AZOS - Uma ferramenta para migração em *multi-clouds**

Raul Leiria[1], Vinícius Gubiani[1], Fabricio Cordella[1], Thiago Nascimento[1], Miguel da Silva[1], Gabriel Susin[1], Marcelo Drumm[1], Tiago Ferreto[1]

[1] Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil

{raul.leiria,vinicius.gubiani,fabricio.cordella}@acad.pucrs.br

{thiago.nascimento,miguel.silva.001}@acad.pucrs.br

{gabriel.susin,marcelo.drumm}@acad.pucrs.br

tiago.ferreto@pucrs.br

Abstract. *Cloud computing paradigm brought to companies and end-users more flexibility in costs, online services and hosting. The possibility in migrate services and infrastructures to the cloud is an attractive, however the lock-in caused by cloud computing providers usually is not convenient to the customers at latency or cost reasons. Therefore, this work presents the AZOS tool for migrating instances in multi-clouds. It was performed a study case where instances were migrated in a multi-cloud composed by OpenStack and Microsoft Azure clouds.*

Resumo. *O paradigma da computação em nuvem possibilitou às empresas e usuários finais maior flexibilidade em custos, serviços e hospedagens online. A possibilidade de migrar serviços e infraestruturas para a nuvem é um atrativo, porém nem sempre o lock-in causado por os provedores de nuvem é interessante a seus clientes, seja por questões de custo ou latência. Em razão disso, este trabalho apresenta a ferramenta AZOS para migração de instâncias em multi-clouds. Foi realizado um estudo de caso onde instâncias foram migradas em uma multi-cloud formada por nuvens OpenStack e Microsoft Azure.*

1. Introdução

Nos últimos anos a popularização da Internet elevou a demanda por serviços *online* [Ren and Zhai 2014]. Ao invés de arquivos e processamentos serem mantidos e feitos localmente em computadores, a tendência é que dispositivos locais, bem como computadores *desktops*, *laptops* e *smartphone*s, sejam utilizados em sua maior parte como terminais de acesso [Jin and Lin 2012]. O paradigma da computação em nuvem possibilita contratar recursos computacionais sob demanda através do sistema *Pay-As-You-Go*. Entretanto, apesar das vantangens desse modelo os clientes podem sofrer o efeito de *lock-in* em relação ao seu provedor de nuvem, o qual os coloca sob regras e preços determinados por ele. As *multi-clouds* possibilitam aos usuários manter suas infraestruturas e serviços em provedores de nuvem distintos [Petcu 2013], todavia a portabilidade entre eles ainda é um desafio em razão dos diferentes padrões e tecnologias utilizados.

*Este trabalho foi apoiado pelo Programa PDTI, financiado pela Dell Computadores do Brasil Ltda (Lei 8.248/91).

A ferramenta proposta neste trabalho denomina-se AZOS e possibilita a migração de instâncias em *multi-clouds* pertencentes ao modelo IaaS (*Infrastructure-as-a-Service*). Seu objetivo é evitar o *lock-in* ocasionado por provedores de nuvem e garantir ao usuário a mobilidade entre eles. As entidades envolvidas no funcionamento desta ferramenta são nuvem de origem, nuvem de destino e usuário de âmbas as nuvens. Primeiramente é realizado um processo de *discovery* na nuvem de origem para gerar um *dump* do *tenant* do cliente contendo descrições do ambiente e de suas instâncias. Por segundo é realizado o processo de *build* que efetua a leitura do arquivo de descrições gerado pelo *discovery* e orquestra a nuvem do cliente no provedor de destino. Os processos de *discovery* e *build* operam sem concomitância e possibilitam ao usuário usufruir de melhores preços e condições conforme ofertas efêmeras dos provedores, graças à portabilidade adquirida com a migração de instâncias entre eles.

A partir da Introdução (seção 1) o presente trabalho está estruturado como segue. Na seção 2 está a fundamentação onde é possível encontrar conceitos chaves para o entendimento das seções seguintes. Na seção 3 estão os trabalhos relacionados a este e por seguinte na seção 4 está a descrição detalhada da ferramenta. Na seção 5 está um estudo de caso realizado em uma *multi-cloud* composta por as nuvens OpenStack [Foundation 2017] e Microsoft Azure [Microsoft 2017]. Na seção 6 está a validação da ferramenta, e por fim na seção 7 estão a conclusão e os trabalhos futuros.

2. Fundamentação

A computação em nuvem envolve diferentes níveis de abstrações e tecnologias. Sua base advém da virtualização que possibilita a divisão dos recursos dos servidores em instâncias para ser possível suportar múltiplos clientes. Em outras palavras, a virtualização permite a execução de sistemas operacionais sobre o mesmo *hardware* de forma concomitante. Sistemas operacionais podem ser virtualizados utilizando uma camada de *software* chamada *hypervisor*, que intermedia a comunicação entre sistemas operacionais *guests* e o *hardware* real. Os *hypervisors* são classificados de acordo com a sua proximidade do *hardware* (tipo 1 ou 2) [Tanenbaum and Bos 2014] e em relação a sua técnica de virtualização (paravirtualização, virtualização completa ou assistida por *hardware*). Dispositivos de entrada e saída costumam ser emulados ao passo que a interação com o processador pode ser realizada por meio de tradução binária, *hypercalls* ou utilizando as extensões de virtualização presentes em Unidades de Processamento Central (CPUs) que ofereçam suporte à tecnologia Intel VT-x ou AMD-V.

Em herança à virtualização, os gerenciadores de nuvem podem utilizar a migração do tipo *live* para migrar instâncias (máquinas virtuais). A maioria dos *hypervisors* oferece esse tipo de migração de forma nativa, bastando ao gerenciador de nuvem oferecer apenas suporte a ela. Nesse tipo de migração é necessário haver um *block storage* compartilhado entre os servidores de *compute* (*hypervisors*) para que somente sejam transferidos de um *hypervisor* para outro os dados em memória principal referentes à instância migrada. Ao contrário da migração do tipo *live*, a migração do tipo *offline* não requisita que os processadores dos servidores de *compute* (virtualização) possuam a mesma arquitetura e tampouco que os *hypervisors* sejam iguais. Essas duas limitações muitas vezes inviabilizam a portabilidade entre diferentes provedores de nuvem, o que leva à migração *offline* a ser a mais propícia para esses casos.

