisponível ou com degradação na qualidade dos serviços por diversos motivos, pode-se querer migrar de um provedor para outro os serviços e recursos para não impactar o usuário final. Por exemplo, em caso de falha ou manutenção do datacenter

que hospeda a Cloud privada, pode-se migrar serviços para uma Cloud pública, mesmo que temporariamente.

Eficiência de Custos: Em alguns casos, provedores oferecem descontos ou modificam seus preços em função da oferta e demanda, pode-se migrar uso de recursos e serviços de uma Cloud 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, os quais não serão diretamente abordados neste artigo, mas podem ser entendidos de forma mais completa em [Fisher 2018].

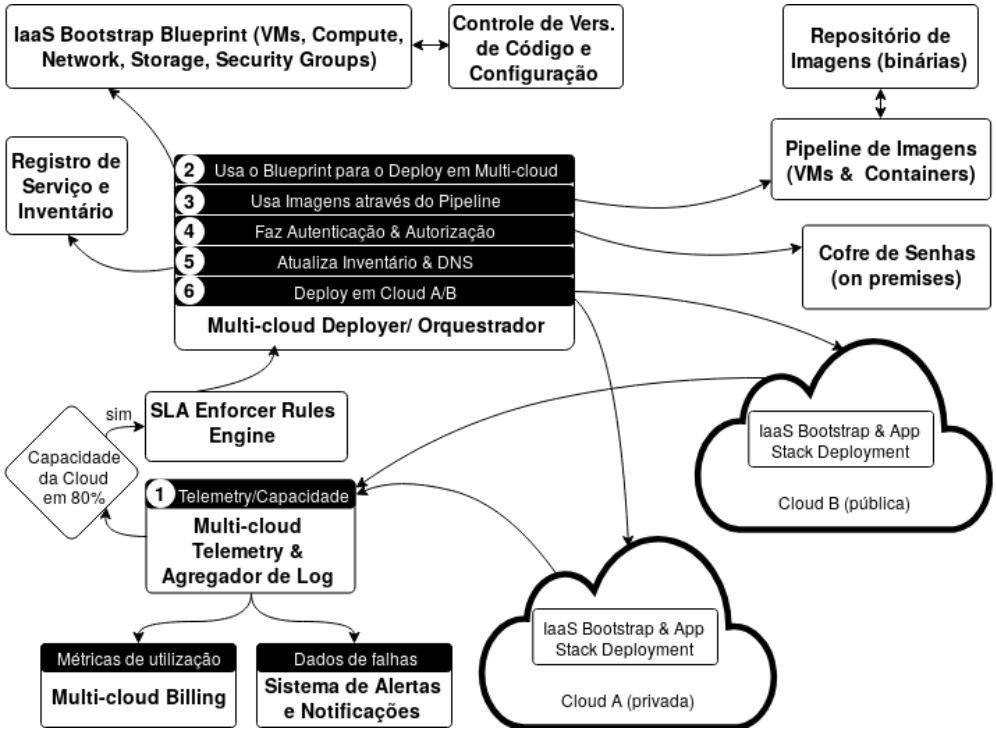

Figura 1. legenda longa figura1

Multi-cloud Telemetry (1) coleta e agrega dados de capacidade das Clouds integrantes da Multi-cloud. Na Figura estao representadas uma Cloud Privada e outra Publica, mas poderiamos 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 Cloud para outra, como exemplo, se a capacidade de uma das clouds chegar a 80%, faz-se deploy de novos recursos apenas na outra Cloud. O **SLA Enforcer Rules Engine** pode avaliar um conjunto de condições que se faça adequado para cada negócio ou situação, por exemplo ele pode avaliar condições de custo ou tempo de resposta, além da capacidade. A avaliação 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 disponibilidadas para o usuário através do processo **Multi-cloud Billing**.

Também os alertas de erros são tratados e processados pelo processo **Sistema de Alertas e Notificações**.

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

Toda informação sensível, em especial senhas ou chaves são obtidas 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-cloud.

