Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Controle de Acesso na Plataforma de Nuvem Federada BioNimbuZ

Heitor Henrique de Paula Moraes Costa

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo

Coorientador
Prof. MSc. João José Costa Gondim

Brasília
2015

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Homero Luiz Piccolo

Banca examinadora composta por:

Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo (Orientador) — CIC/UnB
Prof. MSc. João José Costa Gondim — CIC/UnB
Prof.ª Dr.ª Maria Emília Machado Telles Walter — CIC/UnB
Prof.ª Dr.ª Maristela Terto de Holanda — CIC/UnB

CIP — Catalogação Internacional na Publicação

Costa, Heitor Henrique de Paula Moraes.

Controle de Acesso na Plataforma de Nuvem Federada BioNimbuZ / Heitor Henrique de Paula Moraes Costa. Brasília : UnB, 2015.
147 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2015.

1. Computação em Nuvem, 2. Nuvem Federada, 3. Segurança,
4. Controle de Acesso, 5. BioNimbuZ.

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Controle de Acesso na Plataforma de Nuvem Federada BioNimbuZ

Heitor Henrique de Paula Moraes Costa

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo (Orientador)
CIC/UnB

Prof. MSc. João José Costa Gondim
CIC/UnB

Prof.ª Dr.ª Maria Emília Machado Telles Walter
CIC/UnB

Prof.ª Dr.ª Maristela Terto de Holanda
CIC/UnB

Prof. Dr. Homero Luiz Piccolo
Coordenador do Bacharelado em Ciência da Computação

Brasília, 06 de julho de 2015

Dedicatória

Dedico este trabalho aos meus pais, que sempre me incentivaram, me educaram e me ajudaram a ser o que eu sou hoje. Dedico também ao Alexandre Dantas, o kure, um grande amigo que sempre me ajudou e me ensinou bastante, mas agora não está mais entre nós.

Agradecimentos

Agradeço primeiramente a Deus, que sempre me deu forças para continuar, principalmente nos momentos mais difíceis desta caminhada.

Aos meus pais, por acreditarem em mim e investirem na minha educação, pois sem este apoio eu jamais teria chegado aonde cheguei. Ao meu irmão, pelo companheirismo, pela amizade e pelas muitas horas de conversa compartilhando ideias e reflexões.

Agradeço imensamente a minha professora e orientadora, Aletéia Patrícia, pela dedicação e paciência, sempre me guiando na condução deste trabalho. Aos amigos que conheci graças ao trabalho em conjunto no BioNimbuZ, que sempre me ajudaram e me motivaram, mesmo nos momentos mais cansativos e depois de longas horas de reunião.

A todos os meus amigos do 2º/2010, por dividirem comigo muitas horas de estudo, incluindo madrugadas e finais de semana, e também os momentos de diversão, nas jogatinas, saídas ao cinema e partidas de futebol. Em especial: Filipe Caldas, Paulo Afonso, Wallace Bruno, André Belle, Ítalo Batista, Felipe Rodopoulos, Silas Souza e Guilherme Alexsander.

Agradeço a Rafael Procópio, Maxuell Martins, Priscila Batista, Leonardo Rafael e Victor Augusto e a todos os meus amigos, por me ajudarem, por estarem presentes em vários momentos da minha vida e por me distraírem quando eu precisava esquecer as obrigações.

Muito obrigado!

Resumo

Um ambiente de nuvens federadas é composto por nuvens interligadas entre si, que compartilham recursos utilizando uma interface transparente ao usuário. Nestes ambientes, garantir a segurança dos usuários não é uma tarefa fácil, pois é preciso levar em consideração as características individuais dos provedores pertencentes à federação, sendo este um dos motivos que prejudicam a completa adoção destes ambientes pelas empresas e organizações. Este trabalho trata do problema da segurança em nuvens federadas, em especial na plataforma BioNimbuZ. Neste contexto, este trabalho propõe a utilização de um modelo de controle de acesso baseado em atributos para decidir se um usuário pode ou não acessar um determinado recurso. Os experimentos realizados mostram que o modelo é capaz de restringir, com eficiência, o acesso aos recursos somente aos usuários autorizados.

Palavras-chave: Computação em Nuvem, Nuvem Federada, Segurança, Controle de Acesso, BioNimbuZ.

Abstract

A federated cloud environment consists of interconnected clouds that share resources with each other using an interface transparent to the user. In these environments, ensuring the safety of users is not an easy task, as it is necessary to take into account the individual characteristics of the federation providers, which is one of the reasons that hinder the full adoption of these environments by companies and organizations. This work deals with the problem of security in federated clouds, especially BioNimbuZ platform. In this context, this work proposes the use of an attribute-based access control model to decide whether or not a user can access a given resource. The experiments conducted show that the model is able to efficiently restrict resources access to authorized users.

Keywords: Cloud Computing, Federated Cloud, Security, Access Control, BioNimbuZ.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Desde o surgimento da Internet, o mundo vem sofrendo diversas transformações, sobretudo na forma de utilização do poder computacional, que se aproveitou da rede mundial de computadores para ganhar eficiência e escalabilidade. Antes da Internet, a maior parte do processamento era realizado em servidores locais e *datacenters* proprietários, que mesmo em seus momentos ociosos geravam custos com manutenção e energia elétrica, e não utilizavam seu poder computacional por completo. Neste cenário, surgiu a ideia de um modelo de computação sob demanda, de forma escalável, no qual os usuários pagam apenas por aquilo que foi de fato utilizado. Este modelo recebeu o nome de computação em nuvem [1].

Contudo, não existe na literatura um consenso a respeito da definição de computação em nuvem [1] [2] [3], porém é possível identificar alguns pontos chaves presentes nestas definições, como o fato de ser um sistema distribuído, utilizar a Internet para prover recursos, ser sob demanda e escalável. Assim, é possível definir a computação em nuvem como um modelo de computação distribuída que permite o acesso a diversos recursos, tais como capacidade de processamento e de armazenamento, que são adquiridos como serviço. Além disso, a infraestrutura de nuvem passa a ilusão de que os recursos usados são ilimitados. Todavia, com o passar dos anos, novas necessidades surgiram, e a utilização de nuvens isoladas passou a ser insuficiente para suprir a demanda das aplicações pelo consumo de recursos [4].

Para solucionar este impasse, emergiu o conceito de nuvens federadas, isto é, um conjunto de nuvens interligadas, gerenciadas por uma interface que proporciona uma abstração para o usuário de modo que ele não perceba que são várias nuvens. Assim sendo, em um ambiente de nuvens federadas, várias nuvens podem compartilhar recursos entre si, possibilitando uma expansão maior do que aquela permitida apenas por uma única nuvem, e gerando uma melhor alocação de recursos.

Algumas arquiteturas para federação de nuvens foram propostas [4] [5], visando prover recursos em uma escala maior do que uma nuvem isolada conseguiria. Saldanha [6] propôs uma arquitetura chamada BioNimbuZ, que tem como objetivo garantir de forma transparante, dinâmica e eficiente a execução de aplicações em um ambiente de nuvens federadas.

Todavia, nota-se que apesar de ter aumentado a utilização da computação em nuvem e até mesmo de nuvens federadas, ainda existe uma enorme preocupação com a segurança no uso destas plataformas. E esse é um motivo que prejudica consideravelmente a com-

pleta adoção deste paradigma de computação por organizações e empresas que possuem dados confidenciais [7]. Garantir a segurança em um ambiente de nuvem federada é complexo, pois deve-se levar em cosideração as diferenças entre cada provedor de nuvem, as diversas formas de acesso e a grande quantidade de aplicações que podem executar neste ambiente [8].

Na tentativa de contribuir para a resolução do problema da segurança no ambiente de nuvem federada do BioNimbuZ, este trabalho propôs e implementou um modelo de controle de acesso neste ambiente. O modelo proposto para utilização tem como objetivo garantir que somente pessoas autorizadas tenham acesso aos recursos da federação.

1.1 Motivação

Devido a enorme quantidade de dados que precisam ser armazenados ou processados, a utilização de nuvem federada vem se tornando cada vez mais comum. Assim, é natural que cresça também o interesse nesse campo de estudo. Porém, ainda existe um receio por parte dos usuários com a questão da segurança na utilização desse tipo de plataforma.

Diante deste contexto, este trabalho propõe a criação de um módulo de segurança que implemente um controle de acesso, que garante que a confidencialidade seja preservada na plataforma BioNimbuZ.

1.2 Problema

Não há na plataforma BioNimbuZ uma política de segurança implementada. Assim, a política proposta neste trabalho deve:

- Implementar algum modelo de controle de acesso que seja adequadado a uma arquitetura de federação de nuvem;
- Garantir que um determinado arquivo, dado ou sistema não seja acessado por alguma pessoa não autorizada;

1.3 Objetivos

1.3.1 Principal

O objetivo principal deste trabalho é criar o serviço de segurança no BioNimbuZ, focando, principalmente, no controle de acesso, de modo a garantir que todos os dados e recursos só possam ser acessados por usuários que tenham permissão para tal.

1.3.2 Específicos

Além do principal, este trabalho possui os seguintes objetivos específicos:

- Implementar um modelo de controle de acesso para plataforma de nuvem federada;
- Realizar a integração com o BioNimbuZ;

- Realizar um estudo de caso, a fim de comprovar a eficácia do modelo;
- Discutir os resultados obtidos, verificando as vantagens e as desvantagens do controle de acesso proposto.

1.4 Descrição dos Capítulos

Este trabalho está dividido, além deste, em mais seis capítulos. No Capítulo 2 são apresentados os conceitos de sistemas distribuídos e as definições de *clusters*, *grids* e *cloud*. Também são mostradas as diferenças entre os dois primeiros sistemas e a computação em nuvem, de forma a tornar claro esses conceitos.

O Capítulo 3 apresenta a plataforma BioNimbuZ, mostrando sua estrutura, sua organização, seus objetivos e as modificações sofridas desde a sua proposta original.

No Capítulo 4 são apresentados os conceitos de segurança em nuvem e quais os principais pontos a serem tratados na política de segurança proposta. O foco desse capítulo, no entanto, é a questão do controle de acesso.

No Capítulo 5 é detalhada a criação do serviço de segurança proposto, que contém o controle de acesso da plataforma BioNimbuZ.

No Capítulo 6 são apresentados os resultados obtidos e uma análise dos mesmos. E por fim, no Capítulo 7 são apresentadas as conclusões obtidas e demais trabalhos futuros, que podem melhorar o serviço de segurança do BioNimbuZ.

Capítulo 2

Computação em Nuvem

Neste capítulo serão apresentados alguns sistemas distribuídos, como o *cluster*, o *grid* e a computação em nuvem. Serão apresentadas nas Seções 2.1 e 2.2 suas principais características, as suas arquiteturas e o modo como funcionam. Além disso, serão exploradas as suas diferenças, pois estes sistemas muitas vezes são confundidos.

A Seção 2.3 diz respeito às nuvens federadas que surgiram com a necessidade de aumentar o poder computacional e melhorar a provisão de serviços das nuvens. Serão apresentados também os desafios de projeto relacionados à construção de uma federação de nuvens, e duas propostas de arquitetura.

2.1 Sistemas Distribuídos

Com a utilização cada vez maior dos computadores para realizar cálculos complexos, os computadores monoprocessados deixaram de ser suficientes e não estavam mais atendendo às necessidades. Assim, atualmente, existem três possíveis soluções para este problema: A primeira é aumentar a frequência do processador; A segunda é melhorar o algoritmo; E a terceira é utilizar mais computadores [9].

A primeira solução foi bastante eficiente durante muitos anos, inclusive seguindo a Lei de Moore [10]. Esta solução, porém, esbarra em um limite físico e pode estar chegando ao fim [11], por isso novas alternativas devem ser pesquisadas. A segunda solução pode não ser muito eficaz se tratando de alguns problemas, como cálculos de bioinformática, previsão de clima e simulações de movimentação dos oceanos, que mesmo com algoritmos mais eficientes não podem ser processadas por uma máquina monoprocessada devido a sua complexidade. Isto nos leva à terceira solução, que consiste na utilização de mais de um computador, dando origem aos sistemas distribuídos.

O conceito de sistemas distribuídos já existe há algum tempo e possui diversas definições na literatura. Algumas definições clássicas são apresentadas a seguir:

- Para Lages [12], um sistema distribuído é "um conjunto de elementos de computação que cooperam, trocando informação";
- Pitanga [13] diz que "um sistema distribuído é um conjunto de elementos que se comunicam através de uma rede de interconexão e que utilizam software de sistema distribuído. Cada elemento é composto por um ou mais processadores e uma ou mais memórias";

- Colouris [14] afirma que "um sistema distribuído é aquele no qual os componentes localizados em computadores interligados se comunicam e coordenam suas ações apenas passando mensagens";
- Segundo Tanenbaum [15], um sistema distribuído é "uma coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente".

Pelas definições é possível identificar alguns aspectos relevantes que caracterizam um sistema distribuído, pois eles são formados por múltiplos computadores (unidades de processamento), estão interconectados por uma rede e se comunicam por meio de troca de mensagem e, portanto, não compartilham memória [12].

Contudo, projetar um sistema distribuído esbarra em algumas dificuldades, como a ausência de memória global, a ausência de um *clok* global e a imprevisibilidade no retardo das mensagens. Como não existe uma memória única, a comunicação ocorre por trocas de mensagem, ficando sujeita à imprevisibilidade no retardo das mesmas, ou seja, é difícil definir se uma determinada mensagem foi perdida ou apenas sofreu um atraso. E a falta de um *clock* global dificulta o ordenamento de eventos, um entrave para o estabelecimento de dependências entre ações [15].

Alguns aspectos devem ser considerados durante o projeto de um sistema ditribuído, tais como transparência, tratamento de falhas, heterogeneidade, segurança, entre outros. As escolhas irão definir como funcionará o sistema e quais tecnologias devem ou não ser utilizadas.

A transparência é muito importante para fazer com que os usuários tenham uma visão única do sistema. Ela pode ser dividida em dois níveis, o nível de usuário e o de programador. O usuário deve ter a impressão de que está utilizando um sistema monoprocessado, enquanto que o programador deve ter a impressão de estar programando em um sistema monoprocessado, algo bem mais difícil. A transparência possui diversos tipos, como por exemplo transparêcia de acesso, localização, concorrência, replicação, falha, migração e paralelismo [14]. As quais são descritas a seguir:

- Transparência de Acesso: junto com a transparência de localização são os tipos básicos que caracterizam um sistema distribuído. É o que permite que um recurso possa ser acessado da mesma forma, independente de estar em uma máquina local ou remota.
- Transparência de Localização: se caracteriza pelo fato de que os usuários não estão cientes da localização física dos recursos, ou seja, não sabem em qual máquina cada recurso se encontra;
- Transparência de Concorrência: é o que garante que um usuário não perceba que existem outros usuários utilizando o sistema. E caso o mesmo recurso esteja sendo utilizado por múltiplos usuários, a coerência deve ser garantida;
- Transparência de Replicação: permite utilizar várias instâncias do mesmo recurso lógico, sem o conhecimento da existência de réplicas;
- Transparência de Falha: diz respeito ao fato do sistema poder esconder a ocorrência de falhas e continuar a execução mesmo que elas aconteçam;

- Transparência de Migração: oculta que um determinado recurso possa mudar de localização dentro do sistema. Um sistema que é transparente à migração, obrigatoriamente é transparente à localização, assim, tanto dados quanto processos, e até mesmo a computação podem sofrer migração;
- Transparência de Paralelismo: permite que o sistema escolha quantos processadores serão disponibilizados para uma determinada aplicação, de forma que sejam atendidos os critérios de balanceamento de carga, tempo de resposta, entre outros.

Outro aspecto relevante nos sistemas distribuídos é a heterogeneidade, que ocorre principalmente entre a rede, o hardware e o sistema operacional das máquinas. As diferenças na rede, geralmente, são resolvidas com a utilização de um protocolo, que consiste em uma convenção que permite a comunicação e a troca de dados. A heterogeneidade de sistema operacional deve ser resolvida com a utilização de uma camada de software de alto nível, a qual é chamada de *middleware*, como por exemplo CORBA [16] e J2EE (*Java Platform, Enterprise Edition*) [17].