As plataformas de nuvem fornecem a sensação de que não há limites para expansão de *hardware*, fazendo com que muitas vezes o usuário expanda tanto sua infraestrutura virtual a ponto de se tornar inviável uma migração para outro provedor, seja por a sua extensão ou por a incompatibilidade entre eles. Nesses casos, geralmente os usuários acabam optando por realizar uma extensão de sua infraestrutura para outras plataformas de nuvem, porém mantendo ainda contrato com mais de um provedor. Esse cenário especifica o conceito de *multi-cloud*, onde infraestruturas virtuais independentes são mantidas em diferentes provedores de nuvem. As *multi-clouds* em conjunto com ferramentas para migração possibilitam portabilidade entre diferentes plataformas de nuvem, minimizando assim o efeito de *lock-in* ocasionado por elas.

Em vista das *multi-clouds* serem nuvens independentes, elas seguem os mesmos modelos de implantação e de serviço da computação em nuvem. Os modelos de serviço proporcionam diferentes níveis de operação para usuários nos serviços ofertados por as plataformas de nuvem. Nesse sentido, os modelos mais significativos são IaaS (*Infrastructure-as-a-Service*), PaaS (*Platform-as-a-Service*) e SaaS (*Software-as-a-Service*) [Buyya et al. 2013]. O modelo IaaS disponibiliza infraestrutura como serviço, geralmente servidores virtualizados, por meio da virtualização de *hardware*, *storage* e rede. O PaaS por sua vez fornece ambientes para execução, processamento e desenvolvimento de aplicações, geralmente é utilizado por programadores. Já o modelo SaaS é o mais próximo do usuário final e oferece serviços que normalmente são acessíveis via *Application Program Interface* (APIs) ou por o navegador de Internet. Exemplos disso são redes sociais e Internet *bankings*.

Ao passo que os modelos de serviço se referem ao nível de operação dos usuários numa pilha de *software*, os de implantação se referem ao local onde a nuvem está hospedada. Quando hospedada num *data center* próprio da empresa trata-se de uma nuvem privada. Quando em *data centers* de terceiros nuvem pública e quando existe a união de recursos entre nuvens privadas e públicas trata-se de uma nuvem híbrida. Nuvens privadas são interessantes para empresas que priorizam baixa latência e confidencialidade de seus dados, pois no provedor de nuvem pública existe a possibilidade de vários clientes estarem compartilhando o mesmo servidor físico, e caso a segurança de algum deles seja comprometida pode haver o comprometimento da dos demais. Cabe aos *softwares* gerenciadores de nuvem manter o isolamento dos clientes em seus respectivos *tenants*.

Atualmente, dentre os mais populares gerenciadores de nuvem *open-source* está o OpenStack. Esse gerenciador é modularizado e por isso possui seus serviços de forma independente. O Keystone é o seu serviço de identificação, ele autentica e autoriza o acesso para os demais serviços. Imagens e discos virtuais (VHDs) de sistemas operacionais são gerenciados pelos serviços Cinder e Glance, respectivamente. A gerência do *hypervisor* e do ciclo de vida das máquinas virtuais (instâncias) é efetuada pelo serviço Nova *compute*, enquanto que os recursos de rede são gerenciados por o serviço Neutron. O Microsoft Azure é popularmente conhecido por ser uma nuvem pública, entretanto ele também é um gerenciador de nuvem que possui serviços com funcionalidades equivalentes as do OpenStack. Não estão explícitas nomenclaturas para eles e tampouco detalhamentos acerca de seu funcionamento nas literaturas consultadas. Todavia, ambos gerenciadores de nuvem possibilitam acesso aos seus serviços via interfaces de comandos, o que flexibiliza o desenvolvimento de ferramentas para interagir com eles.

Ambientes de nuvem e suas instâncias costumam ser descritos por arquivos de *template*. Esses arquivos contém parâmetros e dados acerca da topologia da nuvem, de seus recursos e *tenants*. Há diferentes implementações para arquivos de *template*, em geral todas objetivam a orquestração, todavia costumam ser incompatíveis entre si. Há propostas de unificação de formatos como por exemplo o formato TOSCA [Binz et al. 2013], mas a maioria das plataformas de nuvem ainda utiliza formatos próprios de *template*. Casos conhecidos são a Microsoft Azure, que utiliza o formato *Azure Resource Manager*, o serviço de orquestração do OpenStack que utiliza o formato *OpenStack HOT* e a Amazon que utiliza o formato *AWS Cloud Formation*. Apesar de serem implementações diferentes, dentre esses três o formato da Amazon é aceito em outras plataformas de nuvem como a OpenStack.

Neste trabalho será efetivada uma *multi-cloud* utilizando o modelo de serviço IaaS e os modelos de implantação nuvem privada para o gerenciador de nuvem OpenStack e nuvem pública para o Microsoft Azure. O OpenStack utilizará o *hypervisor* KVM (*hypervisor* tipo 1 e virtualização assistida por *hardware* [Chirammal et al. 2016]) [Kivity et al. 2007] em conjunto com o emulador de periféricos QEMU [Bellard 2005]. A ferramenta AZOS utilizará em seus processos de *discovery* e *build*, migração *offline* e *templates* para orquestração da nuvem, de suas instâncias e de sua topologia.

3. Trabalhos Relacionados

[Katzmarski et al. 2014] apresentam uma técnica e prova de conceito acerca da migração *offline* de máquinas virtuais entre o OpenStack e dispositivos locais, podendo esses serem de arquitetura x86 e ARM. Para isso foi desenvolvido um *plugin* para o OpenStack que adiciona duas rotinas à API do *compute*, sendo elas para arquivar e restaurar máquinas virtuais. O processo para arquivar a máquina virtual (VM) consiste em o usuário enviar uma requisição ao *compute* node, que por sua vez cria um *snapshot* e o empacota com outros arquivos referentes à VM para futuro *download* via Glance. A fim de que seja possível executar as instâncias em dispositivos locais, os autores criaram um *web service* para migrar e gerenciar máquinas virtuais advindas do OpenStack. Esse *web service* contém as funcionalidades básicas das APIs do OpenStack, *download* e *upload* de instâncias para ir e vir da nuvem com elas.

