Avaliando o Aprisionamento entre Várias Plataformas de Computação em Nuvem

Arthur E. C. da Silva Souza[1], José A. Medeiros de Lima[1], Renato Gondim[1], Thomás Diniz[2], Nélio Cacho[1], Frederico Lopes[2] , Thais Batista[1]

[1]Departamento de Informática e Matemática Aplicada – Universidade Federal do Rio Grande do Norte (UFRN)
Natal – RN – Brasil

arthurcassio@yahoo.com.br, .j.alex.medeiros@gmail.com, renato@dimap.ufrn.br, neliocacho@dimap.ufrn.br, thaisbatista@gmail.com

[2]Escola de Ciências e Tecnologia – Universidade Federal do Rio Grande do Norte (UFRN)
Natal – RN – Brasil

thomasfdsdiniz@gmail.com, fred.lopes@gmail.com

Abstract. *Despite cloud lock-in being one of the main drawbacks of cloud computing, there are no studies that systematically evaluate the compatibility among different cloud platforms. The goal of this paper is twofold: (i) to propose a generic process based on compatibility tests to evaluate the lock-in among cloud platforms; (ii) to apply the proposed process to evaluate the compatibility among a public platform (Amazon EC2) and three private platforms (OpenStack, Eucalyptus, and OpenNebula). This paper shows a compatibility evaluation and investigates if the existing compatibility is enough to avoid cloud lock-in.*

Resumo. *Apesar do cloud lock-in (aprisionamento) ser um dos principais problemas da computação em nuvem, não há estudos que avaliem sistematicamente a compatibilidade entre diferentes plataformas de nuvem. Esse artigo tem dois objetivos principais: (i) propor um processo genérico para avaliação do aprisionamento entre plataformas de nuvem através de uma abordagem de teste de compatibilidade; (ii) aplicar o processo proposto para avaliar o aprisionamento entre uma plataforma pública (Amazon EC2) e três plataformas privadas (OpenStack, Eucalyptus e OpenNebula). Esse trabalho mostra uma avaliação da compatibilidade e investiga se a compatibilidade existente é suficiente para evitar o problema de aprisionamento.*

1. Introdução

Computação em Nuvem surgiu como um modelo de computação distribuída onde recursos computacionais (*hardware*, armazenamento, plataformas de desenvolvimento, e comunicação) são virtualizados e disponibilizados como serviços (re)configuráveis, pagos pela sua utilização, sendo suportados por grandes centros de dados na Internet [Armburst et al. 2010, Foster et al. 2008, Vaquero et al. 2009]. As nuvens podem ser organizadas basicamente em três modelos [Armburst et al. 2010]: nuvem pública, nuvem privada e nuvem híbrida. Os serviços de nuvem pública são caracterizados pela

sua disponibilidade para clientes através de um modelo do tipo pagamento por uso. Por sua vez, no serviço de nuvem privada o centro de dados (*datacenters*) é interno a empresa ou outro tipo de organização, não estando disponíveis publicamente [Armbrust et al. 2010]. A nuvem privada oferece muitos dos benefícios de um ambiente de Computação em Nuvem pública, como a elasticidade, ainda que limitada, e o modelo baseado em serviços. Finalmente, a nuvem híbrida materializa-se quando uma nuvem privada é suplementada com características presentes em nuvens públicas (i.e. informações de negócio que sejam não críticas e operações de processamento podem ser mantidas na nuvem pública, enquanto serviços e dados de negócio que sejam considerados críticos podem ser mantidos sob o controle dos usuários no escopo da nuvem privada). Dessa forma, há uma utilização mista e integrada desses dois modelos de nuvem. Tais plataformas permitem que dados e processos sejam gerenciados internamente a uma organização sem restrições de banda de rede, exposição à segurança e requisitos legais que o uso de serviços de nuvem pública pode requerer. De fato, boa parte das restrições de uso imposta pelas nuvens públicas leva ao *cloud lock-in* [Buyya, Broberg e Goscinski 2011, Armbrust et al. 2010].

O *cloud lock-in* pode ser visto como a dependência entre uma aplicação e os recursos de um determinado provedor de computação em nuvem [Buyya, Broberg e Goscinski 2011]. Esse problema geralmente ocorre quando o provedor implementa um conjunto próprio de APIs (*Application Programming Interface*), e ao migrar a aplicação de um provedor para outro, é necessário modificá-la substancialmente para usar a API do novo provedor. Em termos práticos isto aprisiona a aplicação à nuvem, uma vez que a mudança de provedor é uma tarefa complexa e de custo elevado [Buyya, Broberg e Goscinski 2011]. Deste ponto em diante, o termo *cloud lock-in* será traduzido como *aprisionamento*.

Apesar do aprisionamento ser considerado um dos principais desafios da computação em nuvem [Armbrust et al. 2010], não há estudos que avaliem sistematicamente a compatibilidade entre diferentes plataformas de nuvem. Assim, este artigo busca avaliar o aprisionamento que uma organização teria que superar para migrar uma aplicação de uma nuvem pública para uma privada/híbrida, ou vice-versa. Tal avaliação é relevante uma vez que pode: (i) nortear os usuários de nuvens na tomada de decisões sobre migrações entre plataformas de nuvens públicas e privadas, além de (ii) fornecer embasamento a uma possível interoperabilidade entre plataformas de nuvens.

Esse trabalho tem dois objetivos principais: (i) propor um processo genérico para avaliação do aprisionamento entre plataformas de nuvem através de uma abordagem de teste de compatibilidade; (ii) aplicar o processo proposto para avaliar o aprisionamento entre uma plataforma pública (Amazon EC2) e três plataformas privadas (OpenStack, Eucalyptus e OpenNebula). A justificativa para escolha da Amazon EC2 como base é que em termos de ambiente de nuvem comercial, a Amazon é líder de mercado e, por esse motivo, algumas plataformas de nuvem *open-source*, como Eucalyptus, OpenStack e OpenNebula, preconizam que disponibilizam acesso a seus serviços via a API da Amazon EC2. No entanto, na literatura não há relato sobre a existência de uma abordagem sistemática utilizada para a avaliação da compatibilidade dessas nuvens com a EC2. Esse trabalho mostra uma avaliação da compatibilidade e investiga se a compatibilidade existente é suficiente para evitar o problema de aprisionamento.