O tratamento de falhas ocorre de diversas formas, começando pela detecção das falhas, passando pelo mascaramento das mesmas, até a recuperação. Nesse caso, a redundância é algo fundamental, visto que existem múltiplos computadores e esse potencial deve ser utilizado.

Além das caracterísitcas citadas, a segurança é fundamental em sistemas distribuídos, e será o foco deste trabalho. A seguir serão mostrados alguns dos exemplos de sistemas distribuídos existentes e serão feitas comparações a fim de não deixar dúvidas a respeito das diferenças entre eles.

2.1.1 *Cluster*

Um *cluster* é um conjunto de computadores conectados por uma rede de alta velocidade, que trabalham juntos como um único recurso integrado, e tem como objetivo resolver problemas de alta complexidade [18]. A ideia é aumentar o poder computacional a medida em que mais computadores são conectados, aproveitando o conceito de componentes *commodities* [12], que são bens uniformes e produzidos em larga escala. Geralmente, cada nó executa uma instância de um mesmo sistema operacional, porém em algumas configurações é possível encontrar nós executando sistemas operacionais diferentes. Assim, como acontece com o sistema operacional, é mais comum que os *clusters* contenham o mesmo hardware para cada nó, mas existem exceções [9].

Quando estes vários computadoes estão conectados, eles compartilham a execução dos programas, promovendo um balanceamento e trabalhando de forma a parecer um único computador virtual [19]. Do ponto de vista do usuário, existem vários computadores que funcionam como uma única máquina.

Assim, os programas que são executados em um *cluster* devem ser escritos com as divisões explicitamente declaradas para que cada nó execute paralelamente, caso contrário, o poder computacional não será de fato utilizado.

A Figura 2.1 apresenta uma arquitetura de um *cluster*. Nela, cada nó está dedicado quase que exclusivamente ao *cluster*, executando um sistema operacional em suas configurações mínimas. O mestre é o responsável por gerenciar e atribuir as tarefas aos outros nós, e por isso deve ser um computador mais potente para que não torne o gerenciamento

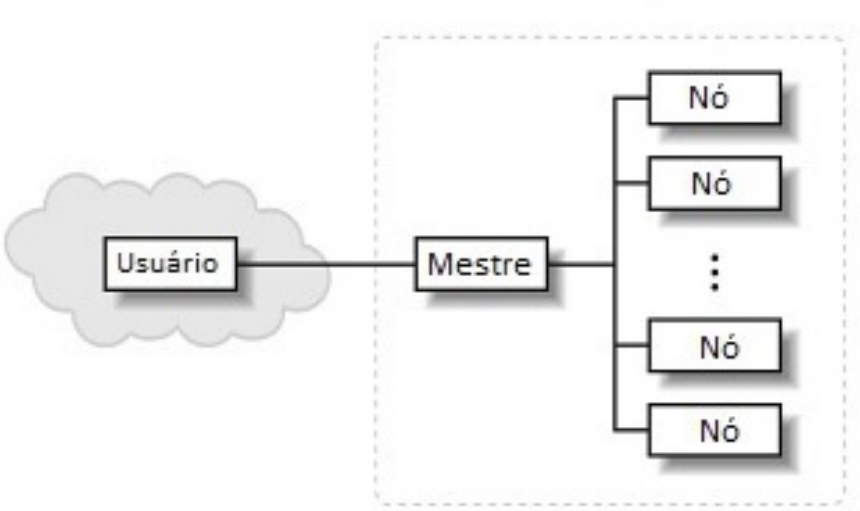

Figura 2.1: Arquitetura de um *Cluster* [9].

um gargalo. E o usuário que utiliza o *cluster* pode estar conectado à mesma rede local ou então à Internet para poder acessar os recursos computacionais.

2.1.2 *Grid* Computacional

Este termo é uma analogia com as redes de energia elétrica (do inglês *power grids*), pois é um sistema que está presente em vários lugares e que é acessado pelas pessoas de forma simples e natural, sem a preocupação de como aquela energia foi produzida.

Para Buyya [1], um *grid* é um tipo de sistema distribuído e paralelo que permite o compartilhamento, a seleção e a agregação de recursos geograficamente distribuídos em tempo de execução e de forma dinâmica. Por esta definição percebe-se que *grid* computacional é um sistema distribuído que utiliza recursos que estão dispersos geograficamente ao longo do planeta. Esses recursos são interligados por um rede de alta velocidade e, na maioria das vezes, são altamentes heterogêneos, ou seja, compreendem computadores com configurações, hardware e software diferentes, que são combinados para aumentar o poder computacional do sistema como um todo.

Este paradigma de computação evoluiu bastante no ramo científico, e projetos utilizando este tipo de sistema distribuído surgiram, como por exemplo o *Open Science Grid* [20], que possui estudos em diversas áreas, entre elas a física, as nanopartículas e a biologia molecular.

Apesar de ser bastante utilizado no meio científico, é complexo desenvolver aplicações que executem neste tipo de plataforma, pois diversas questões devem ser levadas em consideração, como a heterogeneidade, o dinamismo, a escalabilidade e a segurança. Como existem diversos computadores conectados, deve-se levar em conta as possíveis diferenças entre hardware e software. Quanto ao dinamismo e escalabilidade, deve-se estar atento quanto à disponíbilidade, ou seja, um computador pode não estar mais disponível para o *grid* caso esteja sendo utilizado pelo usuário. E garantir a segurança é um problema, pois os recursos estão espalhados em diversos lugares e o controle sobre cada um deles se torna complicado.

Para tentar superar estas dificuldades, Foster et al. [21] propuseram uma arquitetura que se organiza em camadas, e pode ser vista na Figura 2.2.

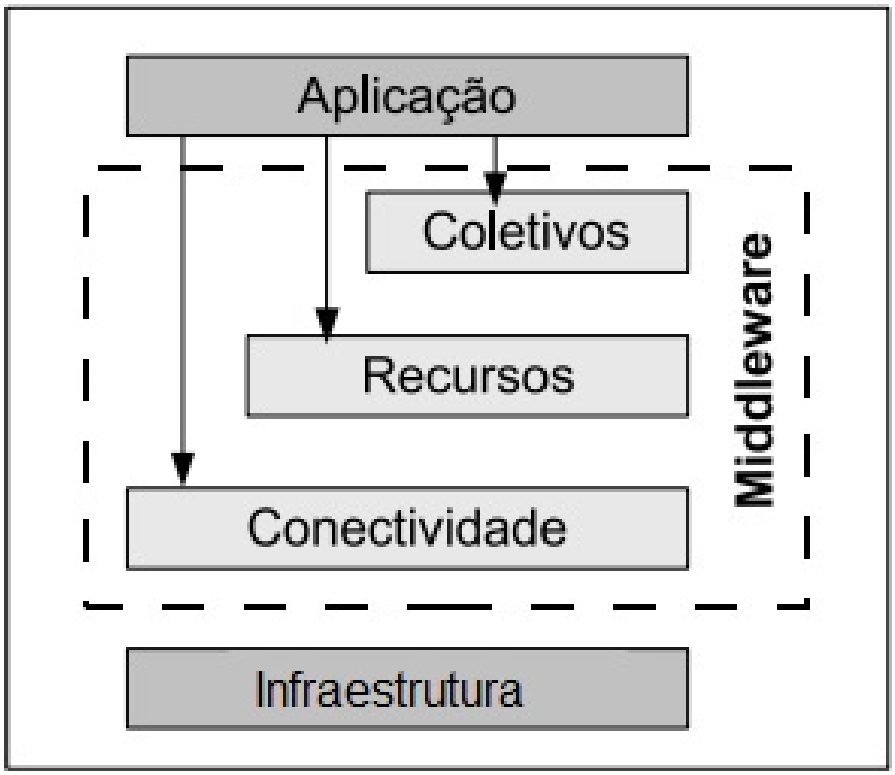

Figura 2.2: Camadas da Arquitetura Grid [21].

A camada mais em baixo é a de infraestrutura, nela se encontram os recursos computacionais propriamente ditos, que realizam o processamento e o armazenamento, entre outros. No meio existe o *midleware*, uma camada de software responsável por prover transparência ao usuário e facilitar a utilização deste ambiente. Nesta arquitetura, tem-se a divisão do *midleware* em três camadas distintas. Na camada de conectividade se encontram o núcleo de comunicação e os protocolos para transmissão em redes. É através dela que é possível se estabelecer uma comunicação entre os diversos recursos heterogêneos presentes no *grid*. A camada de recursos tem a responsabilidade de inicializar e controlar o compartilhamento de recursos individuais. A camada coletivos abrange aquilo que não pertence a algum serviço específico, como escalonamento, monitoramento e replicação de dados. E por fim, a camada de aplicação deve fornecer facilidades para que as aplicações possam ser executadas sem que seja necessário conhecer especificamente cada componente do *grid* [21].

Nesse cenário, é comum haver confusão nas definições das plataformas de *Grids* e de Nuvem, pois ambas possuem algumas características em comum, apesar de terem focos diferentes. A Subseção 2.2.3 explanará um pouco mais sobre estas diferenças, mas antes será apresentado o conceito de computação em nuvem.

2.2 Nuvem Computacional

A computação em nuvem é um paradigma de computação distribuída que surgiu como uma tendência para a provisão de recursos e serviços de forma rápida, barata, escalável e flexível [2]. Diversos tipos de recursos podem ser disponibilizados, tais como memória, capacidade de processamento, armazenamento entre outros. Todavia, a definição de computação em nuvem não é um consenso entre os pesquisadores da área, e algumas definições serão descritas a seguir:

- Armbrust et al. [3] definem como "a união de aplicações oferecidas como serviço pela Internet, com o hardware e o software localizados em *datacenters* de onde o serviço é provido";
- De acordo com Foster et al. [2] computação em nuvem é "um paradigma computacional altamente distribuído, direcionado por uma economia de escala, na qual poder computacional, armazenamento, serviços e plataformas abstratas, virtualizadas, gerenciadas e dinamicamente escaláveis são oferecidos sob demanda para usuários externos por meio da Internet";
- Buyya et al. [1] definem como "um tipo de sistema paralelo e distribuído, que consiste em uma coleção de computadores virtuais interconectados que são provisionados dinamicamentes e apresentados como um ou mais recursos computacionais unificados, baseados em acordos de nível de serviço estabelecidos entre provedor de recursos e o consumidor".

A primeira definição é bastante abrangente e deixa espaço para que haja confusão com outros paradigmas de computação distribuída. É possível, por exemplo, confundir com a definição de um *grid* computacional. A segunda já apresenta um elemento importante para a definição mais completa de nuvem, que é a frase "oferecido sob demanda". Isto significa que na computação em nuvem o usuário paga por aquilo que utiliza, diferentemente de como ocorre nos outros paradigmas de computação ditribuída, e por isso é uma diferença fundamental. A terceira definição fala sobre os acordos de nível de serviço (*Service Level Agreement* - SLA). Esse acordo é negociado entre o consumidor e o fornecedor, e é ele quem define quais indicadores irão medir a qualidade do serviço. Assim, a violação deste acordo pode levar ao pagamento de multas [1].

Desta forma, a computação em nuvem possui diversas características que a difere dos outros sistemas distribuídos existentes. As principais características são [22]:

- *Self-service*: na computação em nuvem, os serviços, tais como armazenamento e processamento, são contratados pelo usuário diretamente, sem a participação de algum administrador dos provedores de nuvem, e de forma que atenda às necessidades do usuário [23];
- Amplo Acesso: a ideia é que os recursos estejam disponíveis na Internet e possam ser acessados a partir de dispositivos heterogêneos, desde que estes possuam acesso à rede mundial de computadores;
- *Pooling* de Recursos: os recursos de um determinado provedor são organizados em um *pool* de recursos físicos e virtuais, de forma a facilitar o acesso de vários usuários e facilitar também os ajustes de demanda;

- Elasticidade: caso seja necessário aumentar a utilização de determinado recurso, seja ele qual for, este aumento deve ocorrer de forma fácil ou até mesmo de forma automática;
- Serviço Medido: a utilização dos recursos deve ser monitorada a todo momento, de forma a garantir que seja cumprido o contrato de qualidade de serviço realizado entre o provedor de serviço e o usuário.

2.2.1 Arquitetura de uma Nuvem

Na literatura, há diferentes propostas de arquitetura [24] [25], e a Figura 2.3 apresenta um destes modelos. Nela é possível identificar três atores, que são os Provedores de Serviço (*Service Providers*), os Provedores de Infraestrutura (*Infrastructure Providers*) e os Usuários de Serviço (*Service Users*).

Os provedores de infraestrutura disponiblizam recursos para que os provedores de serviços hospedem suas aplicações sem precisar se preocupar com as questões de estrutura física, a fim de ganhar escalabilidade e flexibilidade. Os usuários de serviço acessam as aplicações que foram colocadas a disposição através da Internet.Tanto usuários de serviço como provedores de serviço pagam apenas por aquilo que consumirem, o chamado *pay-per-use*.

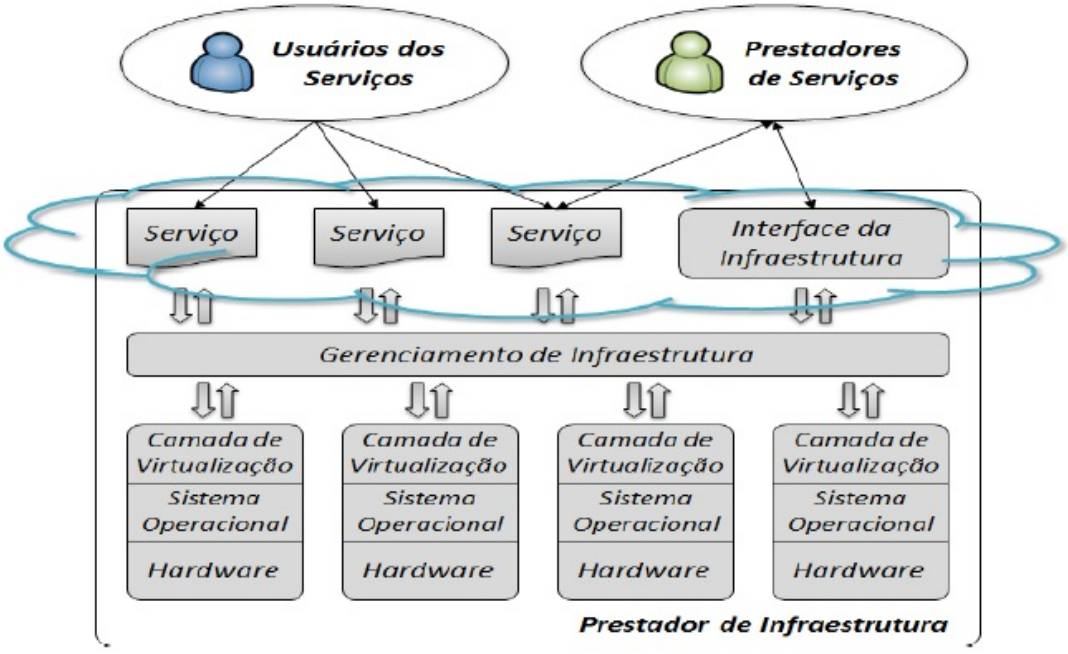

Figura 2.3: Arquitetura de Nuvem computacional [22].

Esta arquitetura de nuvem está dividida em três camadas distintas. A mais próxima do usuário é onde se encontram os serviços, ela possui uma interface para que os clientes tenham acesso às aplicações que foram disponibilizadas. A camada mais baixa é a infraestrutura de fato. Ali estão localizados os servidores, os *datacenters*, a rede e toda a parte física que compõe a nuvem. A camada do meio fornece facilidades para que o

provedor de serviços consiga disponibilizar suas aplicações sem ter que se preocupar com as individualidades do hardware.