Em seu trabalho, [Mangal et al. 2015] desenvolveram um método para realizar a integração de nuvens OpenStack públicas e privadas. Em seu experimento, os autores instalaram às nuvens em servidores físicos diferentes, assim simulando no segundo servidor uma nuvem pública. Quando ocorre a solicitação para *placement* de uma instância, o *Load analyzer* (implementado pelos autores) verifica a quantidade de uso dos recursos físicos do *host* e caso ele esteja sobrecarregado, a instância é criada na nuvem pública. Instâncias que estão na nuvem pública geram gastos financeiros e por isso o *Load analyzer* também é executado em períodos de tempo para verificar se o servidor *host* da nuvem privada está com recursos disponíveis para hospedar instâncias que sofreram *placement* na nuvem pública. Se ele estiver disponível, ocorre *live migration* no sentido nuvem pública para nuvem privada.

Em [Awasthi and Gupta 2016], os autores relatam ser possível realizar *live migration* entre nuvens de mesmo *hypervisor* já que eles possuem o mesmo formato de disco

virtual, ou entre nuvens de *hypervisors* diferentes sendo em alguns casos necessário realizar conversão para o formato de disco da nuvem de destino. Segundo os autores, os *hypervisors* suportados pelo OpenStack são KVM, QEMU, Xen, vSphere e Microsoft Hyper-V. No trabalho não é mencionado se houveram implementações ou testes com as afirmações feitas pelos autores, ao que parece é apenas um estudo teórico.

[Desai and Patel 2015] propõem uma abordagem para diminuir o tempo de migração do tipo *live* em ambientes de nuvem. Sua abordagem se baseia em modificar a etapa de pré-copia para reduzir o número de páginas da memória a serem transferidas. Para isso, os autores utiliziram um limiar que aumenta a precisão de quando e quais páginas precisam ser atualizadas no destino. Sobre essas páginas é aplicado o algoritmo *Characteristic Based Compression* (CBC) para compressão dos dados e com isso elas são transferidas. Os autores relataram ter conseguido diminuir o tempo total de migração e o tempo de *downtime* significativamente.

A ferramenta AZOS se diferencia do trabalho de [Mangal et al. 2015] e de [Katzmarski et al. 2014] por não realizar intrusões no código fonte do OpenStack, utilizando apenas as APIs nativas de seus serviços para interação. Ainda, [Mangal et al. 2015] simulam sua nuvem pública na mesma rede local de sua nuvem privada, ao passo que a ferramenta AZOS possui especificação para e foi testada com a nuvem pública Microsoft Azure. [Mangal et al. 2015] relatam o uso, [Desai and Patel 2015] utiliza com modificações para redução de tempos e [Awasthi and Gupta 2016] apenas afirma ser possível utilizar *live migration* para migração de instâncias, enquanto a ferramenta AZOS utiliza de forma prática migração *offline* e pode com isso efetuar as conversões necessárias para compatibilidade entre diferentes formatos.

Optou-se por esse tipo de migração (*offline*) em razão de não ser necessário haver compatibilidade de CPUs entre *host* de origem de destino e tampouco entre *hypervisors* e seus formatos de discos virtuais. A migração *offline* possui apenas limitações quando se trata de *hypervisors* que utilizam diferentes técnicas de virtualização, portanto sistemas operacionais *guests* que executam em virtualização completa podem ter problemas de compatibilidade caso sejam migrados para *hypervisors* que suportem apenas para-virtualização. No estudo de caso realizado neste trabalho, a ferramenta AZOS foi utilizada em ambientes onde não há para-virtualização.

4. Descrição da ferramenta

A ferramenta proposta neste trabalho foi implementada na linguagem de programação *Python* e se comunica com os gerenciadores de nuvem por meio dos *endpoints* de seus serviços. Na arquitetura da aplicação existe um usuário que utiliza a ferramenta AZOS para adquirir mobilidade em sua *multi-cloud*. Para isso são executados processos de *discovery* e *build* em suas nuvens propositando a migração de instâncias. Não há incompatibilidades entre os *hypervisors* nesse caso, pois a migração utilizada é do tipo *offline*. O ambiente de nuvem é representado por arquivos descritores tanto para importação como exportação de dados. O formato utilizado é o *JavaScript Object Notation* (JSON) e por isso os arquivos são passíveis de leitura e alteração por parte do usuário.

Tanto o formato JSON quanto o *eXtensible Markup Language* (XML) são utilizados para o intercâmbio de dados entre aplicações. O formato XML se baseia em *tags* para demarcar informações e pode ser facilmente manipulado para criar dialetos próprios

para aplicações. O formato JSON não utiliza *tags* e sim pares atributo-valor para armazenar dados referentes a objetos que podem ser ainda utilizados em conjunto com vetores ou com outros objetos. Em uma breve comparação, o formato JSON é mais robusto nos sentidos de que por não utilizar *tags*, torna menor a quantidade de informações a serem transmitidas e processadas. Sua sintaxe é humanamente mais legível que a do XML e explicitamente é visível a utilização da orientação a objetos em sua essência. Por essas razões o formato JSON é utilizado neste trabalho tanto para o arquivo de *template Open Cloud Exportation File* (OCEF) quanto para os arquivos de credenciais das nuvens.

Antes de qualquer interação com as nuvens da *multi-cloud* é necessário autenticar-se nelas. Conforme Exemplo 1, os dados necessários para isso ficam presentes nos arquivos de credenciais (JSON) lidos por os processos de *discovery* e *build*. Os gerenciadores de nuvem disponibilizam URLs específicas para que ferramentas externas possam se comunicar com eles por meio de suas APIs. Usuários podem possuir mais de um *tenant* (projeto) na mesma nuvem, logo é necessário que seu nome esteja especificado no arquivo de credenciais. Cada *tenant* pode possuir mais de um domínio o que leva à necessidade de sua menção também no arquivo de credenciais. Nome de usuário e senha são atributos do arquivo que em conjunto com os outros dados são enviados à URL das APIs dos *endpoints*. Se os dados estiverem corretos ocorre a autenticação e autorização para interação com os seviços da nuvem.