O artigo está estruturado em cinco seções. A Seção 2 apresenta trabalhos que realizam comparações entre plataformas de nuvens. A Seção 3 apresenta o processo definido nesse trabalho para avaliar o aprisionamento entre plataformas de nuvens. A Seção 4 descreve um estudo de caso que aplica o processo avaliativo proposto e os resultados obtidos em cada fase. Por fim a Seção 5, contém as conclusões.

2. Trabalhos Relacionados

Existem vários estudos que buscam comparar e avaliar plataformas de nuvem privadas e híbridas, de modo que potenciais usuários, desenvolvedores e administradores possam ter um conjunto de dados para ajudá-los a decidir qual seria a plataforma mais adequada para suas demandas. No trabalho desenvolvido por [Peng et al 2009] são comparadas as plataformas Eucalyptus, OpenNebula, entre outras. Na descrição da compatibilidade é afirmado que a Eucalyptus tem suporte a API EC2. Apesar de tal conclusão, o estudo não deixa claro como tal resultado foi obtido. Outro exemplo é encontrado em [Mahjoub et al. 2011] no qual as plataformas Eucalyptus e OpenNebula, dentre as avaliadas, são ditas compatíveis com Amazon EC2 enquanto que a OpenStack não. De forma semelhante ao artigo citado anteriormente, não há clareza na sistemática que levou a essa conclusão. O trabalho realizado por [von Laszewski, et. al 2012] afirma que a API EC2 é um padrão "de fato" e que as três plataformas objetos de estudo neste trabalho são apresentadas como compatíveis a API EC2. Embora seja ressaltado que essa compatibilidade corresponde às funcionalidades básicas como: registro e *upload* de imagens e execução, finalização e descrição de máquinas virtuais, mais uma vez a abordagem metodológica não é salientada.

Algumas soluções de código aberto para computação na nuvem são comparadas em [Endo, et al 2010] e são apresentados os desafios que os desenvolvedores enfrentam para trabalharem com soluções usando plataformas de nuvens. Esse trabalho analisou a capacidade da API de trabalhar com diferentes tipos de sistemas operacionais e diversos tipos de máquinas virtuais. Além disso, esse trabalho analisou as restrições que a referida API impõe aos programadores, assim como os acessos destes na solução.

O presente trabalho diferencia-se dos demais nos seguintes aspectos: (i) apresentação de um processo genérico de avaliação do aprisionamento com sistemática bem definida; (ii) avaliação específica do aprisionamento em termos da API fornecida pelas plataformas, não abordando outros problemas de aprisionamento como os relacionados a imagens e virtualização e (iii) uso de teste de compatibilidade para avaliar o processo proposto através de um estudo de caso que compara o aprisionamento entre plataforma pública (Amazon EC2) e três plataformas privadas (OpenStack, Eucalyptus e OpenNebula).

3. Processo de Avaliação de Aprisionamento

Esta seção descreve a metodologia definida nesse trabalho para avaliar o aprisionamento entre plataformas de nuvem através de uma abordagem de teste de compatibilidade. O teste de compatibilidade [Lewis, W. E. 2000] busca identificar incompatibilidades entre componentes. Neste caso, o teste de compatibilidade será utilizado para avaliar se as interfaces fornecidas pelas plataformas de nuvem são compatíveis ou não com uma interface específica. O processo de avaliação aqui proposto foi inspirado em [Mariani, L. et al. 2007; Flores e Polo 2009] e é formado por três etapas, como descritas na Figura 1.

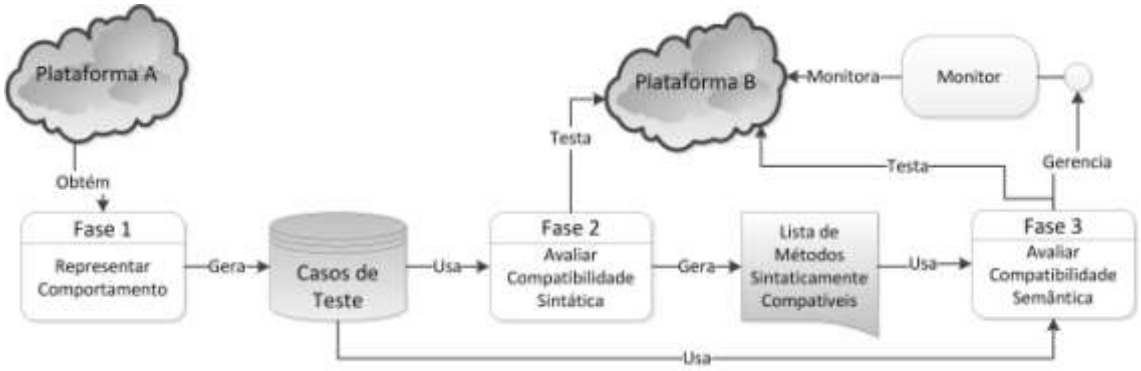

Figura 1. Processo para avaliação de Nível de Aprisionamento

A primeira fase do processo de avaliação de aprisionamento é responsável por analisar o comportamento de uma das plataformas envolvidas no processo de avaliação. A Plataforma A é geralmente representada pela plataforma onde uma determinada aplicação esta implantada. Nesta fase são gerados casos de teste unitários (CT), isto é casos de teste da menor unidade, no caso métodos, que buscam descrever diferentes aspectos das funcionalidades fornecidas pela Plataforma A. Note que o objetivo dos casos gerados nesta fase não é procurar defeitos na Plataforma A, mas representar seu comportamento. Dado que os CTs são executados com sucesso na Plataforma A, espera-se resultado similar durante a execução dos CTs na Plataforma B, para que a mesma seja compatível com a Plataforma A. Maiores detalhes sobre esta fase serão descritos na Seção 4.2.