Uma outra proposta de arquitetura foi feita por Foster et al. [2] e está representada na Figura 2.4.

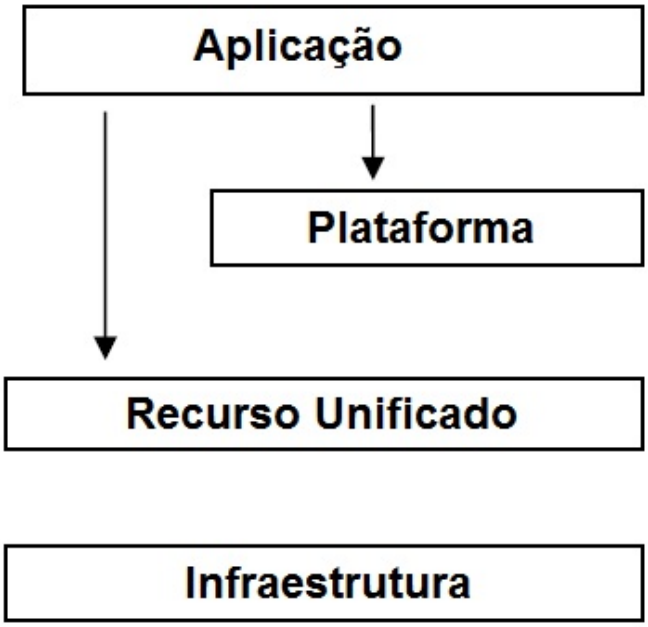

Figura 2.4: Arquitetura para a Plataforma de Nuvem por Foster et al [2].

A camada de infraestrutura contém os recursos computacionais, como os recursos de armazenamento e os de processamento, é o hardware propriamente dito. A camada de recursos unificados contém os recursos que foram encapsulados, geralmente por meio da virtualização, e estão disponíveis tanto para os usuários finais quanto para a camada superior. Assim, *clusters* e computadores virtuais são exemplos de recursos desta camada. A camada de plataforma consiste em um conjunto de ferramentas especializadas que estão executando em cima da camada de recursos unificados, tais como plataformas de desenvolvimento. E por fim, a camada de aplicação que contém as aplicações que executam na nuvem.

Os serviços que são oferecidos no paradigma de computação em nuvem podem ser divididos em categorias. Apesar de ser possível encontrar na literatura propostas com mais de três classes [26], o mais comum é dividir os serviços em *Infrastructure-as-a-Service*, *Platform-as-a-Service* e *Software-as-a-Service*, descritos a seguir:

- *Infrastructure-as-a-Service*: os serviços que são oferecidos se caracterizam por serem de infraesturura, podendendo ser desde capacidade de processamento, armazenamento ou até máquinas virtuais completas. Um provedor desta camada que se destaca é a Amazon, com o *Elastic Compute Cloud* (EC2) [27] e com o *Simple Storage Service* (S3) [28];
- *Platform-as-a-Service*: o que caracteriza este nível é a disponibilização de um ambiente no qual o usuário possa programar, testar e executar aplicações. O Google

App Engine [29] se encontra nesta categoria, pois é um ambiente no qual aplicações podem ser desenvolvidas sem ter nenhum tipo de programa instalado na sua máquina, tudo é feito através da Internet;

- *Software-as-a-Service*: são as aplicações disponibilizadas pelos provedores como serviços aos usuários comuns, de forma gratuita ou cobrando pela sua utilização. Um exemplo é o Google Docs [30], no qual são disponibilizados editores de textos, editores de planilhas e ferramentes para a criação de apresentações. Os usuários podem acessar estes serviços em qualquer lugar, sem a limitação de ter o software instalado na sua máquina.

2.2.2 Tipos de Nuvens

As nuvens podem ser divididas em quatro tipos diferentes de implantação, que são nuvens públicas, privadas, comunitárias e híbridas.

- Nuvens Públicas: esse tipo de nuvem é aquela mantida por um fornecedor, geralmente uma grande companhia, para um usuário comum ou uma empresa. É a nuvem no sentido tradicional do senso comum, na qual os recursos são provisionados dinamicamente e através da Internet, *web-services* e aplicações web são disponibilizadas [26];
- Nuvens Privadas: é a nuvem que está dentro de um contexto empresarial e não está acessível a todas as pessoas, sendo restrita apenas aos funcionários e aos parceiros da empresa. Pelo fato de não estar disponível na Internet, apresenta vantagens em termos de segurança e largura de banda da rede;
- Nuvens Comunitárias: infraestrutura que é compartilhada por organizações que mantém algum tipo de interesse em comum (jurisdição, segurança, economia), e pode ser administrada, gerenciada e operada por uma ou mais destas oganizações;
- Nuvens Híbridas: ambiente no qual ambos os tipos de nuvens anteriormente descritos podem ser utilizados em conjunto. É interessante para alguns modelos de negócios, pois arquivos sigilosos ou sistemas que manipulam dados sigilosos podem ser mantidos na nuvem privada, enquanto que os outros dados e sistemas podem ficar na nuvem pública, por exemplo.

2.2.3 Comparação entre Sistemas Distribuídos

Algumas características fundamentais para o entendimento das diferenças entre as plataformas *cluster* e *grid* são apresentadas na Tabela 2.1.

O primeiro item é a heterogeneidade, que nos *clusters* está pouco ou não está presente, pois eles são formados por componentes de hardware semelhantes e, geralmente, todos os nós executam o mesmo sistema operacional, enquanto que os *grids* são altamente heterogêneos, tanto no hardware como no software. Em relação à dedicação o *grid* não é dedicado, isto é, ele utiliza o poder de processamento dos computadores conectados quando estes não estão sendo utilizados pelos seus proprietários, enquanto que um *cluster* necessita de exclusividade para realizar o processamento. Os *clusters* utilizam compontentes que estão

Tabela 2.1: Características de *Clusters* e *Grids*.

Característica	**Cluster**	**Grid**
`Heterogêneo`	Não/Pouco	Sim
`Dedicado`	Sim	Não
`Proximidade dos Componentes`	Sim	Nao
`Componentes Commodity`	Sim	Sim

muito próximos um do outro, geralmente são idênticos, enquanto que os *grids* podem utilizar componentes bastante diferentes entre si. Ambos possuem componentes *comodity*, isto é, componentes que são facilmente adquiridos e a um custo baixo [12].

A Figura 2.5 mostra uma comparação entre alguns dos sistemas distribuídos quanto à escala, à orientação a serviços e à aplicações.

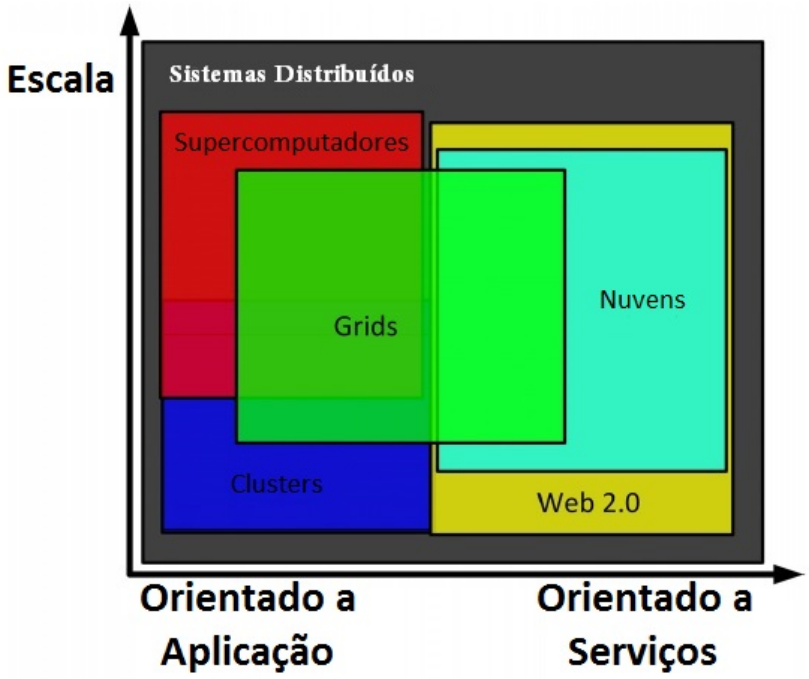

Figura 2.5: Comparação entre Sistemas Distribuídos quanto à Escala e a Orientação [2].

Na Figura 2.5 são mostrados os supercomputadores, um tipo de sistema distribuído que possui alta velocidade de processamento, que é construído com hardware específicamente desenvolvido para esta finalidade, tornando-o de alto custo. A computação em nuvem é a que apresenta o maior foco na orientação à serviços e é de grande escala, enquanto que supercomputadores e *clusters* apresentam um foco em aplicações tradicionais, não voltadas a serviços. O *grid* se sobrepõe a todas as áreas e é considerado de menor escala do que as nuvens e os supercomputadores.

Apesar de *grids* e nuvens serem sistemas distribuídos, e possuírem características em comum, tais como a ideia de redução de custos na computação, e aumento de flexibilidade,

eles não são a mesma coisa. Nesta subseção essas diferenças serão mostradas de forma que não haja mais dúvida e confusão envolvendo estes conceitos.

Ian Foster [2] dividiu as características em alguns pontos e mostrou as diferenças em cada um deles:

- Modelo de Negócio: tradicionalmente os usuários estão acostumados a pagar uma vez por uso ilimitado de um determinado software. Com a computação em nuvem isto não é mais verdade e é possível pagar apenas por aquilo que é consumido, assim como é feito com a água e a eletricidade. Porém, no caso da computação em nuvem o pagamento pode ser feito por horas de utilização ou por quantidade de espaço utilizado. Já os *grids*, que são utilizados para fins científicos, utiliza um modelo baseado em projetos, que tenta prever o quanto de recursos será utilizado em um determinado projeto;
- Modelo de Computação: a computação em nuvem permite que várias aplicações sejam executadas ao mesmo tempo graças a utilização da virtualização. No *grid* isto não é possível, apenas uma aplicação pode executar por vez;
- Virtualização: a virtualização é essencial para que exista a computação em nuvem, ela fornece a abstração e o encapsulamento necessários. Na nuvem é preciso que vários usuários utilizem os recursos sem perceber que existem outros usuários concorrendo, e cada um deles pode utilizar a quantidade de recursos que precisar. A virtualização fornece essa abstração sobre a infraestrutura. Já os *grids* não são dependentes de virtualização e não o utilizam na maioria das vezes, devido às políticas das organizações de ter controle total sobre os seus dados;
- Localização dos Dados: com o volume dos dados aumentando cada vez mais, deve-se considerar o custo de transmissão de um dado do local aonde está armazenado para o local no qual será processado. Existe uma grande diferença do custo de transmissão de um dado armazenado no próprio disco da máquina na qual será processado, do custo de transmissão pela rede. Quanto mais perto o servidor no qual o dado está armazenado se encontra da máquina para o qual será levado, menor o custo de transmissão e, portanto, melhor o desempenho. É importante então levar em conta a localização física dos servidores no momento do armazenamento de um dado, e isto é comum na computação em nuvem. No *grid*, geralmente, é utilizado um sistema de arquivos distribuídos, como por exemplo, o NFS (*Network File System*) [31], o que dificulta a aplicação de decisões de armazenamento que levam em consideração a localidade.

Buyya et al. [1] foram ainda mais longe e fizeram uma comparação entre *cluster*, *grid* e nuvem, que pode ser vista Tabela 2.2.

Apesar de a computação em nuvem ser mais escalável que outros sistemas distribuídos, como os *clusters* e *grids*, atualmente, ela já não é mais suficiente para suprir sozinha a demanda por recursos computacionais. A solução então foi integrar mais de uma nuvem computacional, o qual foi chamado de federação de nuvens, que será descrita em detalhes na próxima seção.

Tabela 2.2: Diferenças entre os Sistemas Distribuídos (adaptado de Buya et al. [1]).

Sistemas / Características	*Cluster*	*Grid*	Nuvem
Componentes	Computadores commodities	Computadores de alta tecnologia	Computadores commodities e de alta tecnologia
Sistema Operacional	Um SO padrão	Qualquer SO padrão	Máquinas virtuais com múltiplos SOs
Proprietário	Único	Múltiplo	Único
Rede de Conexão	Dedicada, largura de banda alta com baixa latência	Internet, alta latência e pouca largura de banda	Dedicada, largura de banda alta com baixa latência
Segurança	Tradicional, baseada em login e senha	Par de chave pública e privada com autenticação	Cada usuário utiliza uma máquina virtual, com suporte para controle de acesso.
Preço dos Serviços	Limitado, não aberto ao mercado	Dominado pelo interesse público ou privado	Preços utilitários com descontos para grandes clientes
Negociação de Serviço	Limitada	Sim, baseada em acordos de níveis de serviço	Sim, baseada em acordos de níveis de serviço
Gerenciamento de Usuário	Centralizado	Descentralizado	Centralizado ou pode ser delegado a um terceiro
Gerenciamento de Recursos	Centralizado	Distribuído	Centralizado/ Distribuído
Alocação/ Escalonamento	Centralizado	Descentralizado	Centralizado/ Descentralizado

2.3 Federação de Nuvens

Com o passar dos anos novas necessidades foram surgindo e a utilização de nuvens de forma isolada passou a não ser mais suficiente para algumas aplicações. Além disso, empresas começaram a criar *datacenters* ao redor do mundo para a aumentar a segurança em caso de queda em algum dos servidores, e também como uma forma de atender às solicitações em menor tempo, diminuindo a distância entre o usuário e os servidores. Dessa forma, integrar nuvens passou a ser algo necessário para continuar fornecendo serviços de forma rápida, eficiente e escalável.

Bittman [32] dividiu a evolução do paradigma de computação em nuvem em três fases. A primeira fase é chamada de monolítica e se caracteriza pelas ilhas proprietárias, com serviços fornecidos por empresas de grande porte, como Google, Amazon e Microsoft. Na segunda fase, chamada de cadeia vertical de fornecimento, ainda se tem o foco nos ambi-

entes proprietários, mas as empresas começam a utilizar alguns serviços de outras nuvens, sendo vista como o começo da integração. E por último, tem-se a federação horizontal, na qual pequenos provedores se juntam para aumentar sua escalabilidade e eficiência na utilização dos recursos, e começam a ser discutidos padrões de interoperabilidade.

Atualmente, tem-se vivido a terceira fase, isto é, a etapa em que os provedores de nuvem trabalham exclusivamente sozinhos ja passou, e tem-se alcançado a etapa em que os pequenos provedores se aliam horizontalmente.

Assim sendo, a federação de nuvens computacionais pode ser definida como um conjunto de provedores de nuvens públicos e privados, conectados através da Internet [6]. O BioNimbuZ [6] é uma proposta de arquitetura para a realização de uma federação horizontal, que visa facilitar a integração entre diversos provedores de nuvem. O intuito é garatir a máxima eficiência na utilização dos recursos, e aumentar a escalabilidade dos provedores.

Realizar uma federação, no entanto, não é algo simples devido as diferenças entre as nuvens, cada uma com suas particularidades, tanto no hardware (arquitetura do processador por exemplo) como no software (sistema operacional ou outros softwares utilizados). A próxima seção apresenta os principais desafios.

2.3.1 Desafios de Projeto

Para que haja a federação é necessário que os seguintes requisitos sejam atendidos [4]:

- Automatismo e Escalabilidade: uma nuvem, utilizando mecanismos de descoberta, deve ser capaz de escolher, dentre as nuvens existentes, qual que atende às suas necessidades e deve ser capaz de reagir a mudanças;
- Segurança Interoperável: é necessário a integração entre diversas tecnologias de segurança, de forma que uma nuvem não precise alterar suas políticas de segurança para se juntar à federação.