Na arquitetura da ferramenta representada pela Figura 1 existem três entidades: nuvem de origem e de destino que formam a *multi-cloud*, e usuário dessa *multi-cloud*. O papel do usuário é preencher os arquivos de credenciais com informações fornecidas por seus provedores de nuvem e utilizar a ferramenta AZOS para realizar o processo de *discovery* na nuvem de origem e *build* na nuvem de destino. Durante esses dois processos a ferramenta AZOS interage com os gerenciadores de nuvem em tarefas locais e remotas. Ao final o cliente irá possuir em seu computador local os discos virtuais de suas instâncias e o arquivo de *template* OCEF que descreve sua nuvem. Esses itens serão utilizados como entrada para o processo de *build* na nuvem de destino.

O arquivo OCEF é o formato de arquivo de *template* utilizado pela ferramenta AZOS. Seu uso pode ser visualizado no Exemplo 2. O arquivo de *template* descreve informações acerca de uma máquina virtual. Seu primeiro item é referente ao seu VHD. Nele consta o seu tamanho em *gigabytes* e o caminho no computador do cliente que leva ao seu arquivo. O item seguinte se refere ao *firewall* que organiza suas regras em *security groups*. Logo estão o nome da instância, sua quantidade de memória principal em *megabytes*, seu endereço IP local e sua quantidade de processadores virtuais. Cada dado do arquivo está armazenado no formato atributo-valor.

```
{
"auth_url_sdk" : "http://192.168.10.10:5000/v3/",
"auth_url" : "http://192.168.10.10:5000",
"username" : "demo",
"user_domain_name":"Default",
"project_name":"demo",
"project_domain_name":"Default",
"password": "password"
}
```

Exemplo 1. Arquivo de credenciais

```
{
    "VM": [
        {
            "disk": {
                "/dev/vda": {
                    "file": "/tmp/instancia01_2016-12-13_16-36-11.img",
                    "size": 10
                }
            },
            "firewall": {
                "description": "Default security group",
                "name": "default",
                "rules": []
            },
            "memory": 1024,
            "name": "instancia01",
            "networks": {
                "private": [
                    "10.0.0.12/26"
                ]
            },
            "vcpu": 1
        }
    ]
}
```


Exemplo 2. Open Cloud Exportation File (OCEF)

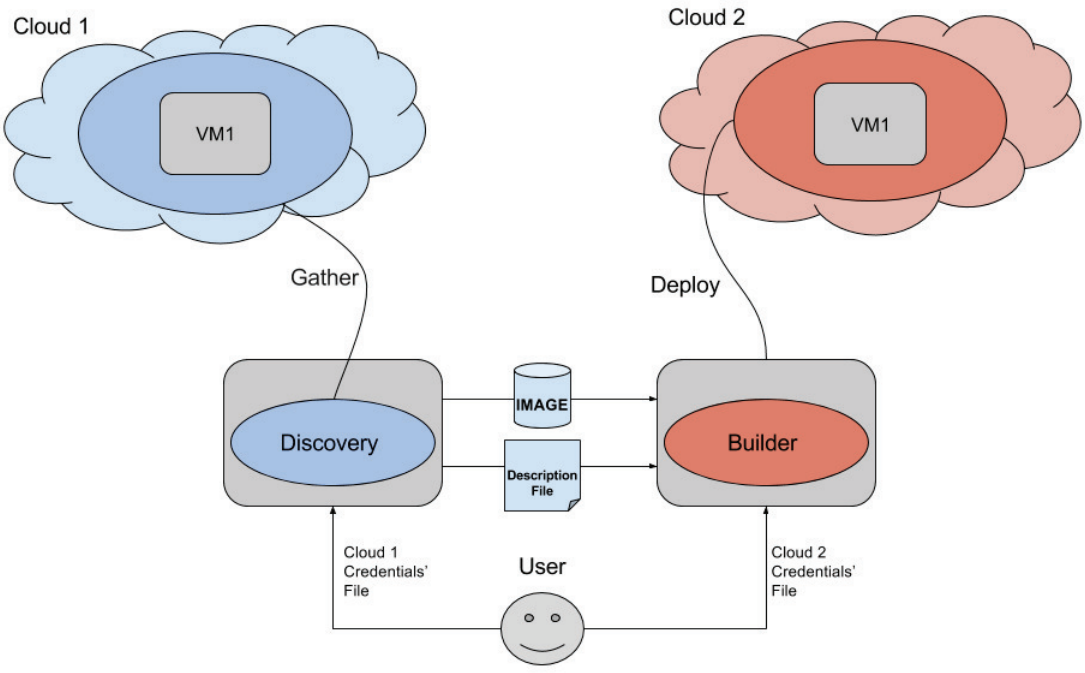

Figura 1. Arquitetura da aplicação

5. Estudo de caso

No presente estudo de caso foi desenvolvido um módulo para a ferramenta AZOS ser capaz de migrar instâncias de nuvens privadas OpenStack para nuvens públicas Microsoft Azure, e vice-e-versa. Ao passo que o OpenStack possui seus próprios formatos, nomenclaturas, *templates* para orquestração, topologias e serviços; a Azure também dispõem de seus próprios, o que ocasiona incompatibilidade entre as duas referidas nuvens. Em vista disso, a migração *offline* possibilita a portabilidade de serviços e instâncias entre nuvens (*multi-clouds*) estabelecendo um protocolo comum entre elas. Esse tipo de migração consiste em realizar processos de *discovery* e *build*, e implicitamente a eles trabalhar na homogenização dos recursos a serem migrados. Os processos de *discovery* e *build* acontecem em ordem, ocorrendo primeiro o *discovery* para gerar um *template* referente à nuvem de origem e após o *build* para importá-lo e orquestrá-lo, conforme mostrado na Figura 2.