Por sua vez, a segunda fase do processo de avaliação consiste em avaliar se a interface fornecida pela Plataforma B é sintaticamente compatível com a interface fornecida pela Plataforma A. A Plataforma B é geralmente representada por uma plataforma alvo de uma possível migração. A compatibilidade sintática avalia possíveis diferenças entre nomes e parâmetros dos elementos da interface. Tal compatibilidade é testada através de execução dos casos de teste gerados na fase anterior (CTs) na Plataforma B. Se o caso de teste passar, o referido método testado é inserido em uma lista de métodos com provável compatibilidade. Por outro lado, se o caso de teste falhar, o método é marcado como incompatível.

A última fase do processo proposto avalia se o comportamento da Plataforma B é o mesmo da Plataforma A para os métodos avaliados. Isto implica em executar os casos de teste na Plataforma B e avaliar através de monitores se o comportamento observado é o mesmo esperado pela Plataforma A. Para reduzir o numero de monitores gerados, este processo monitora apenas os métodos que compõem a lista de métodos gerada na fase anterior. Maiores detalhes sobre esta fase serão descritos na Seção 4.3.

A geração de monitores não é uma tarefa trivial uma vez que exige o monitoramento assíncrono de vários atributos da nuvem avaliada. Por este motivo, o processo foi dividido entre compatibilidade sintática (fase 2) e semântica (fase 3), visando com isso reduzir o número de monitores a serem gerados.

4. Estudo de Caso

Como descrito na Seção 2, vários artigos afirmam que as plataformas OpenStack, Eucalyptus e OpenNebula são compatíveis com a API Amazon EC2. No entanto, nenhum deles quantifica tal compatibilidade. Diante disto, esta seção irá detalhar como

o processo proposto pode ser utilizado para quantificar o aprisionamento entre uma plataforma pública (Amazon EC2) e três plataformas privadas (OpenStack, Eucalyptus e OpenNebula).

Para efeito de ilustração, a Amazon EC2 será considerada a plataforma original (que hospeda uma aplicação fictícia A), enquanto que as plataformas privadas serão tratadas como plataformas candidatas a hospedarem tal aplicação A. Neste caso, o objetivo do estudo de caso é quantificar a compatibilidade entre a Amazon EC2 e as outras plataformas avaliadas. Quanto maior a compatibilidade, menor será o aprisionamento.

4.1. Plataformas Avaliadas e Ambiente de Execução

Para este estudo de caso, a Amazon *Elastic Compute Cloud* (EC2) foi escolhida como plataforma original por ser, atualmente, a nuvem comercial mais popular, estável e rica em recursos [Juve et al. 2009], mantendo a liderança do mercado e sendo utilizada em várias soluções [Juve et al. 2009, Juve et al. 2010, Turcu, Foster e Nesterov 2009, Lenk et al. 2011], de forma que sua API de acesso a AWS EC2 é considerada um padrão "de fato" [von Laszewski, et. al 2012]. Por sua vez, as plataformas OpenStack, Eucalyptus e OpenNebula foram escolhidas para compor a avaliação por serem consideradas as principais plataformas que suportam a implantação de nuvens Privada e Hibridas [Khan, Ylitalo e Ahmed 2011]. As seções seguintes descrevem suscintamente cada uma das plataformas avaliadas.

4.1.1. Amazon Web Services (AWS)

O *Amazon Web Services* (AWS) são serviços de nuvem providos pela *Amazon*. Estes serviços oferecem poder computacional, facilidades de armazenamento e várias outras funcionalidades que permitem que empresas implantem aplicações e serviços com suporte a escalabilidade e confiabilidade. O AWS é formado por vários serviços: (i) Amazon *Elastic Compute Cloud* (EC2), Amazon *Simple Storage Service* (S3), Amazon *Relational Database Service* (RDS), Amazon *SimpleDB*, entre outros serviços.

Para este estudo de caso, iremos restringir a avaliação apenas ao A*mazon EC2* dado que este é o único serviço suportado por todas as outras plataformas avaliadas. O *Amazon EC2 é* um serviço que oferece capacidade de computação redimensionável na nuvem. Esse serviço apresenta-se como um verdadeiro ambiente de computação virtual, permitindo aos usuários, através de uma interface Web, criar, usar e gerenciar máquinas virtuais com sistemas operacionais Windows e Linux, ou mesmo iniciar tais máquinas de acordo com as necessidades das aplicações.

4.1.2. OpenStack

OpenStack [OpenStack 2012] é uma coleção de projetos de software *open-source* que podem ser utilizados por empresas ou provedores de serviços para implantar uma infraestrutura de nuvem privada ou híbrida. O OpenStack é constituído por três serviços principais: *OpenStack Compute Infrastructure* (*Nova*), *OpenStack Storage Infrastructure (Swift*) e *OpenStack Imaging Service* (*Glance*).

Para efeito deste estudo de caso, o principal serviço utilizado será o *OpenStack Compute Infrastructure* (*Nova*). O *Nova* implementa um serviço de computação virtual,

incluindo a criação, iniciação e gerência de instâncias de máquinas virtuais, o gerenciamento da infraestrutura de rede e o controle de acesso de usuários aos recursos da nuvem. Seu desenvolvimento teve como base o serviço Nebula, desenvolvido pela NASA. É semelhante em escopo ao serviço EC2 da Amazon e expõe suas funcionalidades para usuários através de suas APIs [OpenStack 2012].

4.1.3. Eucalyptus

Eucalytpus [Eucalyptus 2012] é um *framework open-source* de computação em nuvem que permite a criação de *clusters* privados em *datacenters*. O Eucalyptus é constituído pelos seguintes serviços: *Cloud Controller* (CLC), *Walrus Storage Controller* (WS3), *Cluster Controller* (CC), *Node Controller* (*NC*) e *Elastic Block Storage Controller* (EBS).

Neste estudo de caso, o principal serviço utilizado será o *Cloud Controller* (CLC). O CLC implementa o ponto de entrada na nuvem para usuários e administradores, oferecendo um conjunto de APIs para facilitar sua administração. O CLC é responsável por coletar informações dos controladores de nós de execução e tomar decisões de escalonamento repassadas aos controladores de clusters [Eucalyptus 2012].