Outros desafios para a criação de nuvens federadas foram identificados e são descritos a seguir [5]:

- Previsão de Comportamento da Aplicação: é importante que o sistema seja capaz de prever o comportamento das aplicações, para que ele possa tomar decisões inteligentes quanto à alocação de recursos, dimensionamento dinâmico, armazenamento ou largura de banda. Um modelo deve ser construindo para tentar realizar esta previsão. Este modelo deve levar em consideração estatísticas de padrões de utilização dos serviços e ajustar as variáveis sempre que for necessário, para que o modelo seja o mais próximo possível da realidade;
- Mapeamento Flexível de Recursos e Serviços: algo que sempre é levado em consideração em qualquer projeto é o custo. É importante que o sistema seja o mais eficiente possível, sempre tentando encontrar a melhor configuração de hardware e software que atenda às demandas de qualidade de serviço que foram estabelecidas entre o usuário e o provedor. Esta é uma tarefa bastante complicada devido ao comportamento não previsível das aplicações e serviços que são utilizados na nuvem;

- Modelo Econômico Impulsionado por Técnicas de Otimização: o problema da tomada de decisões orientadas ao mercado é um problema de otimização combinatória, que visa encontrar a melhor combinação entre serviços e planos. Os modelos de otimização visam, otimizar tanto os recursos centralizados (utilização, disponibilidade e incentivo) quanto os centrados nos usuários (tempo de resposta, o orçamento gasto e a justiça);
- Integração e Interoperabilidade: muitas empresas possuem dados sigilosos e não se sentirão confortáveis de colocá-los na nuvem. Esse medo é tanto pela questão da segurança, ou seja, medo de que alguma pessoa não autorizada tenha acesso a dados confidenciais, como também é pela forma como as aplicações que já existem na empresa irão interagir com os dados e as aplicações que estão na nuvem;
- Monitoramento Escalável dos Componentes do Sistema: atualmente as técnicas que são utilizadas para o monitoramento e o gerenciamento dos componentes da nuvem utilizam uma abordagem centralizada. Em sistemas distribuídos manter uma singularidade sempre é um problema, principalmente, por este poder se tornar um gargalo para o sistema, caso o volume de requisições seja muito alto, como também por questões de segurança, visto que algum problema pode prejudicar toda a federação de nuvens. É importante que este monitoramento seja feito de forma distribuída, para que não haja problema de escalabilidade, de desempenho e de confiabilidade.

2.3.2 Arquitetura

Para que os requisitos anteriormente citados sejam atendidos e os desafios superados, arquiteturas para a federação de nuvens foram propostas [5] [4]. Celesti et al. [4] propuseram uma arquitetura que eles chamaram de três fases, veja a Figura 2.6. Neste modelo eles dividiram as nuvens em dois grupos, local e estrangeiro. A nuvem local consiste naquele provedor que já atingiu a saturação, não consegue mais atender às demandas e, portanto, deve repassar requisições para o restante da federação. Os provedores que vão receber essas requisições são chamados de estrangeiros, e podem ou não cobrar por ceder seus recursos ociosos para atender às demandas que foram repassadas a ele. Vale ressaltar que uma nuvem pode ser local e estrangeira ao mesmo tempo, basta que ela esteja saturada em um determinado recurso, e por isso precise da ajuda de outros provedores para suprir essa demanda, e ao mesmo tempo está com outro recurso ocioso e o disponibiliza para outro provedor. Este repasse de requisições de uma nuvem para outra deve ser feito de forma transparente ao usuário e de forma a atender os contratos de serviço que foram efetuados previamente.

Na arquitetura proposta por Celesti et al. [4], para cada provedor existente na federação existe um gerenciador chamado de *Cross-Cloud Federation Manager* (CCFM). O CCFM é o responsável por realizar a gerência das nuvens, verificando quando uma nuvem já está saturada e tentando encontrar nuvens que estejam dispostas a contribuir com seus recursos ociosos. Ele está divido em três subcomponentes, cada um responsável por realizar uma fase dentre as três que dão nome a arquitetura, os quais são:

- *The Discovery Agent*: é o agente responsável por realizar o processo de descoberta na federação. Para que este processo seja feito de forma dinâmica e distribuída é

necessário que cada provedor de nuvem disponibilize suas informações para um local centralizado (apesar de ser logicamente centralizado é implementado de forma distribuída), e sempre que uma nuvem precisar de informações sobre os outros provedores basta consultar este local;

- *The Match-Making Agent*: este agente é o responsável por escolher dentre todas as nuvens estrangeiras, qual é a que melhor se encaixa nos requisitos pretendidos. Depois de feita a escolha, ele também é responsável por garantir que os acordos de níveis de serviço sejam cumpridos e a qualidade de serviço seja mantida;
- *The Authentication Agent*: agente que tem como função realizar a autenticação dos usuários dentre as diversas nuvens da federação. É uma tarefa complicada, pois cada nuvem pode ter vários usuários autenticados, e essas autenticações podem variar a todo instante. Outro problema é que cada nuvem pode utilizar uma tecnologia de autenticação diferente da outra, e este agente é responsável por realizar a interoperabilidade entre essas tecnologias.

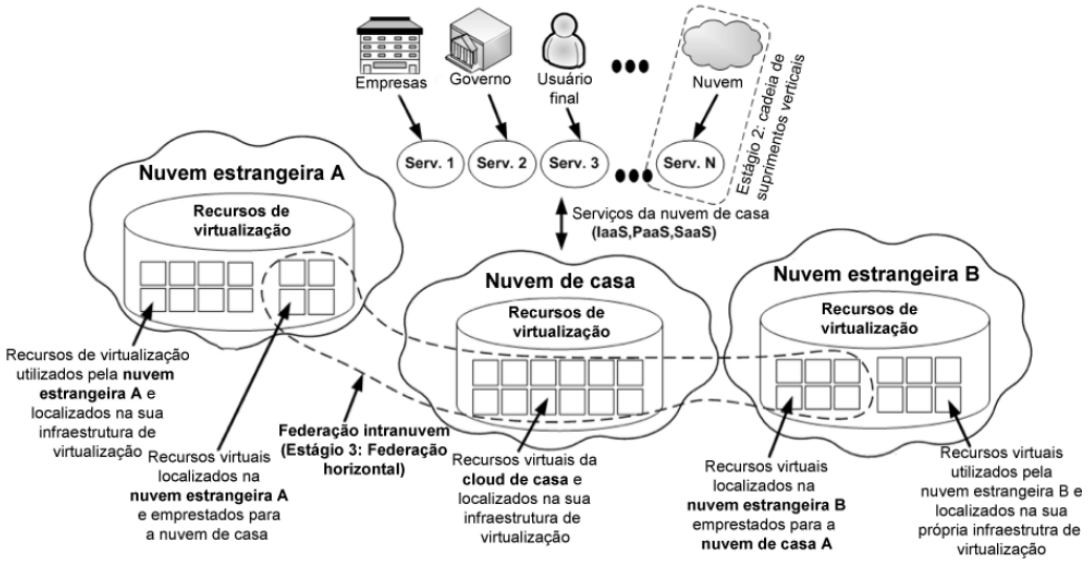

Figura 2.6: Arquitetura para Federação de Nuvens, proposta por Celesti et al. [4].

Uma proposta alternativa de arquitetura foi apresentada por Buyya et al. [5]. Nesta proposta, o usuário utiliza um componente externo aos provedores para realizar qualquer tipo de interação com a federação. Este componentente é chamado de *Cloud Broker* (CB). Ele é o responsável por intermediar a comunicação entre o usuário e os provedores, e também por identificar os recursos que estão disponíveis em cada provedor, de forma a atender os requisitos de qualidade de serviço (QoS) que foram definidos através de um contrato de SLA.

Para conseguir estes dados sobre os provedores de nuvem, o CB consulta o *Cloud Exchange* (CEx), um outro componente da arquitetura. A principal função do CEx é servir como um registro, que é consultados pelo CB a fim de obter informações sobre a infraestrutura, o custo de utilização, os padrões de execução e os recursos disponíveis, além de mapear as requisições dos usuários aos provedores.

Em cada provedor presente na federação existe um outro componente, chamado *Cloud Coordinator* (CC), responsável por incluir a infraestrutura disponível na federação e expor estas informações aos interessados. Para atender às requisições dos usuários, o CC realiza uma autenticação e estabelece um acordo de qualidade de serviço com cada CB, que também é responsável por encaminhar as tarefas para a execução. Um exemplo desta arquitetura é apresentado na Figura 2.7.

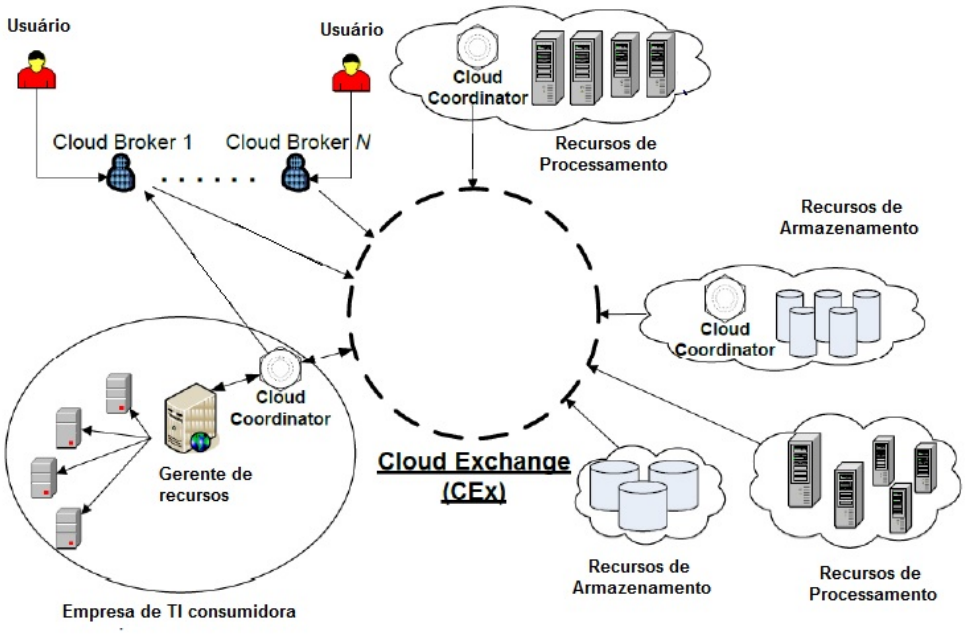

Figura 2.7: Arquitetura para Federação de Nuvens, proposta por Buyya et al. [5].

Como não existe um padrão para a federação de nuvens computacionais, uma nova arquitetura foi proposta, chamada de BioNimbuZ, que tem como objetivo garantir de forma dinâmica, transparente e escalável a execução de aplicações no nível de *Software-as-a-Service*. Esta proposta é detalhada no próximo capítulo.

Capítulo 3

BioNimbuZ

Este capítulo tem como objetivo apresentar a plataforma para federação de nuvens computacionais BioNimbuZ [6]. Serão detalhados seu funcionamento, sua arquitetura e os seus serviços e controladores. Além disso, serão apresentadas as modificações que ocorreram na proposta original e quais tecnologias foram incorporadas com estas alterações.

3.1 Visão Geral

O BioNimbuZ é uma arquitetura para federação de nuvens híbridas, que foi proposta originalmente por Saldanha [6], e que tem sido aprimorada constantemente em outros trabalhos [33] [34]. Ele foi desenvolvido para suprir a demanda de plataformas de nuvens federadas, visto que a utilização de nuvens de forma isolada já não atende, em muito casos, às necessidades de processamento, de armazenamento, entre outros, na execução das aplicações de bioinformática.

O BioNimbuZ permite a integração entre nuvens de diversos tipos, tanto privadas quanto públicas, deixando com que cada provedor mantenha suas características e políticas internas, e oferece ao usuário transparência e ilusão de infinidade de recursos. Desta forma, o usuário pode usufruir de diversos serviços sem se preocupar com qual provedor está sendo de fato utilizado.

Outra característica desta arquitetura é a flexibilidade na inclusão de novos provedores, pois são utilizados *plugins* de integração que se encarregam de mapear as requisições vindas da arquitetura para as requisições de cada provedor especificamente. Isso é fundamental para que alguns objetivos possam ser alcançados, principalmente, na questão da escalabilidade e da flexibilidade.

Originalmente, toda a comunicação existente no BioNimbuZ era realizada por meio de uma rede *Peer-to-Peer* (P2P) [35]. Porém, para alcançar os objetivos desejados de escalabilidade e de flexibilidade, percebeu-se a necessidade de alterar a forma de comunicação entre os componentes da arquitetura do BioNimbuZ, pois a utilização de uma rede de comunicação *Peer-to-Peer* (P2P) não estava mais suprindo as necessidades nestes dois quesitos. É importante ressaltar que os outros objetivos propostos por Saldanha [6], tais como obter uma arquitetura tolerante a falhas, com grande poder de processamento e armazenamento, e que suportasse diversos provedores de infraestrutura, foram mantidos e melhorados.

Assim sendo, nos trabalhos [33] [34] foi proposta a utilização de Chamada de Procedimento Remoto (RPC) [36] para realizar a comunicação de forma transparente, pois ela permite a chamada de procedimentos que estão localizados em outras máquinas, sem que o usuário perceba [15]. E para auxiliar na organização e na coordenação do BioNimbuZ, foi utilizado um serviço voltado à sistemas distribuídos chamado ZooKeeper [37], da Fundação Apache [38]. A seguir, será explicado de forma detalhada o funcionamento do Zookeeper e do Avro, o sistema de RPC também da Fundação Apache.

3.1.1 Apache Avro

O Apache Avro [39] é um sistema de RPC e de serialização de dados desenvolvido pela Fundação Apache [38]. Algumas vantagens deste sistema são o fato de ser livre, a utilização de mais de um protocolo de transporte de dados em rede, e o suporte a mais de um formato de serialização de dados. Quanto ao formato dos dados, existe o suporte aos dados binários e aos dados no formato JSON [40]. Em relação aos protocolos pode-se utilizar tanto o protocolo HTTP [41] como um protocolo próprio do Avro [42].

O Avro foi criado para ser utilizado com um grande volume de dados, e possui algumas características [42] definidas pela própria Fundação Apache, como uma rica estrutura de dados com tipos primitivos, um formato de dados compacto, rápido e binário e a integração de forma simples com diversas linguagens de programação.

O Avro foi escolhido como *middleware* de Chamada Remota de Procedimento para o BioNimbuZ, por ser livre, flexível e possuir integração com várias linguagens.

3.1.2 Zookeeper Apache

O Zookeeper [37] é um serviço de coordenação de sistemas distribuídos, criado pela Fundação Apache, para ser de fácil manuseio. Ele utiliza um modelo de dados que simula uma estrutura de diretórios, e tem como finalidade facilitar a criação e a gestão de sistemas distribuídos, que podem ser de alta complexidade e de difícil coordenação e manutenção.

No Zookeeper são utilizados espaços de nomes chamados de *znodes*, que são organizados de forma hierárquica, assim como ocorre nos sistemas de arquivos. Cada *znode* tem a capacidade de armazenar no máximo 1 Megabyte (MB) de informação, e são identificados pelo seu caminho na estrutura. Neles podem ser armazenadas informações que facilitem o controle do sistema distribuído, tais como metadados, caminhos, dados de configuração e endereços [34].

Existem dois tipos de *znodes*, os persistentes e os efêmeros. O primeiro é aquele que continua a existir mesmo depois da queda de um provedor, sendo útil para armazenar informações a respeito dos *jobs* que foram executados e outros tipos de dados que não se deseja perder. Os efêmeros, por outro lado, podem ser criados para cada novo partipante da federação, pois assim que este novo participante ficar indisponível, o *znode* efêmero referente a ele será eliminado e todos os outros componentes do sistema distribuído saberão que ele não se encontra disponível. Na Figura 3.1 pode ser visto um exemplo da estrutura hierárquica dos *znodes*.