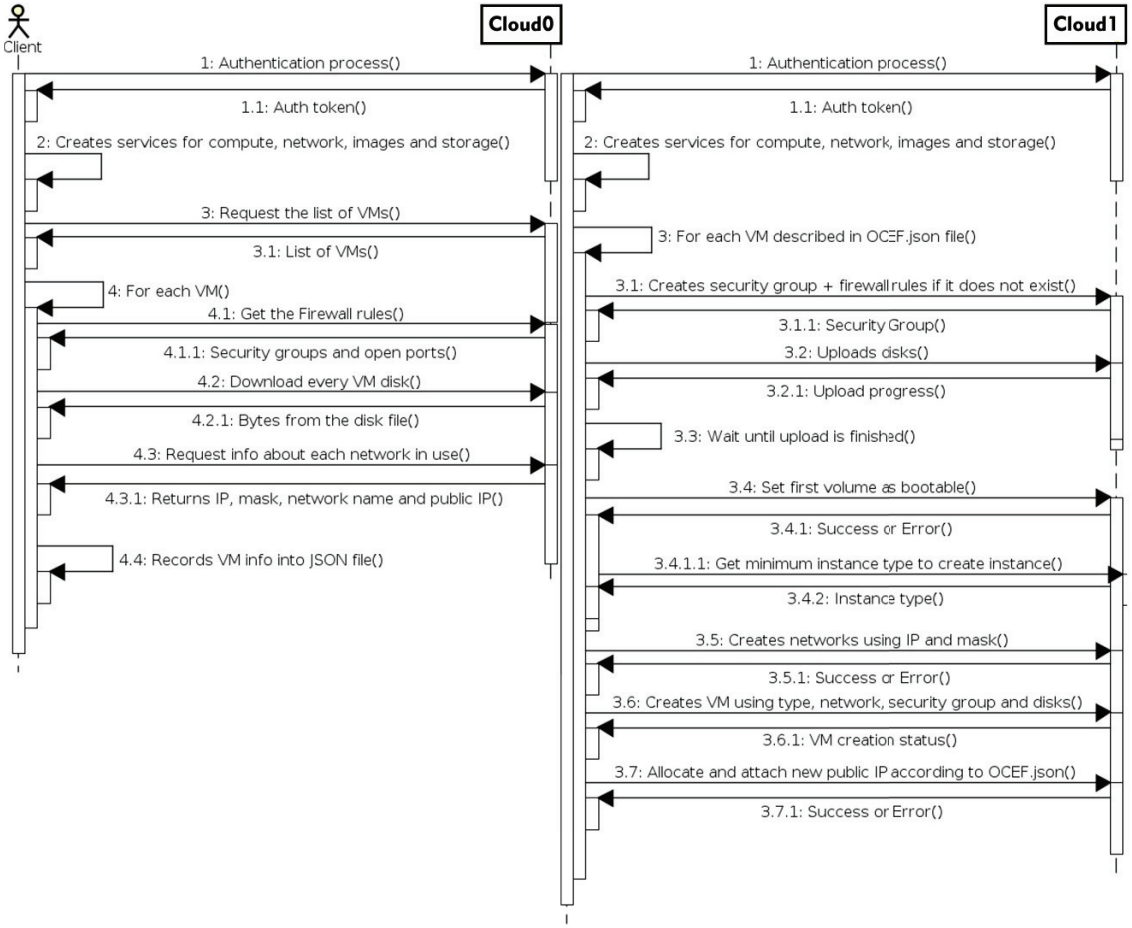

Figura 2. Diagrama de sequência dos processos de *discovery* e *build*

No processo de *discovery*, os dados do usuário são enviados para a API do serviço de identificação OpenStack Keystone que autoriza o acesso para determinado *tenant* retornando um *token* a ser utilizado em cada operação. Em sequência ocorre o instanciamento de objetos para interação com os serviços de *compute* (Nova), rede (Neutron) e imagem (Glance). Após é enviada uma requisição para gerar uma lista de instâncias contidas no *tenant* do cliente. A cada instância encontrada efetua-se o *download* de seus discos virtuais e são extraídas informações acerca do tamanho dos discos e memória principal, número de

processadores, IPs públicos, IPs locais, *security groups* e outras políticas de *firewall*. Em seguida os dados coletados são escritos no formato JSON OCEF para futura orquestração via módulo de *build*. Uma das particularidades desse estudo de caso é a necessidade de conversão do formato de disco do OpenStack. Nativamente a nuvem Microsoft Azure não aceita o disco no formato QCOW2, sendo necessáro convertê-lo manualmente para o formato VHD antes de iniciar o processo de *build*.

Da mesma forma que o processo de *discovery*, o *build* inicia pela autenticação na API do serviço de identificação da Azure que devolve um *token* caso a autenticação seja bem sucedida. Por seguinte os objetos referentes aos serviços da Azure são instanciados e se inicia a leitura do arquivo JSON OCEF gerado no processo de *discovery*. Para cada instância encontrada no arquivo *template* são criados, caso não existam, seus respectivos *security groups* e regras de *firewall*. Após é realizado o *upload* de seu disco virtual e ao final dessa etapa são extraídos da atual iteração informações acerca de quantidade de memória principal, endereços IP, número de processadores e tamanho do disco, para que seja selecionado na Azure um *instance type* com no mínimo as mesmas configurações. Por fim é criada a instância e se constar na iteração a presença de IPs públicos anexados a ela, é solicitado à API da Azure um número igual de IPs para que a instância fique acessível na Internet.

Durante o desenvolvimento do módulo Azure-OpenStack foram identificadas limitações no sentido da Azure que estão listadas a seguir: (i) usuários não administrativos não possuem privilégios para criar *resource groups*, isso faz com que usuários com privilégios de administrador sejam requeridos no processo de *build*; (ii) instâncias do Azure possuem no mínimo 30GB de tamanho de disco quando criadas pelo console Web, o que leva a um maior tempo de *download* do VHD no momento da migração para o OpenStack; (iii) no momento da migração para a nuvem Azure, o SDK utilizado no desenvolvimento realiza o carregamento das instâncias em memória principal no cliente para compactação dos dados e por conseguinte diminui o tempo de *upload*. É recomendado que o cliente possua no minimo 16GB de RAM disponível para esse processo, visto que essa é uma peculiaridade imposta por o SDK e não por a ferramenta AZOS; (iv) a nuvem Azure possui alguns IPs privados reservados para seu uso interno, podendo haver colisão de IPs caso instâncias orquestradas possuam esses mesmos endereços. Por essa razão o processo de *build* na Azure suporta apenas alocação dinâmica de IPs.