4.1.4. OpenNebula

O OpenNebula [OpenNebula 2012] é um projeto *open-source* que compreende um conjunto de ferramentas que viabilizam a gerência de *datacenters* e a criação de nuvens privadas ou híbridas virtualizadas segundo o modelo de infraestrutura como serviço. O OpenNebula é composto por cinco componentes principais: (i) interfaces e APIs, que permitem a interação de um usuário com uma infraestrutura estabelecida com o OpenNebula; (ii) grupos e usuários, responsáveis pela criação de contas e grupo e pelos mecanismos de autenticação e autorização que o sistema fornece; (iii) controle de rede, subsistemas responsáveis gerenciar a integração com outras redes ou *datacenter*, (iv) *host* e virtualizações, que suporta gerenciadores de virtualização para gerência do ciclo de vida; e (v) monitoramento das máquinas virtuais e o componente para armazenamento que pode ser configurado para suportar um sistema de arquivos compartilhado ou não [OpenNebula 2012].

Em nosso estudo de caso, o principal serviço avaliado será o *OpenNebula EC2 Query*. O EC2 *Query* API é um serviço Web que permite o gerenciamento de máquinas virtuais em uma nuvem OpenNebula [OpenNebula 2012].

4.1.5. Ambiente de Execução

Para esta avaliação foram utilizadas as versões Diablo do OpenStack, Eucalyptus 3.1 e OpenNebula 3.4. Tais plataformas foram instaladas em um servidor HP Proliant ML350 G6, 24 CPUs de 2.4GHz, 32 GB de memória e 2 TB de disco. Foi utilizado ainda o gerenciador de máquina virtual Xen Server na versão 5.6 e a distribuição GNU/Linux Ubuntu Server 11.04 com kernel atualizado para versão 3.2.0-23. A configuração *single node*, onde todos os componentes são instalados em um mesmo servidor, foi utilizada para todas as plataformas. Tal configuração foi utilizada para garantir que todas as plataformas analisadas usassem a mesma arquitetura de hardware e software.

4.2. Primeira Fase: Representação do Comportamento da EC2

O objetivo desta primeira fase da avaliação é gerar um conjunto de casos de teste (CTs) unitários que representem o comportamento da plataforma original (Amazon EC2), de modo que todas as suas funcionalidades sejam exercitadas.

A documentação e os métodos que compõem a API da plataforma avaliada são pontos que devem ser considerados durante a geração dos CTs. Por exemplo, as funcionalidades da Amazon estão descritas por meio de documentação e de uma API disponíveis através de kits de desenvolvimentos de aplicativos (SDK). Para este estudo de caso, foi utilizado o SDK Java (release 1.2.15), em virtude do mesmo ser amplamente utilizado e independente de sistema operacional.

Os casos de teste foram gerados com base nos métodos fornecidos pela classe "*com.amazonaws.services.ec2.AmazonEC2Client*". Tal classe disponibiliza um conjunto de 113 métodos que permitem gerenciar as funcionalidades da Amazon EC2.

```
01  public class BehaviorTestCase extends TestCase {
02    static boolean monitorFlag = false;
03
04    public static AmazonEC2 ec2;
05   public static void init(String endPoint, String accessKey, String secretKey) throws Exception {
06       BasicSessionCredentials credentials = new BasicSessionCredentials(accessKey, secretKey,"");
07       ec2 = new AmazonEC2Client(credentials);
08       ec2.setEndpoint(endPoint);
09    }
10    public void testDescribeImages(){
11       ec2.describeImages();
12       int statusCodeReturn = TestCaseLogger.getstatusCode("DescribeImages", monitorFlag);
13       assertEquals("Compativel", true, (statusCodeReturn >= 200 && statusCodeReturn <= 206));
14    }
15  }
```

Figura 2. Código de Exemplo do caso de teste DescribeImages.

A Figura 2 mostra um exemplo de caso de teste utilizado para testar o método *describeImages()*. O método *init* (Linhas 4-8) é responsável por criar uma instância da classe *AmazonEC2Client* e recebe como parâmetro três variáveis: um *endpoint*, uma *accesskey* e uma *secretkey*. O *endpoint* representa o endereço de rede do servidor alvo da solicitação. Por sua vez, a variável *accesskey* e *secretKey* representam a identificação alfanumérica única do usuário e sua senha, respectivamente.

O método *testDescribeImages()* representa o caso de teste unitário, onde uma invocação ao método *describeImages()* da API da Amazon EC2 é realizada (linha 10). Cada invocação à API da Amazon EC2 envia mensagens através do protocolo *Hypertext Transfer Protocol (HTTP)* contendo os detalhes da solicitação. Ao tratar tal mensagem, a plataforma alvo retorna uma mensagem de resposta HTTP contendo o *status code* [HTTP/1.1]. Através de uma analise do código fonte do SDK da Amazon (Classe *com.amazonaws.http.AmazonHttpClient*), foi observado que toda mensagem de resposta com *status code* entre valor 200 a 206 é considerada como requisição bem sucedida. O *status code* de cada mensagem resposta é obtido através de uma análise no log gerado pelo próprio SDK da Amazon.

Para realizar tal análise, foi implementada a classe *TestCaseLogger* que relaciona cada invocação de método com sua respectiva mensagem de resposta (*status code*). A Figura 3 descreve a relação entre os casos de teste e a classe *TestCaseLogger*. O *status code* da invocação do método *describeImages* é obtido através da invocação do

método *getstatusCode*. Tal método recebe como parâmetro o nome do método cuja *status code* deve ser retornado. O segundo parâmetro será detalhado na Seção 4.4.

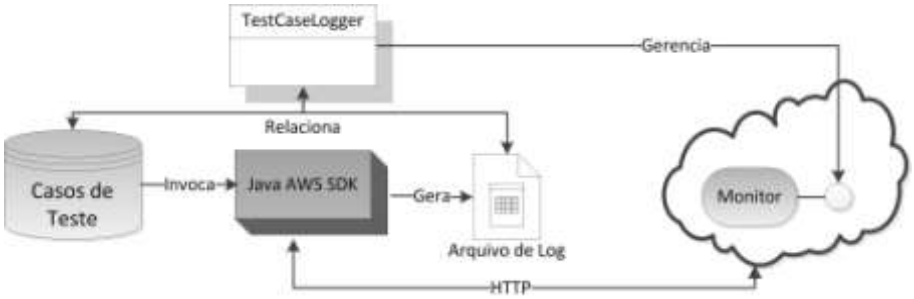

Figura 3. Arquitetura dos casos de teste gerados na Fase 1.