Os serviços e itens de configuração ativos são registrados no **Registro de Serviços e Inventário (5)**, bem como o DNS é atualizado para refletir os novos registros e rotas.

Alguns componentes 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 diversaas 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 demosntrados na seção a seguir).

O cenário descrito acima inicia em **Telemetry (1)**, assim possibilitando situações automatizadas de orquestração dos recursos na Multi-cloud, entretanto, um humano também poderia iniciar esse processo, passando instruções diretamente ao processo **Multi-cloud Deployer/Orquestrator**.

Os processos descritos acima e esquematizados na Figura1 podem ser bastante complexos e envolver outros subprocessos e também um conjunto diversificado de ferramentas e softwares. Como exemplo, exploremos em mais detalhes o **Pipeline de Imagens**.

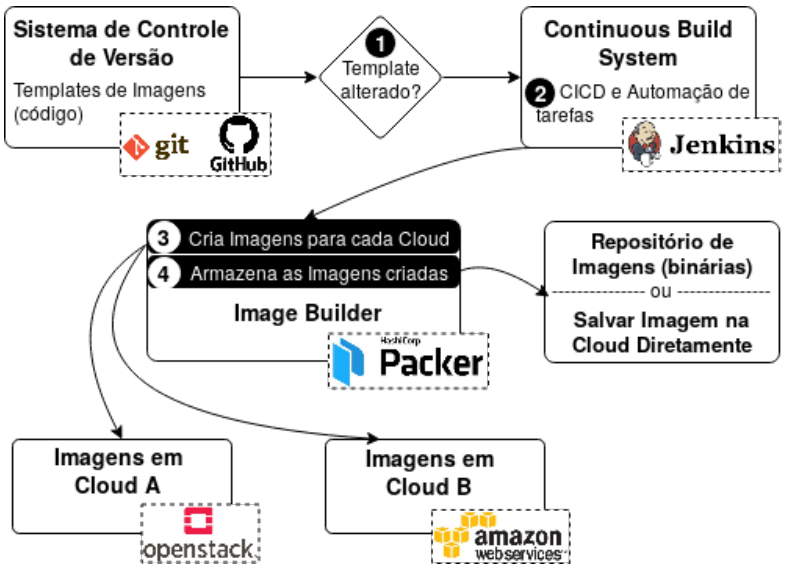

Figura 2. legenda longa figura2

O objetivo do processo da Figura2 é prover imagens binárias de servidores que estão dentro dos padrões para rodar em qualquer Cloud IaaS que fazem parte da Multi-cloud.

Este processo de criação padronizada de imagens pode alcançar seus objetivos utilizando produtos open-source. Os referenciados na Figura2 são; **Spinnaker** (http://www.spinnaker.io) que é uma plataforma para entrega contínua (*Continuous Delivery*) de *releases* de software em ambiente Multi-cloud. **Packer** (https://www.packer.io) é um framework capaz de produzir multiplas imagens em formatos diferentes, apropriadas para a maioria dor provedores de Cloud (Públicas e Privadas). Packer pode ser usado com uma variedade grande de *build systems*, dentre os quais destacamos o **Jenkins** (https://jenkins.io).

ver 829 e 2363

O processo ilustrado na Figura2 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 Cloud **(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 Cloud **(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 Cloud.

4. Validação Experimental - Application Deployment in Multi-Cloud Environment

Uma das maneiras de experimentar o processo de deploy de IaaS e PaaS em Multi-cloud pode ser feito através do uso de alguns softwares que automatizam a criação de recursos em Cloud. O uso das consoles WEB ou SDKs disponibolizadas pelos principais provedores de Cloud propicia maior chance de erro humano, requer que a documentação seja feita adicionalmente e não aproveita os benefícios da Infraestrutura como Código, discutida a seguir.