O Zookeeper também suporta o conceito de *watchers*, que funcionam como observadores de mudanças, e enviam alertas sobre as alterações ocorridas em algum dos *znodes*. Este conceito é útil para sistemas distribuídos, pois permitem o monitoramento constante

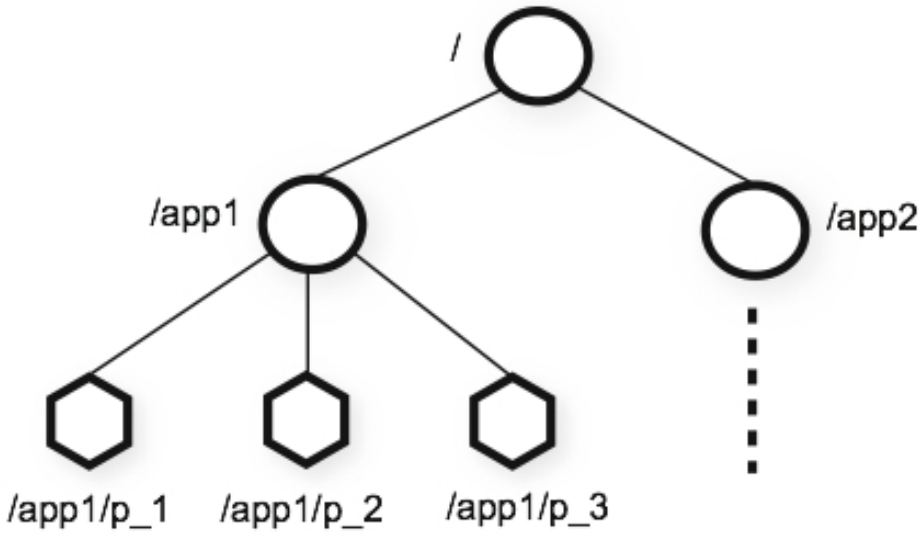

Figura 3.1: Estrutura Hierárquica dos *Znodes* [37].

do sistema, deixando para que *watchers* avisem quando um provedor estiver fora do ar, por exemplo, ou então quando um novo recurso estiver disponível.

3.2 Arquitetura do BioNimbuZ

O BioNimbuZ faz uso de uma arquitetura hierárquica distribuída, conforme apresentada na Figura 3.2. Ela representa a interação entre as três camadas principais: aplicação, núcleo e infraestrutura. A primeira camada - aplicação, permite a integração entre a aplicação do usuário e os serviços do núcleo da plataforma. A interface com o usuário pode ocorrer tanto através de linha de comando ou por uma interface gráfica, e é nela que devem ser inseridos os *workflows* que serão executados nas diversas nuvens. A aplicação então se comunica com o núcleo que é o responsável por realizar toda a gerência, e por se comunicar com os *plugins* de cada provedor. Os *plugins* mapeiam as requisições e as enviam para cada provedor. A seguir serão descritos o funcionamento e as características de cada uma destas camadas.

3.2.1 Camada de Aplicação

A camada de aplicação é a responsável por fazer toda a comunicação com o usuário e é nela que os usuários devem inserir as ações que desejam executar na federação. A interação com o usuário pode ser realizada através de páginas web, linhas de comando ou interface gráfica (GUI).

Assim que a tarefa for concluída, é função desta camada apresentar o *feedback* da execução, mostrando para o usuário se a tarefa foi completada com sucesso ou não. As principais ações que um usuário pode querer executar na federação são: *upload* de arquivos, listagem de arquivos, *download* de arquivos, submissão de *jobs*, consulta de *jobs*, cancelamento de *jobs*, verificação do extrato da fatura, pagamento de fatura e definição de SLA.

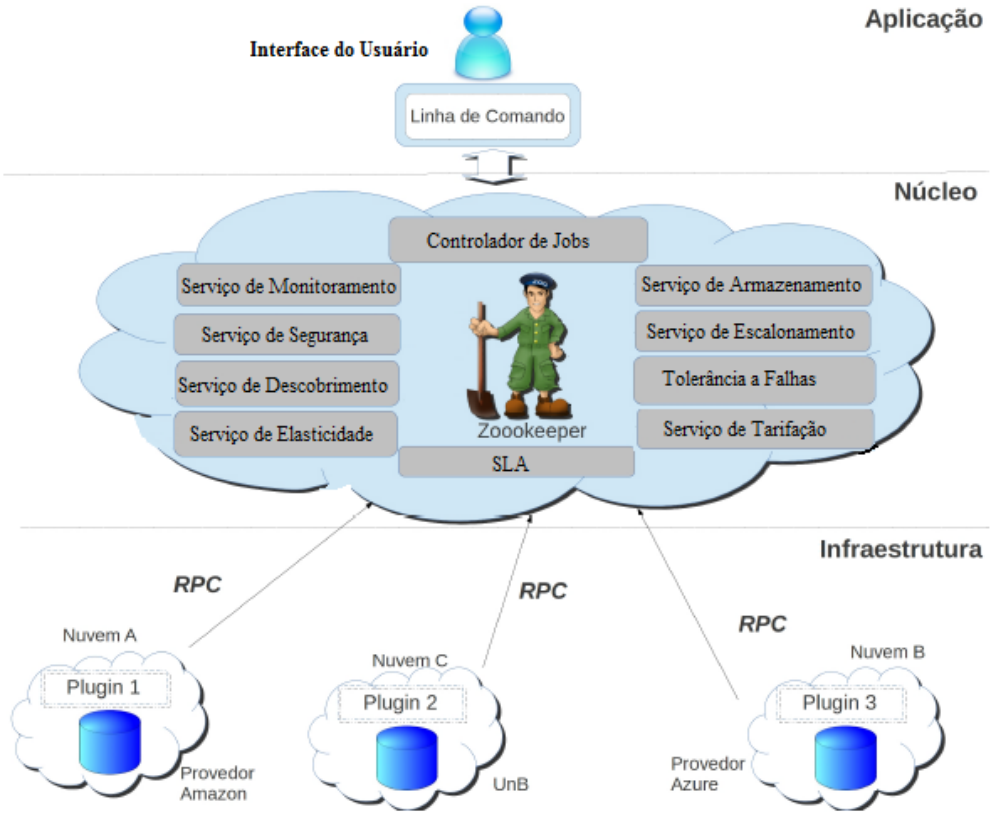

Figura 3.2: Arquitetura do BioNimbuZ.

3.2.2 Camada de Núcleo

O núcleo é o responsável por toda a gerência da federação, tendo como atividades o descobrimento de provedores e recursos, o escalonamento de tarefas, o gerenciamento de tarefas, o armazenamento de arquivos e o controle de acesso dos usuários, entre outras. Para cada uma destas atividades existe um serviço responsável, e novos serviços podem ser incorporados à medida em que forem demandados.

A seguir serão descritos os serviços de monitoramento, descobrimento, escalonamento, armazenamento, tolerância a falhas e segurança, e mais dois controladores: o controlador de SLA (*Service Level Agreement*) e o controlador de *jobs*.

- Controlador de *Jobs*: é quem recebe a requisição que vem da aplicação e manda para a camada de núcleo, verificando as credenciais dos usuários com o serviço de segurança, para permitir ou não a execução das tarefas. É responsável por gerenciar os vários pedidos e garantir que os resultados sejam entregues de forma correta para cada uma das requisições, e mantê-los para que possam ser consultados posteriormente;
- Controlador de SLA: quando o usuário submete uma tarefa para a federação por meio da camada de aplicação, ele deve preencher um *template* de SLA, que representa, entre outras coisas, os parâmetros de QoS (*Quality-of-Service*) que ele deseja ter disponibilizado pela plataforma de nuvem. Estes parâmetros podem descrever

desde requisitos funcionais, como número de núcleos de CPU, tamanho de memória, tamanho de armazenamento, até requisitos não funcionais, como custo a pagar e taxa de transferência. O controlador de SLA é responsável por implementar o ciclo de vida de SLA, o qual compreende seis atividades: descoberta de provedores de serviço, definição de SLA, estabelecimento do acordo de serviço, monitoramento de violação do acordo, término de acordo e aplicação de penalidades por violação.

Para cada pedido de execução feito por um usuário, este controlador deve verificar junto à federação se ela pode suportar os parâmetros naquele momento, e isto é feito através dos *plugins* de cada provedor. O pedido de execução junto com o *template* de SLA é repassado ao serviço de monitoramento que se encarrega de enviar ao serviço de escalonamento, e verificar se a tarefa foi executada com sucesso ou não;

- Serviço de Monitoramento: é o serviço responsável por realizar um acompanhamento dos recursos da federação, junto ao Zookeeper, com a utilização de *watchers* e tratando os alertas enviados pelos mesmos. Uma outra responsabilidade deste serviço é permitir a recuperação dos dados principais utilizados pelos módulos de cada servidor BioNimbuZ, armazenados na estrutura do Zookeeper, para possíveis rescontruções ou garantias de execuções de serviços solicitados;

- Serviço de Descobrimento: é o serviço que identifica e mantém informações a respeito dos provedores de nuvem que estão na federação, como capacidade de armazenamento, processamento e latência de rede, e também mantém detalhes sobre parâmetros de execução e arquivos de entrada e de saída. Para obter estas informações a respeito dos provedores, o Zookeeper é consultado, pois todos os provedores presentes na federação possuem *znodes* que armazenam seus dados. Sempre que uma modificação é realizada, como a saída ou entrada de algum provedor, os *watchers* alertam os outros serviços, mantendo assim todos os participantes da federação atualizados;

- Serviço de Escalonamento: é o serviço que recebe o pedido para a execução de alguma tarefa - aqui chamado de *job* - e as distribui dinamicamente entre os provedores disponíveis, divididas em instâncias menores - *tasks*. O serviço de escalonamento também é responsável por acompanhar toda a execução do *job*, e manter um registro das execuções já escalonadas.

  Para realizar a distribuição de tarefas na federação, algumas métricas são levadas em consideração, como latência, balanceamento de carga, tempo de espera e capacidade de processamento, entre outras, visando atender o que foi determinado no acordo de SLA. O serviço de escalonamento que está implementado atualmente no BioNimbuZ pode ser visto em mais detalhes no trabalho de Oliveira [34];

- Serviço de Armazenamento: responsável pela estratégia de armazenamento dos arquivos que são utilizados ou mantidos pelas aplicações. O armazenamento deve ocorrer de forma eficiente, para que as aplicações possam utilizar os arquivos com o menor custo possível. Este custo é calculado utilizando-se algumas métricas, tais como, latência de rede, distância entre os provedores e capacidade de armazenamento.

Outra estratégia adotada no armazenamento dos arquivos é a replicação em mais de um provedor, para tentar garantir que os arquivos estejam disponíveis quando as aplicações forem utilizá-los. A política de armazenamento que está implementada atualmente no BioNimbuZ foi introduzida no trabalho do Bacelar e Moura [33];

- Serviço de Tolerância a Falhas: tem como objetivo garantir que todos os principais serviços estejam sempre disponíveis, e em caso de falhas, possa ser feito uma recuperação. Essa recuperação exige que todos os objetos que forem afetados pela falha voltem ao estado anterior. O serviço de tolerância a falhas possui atuação de forma distribuída na plataforma BioNimbuZ, e está presente em outros serviços.

  No Serviço de Armazenamento, quando um alerta de indisponibilidade de um recurso é lançado por um *watcher*, é iniciado uma rotina de recuperação para os arquivos que este recurso continha. Como os arquivos são armazenados de forma duplicada na federação, a recuperação ocorre de forma a identificar quais arquivos foram perdidos e realizar uma nova duplicação em outro servidor do BioNimbuZ.

  No Serviço de Escalonamento, a recuperação está presente quando é recebido um alerta de indisponiblidade de um determinado recurso, e todas as tarefas que foram escalonadas e ainda não haviam sido executadas, ou estavam em execução, são novamente escalonadas, agora para uma outra máquina na federação.

  Essas recuperações são possíveis devido aos *watchers*, que disparam alertas aos responsáveis, avisando sobre os problemas em algum recurso da federação;

- Serviço de Elasticidade: é o serviço responsável por dinamicamente aumentar ou diminuir o número de instâncias de máquinas virtuais, ou então reconfigurar os parâmetros de utilização de CPU, de memória, de largura de banda, entre outros recursos que são disponibilizadas pelos provedores de nuvem, a fim de obter uma melhor utilização da infraestrutura disponível. A elasticidade pode ser vertical, quando há um redimensionamento dos atributos de CPU, de armazenamento, de rede ou de memória, como também pode ser horizontal, quando o número de instâncias de máquinas virtuais é modificado para mais ou para menos;

- Serviço de Tarifação: é o serviço responsável por calcular o quanto os usuários devem pagar pela utilização dos serviços oferecidos na plataforma BioNimbuZ. Para que isto seja possível, este serviço se mantém em constante contato com o serviço de monitoramento, para obter informações, tais como tempo de execução e quantidade de máquinas virtuais alocadas das tarefas que foram executadas;

- Serviço de Segurança: segurança em nuvens federadas é uma área de estudo em constante evolução, e o serviço de segurança deve trabalhar em diversos pontos para fornecer um serviço efetivamente seguro. O primeiro passo é a autenticação de usuários, ou seja, é preciso saber se o usuário que está tentando acessar algum recurso na federação é quem ele realmente diz ser. Depois da autenticação, vem a autorização, que consiste em verificar se o usuário pode realizar as ações que deseja. Muitos outros aspectos podem ser abordados por este serviço, como a criptografia de mensagens, para garantir a confidencialidade na troca de informações entre provedores, e também a verificação de integridade de arquivos, de modo que seja possível garantir que um arquivo não seja alterado por fatores externos à federação.

O foco deste trabalho é implementar um modelo de controle de acesso, para que somente usuários autorizados possam acessar os recursos na federação. Este é o primeiro trabalho relacionado à implementação de segurança na plataforma BioNimbuZ. Assim, todos os detalhes deste serviço serão mostrados no Capítulo 5 deste trabalho.

3.2.3 Camada de Infraestrutura

A camada de infraestrutura consiste em todos os recursos que os provedores de nuvens colocam a disposição da federação somados aos seus respectivos *plugins* de integração. O principal objetivo desta camada é prover uma interface de comunicação entre o BioNimbuZ e os provedores de nuvens, utilizando para tal os *plugins* que mapeiam as requisições provenientes da arquitetura, para comandos específicos para cada provedor, individualmente.

Assim sendo, faz-se necessário desenvolver um *plugin* para cada plataforma de nuvem, tais como o *Elastic Compute Cloud* (EC2) [27], e o *Simple Storage Service* (S3) [28], ambos da Amazon.

3.3 Organização Lógica

Para garantir a eficiência e a disponibilidade na gerência dos recursos entre os diferentes provedores de nuvens, a plataforma BioNimbuZ foi implementada de maneira hierárquica distribuída. Para isso foram definidos três níveis, o *master* global, o *master* local e o *slave*.

- *Master* Global: é o responsável por realizar todas as decisões de escalonamento e comunicar a decisão para os outros membros da federação. Ele ordena a execução de tarefas em outros provedores através dos *masters* locais, e recebe deles os resultados das execuções. Existe apenas um *master* global ativo na federação;
- *Master* Local: é o responsável por receber o pedido de execução das tarefas a partir do *master* global, e repassar aos *slaves* para que sejam executadas. Ele também tem a função de realizar o escalonamento local e informar ao *master* global os resultados da execução. Em cada provedor de nuvem existe um *master* local;
- *Slave*: é o responsável por executar as tarefas e repassar os resultados ao *master* local. Cada instância de máquina virtual executa um *slave*.

Essas divisões porém, não são perceptíveis ao usuário, pois ele mantém contato apenas com a interface do usuário, que entra em contato com o *master* global. A Figura 3.3 representa essa forma de organização do BioNimbuZ.