Baseado na arquitetura mostrada na Figura 3, foi utilizada a ferramenta IDE Eclipse [Eclipse 2012] em conjunto com o *framework* JUnit [JUnit 2012] para a geração de um caso de teste para cada método da API disponível no AWS SDK Java. A execução dos casos de teste foi organizada de modo que respeitasse a sequência exibida na Tabela 1. Tal sequência permite criar um conjunto de elementos (*Gateways*, DHCPs, *Images*, etc), realizar operações de modificação sobre estes elementos, consultar se tais mudanças foram realizadas e finalmente remover os elementos criados.

Tabela 1. Sequência de execução dos casos de teste.

Ord.	Operação	Métodos
1	Criar	*CreateCustomerGateway, CreateDhcpOptions, CreateImage, CreateInternetGateway, CreateKeyPair[12], CreateNetworkAcl, CreateNetworkAclEntry, CreatePlacementGroup, CreateRoute, CreateRouteTable, CreateSecurityGroup[12], CreateSnapshot[12], CreateSpotDatafeedSubscription, CreateSubnet, CreateTags, CreateVolume[12], CreateVpc, CreateVpnConnection, CreateVpnGateway*
2	Modificar	*AllocateAddress[123], AssociateAddress[3], AssociateDhcpOptions, AssociateRouteTable, AttachInternetGateway, AttachVolume[2], AttachVpnGateway, AuthorizeSecurityGroupEgress, AuthorizeSecurityGroupIngress[12], BundleInstance, CancelBundleTask, CancelSpotInstanceRequests, DeregisterImage[2], DetachInternetGateway, DetachVolume[2], DetachVpnGateway, DisassociateAddress[3], DisassociateRouteTable, ImportKeyPair[2], ModifyImageAttribute, ModifyInstanceAttribute, ModifySnapshotAttribute, MonitorInstances, PurchaseReservedInstancesOffering, RebootInstances[12], RegisterImage[2], ReleaseAddress[3], ReplaceNetworkAclAssociation, ReplaceNetworkAclEntry, ReplaceRoute, ReplaceRouteTableAssociation, RequestSpotInstances, ResetImageAttribute, ResetInstanceAttribute, ResetSnapshotAttribute, RevokeSecurityGroupEgress, RevokeSecurityGroupIngress[12], RunInstances[123], StartInstances[12], StopInstances[12], TerminateInstances[123], UnmonitorInstances*
3	Consultar	*ConfirmProductInstance[2], DescribeAddresses[123], DescribeAvailabilityZones[12], DescribeBundleTasks, DescribeConversionTasks, DescribeCustomerGateways, DescribeDhcpOptions, DescribeImageAttribute, DescribeImages[123], DescribeInstanceAttribute, DescribeInstances[123], DescribeInstanceStatus, DescribeInternetGateways, DescribeKeyPairs[12], DescribeNetworkAcls, DescribePlacementGroups, DescribeRegions[12], DescribeReservedInstances[2], DescribeReservedInstancesOfferings[2], DescribeRouteTables, DescribeSecurityGroups[12], DescribeSnapshotAttribute, DescribeSnapshots[12], DescribeSpotDatafeedSubscription, DescribeSpotInstanceRequests, DescribeSpotPriceHistory, DescribeSubnets, DescribeTags, DescribeVolumes[123], DescribeVpcs, DescribeVpnConnections, DescribeVpnGateways, GetConsoleOutput[12], GetPasswordData*
4	Deletar	*DeleteCustomerGateway, DeleteDhcpOptions, DeleteInternetGateway, DeleteKeyPair[12], DeleteNetworkAcl, DeleteNetworkAclEntry, DeletePlacementGroup, DeleteRoute, DeleteRouteTable, DeleteSecurityGroup[12], DeleteSnapshot[12], DeleteSpotDatafeedSubscription, DeleteSubnet, DeleteTags, DeleteVolume[12], DeleteVpc, DeleteVpnConnection, DeleteVpnGateway*

A partir da geração dos casos de teste, foi observado que dos 125 métodos da API, 12 métodos (mostrados na Tabela 2) não são implementados pelo SDK fornecido pela Amazon. Ao se consultar a documentação da Amazon, constatou-se que esta é uma limitação intrínseca da versão do SDK utilizada.

Tabela 2. Métodos não avaliados.

Métodos da EC2 API não implementados pelo AWS SDK Java		
AttachNetworkInterface	DescribeNetworkInterfaceAttribute	ImportVolume
CancelConversionTask	DescribeNetworkInterfaces	ModifyNetworkInterfaceAttribute
CreateNetworkInterface	DetachNetworkInterface	ReportInstanceStatus,
DeleteNetworkInterface	ImportInstance	ResetNetworkInterfaceAttribute

4.3. Segunda Fase: Compatibilidade sintática

Na avaliação da compatibilidade sintática, o objetivo é verificar se a implementação da interface AWS EC2 disponibilizada pela Plataforma B (OpenStack, Eucalyptus e OpenNebula) é compatível com sintaxe dos métodos fornecidos pela Plataforma A (Amazon EC2). Assim, considerando que os casos de teste (CTs) gerados na fase anterior representam as funcionalidades da API EC2, na Amazon, o objetivo desta fase é avaliar se o mesmo conjunto de CTs são executados, com sucesso, para cada Plataforma B avaliada. Se o caso de teste for executado com sucesso (obter um *status code* entre valor 200 a 206), então tal método testado é considerado sintaticamente compatível com a API EC2.