4.1. Infraestrutura como Código com Terraform

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 (ou nenhuma em alguns casos) para criar ou modificar recursos em diferentes ambientes (Produção, Teste, Desenvolvimento) Clouds (GCP, Azure ou AWS por exemplo). Uma vez que a infraestrutura está codificada de forma estruturada e geralmente em módulos, o código pode ser

reaproveitado. O fato da infraestrutura estar definida toda em código, e este estar **versionado** em ferramentas como Git, possibilita **maior controle nas aplicações de mudança** nos ambientes possibilitando, inclusive, a **recuperação de desastres**. O versionamento do código em Git, também possibilita a simplificação da **documentação do código**, bem como 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.

Existem diversas ferramentas que trabalham sob o conceito de IaC, uma delas é nativa do Openstack, o Heat, esse serviço interage com a API do Openstack para criação ou modificação de Infraestrutura. Heat também consegue atuar em AWS, mas alguns recursos de Infraestrutura ainda não estão definidos no Heat e portanto ainda não podem ser usados.

Outras ferramentas, como Chef, Puppet, Ansible e SaltStack que são muito utilizadas para gerenciamento de configuração de servidores também podem ser usados para criação de recursos e infraestrutura em Nuvens públicas ou Privadas. Estes softwares, noentanto, dependem de módulos, bibliotecas ou outros softwares adicionais para "conversarem" com as APIs das diversas nuvens disponíveis no mercado. Nuvens com menor expressividade no mercado tendem a ter menor suporte por parte dessas ferramentas.

Outra forma de intergir com as APIs das Nuvens é via Software Development Kit -SDK (por exemplo, a AWS utiliza boto3, em Python) e cada provedor de Cloud disponibiliza seu próprio SDK e geralmente diferem muito entre si, e o SDK de uma Cloud 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-cloud. As SDks da maioria dos provedores de Nuvem atende apenas as especificações de API do respectivo provedor de Cloud (bem como outras ferramentas, como o CloudFormation, que atende somente AWS) e portanto não são boas soluções para gerenciar multi-cloud com IaC, pois, não favorecem a fácil replicação de implementações, o reaproveitamento de código, bem como sua manutenção e suporte. Como descrito na Introdução, a maioria dos provedores de Cloud não oferecem APIs padronizadas, dificultando muito o desenvolvimento de ferramentas agnósticas. As ferramentas como Chef, Puppet, Ansible e SaltStack têm foco no gerenciamento de confiração de servidores e não oferecem bom suporte e escopo para criação de infra em múltiplos provedores de nuvem [Morris 2016].

Para gerenciar múltiplas Nuvens, é preciso que o Software possa interagir com as diversas APIs dos provedores de Cloud (publicas e privadas) e com suporte o mais amplo possível ao gerenciamento dos recursos das Clouds. Além disso, é importante que a solução tenha constante manutenção e atualizações (pode ser medido pela atividade e quantidade de contribuidores do GitHub) e tenha uma estrutura de suporte técnico formal (pago) para atender à necessidade (eventualmente regulatória) de algumas corporações.

4.2. validação Experimental com Terraform

O Terraform, é uma solução open-source específica para a criação da infraestrutura em Nuvem (dentro do modelo IaC), diferente do CloudFormation ou SDKs, o Terraform é uma solução agnóstica, permitindo-lhe criar infraestrutura em praticamente qualquer ambiente, seja ele em Cloud (Amazon AWS, Microsoft Azure, Google GCP, IBM Cloud,

Digital Ocean, etc.), ambiente virtualizado, local ou em Data Centers (VMWare, Xen, Virtual Box, etc.), Docker, Kubernetes, além de recursos diversos de infraestrutura e softwares, tais como Redes (F5 BIG-IP, Palo Alto Networks,etc.), Bancos de Dados (MySQl ou PostgreSQL), dentre outras (ver www.terraform.io). Terraform conta com uma sólida e ativa comunidade de contribuidores (https://github.com/hashicorp/terraform) e a Hashicorp, empresa que criou o Terraform, oferece suporte, consultorias e treinamento com foco no mercado corporativo.