No sentido do OpenStack, são relatadas as seguintes limitações: (i) algumas instâncias utilizam seu sistema operacional no formato *Amazon Machine Image* (AMI), que consiste em arquivos separados para *kernel* e *Random Access Memory* (RAM) do *guest*. Nesses casos é necessário que haja na nuvem de destino os arquivos mencionados e eles devem estar nas mesmas versões para que a instância possa dar boot a partir deles; (ii) A ferramenta suporta apenas um volume por instância; (iii) O OpenStack oferece uma opção para *download* do disco virtual da instância tanto via interface de linha de comando quanto via Horizon. Porém, foi encontrado um *bug* que impossibilita esse *download*. Para contornar essa limitação, a ferramenta AZOS no processo de *discovery* no OpenStack arquiva o VHD da instância como imagem no Glance e após é possível efetuar o seu *download*. Esse processo em seu inverso é válido para o processo de *build*, onde primeiramente a imagem é carregada para o Glance e após é gerado o VHD da instância a partir dela.

6. Avaliação

O estudo de caso realizado neste trabalho envolveu o desenvolvimento de módulos para que a ferramenta AZOS pudesse realizar migrações entre nuvens OpenStack e Microsoft Azure. Com o propósito de aferir o funcionamento desses módulos, foi realizada a avaliação de suas funcionalidades por meio da medição de tempo de execução de cada tarefa (etapa) dos processos de *discovery* e *build*. Para isso foram realizadas medições granulares acerca desses processos, sendo executados os processos de *discovery* e *build* tanto na nuvem OpenStack quanto na Azure. Dessa maneira foi possível realizar comparações entre os tempos de mesmas tarefas utilizando implementações diferentes, ainda que as duas nuvens em questão possuam APIs e SDKs distintos para realizar as migrações que o cliente necessita.

No que diz respeito ao cenário, as medições foram realizadas em uma nuvem privada OpenStack e em uma nuvem pública Azure. Tentou-se ao máximo reproduzir um cenário real para que as medidas coletadas tivessem maior significado no que diz respeito à ambientes de produção. O ambiente de nuvem privada foi provisionado utilizando o Devstack [Foundation 2017] em uma instalação *all-in-one* onde todos os serviços do OpenStack ficam no mesmo servidor. Já o ambiente de nuvem pública encontrava-se pré-configurado na Azure bastando apenas alguns ajustes para deixá-lo em produção com uma instância. Em ambas nuvens há uma instância com a configuração de um processador, 1GB de memória principal, disco rígido virtual de 30GB e um endereço IP local. O cliente das duas nuvens está na mesma rede local que a nuvem privada OpenStack enquanto que a nuvem pública Azure está acessível a partir da Internet em um *data center* do provedor Microsoft.

No estudo de caso foram realizadas migrações de instâncias no sentido de OpenStack para Azure e vice-versa. Para ser possível comparar as mesmas etapas de migração para as duas nuvens, foi estabelecida a métrica tempo. É possível medir o tempo das etapas em razão de haver uma padronização de tarefas nos métodos da ferramenta, onde cada método representa uma ou mais tarefas a serem desempenhadas ou no processo de *discovery* ou no processo de *build*. Com isso as etapas (tarefas) medidas foram em relação à rede, regras de *firewall*, *flavor*, autenticação e migração (*download* e *upload*) de discos virtuais. As nomenclaturas e modos de operação das duas nuvens são distintos, por isso foi necessário estabelecer etapas bem definidas de maneira que mesmo diferentes implementações para a mesma tarefa pudessem ser comparadas tanto nos dois processos de *discovery* quanto nos dois de *build* realizados no estudo de caso.

As medições de tempo das etapas equivalentes foram feitas utilizando contadores de tempo dentro do código. Os processos de *discovery* e *build* são divididos em etapas, antes de cada uma delas é inserido um temporizador inicial e após um temporizador final. Assim é possível realizar a diferença entre tempo final e inicial para que seja possível obter o tempo total de execução de determinado trecho de código referente à etapa analisada. Com o propósito de diminuir o erro nas medições, foram realizadas três execuções para cada etapa em diferentes turnos, já que a vazão e disponibilidade dos *links* de rede poderiam ser afetadas. Essa metodologia foi adotada principalmente em razão dos tempos de *download* e *upload* dos discos virtuais serem maiores do que os das outras tarefas, haja vista que ambos possuem tamanho na unidade de gigabyte e por isso dependendo da rede e por conseguinte do horário pode haver uma drástica variação de tempo.

A Figura 3 contém o gráfico com os tempos das etapas dos processos de discovery nas nuvens OpenStack e Azure. Na etapa de leitura das regras de *firewall* na nuvem OpenStack o tempo é aproximadamente duas vezes maior que na Azure, isso ocorre em razão de serem necessários três laços de repetição aninhados na OpenStack ao passo que na Azure são necessários somente dois para leitura dos *security groups*. Quando lidas as informações referentes à rede das instâncias, a implementação do *discovery* para a nuvem Azure consegue obter os dados apartir de um único objeto enquanto que por limitações do SDK utilizado na OpenStack é necessário realizar iterações para obter os respectivos dados. O referido SDK ainda possui um método nativo para obter os dados relativos ao *flavor* associado à instância, o que motiva essa etapa a possuir menor tempo do que a mesma etapa no *discovery* da Azure. A etapa de autenticação em ambas implementações do *discovery* (Azure e OpenStack) é semelhante, com isso acredita-se que o maior tempo necessário para autenticação na nuvem Azure ocorre por alguma particularidade em sua API.