Cada requisição a API EC2 baseia-se no protocolo HTTP. Dessa forma, a ordem dos parâmetros não interfere no código de retorno. Caso haja a exclusão de algum parâmetro é retornado um código de erro e considera-se não compatível. Para observar esse comportamento foi necessário manipular o código do AWS SDK. Dado que ao utilizar apenas o AWS SDK tal comportamento é encapsulado em objetos de retorno. Salientamos que foi preciso diminuir o nível de segurança das requisições submetidas às nuvens OpenStack e Eucalyptus, uma vez que por padrão o AWS SDK utiliza a versão "V2" do algoritmo "*HmacSHA1*", tendo sido necessário fazer uso da versão "V1" nessas duas plataformas.

Com a avaliação da plataforma da Amazon EC2, realizada na fase 1, foram identificados 113 métodos que tiveram suas chamadas implementadas em casos de testes. Os resultados obtidos na segunda fase revelaram um comportamento singular da plataforma OpenNebula, uma vez que todos os métodos foram compatíveis sintaticamente. Na Eucalyptus, 34 foram compatíveis e na OpenStack, 26. Os resultados comparativos da Fase 1 e Fase 2 são apresentados na Tabela 3.

Tabela 3. Resultados das Fases 1 e 2 do processo avaliativo.

Plataformas Avaliadas	Número de Métodos compatíveis	
	Fase 1	Fase 2
Amazon EC2	113	n/a
OpenStack	n/a	26
Eucalyptus	n/a	35
OpenNebula	n/a	113

4.4 Terceira Fase: Compatibilidade semântica

A terceira fase objetiva monitorar o comportamento na plataforma B (OpenStack, Eucalyptus e OpenNebula) durante a execução dos casos de testes (CTs) selecionados na fase 2. O monitoramento irá garantir se a ação solicitada pelo caso de teste foi realizada como esperado na plataforma avaliada. Para isso, deve ser implementado um monitor para cada caso de teste que foi executado com sucesso na fase anterior. Considerando que, em muitos casos, a implementação do monitor não é uma tarefa

trivial, a divisão da avaliação em compatibilidade sintática e semântica permitiu reduzir o número de monitores implementados para as plataformas OpenStack e Eucalyptus.

Um cenário que exemplifica a necessidade do monitoramento para comprovar a compatibilidade semântica entre plataformas é representado pelo comportamento do método *RunInstances*. Esse método inicia a execução de uma ou mais instância(s) na nuvem, tal operação não é realizada de forma síncrona. Isto ocorre, pois para efetuar tal ação a plataforma deve carregar as informações da imagem para que o *hypervisor* (gerenciador de máquinas virtuais) responsável possa iniciar a execução da máquina virtual que representará a instância. Todo esse processo demanda certo intervalo de tempo, de forma que uma chamada ao método *RunInstances* apenas garante que o processo foi iniciado. Assim, a comprovação da eficácia da chamada efetuada, isto é, se a instância está de fato executando, deve ser realizada por um monitor de comportamento.

Como mostrado na Figura 1, os mesmos casos de teste gerados na fase 1 e que foram sintaticamente compatíveis na fase 2 são utilizados na fase 3. No caso de teste ilustrado na Figura 2, para realizar o monitoramento das ações invocadas é necessário apenas mudar o valor da variável *monitorFlag* de *false* para *true*. Tal mudança fará com que a classe *TestCaseLogger* procure por um monitor previamente registrado para monitorar tal ação. Além disso, quando o segundo parâmetro do método *getstatusCode* é informado como *true,* este método fica bloqueado até que o monitor retorne se a operação foi bem sucedida ou não.

```
01  public class RunInstanceResource extends ServerResource implements RunInstace {
02     @Get
03     public int monitore(String instanceId) {
04        return monitoreTentativa(0, instanceId);
05     }
06     private int monitoreTentativa(int tentativa, String instanceId){
07        int retorno = 400;
08        try {
09           if(OpenStackAPIClient.getStatusInstance(instanceId).equals("running")){
10              return 200;
11           }else{
12              if(tentativa <= 9){
13                 Thread.sleep(30000);
14                 return monitoreTentativa(++tentativa, instanceId);
15              }
16           }
17        } catch (InterruptedException e) {
18           e.printStackTrace();
19        }
20           return retorno;
21        }
22  }
```

Figura 4. **Exemplo de Implementação do Monitor.**

Um exemplo de monitor implementado em Java, com o Restlet *Framework* [Restlet 2012], é ilustrado na Figura 4. Neste caso, o comportamento do *RunInstances* é monitorado, através do método *monitore*. Tal método recebe como parâmetro o *instanceId* (linha 03) que representa a identificação única da instância iniciada via o caso de teste *testRunInstances*. Através desse parâmetro o monitor requisita informações sobre a instância que está sendo iniciada. Dentre os atributos retornados, o *instanceState* define o estado da instância, que pode ser: *pending, running, shutting-down, terminated, stopping* ou *stopped*. Para isso, o monitor efetua 10 requisições (linha

12-14) à plataforma, sendo uma a cada 30 segundos (linha 13), consultando se a instância daquele *instanceId* já possuí um *instanceState* de valor *running* (linha 09), caso isso ocorra, o monitor retorna para o *TestCaseLogger* um *status code* com valor *200* (linha 20), indicando que a operação foi bem sucedida. Note que o monitor descrito na Figura 4 está executando em uma instancia da nuvem OpenStack e usa a própria API do OpenStack para realizar tal monitoramento.

Após a execução dos casos de teste e dos seus respectivos monitores, os resultados obtidos mostram que, dos 113 métodos avaliados: na OpenStack, 26 são compatíveis e 87 não são compatíveis; na Eucalyptus, 34 são compatíveis e 79 não; e na OpenNebula, 10 são compatíveis, enquanto 103 não são. A Tabela 4 resume o número de métodos compatíveis para cada fase do processo de avaliação. A lista de todos os métodos compatíveis pode ser obtida na Tabela 1. Os métodos que possuem um número são métodos que obtiveram compatibilidade semântica com alguma plataforma. A presença do número 1 indica que o método é compatível com a plataforma OpenStack, enquanto que 2 representa compatibilidade com a plataforma Eucalyptus e 3 indica a compatibilidade com OpenNebula. Por exemplo, o método *RunInstances*[123] é compatível com todas as plataformas avaliadas, enquanto que *DetachVolume*[2] é compatível apenas com a plataforma Eucalyptus.