Em https://github.com/arielfrozza/multi-cloud está o código que faz o deploy de uma aplicação WEB simples em duas Clouds diferentes; AWS e Openstack. Esse cenário tem o objetivo de representar o uso de um caso particular de Multi-cloud, que é a cloud híblida. A execução desses códigos pode ser feita mediante instalação do Terraform em host com acesso às duas nuvens, 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 passo a passo encontran-se no repositório 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 a ser criada nas duas Clouds. A arquitetura dessa validação segue o modelo apresentado na Figura1, mas não de forma integral. Nosso interesse é apenas verificar e identificar as vantagens e limitações do uso de tal solução à luz dos conceitos de IaC definidos anteriormente, para tal nos concentraremos na criação de uma instância Ubuntu 16.04 em cada uma das Clouds seguido de instalação de alguns softwares (Python 2.7 e Flask) e deploy de um codigo HTTP RESP (escrito em Flask) simples.

Durante o desenvolvimento dos códigos, pudemos obter grandes vantagens com o uso do GitHub para controle de versões e mudanças no código. O código pode ser testado com mais frequencia, a cada incremento no código. Também a documentação ficou mas robusta e coerente, pois, a cada novo commit feito, as anotações de comentário eram obrigatórias e sincronas.

Pudemos notar também que, apezar da estrutura e quantidades de arquivos de código escritos para as duas Nuven serem iguais, devido ao funcionamento do Terraform, a sintaxe para a criação de recursos, o método de autenticação são substancialmente diferentes para as duas Clouds, o que acabou causando que todos os arquivos diferissem parcialmente de uma Nuvem para outra, o que dificulta o controle e reuso direto de código.

Por outro lado, o reuso ainda pode ser feito mediante poucos ajustes. No caso te termos que replicar essa inplementação em um terceiro provedor de Nuvem (Google GCP ou Azure, por exemplo) teríamos apenas que incluir ou retirar algumas linhas específicas para aquela Nuvem e mudaría-mos os valores e nomes das variáveis. Toda a estrutura, relação e quantidade de arquivos permaneceria igual às outras.

Em nenhuma etapa da escrita ou execução dos códigos tivemos que interagir ou escrever código que interagisse diretamente com as APIs das Nuvens em estudo, isso nos facilitou a escrita e documentação dos códigos.

No caso de uso das SDKs (ou outros serviços como Heat e CloudFormation) das Nuvens, todo o código seria diferente e nada poderia ser reaproveitado para um provedor de Nuvem diferente. Usar uma solução agnóstica como Terraform, facilita o estabelecimento de processos padronizados e estruturados de desevolvimento de código e possibilita

melhor interação e troca de conhecimento e experiências entre pessoas e times.

5. Conclusion

- multi nuvem é uma ótima solução para que empresas obtenham fault-torerance, performance, security and cost efficiency para suas ofertas de serviço na Web.

- Há diversas soluções para IaC em Nuvem Pública ou Privadas, mas poucas são agnósticas, completas e robustas o bastante para atender as necessidades de empresas que utilizam multi-nuvem.

- Na validação experimental pudemos observar que o versionamento de código com ferramentas como Git são fundamentais para IaC e que o Terraform se mostrou uma solução agnóstica muito boa para trabalharmos em muti-cloud com IaC.

- O uso de SDKs ou outras soluções, como Heat ou CloudFormation, específicas para provedores de Nuvem específicos dificulta ou impossibilita o reaproveitamento de código e a replicação de implementações entre provedores de Nuvem diferentes. No caso de uso para multi-nuvem com grande numero de provedores participantes, o uso de SDKs e ferramentas específicas pode gerar muita dificulde de controle para manutenção, testes e implementação dos códigos.

- Os principais provedores de nuvem parecem se proteger do mercado em multi-nuvem por não aderirem a padrões de Nuvem API, como as propostas por OASIS, a qual o OpenStak é uma das poucas aderentes.

- A concorrência em alguns setores de serviços Web e a crescente exigência do usuário final de serviços de alta qualidade (rápidos, disponíveis e baratos) pode impulsionar alguns setores para o uso de multi-nuvem.

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.