A primeira etapa do funcionamento do BioNimbuZ é o contato entre o serviço de interação e o *master* global, mais precisamente com o controlador de *jobs* (passo 1 na Figura 3.3). Este controlador entrará em contato com o serviço de segurança para que seja realizada a autenticação, que será mostrada em detalhes no Capítulo 5. Uma vez autenticado, o usuário pode solicitar a utilização de algum serviço na federação, como a execução de alguma tarefa, cabendo ao serviço de escalonamento na camada de núcleo definir em qual provedor deverá ser executada a tarefa submetida pelo usuário. O *master*

global pode enviar a tarefa tanto para o próprio *slave* (localizado na mesma nuvem) quanto para o *master local* de outro provedor (passos 2a, 2b e 2c da Figura 3.3), que se encarregará de enviar ao respectivo *slave* (passos 3b e 3c da Figura 3.3). Por fim, o *master* global espera a execução das tarefas, e aguarda o resultado que pode vir do próprio *slave* ou dos *masters* locais.

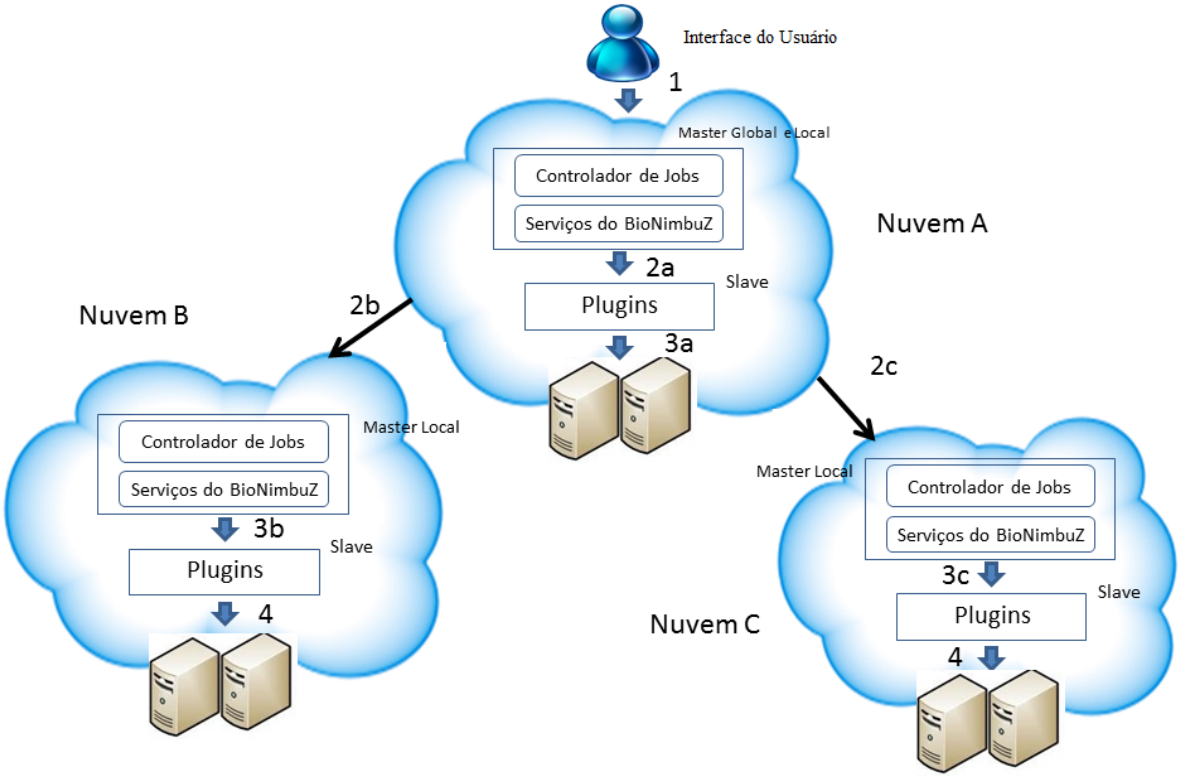

Figura 3.3: Organização Lógica do BioNimbuZ.

Dessa forma, o BioNimbuZ executa as tarefas em diferentes nuvens de maneira transparente ao usuário, pois é dada a ilusão de que todas as tarefas executaram em uma única plataforma de nuvem.

Contudo, não adiante um sistema ser escalável, flexível e tolerante a falhas, se ele não garante a segurança das informações dos usuários. Um usuário não pode ter acesso a arquivos que não lhe pertencem e nem deve poder realizar ações que prejudiquem os dados de outros usuários da federação. Nesse contexto, o próximo capítulo abordará o tema principal deste trabalho que é a segurança em nuvens federadas, e apresentará um modelo de controle de acesso para o BioNimbuZ, proposto neste trabalho.

Capítulo 4

Segurança em Nuvem Federada

O objetivo deste capítulo é apresentar os principais problemas envolvendo segurança em nuvem federada, os quais estão relacionados à infraestrutura, à virtualização e também à aplicação. Assim, serão discutidos pontos que englobam as três principais propriedades de segurança computacional: disponibilidade, integridade e confidencialidade, esta última sendo abordada detalhadamente, pois nela se encontram a autenticação e autorização, foco deste trabalho. Além disso, alguns dos principais modelos de controle de acesso serão apresentados, bem como suas características e o motivo deles serem apropriados ou não para a utilização em nuvem federada.

Para finalizar, também será apresentado o modelo de controle de acesso baseado em atributos (*Attribute Based Access Control* — ABAC), que foi implementado na solução proposta de segurança para o BioNimbuZ.

4.1 Principais Conceitos

Garantir que um sistema é seguro não é uma tarefa fácil, ainda mais se tratando de um sistema distribuído, pois ele possui mais fatores a se considerar do que um sistema que trabalha de forma isolada. É preciso levar em consideração os diversos provedores de nuvens que estão participando da federação, suas características individuais e tecnologias que estão utilizando, de forma a oferecer uma solução que seja de fato segura.

O*National Institute of Standards and Technology* (NIST) define a segurança em computação como a proteção dos recursos de um sistema computacional, ou seja, proteger a integridade, a disponibilidade e a confidencialidade dos recursos (*hardware*, *software*, *firmware*, dados e informação) [43].

Para entender esta definição é preciso entender a que se refere estas três propriedades, integridade, disponibilidade e confidencialidade, as quais serão descritas nas próximas seções.

4.1.1 Integridade

A integridade é uma propriedade que se refere a confiabilidade dos recursos (*hardware*, *software* e dados), e pode ser entendida como a prevenção contra ações indevidas ou não autorizadas que podem afetar estes recursos [44]. Essa propriedade pode ser classificada em [44]:

- Integridade de dados: está ligado à proteção contra modificações, ou exclusão de informações, e se refere diretamente ao conteúdo dos dados. Um usuário que armazenou seus dados em uma federação de nuvens espera que eles não sejam modificados por ninguém que não tenha autorização, e deve poder recuperar todas as informações da forma que estavam originalmente;
- Integridade de fonte: a origem da informação deve possuir credibilidade e ser autentica para que todos os usuários tenham confiança naquele dado. É possível fazer um paralelo com um jornal que recebe uma indicação de matéria de uma fonte falsa a respeito de algo que não existiu, e mesmo assim publica o texto integral. Neste caso, houve integridade de dados, pois o texto foi publicado da forma que foi recebido, porém não houve integridade de fonte, pois não foi possível verificar a veracidade das informações;
- Integridade de software: refere-se à integridade dos serviços disponibilizados por um provedor de nuvem, que não devem ser modificados por pessoas não autorizadas [45]. Um determinado serviço deve funcionar da mesma forma sempre que for requisitado, a menos que o provedor decida ou autorize alguém a alterá-lo.

Os mecanismos de integridade podem ser divididos em duas classes, os de prevenção e os de detecção. Os mecanismos da primeira classe basicamente se encarregam de bloquear as tentativas não autorizadas de alterar um dado, por exemplo, um usuário que não possui autorização e quer modificar um arquivo, ou talvez até tenha autorização para algumas operações e não para outras. Sistemas de autenticação e controle de acesso são representantes dessa classe [45].

Já os mecanismos da segunda classe tem como objetivo informar quando os dados não estão mais confiáveis. Para fazer isto, podem analisar os eventos do sistemas (como ações dos usuários) ou verificar os próprios dados para procurar indícios de violação [45]. Esses mecanismos podem ainda aplicar algoritmos de correção para que os dados voltem a um estado confiável.

A integridade está intimimamente ligada a confidencialidade, porém é muito mais ampla, pois leva em consideração a origem dos dados (como e quando foram obtidos), o armazenamento (é necessário saber se os dados foram armazenados corretamente) além da autorização e da autenticação (também presentes na confidencialidade) [46].

4.1.2 Disponibilidade

É possível definir disponibilidade como sendo a medida de probabilidade de um sistema não estar em falha em um determinado instante de tempo [46], ou como a capacidade de utilizar a informação ou recurso sempre que desejado [44].

No paradigma de computação em nuvem o usuário deve possuir os recursos que quiser e sempre que for necessário, por isso a disponibilidade é um conceito fundamental quando se fala em segurança de nuvem federada.

A disponibilidade deve ser tratada em dois âmbitos diferentes, o primeiro refere-se a disponibilidade do recurso, ou seja, caso um provedor de nuvens não esteja respondendo, deve haver um mecanismo de tolerância a falhas para permitir o uso de outros provedores, tornando transparente para o usuário a migração de um provedor para o outro. O outro ponto que deve ser levado em consideração é que um usuário não pode ter um serviço

negado caso esteja autorizado a acessá-lo, para tanto, é necessário que a autenticação e a autorização sejam feitas de forma correta, não deixando que ocorram casos de violação de disponibilidade nesse sentido.

4.1.3 Confidencialidade

Uma possível definição para confidencialidade é a ausência de conhecimento a respeito de informações não autorizadas [47], isto é, dados e informações protegidas só devem ser acessadas por pessoas autorizadas. Tanto empresas como o governo possuem dados confidenciais e querem restringir o acesso a algumas informações, como dados militares ou patentes de novas tecnologias, o que motivou a criação de diversas estratégias como a criptografia [48], e modelos de controle de acesso [49] [50] que serão melhores explorados ainda neste capítulo.

Para que seja garantida a confidencialidade, dois conceitos são fundamentais, o de autenticação e autorização [51]:

- Autenticação: é o processo que determina se a identidade de um usuário é legítima, e para isto, deve ser baseada em um ou mais dos seguintes fatores:
  - Algo que o usuário conhece, como por exemplo uma senha ou identificação única;
  - Algo que o usuário possui, como uma chave;
  - Alguma característica do usuário, como uma impressão digital ou padrão da retina.
- Autorização: é o processo que diz se um usuário pode ter acesso a um determinado recurso ou não. Para que a autorização seja efetiva ela depende da autenticação, pois se um sistema não é capaz de garantir que um usuário é quem ele realmente diz ser, não é possível determinar corretamente se ele possui acesso ou não a determinado recurso.

A confidencialidade dos dados portanto, está diretamente relacionada a autenticação de um usuário, pois é através dela que é possível determinar o que um usuário está autorizado ou não a realizar. A confidencialidade de software também é importante para a segurança geral do sistema, e consiste na confiança de que um processo ou aplicativo específico irá lidar com as informações do usuário de forma segura. As aplicações de software que interagem com os dados do usuário devem ser certificadas para não introduzir riscos, pois o acesso não autorizado pode tornar-se possível através da exploração de uma vulnerabilidade nestas aplicações [52].

As ameaças à confidencialidade são maiores na computação em nuvem devido ao grande número de dispositivos, aplicações e pontos de acesso aos serviços envolvidos, tornando assim um obstáculo para a completa adoção deste paradigma por diversas empresas e entidades que possuem informações confidenciais.

Dessa forma, o primeiro passo que um sistema deve realizar para garantir estas três propriedades de segurança é a utilização de um controle de acesso, que pode estar embutido dentro do sistema, pode ser incorporado à aplicação ou ainda ser implementado por pacotes de segurança [51]. A próxima seção abordará as principais questões envolvendo o controle de acesso.

4.2 Controle de Acesso

Controle de acesso é a forma pela qual o acesso (utilização, modificação e leitura) a determinado recurso é concedido ou proibido [53]. O controle de acesso pode trabalhar de diversas formas diferentes, além de determinar se um usuário tem permissão para acessar um recurso, ele pode determinar também, quando e como esse recurso será utilizado. Um certo arquivo, por exemplo, pode ter seu acesso restringido dentro de um horário específico. Um determinado processo pode ter permissão apenas de leitura e não de escrita, dentre outros diversos exemplos.

O controle de acesso é um mecanismo de segurança, que assim como outros mecanismos, atua na integridade, na disponibilidade e na confidencialidade dos dados dos usuários do sistema. Na questão da integridade, apenas usuários autorizados devem poder alterar informações, e mesmo assim dentro dos limites para o qual ele está autorizado. Quanto à disponibilidade, um usuário sempre deve poder acessar os dados que ele possui autorização de acesso. E por fim, é importante que apenas usuários autorizados possam ler determinadas informações, garantindo assim o sigilo e a confidencialidade dos dados [51].

Qualquer modelo de controle de acesso pode ser descrito formalmente usando descrições de usuários, sujeitos, objetos, operações e permissões [54]. É importante então, entender cada um destes conceitos que são descritos a seguir [55]:

- Usuário: este termo se refere geralmente às pessoas que interagem com o sistema, mas também pode representar outros agentes, como por exemplo um dispositivo ou uma máquina;
- Sujeito: é uma tarefa ou um processo que está agindo em nome do usuário, e executa uma ou mais operações sobre um ou mais objetos do sistema. Um usuário pode ter múltiplos sujeitos, e todos os sujeitos possuem um identificador único. Um usuário, por exemplo, pode executar vários programas na nuvem ao mesmo tempo. Neste caso, cada programa executando ao mesmo tempo é um sujeito do mesmo usuário;
- Objeto: é qualquer recurso do sistema, sob o qual podem ser executadas operações. São exemplos de objetos em um sistema: arquivos, periféricos (impressoras, câmeras e etc.), banco de dados, programas, diretórios, entre outros;
- Operações: são processos ativos que foram invocados por um sujeito. Existem diversas operações, as mais comuns são as de leitura, de escrita e de alteração, mas existem também operações mais complexas, como transações bancárias;
- Permissões: são autorizações para a realização de ações no sistema. É uma relação entre um objeto, um sujeito e uma operação.

Assim sendo, os modelos de controle de acesso correspondem à descrição formal do comportamento de um sistema regido sob regras de segurança. Na literatura é comum dividir estes modelos em três tipos básicos [56]:

- Controle de Acesso Discricionário (*Discretionary Access Control* — DAC): baseia-se na ideia de que o proprietário da informação deve determinar quem tem acesso a ela. O usuário pode determinar também o tipo de acesso a cada informação, por exemplo, um arquivo pode ser somente lido por todos os usuários, enquanto

que outro arquivo pode ser lido e modificado. Este tipo de controle é considerado inadequado para diversos tipos de aplicações, uma vez que são muito flexíveis, pois basta um equívoco do usuário para que as informações importantes sejam reveladas a pessoas não autorizadas;

- Controle de Acesso Obrigatório (*Mandatory Access Control* — MAC): é uma forma de controle centralizada, na qual quem decide se um usuário tem acesso a um determinado arquivo é o próprio sistema, e não o usuário como no DAC. Para realizar esta tomada de decisão, o sistema avalia os rótulos que foram previamente associados aos usuários e objetos. Os rótulos correspondem aos níveis de acesso que o sistema posssui, sendo que quanto maior o nível do rótulo, maior é a quantidade de informações que um usuário pode acessar;
- Controle de Acesso Baseado em Papéis (*Role-Based Access Control* — RBAC): os direitos de acesso são associados a papéis e não diretamente aos usuários. Os usuários são classificados então, em grupos que possuem o mesmo papel e, portanto, possuem acesso ao mesmo conjunto de informações.

Dada a importância, para este trabalho, do modelo de controle de acesso, é preciso conhecer exemplos de modelos pertencentes a estas classes, os quais serão apresentados nas próximas seções.