Tabela 4. Resultados por Fases (1,2 e 3) do processo avaliativo.

Plataformas Avaliadas	Número de Métodos compatíveis		
	Fase 1	Fase 2	Fase 3
Amazon EC2	113	n/a	n/a
OpenStack	n/	26	26
Eucalyptus	n/a	34	34
OpenNebula	n/a	113	10

Quanto aos métodos compatíveis na terceira fase, as plataformas Eucalyptus e OpenStack obtiveram o mesmo resultado da fase anterior. Por outro lado, a plataforma OpenNebula apresentou apenas 10 métodos compatíveis quando na fase anterior tinha sido a melhor avaliada. Esse comportamento ocorre devido a implementação da API EC2 dessa plataforma possuir um código que retorna um *status code* de sucesso (200) para qualquer chamada efetuada a ela, mesmo essa chamada não estando propriamente implementada.

Considerando apenas os 113 métodos testados, a compatibilidade das plataformas avaliadas foi de 23,01% na OpenStack, 30,09% na Eucalyptus e 8,85% na OpenNebula. Considerando que quanto menor for a compatibilidade entre plataformas, maior será o aprisionamento. Concluímos que o aprisionamento nas plataformas objetos da avaliação foi de 76,99% (100% - 23,01%), 69,61% (100% - 30,09%) e 91,15% (100% - 8,85%) para a OpenStack, Eucalyptus e OpenNebula, respectivamente.

Quando analisamos a compatibilidade por funcionalidade, observamos que algumas funcionalidades chegam a ter 100% de compatiblidade, entretanto a grande maioria ainda possui 0% de compatibilidade. Na Tabela 5 apresentamos os resultados agrupados por funcionalidade e adotamos as abreviaturas **M** para número total de métodos da funcionalidade, **C** para quantidade de compatíveis, **N** para não compatíveis e **%C** para a porcentagem da compatibilidade.

Tabela 5. Compatibilidade por Função EC2 API.

Amazon EC2 API		Plataformas Avaliadas								
		OpenStack			Eucalyptus			OpenNebula		
Funcionalidade	M	C	NC	%C	C	NC	%C	C	NC	%C
Amazon DevPay	1	0	1	0 %	1	0	100 %	0	1	0 %
AMIs	7	1	6	14,29 %	3	4	42,86 %	1	6	14,29 %
Availability Zones and Regions	2	2	0	100 %	2	0	100 %	0	2	0 %
Customer Gateways	3	0	3	0 %	0	3	0 %	0	3	0 %
DHCP Options	4	0	4	0 %	0	4	0 %	0	4	0 %
Elastic Block Store	11	6	5	54,55 %	8	3	72,73 %	0	11	0 %
Elastic IP Addresses	5	2	3	40 %	2	3	40 %	5	0	100 %
General	1	1	0	100 %	1	0	100 %	0	1	0 %
Instances	10	6	4	60 %	6	4	60 %	4	6	40 %
Internet Gateways	5	0	5	0 %	0	5	0 %	0	5	0 %
Key Pairs	4	3	1	75 %	4	0	100 %	0	4	0 %
Monitoring	2	0	2	0 %	0	2	0 %	0	2	0 %
Network ACLs	7	0	7	0 %	0	7	0 %	0	7	0 %
Placement Groups	3	0	3	0 %	0	3	0 %	0	3	0 %
Reserved Instances	3	0	3	0 %	2	1	66,67 %	0	3	0 %
Route Tables	9	0	9	0 %	0	9	0 %	0	9	0 %
Security Groups	7	5	2	71,43 %	5	2	71,43 %	0	7	0 %
Spot Instances	7	0	7	0 %	0	7	0 %	0	7	0 %
Subnets	3	0	3	0 %	0	3	0 %	0	3	0 %
Tags	3	0	3	0 %	0	3	0 %	0	3	0 %
VM Import	1	0	1	0 %	0	1	0 %	0	1	0 %
VPCs	3	0	3	0 %	0	3	0 %	0	3	0 %
VPN Connections	3	0	3	0 %	0	3	0 %	0	3	0 %
Virtual Private Gateways	5	0	5	0 %	0	5	0 %	0	5	0 %
Windows	4	0	4	0 %	0	4	0 %	0	4	0 %
Totais	**113**	**26**	**87**	**23,01 %**	**34**	**79**	**30,09 %**	**10**	**103**	**8,85 %**

Conforme os resultados apresentados, a plataforma Eucalyptus, que foi a de maior compatibilidade, teve sua compatibilidade igual a apenas 30,09%. Comprovando que a compatibilidade com a Amazon EC2 API não ocorre em nenhuma das plataformas avaliadas. Caso seja considerada a análise apenas das funcionalidades principais, como afirma [von Laszewski, et al. 2012], *AMIs* (gerência de imagens) e *Instances* (gerência do ciclo de vidas das instâncias), a plataforma Eucalyptus novamente destaca-se como melhor avaliada (52,94%), a OpenStack fica com 41,18% e a OpenNebula com 29,41%.

Como justificativas para a não compatibilidade foram elencados três motivos: (i) a plataforma não possuía a funcionalidade avaliada, (ii) a plataforma possui tal funcionalidade, mas ela é incompatível com a EC2 e (iii) a plataforma possui tal funcionalidade, mas o módulo EC2 da plataforma não implementa tal chamada. Os resultados dessa análise estão exibidos na Tabela 6 que segue a notação **I** para o primeiro motivo, **II** para o segundo e **III** para o terceiro.

Tabela 6. Motivos para Não Compatibilidade.

Plataformas Avaliadas	Motivos para Não compatibilidade		
	I	II	III
OpenStack	69	6	11
Eucalyptus	61	1	9
OpenNebula	74	0	29

As funcionalidades que expõem serviços próprios da Amazon foram as principais responsáveis pelos resultados de não compatibilidade, como exemplo destacam-se: *Network ACLs*, *Placement Groups*, *Virtual Private Gateways*, entre outras. E mesmo uma das funcionalidades principais como *Instances*, não obteve 100% de compatibilidade em nenhuma das plataformas.