Através da Figura 4 é possível visualizar o gráfico que contém os tempos das etapas de *build* realizadas nas duas nuvens. O tempo para criar a instância é maior na Azure em razão de ser necessário estabelecer uma URL para que o *blob storage* referente ao VHD vindouro da OpenStack seja anexado a ela. As regras de *firewall* por sua vez requisitam maior tempo para serem estabelecidas na instância da Azure pois é necessário criar um *security groups* e definir suas regras, ao passo que no provisionamento do OpenStack um *security group* padrão já é criado para todas as instâncias. Em razão da limitação de a Azure reservar alguns endereços IPs para uso interno, o processo de *build* solicita um endereço de IP dinâmico para o servidor DHCP da Azure, o que leva o seu tempo de estabelecimento da rede a ser maior. Na etapa referente a *flavor* da instância é necessário selecionar o que atenda no mínimo as suas características de *hardware* virtual, caso não seja possível então é necessário criá-lo e isso faz com que o tempo da etapa *flavor* na OpenStack seja maior. Os tempos de autenticação são iguais ao do processo de *discovery* já que a implementação e utilização são as mesmas.

Apesar de os processos de *discovery* e *build* possuirem respectivamente as tarefas de *download* e *upload* dos discos virtuais das instâncias, por razões de disparidade entre as grandezas obtidas nas medições de tempo, optou-se em colocá-los no gráfico representado pela Figura 5 de forma exclusiva. O tempo da etapa de migração (*download*) no processo de *discovery* na Azure é aproximadamente 5,6 vezes maior do que na OpenStack em razão de o cliente das duas nuvens estar na mesma rede local que a nuvem privada. Já na etapa de *upload* do disco virtual do *build* da Azure o tempo é aproximadamente 2,3 vezes maior do que na do OpenStack, a pouca diferença de tempo ocorre em razão de o SDK utilizado no desenvolvimento da ferramenta realizar um mapeamento do disco virtual da instância migrada em memória principal no computador do cliente, realizando dessa maneira uma compactação dos dados antes deles serem enviados pela rede.

Na avaliação do estudo de caso os grandes gargá-los encontrados são relativos à quantidade de banda disponível nos *links* de rede, tamanho dos discos virtuais das instâncias e peculiaridades para interação com as APIs das duas nuvens. O tempo total de migração de uma instância da nuvem OpenStack para a nuvem Azure foi de aproximadamente 71,783 minutos, já a migração da nuvem Azure para a OpenStack levou aproximadamente 147,30 minutos, sendo então duas vezes maior.