4.2.1 O Modelo de Matriz de Acesso

O modelo de matriz de acesso [50] é um exemplo de controle de acesso discricionário. Neste modelo, o estado de segurança do sistema é representado por uma matriz M, na qual as linhas representam os sujeitos, as colunas representam os objetos, e as células M_{ij} representam os direitos de acesso dos sujeitos i sobre os objetos j. Um sujeito, não necessariamente precisa ser um usuário, pois ele pode, por exemplo, ser um programa que precisa de permissão para escrever em um certo arquivo. A Figura 4.1 apresenta um exemplo de matriz que possui três sujeitos, sendo dois deles usuários (usuário1 e usuário2) e um programa (prog1), e quatro objetos, sendo três arquivos (arq1, arq2 e arq3) e um programa (prog1). Os direitos que um sujeito pode ter são escrita (w), leitura (r), execução(ex), ou o sujeito pode ser dono daquele objeto. Espaços em branco indicam que o sujeito não tem nenhuma permissão sobre aquele objeto.

Este modelo apresenta uma grande flexibilidade no controle de acesso às informações, visto que os próprios donos do arquivo são responsáveis por fornecerem as permissões para os outros usuários. Existe porém, um problema quanto à disseminação da informação, pois o usuário2 pode realizar uma cópia do arq3 e disponibilizar acesso de leitura para o usuário1. Neste caso, o usuário1 conseguirá acessar as informações do arq3 que ele não possuía autorização para ver.

Implementar este modelo em forma de uma matriz propriamente dita, pode não ser uma boa opção, devido ao tamanho que ela pode alcançar e a grande quantidade de células em branco. Uma forma popular de implementar este modelo é utilizando as listas de controle de acesso (*Acces Control Lists* — ACLs) [57].

Na lista de controle de acesso, cada objeto é associado a uma lista, que indica para cada sujeito, suas permissões sobre aquele objeto. A Figura 4.2 representa a lista de controle de acesso referente à matriz de acesso da Figura 4.1.

	arq1	arq2	arq3	prog1
usuário1	dono r w	r		ex
usuário2	r	dono r w	r	ex
prog1	r w		r w	

Figura 4.1: Exemplo de Matriz de Acesso.

arq1	(usuário1, {dono, r, w}), (usuário2, {r}), (prog1, {r, w})
arq2	(usuário1, {r}), (usuário2, {dono, r, w})
arq3	(usuário2, {r}), (prog1, {r, w})
prog1	(usuário1, {ex}), (usuário2, {ex})

Figura 4.2: Exemplo de Lista de Controle de Acesso.

Esta abordagem utiliza menos espaço para ser armazenada, visto que os espaços em branco na matriz não são representados. Para revogar todos as permissões de um determinado objeto é bastante simples, basta substituir a lista de permissões do objeto por uma lista nula, mas para determinar quais objetos um usuário pode acessar é bastante complexo, pois é preciso percorrer as listas de todos os objetos.

Dessa forma, a adoção deste modelo de controle de acesso em um ambiente de nuvem federada é complicada, pois a quantidade de usuários e de arquivos pode tornar a lista de controle de acesso muito grande, gerando um alto custo de processamento toda vez que houver uma modificação nas permissões dos objetos.

4.2.2 O Modelo Bell-LaPadula (Controle de Acesso Obrigatório)

Em 1973, David Bell e Leonard LaPadula [58] propuseram um modelo que representa um controle de acesso obrigatório, descrevendo formalmente os caminhos permitidos para o fluxo da informação no sistema. Este modelo trata exclusivamente da confidencialidade dos dados e tem sido utilizado para tratar de dados sensíveis, e que requerem uma política de segurança multinível.

Na descrição original [58], máquinas de estados finitos são utilizados para descrever um sistema, na qual as transições obedecem a regras. Foi demonstrado que se partindo de um estado seguro e seguindo as únicas transições possíveis, se chega a um outro estado seguro, então a segurança do sistema está mantida, pois um estado inseguro nunca é alcançado. Estes estados correspondem a sujeitos que tentam acessar objetos, e ambos possuem classificações que definem os seus níveis de segurança. Existem várias formas de

classificação, sendo que a mais comum é a utilizada por governos, que divide os documentos em não classificados, confidencial, secreto e super secreto [57].

Dessa forma, as transições correspondem às ações que os sujeitos podem efetuar sobre os objetos e seguem duas propriedades:

- Propriedade da Segurança Simples: uma leitura de um sujeito S sobre um objeto O é autorizada se, e somente se, a classificação de S for maior ou igual a classificação de O. Isto quer dizer que um objeto classificado como secreto só pode ser lido por um sujeito que possuir a classificação secreto ou super secreto.
- Propriedade Estrela: nenhum sujeito tem acesso de escrita a objetos que estão classificados de forma diferente da classificação dele próprio; e nenhum sujeito tem acesso de leitura em objetos que possuem nível de segurança corrente maior do que a dele próprio.

Com estas propriedades é garantida a segurança utilizando o raciocínio de que não é possível ler informações de um nível mais alto e nem é possível escrever informações para um nível mais baixo. Assim sendo, este modelo é eficaz em manter a confidencialidade, porém é bastante rígido, e por isso não é muito utilizado comercialmente, sendo preferido apenas por organizações governamentais ou militares, ou organizações que possuem dados extremamente sensíveis [59]. Em um ambiente dinâmico, como o que ocorre nas nuvens federadas, torna-se inviável a adoção deste modelo como forma de controle de acesso, sendo necessário um modelo mais flexível.

4.2.3 O Modelo Clark-Wilson (Controle de Acesso Discricionário)

David Clark e David Wilson apresentaram em 1987 um modelo de integridade diferente [60], utilizando práticas administrativas e contábeis aplicadas à computação. O modelo baseia-se em dois princípios que foram chamados de transações bem formadas e separação de responsabilidades [51].

Uma transação bem formada é um conceito em que o usuário não deve manipular dados arbitrariamente, mas apenas de forma controlada, ou seja, de forma a preservar e garantir a integridade dos dados. Um mecanismo utilizado em transações bem formadas é o arquivo de *log*, um arquivo que contém todas os registros de modificações ocorridos no sistema, de forma que as transações possam ser auditadas futuramente.

O outro princípio, o de separação de responsabilidades, visa garantir a consistência externa dos dados, fazendo com que os dados registrados no sistema coincidam com a situação real representada. Para isto as operações são divididas em várias suboperações e cada uma delas deve ser executada por uma pessoa diferente. Uma compra de mercadoria, por exemplo, pode ser dividida em diversas operações, como verificação de estoque, autorização da compra, registro da fatura e pagamento da fatura. Cada etapa só pode ser realizada após a conclusão da estapa anterior.

Esse modelo apresenta um foco bem diferente daquele apresentado por Bell e LaPadula [58], que estavam mais preocupados em propor um modelo que garantisse a confidencialidade dos dados, e para isto, tiveram que impor regras rígidas para leitura e escrita. Enquanto que Clark e Wilson [60] estavam mais preocupados em desenvolver melhorias e aumentar a discussão sobre ferramentas de segurança voltadas ao ambiente

comercial [61]. Este modelo não foi escolhido para a utilização na plataforma de nuvem federada BioNimbuZ, pois é um modelo voltado ao ambiente comercial.

4.2.4 Modelos Baseados em Papéis

Os papéis dos usuários dentro de uma organização são relativamente estáveis, enquanto que os usuários e as permissões são numerosos e podem mudar com frequência [54]. Assim sendo, os modelos baseados em papéis regulam o acesso dos usuários a uma determinada informação com base nas atividades que o usuário desempenha, ou seja, seu papel dentro da organização. As permissões não são definidas diretamente aos usuários, como nos outros modelos, mas sim aos papéis existentes. Desta forma, um usuário possui todas as permissões que foram definidas para o papel que ele exerce. O relacionamento entre os usuários, os papéis e as permissões são mostradas na Figura 4.3.

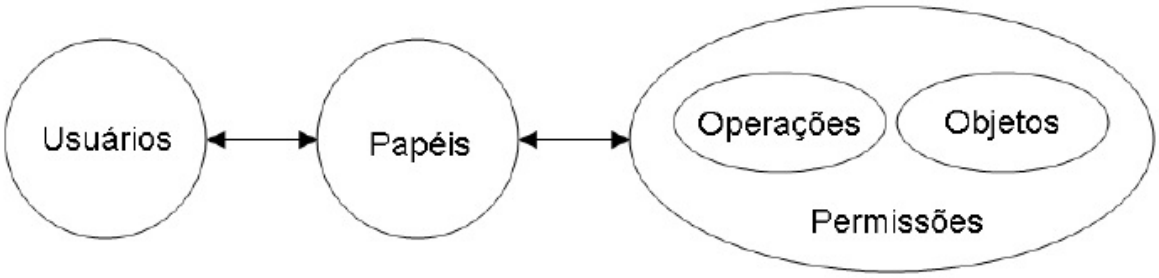

Figura 4.3: Relacionamento entre os Usuários, os Papéis e as Permissões [55].

Os usuário desempenham papéis que por sua vez possuem permissões. As permissões representam as operações que podem ser feitas sobre os objetos.

Os modelos baseados em papéis possuem diversas características, tais como [56] [62]:

- Gerência de Autorização Simples: é preciso se preocupar com apenas dois aspectos na gerência de autorização. O primeiro é a atribuição das autorizações aos papéis da organização, e o segundo é a alocação dos usuários nos respectivos papéis desempenhados. Isso simplifica o controle e deixa fácil a realização de modificações;
- Suporte a Hierarquia de Papéis: é possível definir uma hierárquia entre os papéis utilizando a noção de generalização e especialização, algo comum em muitas aplicações;
- Suporte a Mínimo Privilégio: os papéis fazem com que o usuário possa utilizar apenas o mínimo privilégio requerido por determinada tarefa. Usuários que possuem um papel poderoso na organização só precisam utilizá-lo quando for realmente necessário;
- Suporte a Separação de Deveres: estes modelos baseados em papéis suportam a separação de deveres. Para obter esta separação é necessário restringir as autorizações de papéis considerados mutuamente exclusivos;
- Delegação da Administração de Segurança: é possível descentralizar a administração de segurança nos modelos baseados em papéis. Isto siginifica que o administrador de

segurança pode delegar parte de suas atribuições de acordo com a estrutura da empresa, em outras palavras, é possível definir administradores para papéis específicos do sistema.

Nos modelos baseados em papéis, identificar quais as permissões de um usuário é simples, basta verificar seus papéis e quais as permissões destes papéis. Porém, para revogar uma permissão de um usuário é necessário retirá-lo daquele papel e adicioná-lo a outro. Em um ambiente de nuvem federada em que é preciso conceder diferentes permissões para diferentes usuários, é necessário a criação de muitos papéis, tornando a gerência do controle de acesso complicada.

Assim, é importante notar que os modelos de controle de acesso não são necessariamente excludentes [62]. Eles podem ser combinados de forma a fornecer um modelo de segurança mais adequado às necessidades de cada sistema, como mostrado na Figura 4.4. Quando os modelos são combinados porém, é importante ter cuidado para que não haja conflito entre as políticas, pois um modelo pode conceder uma determinada autorização enquanto que o outro pode negar esta autorização.

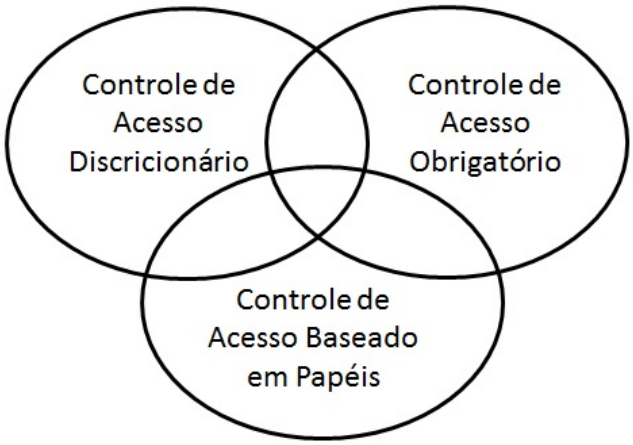

Figura 4.4: Interação entre os Modelos de Controle de Acesso, adaptado de Sandhu and Samarati [62].

Na próxima seção será apresentado o modelo de controle de acesso baseado em atributos. Ele foi o modelo escolhido para implementar o controle de acesso no ambiente de nuvem federada BioNimbuZ por ser um modelo de fácil gerência, flexível e poder ser adaptado para funcionar como os outros modelos.

4.3 Controle de Acesso Baseado em Atributos

O controle de acesso baseado em atributos (*Attribute Based Access Control* — ABAC) pode ser definido como um modelo em que os pedidos dos usuários para a relização de operações sobre os objetos, são concedidos ou negados, de acordo com os atributos envolvidos [63]. São utilizados três tipos de atributos [64]:

- Atributos de Sujeito: sujeitos são entidades que realizam ações sobre os objetos no sistema, como por exemplo usuários, aplicativos ou processos. Os sujeitos possuem

atributos associados a eles que definem suas identidades e características. Dentre os possíveis atributos de sujeito estão o nome, o cargo e a empresa.

- Atributos de Objeto: os objetos são entidades que recebem as ações realizadas por sujeitos. Assim como nos sujeitos, os objetos possuem atributos que podem ser utilizados para tomar decisões de controle de acesso. Um determinado arquivo, por exemplo, pode ter como atributos seu título, seu autor, sua data de criação e seu assunto.
- Atributos de Sistema: são atributos que costumam ser ignorados pelo outros modelos de controle de acesso, pois não estão associados a um determinado recurso e são condições do ambiente, como por exemplo a data, o horário de acesso e a localização geográfica.

As permissões são definidas por meio de regras que recebem como entrada os valores atribuidos aos atributos e retornam verdadeiro, se a autorização for concedida ou falso, se for negada. Nas regras, os sujeitos são representados pela letra s, os objetos pela letra o, e o ambiente pela letra a. As regras podem utilizar operadores lógicos para expressar relações entre os atributos como no exemplo a seguir:

$$pode\_acessar(s, o, a) : (Cargo(s) = "Gerente") \wedge (Tipo(o) = "Confidencial")$$

Nesse caso, os sujeitos que possuem o valor gerente para o atributo cargo, podem acessar os objetos que possuem o valor confidencial para o atributo tipo. Além disso, os atributos podem ser combinados para formar diversos tipos de regras de acesso. É possível, por exemplo, obter o modelo baseado em papéis utilizando a flexibilidade do modelo baseado em atributos, bastando ter um atributo papel e atribuindo os valores dos atributos aos usuários.

O funcionamento deste modelo pode ser visto na Figura 4.5. Inicialmente, um sujeito realiza uma requisição para acessar um objeto (passo 1). Em seguida, o mecanismo de controle de acesso pesquisa, no repositório de políticas, as regras envolvendo o objeto desejado (passo 2.1), e verifica os atributos de sujeito (passo 2.2) e objeto (passo 2.3), para autorizar ou não o acesso. Nesse modelo também é levado em consideração as condições de ambiente que podem interferir no acesso (passo 2.4). Assim, um determinado arquivo, por exemplo, pode ter seu acesso restrito a determinado horário, ou a uma determinada localização geográfica. E por fim, se os atributos atenderem ao que foi estabelecido nas regras, o acesso ao objeto é liberado (passo 3).

Neste modelo, para revogar o acesso a determinado objeto existem duas alternativas. A primeira é a alteração da regra de acesso que está associada ao objeto e ao usuário, e a segunda é a alteração dos valores dos atributos envolvidos nas regras. Assim, este modelo é mais flexível do que o baseado em papéis, pois com uma simples alteração do valor de um atributo é possível revogar o acesso a determinado objeto, enquanto que no RBAC é preciso retirar o usuário do papel.