5. Conclusão

Este artigo propôs um processo que avalia o aprisionamento entre plataformas de nuvens. O processo é baseado em teste de compatibilidade entre diferentes plataformas. Assim, quanto maior a compatibilidade entre as plataformas menor será o aprisionamento. O processo foi instanciado através de uma avaliação do aprisionamento entre a API da Amazon EC2 e as plataformas OpenStack, Eucalyptus e OpenNebula.

Apesar de vários artigos [Peng et al 2009; Mahjoub et al. 2011; von Laszewski, et. al 2012] concluírem que as plataformas OpenStack, Eucalyptus e OpenNebula fornecem compatibilidade com a API da Amazon EC2, neste artigo observamos, de forma sistemática, que a compatibilidade foi de apenas 23,01% na OpenStack, 30,09% na Eucalyptus e 8,85% na OpenNebula. Esses resultados indicam que aplicações que usam extensivamente a API EC2 sofrem aprisionamento (*cloud lock-in)* na Amazon, dado que a compatibilidade das plataformas avaliadas com a API EC2 ainda é reduzida. Abordagens como [OCCI 2013] e [JClouds 2013] foram propostas para reduzir o aprisionamento, constituindo-se em uma possível solução aos questionamentos que surgem pelos resultados apresentados. Como trabalhos futuros pretendemos selecionar outras plataformas de nuvem livres que disponibilizem acesso via EC2 API aplicando sobre elas estudo similar.

Agradecimentos

Este trabalho é parcialmente apoiado pelo Instituto Nacional de Ciência e Tecnologia para Engenharia de Software (INES), financiado pelo CNPq e pela FAPERN, com os projetos 573964/2008-4 e PPP-III-013/ 2009.

Referências

Armbrust, M., Fox, A., Griffith, R., et al. (2010). “A View of Cloud Computing”. Communications of The Acm, Vol 53, No 4, p. 50-58.

AWS EC2API (2012). Amazon Elastic Compute Cloud. “API Reference Version 15/12/2011”. http://docs.amazonwebservices.com/AWSEC2/latest/APIReference/Welcome.html?r=8186, Fevereiro.

AWS JAVA (2012). Amazon Web Service (2012). “AWS SDK for Java”. http://aws.amazon.com/pt/sdkforjava/, Fevereiro.

Buyya R.; Broberg J.; Goscinski A. (2011). “Cloud Computing Principles e Paradigms.”, John Wiley & Sons ltd., England.

EC2 (2012). Amazon Web Services. “Amazon Elastic Compute Cloud (Amazon EC2)”, http://aws.amazon.com/pt/ec2/, Fevereiro.

Eclipse (2012). Eclipse Project, http://www.eclipse.org/, Março.

Endo, P. T.; Estácio, G. G.; Judith, K.; Sadok, D. (2010). “A Survey on Open-source Cloud Computing Solutions.” VIII Workshop em Clouds, Grids e Aplicações.

Eucalyptus (2012). “Eucalyptus open-source cloud computing infrastructure – an overview”. Technical report, Eucalyptus, Inc., August 2009.

Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008). Cloud computing and grid computing

360-degree compared. GCE'08,p. 1–10. IEEE.

Flores, A., and Polo, M.. 2009. Testing-based Process for Evaluating Component Replaceability. *Electron. Notes Theor. Comput. Sci.* 236 (April 2009), 101-115.

HTTP/1.1 (2012). Hypertext Transfer Protocol -- HTTP/1.1. http://www.ietf.org/rfc/rfc2616.txt, Março.

JClouds (2013). Cloud interfaces, simplified. http://www.jclouds.org/

JUnit (2012). Welcome to the JUnit wiki, https://github.com/kentbeck/junit/wiki, Março.

Juve, G., Deelman, E., Vahi, K., Mehta, G., et al (2009). Scientific Workflow Applications on Amazon EC2. e-Science 2009, Oxford, UK.

Juve, G., Deelman, E., Vahi, K., Mehta, G., et al (2010). Data Sharing Options for Scientific Workflows on Amazon EC2. Proceedings of the 2010 ACM/IEEE SC10

Khan, R.H., Ylitalo, J., Ahmed, A.S., (2011). OpenID Authentication As A Service in OpenStack. 7th IAS, p. 372-377.

Lenk, M., Menzel, M., Lipsky, J. et al (2011). What are you paying for? Performance benchmarking for Infrastructure-as-a-Service offerings. 2011 4th IEEE Cloud.

Lewis, W. E. Software Testing and Continuous Quality Improvement. CRC Press LLC, Apr. 2000.

Mariani, L., Papagiannakis, S., and Pezze, M. 2007. Compatibility and Regression Testing of COTS-Component-Based Software. ICSE '07, 85-95.

Mahjoub, M., Mdhaffar A., et al (2011). A comparative study of the current Cloud Computing technologies and offers. 2011 IEEE NCCA. IEEE. p. 131 a 134.

OCCI (2013). Open Cloud Computing Interface. http://occi-wg.org.

OpenStack (2012). OpenStack Compute Stater Guide.

http://docs.openstack.org/diablo/openstack-compute/starter/openstack-starter-guide-diablo.pdf

OpenNebula (2012). .http://opennebula.org/. Novembro.

Peng, J.; Zhang, X., Lei, Z., Zhang, B., Zhang, W., Li, Q. (2009). "Comparison of Several Cloud Computing Platforms". ISISE '09, p. 23-27.

Restlet (2012). RestFul web framework for Java . http://www.restlet.org/. Novembro.

Turcu, G., Foster, I., Nesterov, S. (2010). Reshaping text data for efficient processing on Amazon EC2. Proceedings of the 19th ACM HPDC.

Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. (2009). "A Break in the Clouds: Towards a Cloud Definition". ACM SIGCOMM Computer Communication Review, Vol. 39(1), p.50–55, Janeiro de 2009.

von Laszewski, G.; Diaz, J.; Fugang Wang; Fox, G.C. (2012). "Qualitative Comparison of Multiple Cloud Frameworks". Proceedings: 2012 IEEE 5th international conference on cloud computing, p. 734-741.