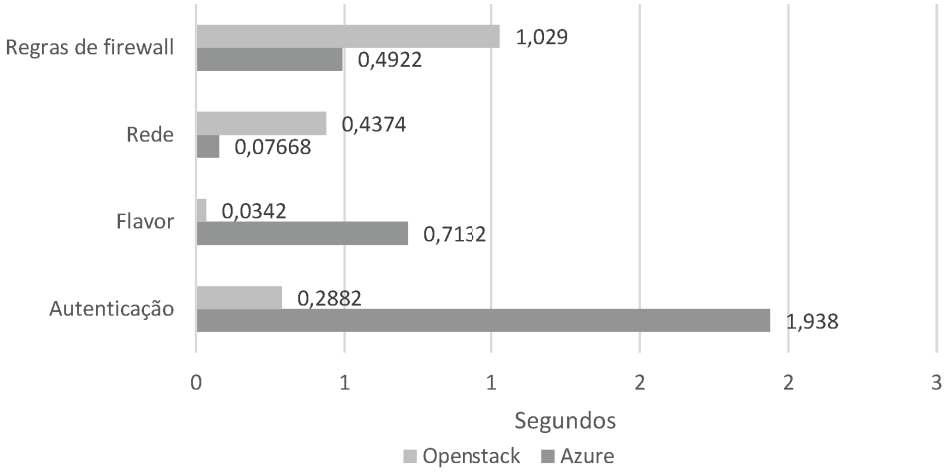

Figura 3. Médias dos tempos dos processos de discovery

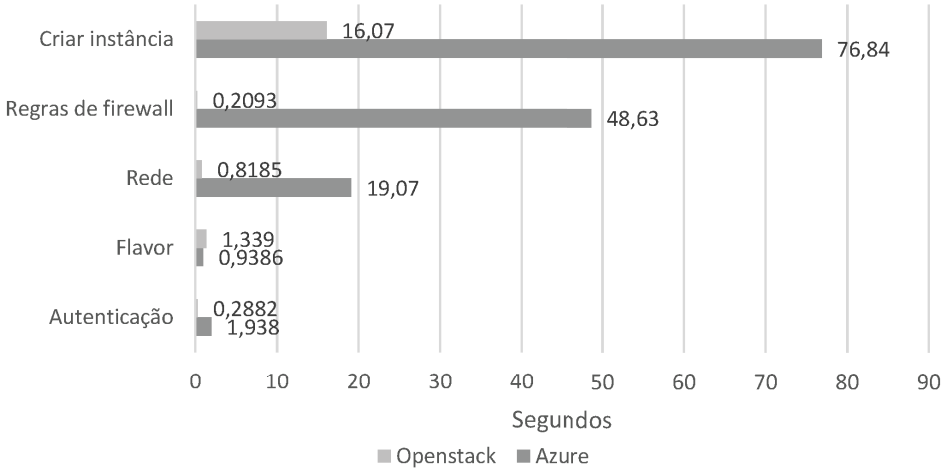

Figura 4. Médias dos tempos dos processos de build

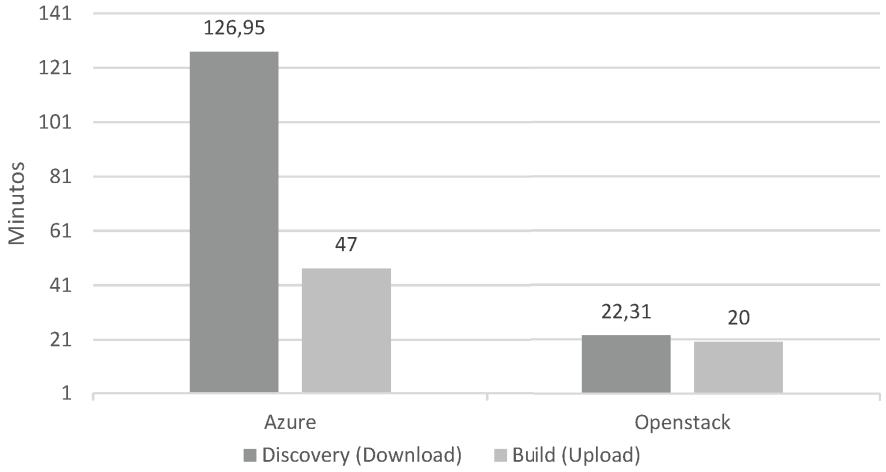

Figura 5. Médias dos tempos de migração dos discos rígidos virtuais

7. Conclusão e trabalhos futuros

O objetivo deste trabalho foi apresentar a ferramenta AZOS para efetuar migrações de instâncias em *multi-clouds*. Em seu modelo arquitetural há duas nuvens e um cliente que deseja portabilidade entre elas. Na nuvem de origem é realizado um processo de *discovery* que consiste na leitura da topologia da nuvem do cliente para gerar um arquivo de *template* contendo informações referentes ao ambiente e as instâncias que nele estão. Na nuvem de destino é efetuado um processo de *build* para orquestrar a nuvem representada no arquivo *template* gerado pelo processo de *discovery*, nesse processo ainda são efetuados os *uploads* dos discos virtuais resultantes do processo de *discovery*. A ferramenta em sua atual versão suporta migrações entre nuvens OpenStack e Microsoft Azure, ela é compatível com diferentes sistemas operacionais *guests*, contanto que eles sejam virtualizáveis e suportem as arquiteturas dos *hypervisores* utilizados.

A abordagem utilizada neste trabalho para realizar a migração de instâncias no modelo IaaS foi a migração do tipo *offline*. Optou-se por esse tipo em razão da *live migration* levar à limitações de compatibilidade em *multi-clouds* devido a heterogeneidade de tecnologias presentes nesse modelo. Ainda, em algumas abordagens são utilizados *web services* para intermediar às migrações e com isso a latência, *overhead* e tempo total podem aumentar em relação à migração feita a partir do computador do cliente. Nesse caso o cliente das duas nuvens orquestra toda a migração a partir de sua estação de trabalho. Ele efetua o processo de *discovery* para gerar o *template* referente a sua nuvem de origem e logo efetua o processo de *build* para orquestrar sua nuvem na plataforma de destino. Dessa forma as conversões entre formatos e descritores de ambiente podem ser efetuadas sem prejuízos a integridade das instâncias.

Foi realizado um estudo de caso onde houve uma migração na *multi-cloud* formada pelas nuvens OpenStack e Microsoft Azure, com o propósito de verificar e validar a ferramenta proposta. A migração de Azure para OpenStack levou 147,30 minutos em razão dos dados migrados terem sido trafegados pela Internet até um dos *data center* da Microsoft e não haver nenhum tipo de compactação de dados na transmissão. Na segunda migração, de OpenStack para Azure, o tempo total de migração foi de aproximadamente 71,783 minutos. A segunda migração levou a metade do tempo da primeira por o SDK utilizado para interagir com a Azure carregar toda a instância a ser migrada em memória principal no cliente e então realizar uma compactação de dados. Contudo, o tempo de migração sempre é proporcional à largura de banda dos *links*, tamanho dos VHDs migrados e a disponibilidade de rotas percorridos durante a migração. Eventuais falhas na transmissão dos dados podem comprometer a migração em um todo.

Em trabalhos futuros pretende-se estender a arquitetura da ferramenta AZOS para ser possível abranger outros modelos de serviço e provedores de nuvem, para inclusive aumentar a abrangência dos testes realizados. A compatibilidade de recursos entre os provedores será um dos maiores desafios. O atual cenário da Internet propicia cada vez mais o uso da computação em nuvem, exemplos disso são serviços de *storage*, *hosting*, *Internet bankings*, *eCommerces*, VDIs (*Virtual Desktop Infrastructures*), dentre outros vários. A latência, controle de custos e sensação de *hardware* inifito muitas vezes levam às empresas a migrar suas infraestruturas para a nuvem. O lado contrário disso é a mobilidade e incompatibilidade entre provedores, o que gera a necessidade de ferramentas que possibilitem a portabilidade e compatibilidade entre eles.

Referências

Awasthi, A. and Gupta, R. (2016). Multiple hypervisor based open stack cloud and vm migration. In *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*, pages 130–134.

Bellard, F. (2005). Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46.

Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., and Wagner, S. (2013). Opentosca–a runtime for tosca-based cloud applications. In *International Conference on Service-Oriented Computing*, pages 692–695. Springer.

Buyya, R., Vecchiola, C., and Selvi, S. T. (2013). *Mastering cloud computing: foundations and applications programming*. Elsevier.

Chirammal, H. D., Mukhedkar, P., and Vettathu, A. (2016). *Mastering KVM Virtualization*. Packt Publishing Ltd.

Desai, M. R. and Patel, H. B. (2015). efficient virtual machine migration in cloud computing. In *Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on*, pages 1015–1019. IEEE.

Foundation, O. (2017). OpenStack Open Source C. C. `http://openstack.org/`.

Jin, D. and Lin, S. (2012). *Advances in Computer Science and Information Engineering*. Springer.

Katzmarski, B., Herrholz, A., Paolino, M., Rigo, A., and Nebel, W. (2014). Considering vm migration between iaas clouds and mobile clients: Challenges and potentials. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 327–332. IEEE.

Kivity, A., Kamay, Y., Laor, D., Lublin, U., and Liguori, A. (2007). kvm: the linux virtual machine monitor. In *Proceedings of the Linux symposium*, volume 1, pages 225–230.

Mangal, G., Kasliwal, P., Deshpande, U., Kurhekar, M., and Chafle, G. (2015). Flexible cloud computing by integrating public-private clouds using openstack. In *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 146–152.

Microsoft (2017). Microsoft Azure: Cloud Computing Platform & Services. `https://azure.microsoft.com/`.

Petcu, D. (2013). Multi-cloud: expectations and current approaches. In *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*, pages 1–6. ACM.

Ren, F. and Zhai, J. (2014). *Communication and popularization of science and technology in China*. Springer.

Tanenbaum, A. S. and Bos, H. (2014). *Modern operating systems*. Prentice Hall Press.