Nesse cenário, foi proposta uma arquitetura de autorização para o ABAC [64], que representa os atores lógicos envolvidos neste modelo, e está representada na Figura 4.6. Os atores envolvidos são [64]:

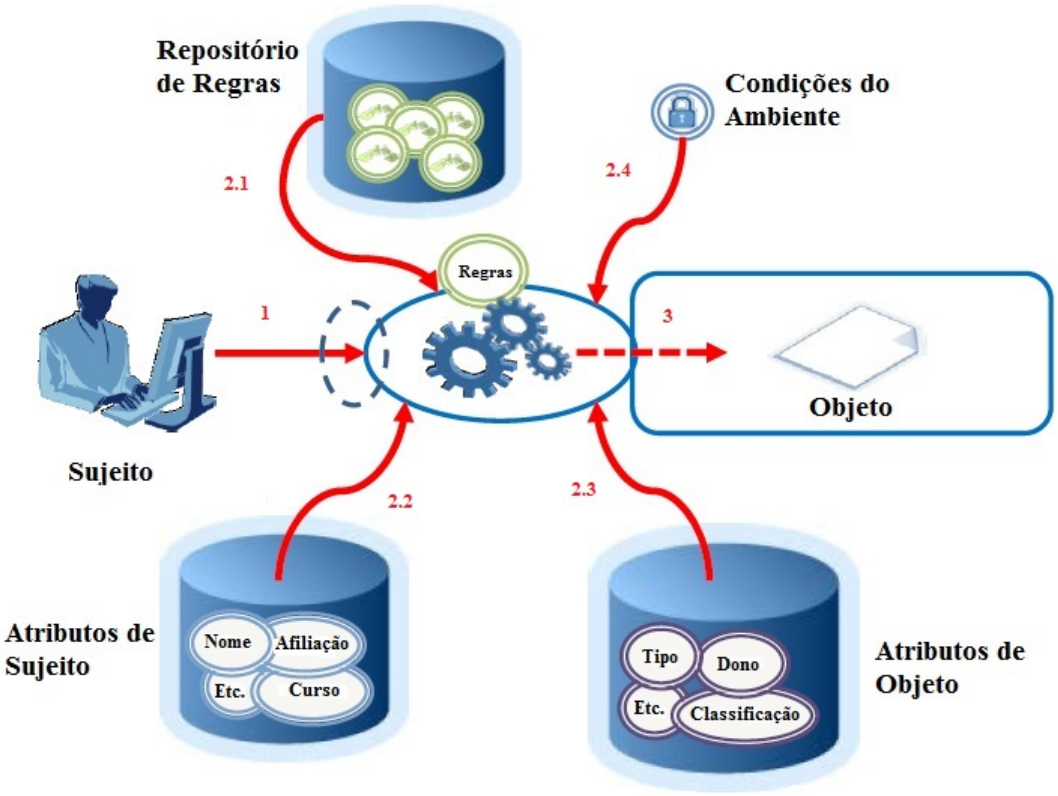

Figura 4.5: Funcionamento do Modelo ABAC (Adaptado de Ferraiolo et al. [63]).

- Gerentes de Atributos: são os responsáveis por gerenciar todos os atributos. São responsáveis também por realizar a atribuição dos valores dos atributos aos sujeitos, aos objetos e aos ambientes, e podem ou não armazenar estes valores;
- Ponto de Aplicação de Políticas (*Policy Enforcement Point* - PEP): é o responsável por solicitar as decisões de autorização e também pela aplicação das mesmas. É o ponto de presença do controle de acesso no sistema, e atua como intermediário entre todas as requisições dos usuários aos objetos. Apesar de estar representado como um ponto único, ele pode estar distribuído no sistema, mas é essencial que ele nunca seja ignorado e esteja presente em todas as requisições dos usuários;
- Ponto de Decisão de Políticas (*Policy Decision Point* - PDP): é o responsável por realizar as avaliações das regras de acesso e tomar a decisão de autorização (permitir ou negar). Para avaliar as regras, ele entra em contato com os gerentes responsáveis pelos atributos, e solicita os valores dos atributos envolvidos na regras;
- Gerente de Políticas: responsável pela criação e pelo gerenciamento das regras de autorização.

Nesta arquitetura, quando um usuário requer o acesso a um determinado recurso, o PEP entra em contato com o PDP para que a decisão de autorização seja tomada. O PDP verifica as regras envolvidas consultado o gerente de políticas, e verifica os atributos

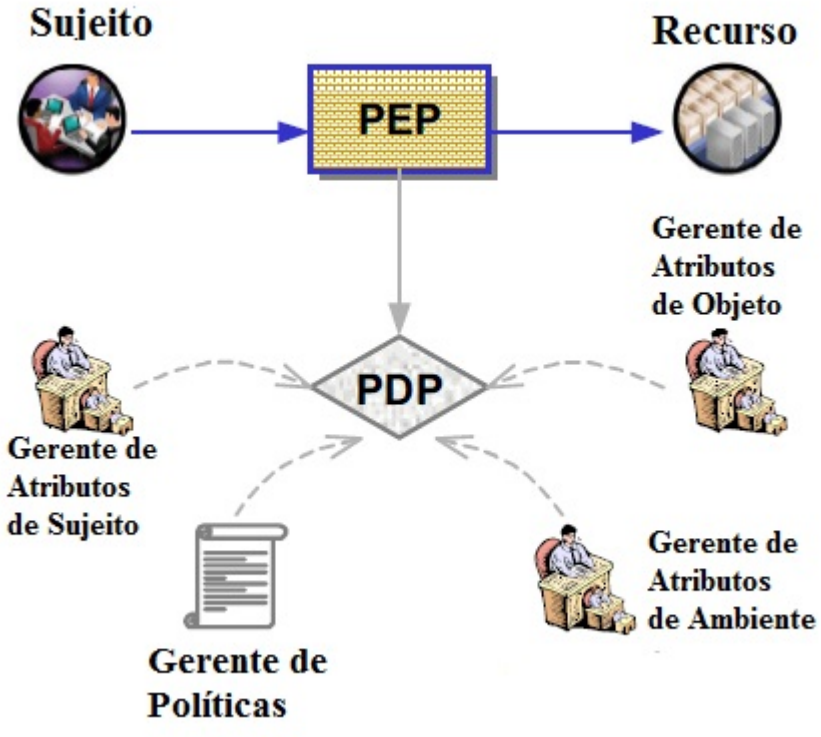

Figura 4.6: Arquitetura de Autorização do ABAC (Adaptado de [64]).

consultado os gerentes de atributos. Com as regras e os atributos é possível tomar uma decisão, que é repassada ao PEP para que ela seja aplicada.

No próximo capítulo será apresentado o modelo de controle de acesso baseado em atributos que foi implementado neste trabalho na plataforma BioNimbuZ.

Capítulo 5

Segurança no BioNimbuZ

Este capítulo tem como objetivo apresentar a implementação do modelo de controle de acesso baseado em atributos (ABAC) na arquitetuta para federação de nuvens computacionais BioNimbuZ. Além disso, este capítulo apresenta quais modificações foram efetuadas no código original do BioNimbuZ para que a integração pudesse ser feita de maneira eficiente. A Seção 5.1 apresenta como é feito o processo de autenticação e a Seção 5.2 mostra com detalhes a implementação da autorização no sistema. E por fim, a Seção 5.3 apresenta como foi feita a integração com o código do BioNimbuZ.

5.1 Autenticação

Como visto no Capítulo 4, autenticação é o processo que determina se a identidade de um usuário é legítima. Assim, o primeiro passo para a implementação do sistema de autenticação no BioNimbuZ foi a definição da forma como as informações dos usuários (identificador e senha) seriam armazenadas. Algumas possibilidades foram levantadas, suas vantagens e desvantagens foram analisadas. A seguir será apresentada a linha de raciocínio que levou à solução escolhida.

1. Texto Simples: armazenar a senha dos usuários em forma de texto simples não é uma boa solução, pois qualquer invasor que conseguir ter acesso ao banco de dados poderá acessar facilmente a conta de todos os usuários apenas lendo as informações, como pode ser visto no exemplo da Figura 5.1;

Identificador	Senha
usuario1	123
usuario2	senha123
usuario3	internet

Figura 5.1: Armazenamento de Senhas em Texto Simples.

2. Descaracterizar a Senha: para evitar que os dados sejam facilmente obtidos por um invasor, a senha não deve ser armazenada diretamente, mas sim modificada de

alguma forma, como por exemplo uma função de *hash* [65]. Existem duas funções de *hash* que foram largamente utilizadas na literatura — MD-5 [66] e SHA-1 [67] — porém, já não são mais consideradas seguras [68], o que faz com que versões mais recentes como a SHA-2 [67] sejam mais recomendadas.

Contudo, essa ainda não é a melhor forma de armazenar as senhas. Caso um invasor consiga acesso ao banco de dados, ele não precisa desfazer (algo que iria contra a definição de *hash*) a função *hash*. Ele pode gerar uma tabela com os *hashes* de várias possibilidades de senhas e compará-las com os valores armazenados, e se algum valor coincidir ele poderá ter acesso, mesmo que a senha não seja exatamente a mesma, mas que tenha gerado o mesmo valor de *hash*. Esta tabela de *hashes* de senhas comuns é chamada de *rainbow table* [69] e tem o funcionamento exemplificado na Figura 5.2. Nesta figura, os identificadores estão armazenados com seus respectivos valores de senhas descaracterizadas. Na *rainbow table* estão os valores de senhas que são muito utilizadas e os seus valores após a utilização de uma função de *hash*. Os valores são comparados para tentar encontrar um valor que coincida e então obter o acesso. Na Figura 5.2 é possível acessar o sistema utilizando o identificador *usuario2* e a senha *senha123*.

Dessa forma, mesmo sem acesso ao banco de dados é possível realizar um ataque tentanto exaustivavamente todas as possibilidades de senha até encontrar a correta (também chamado ataque de força bruta), por isso é importante limitar as tentativas de autenticação em um sistema;

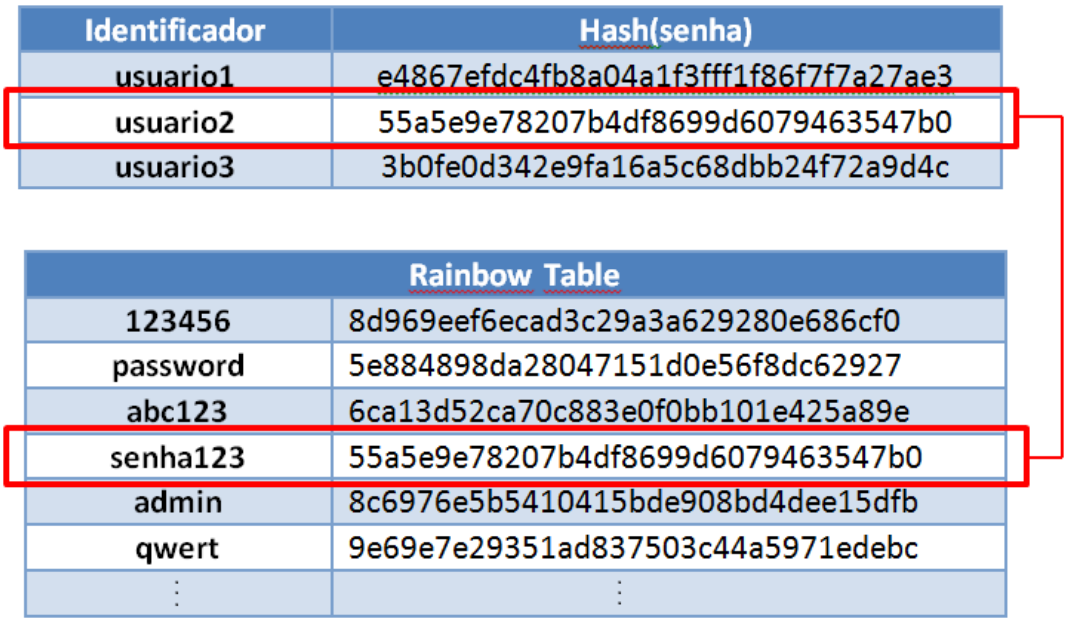

Figura 5.2: Armazenamento de Senhas com *Hash* e *Rainbow Table*.

3. Utilizar *Salts*: é uma técnica em que um número aleatório (chamado *salt*) é concatenado com a senha antes da utilização da função de *hash* [65]. Assim, o número de possibilidades de senhas que um possível invasor deve testar aumenta consideravelmente, tornando o trabalho de descobrir a senha bem mais lento do que na solução

anterior. O *salt* utilizado deve ser armazenado para que seja possível realizar a correta autenticação do usuário quando este digitar a sua senha, como mostrado na Figura 5.3. São armazenados os valores dos identificadores, dos *salts*, e dos *hashes* das senhas concatenadas aos *salts*. Assim, quando um usuário tenta realizar a autenticação, os valores de identificador e senha são inseridos. O serviço de segurança verifica o valor do *salt* que está relacionado ao identificador, e concatena o seu valor com a senha inserida antes de utilizar a função de *hash*. O valor obtido por meio da função de *hash* é comparado com o valor armazenado, e se coincidir o acesso é liberado;

Esta técnica porém, ainda é vulnerável, pois a capacidade de processamento está cada vez maior e existem abordagens que utilizam paralelismo e GPU's que testam milhares de possibilidades de senha por segundo [70] [71].

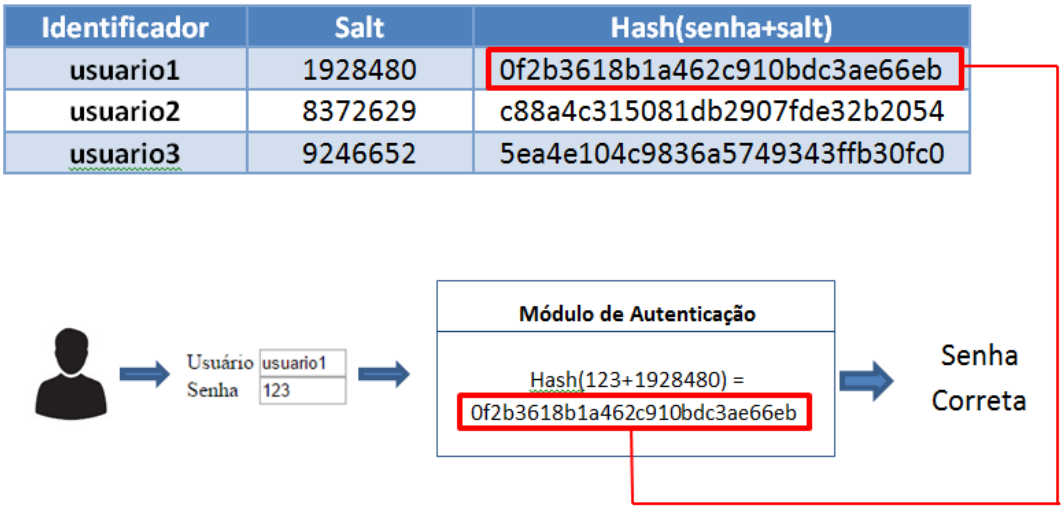

Figura 5.3: Utilização de *Salts*.

4. Reforçar a Senha: para evitar que a senha possa ser descoberta utilizando o grande poder de processamento disponível, é preciso aumentar o tempo que cada possibilidade de senha demora para ser testada. O NIST recomenda [72] a utilização do PBKDF2 [73], uma função de derivação de chaves, ou seja, um mecanismo que dado uma chave original consegue derivar outra chave através de algum processo. O PBKDF2 utiliza a técnica HMAC [74] em conjunto com múltiplas iterações, que podem aumentar de acordo com o nível desejado de segurança, de forma que uma tentativa de descoberta dos valores originais se torne inviável com a capacidade de processamento atual. Um exemplo simplificado do funcionamento do PBKDF2 pode ser ser visto na Figura 5.4. O HMAC é utilizado sobre a senha e o *salt* resultando na primeira iteração. Este valor da primeira iteração serve como um *salt* que é utilizado novamente com a senha na função HMAC para gerar o valor da próxima iteração. Estas etapas se repetem 10000 vezes neste exemplo. O valor final do *hash* equivale ao XOR de todas as iterações.

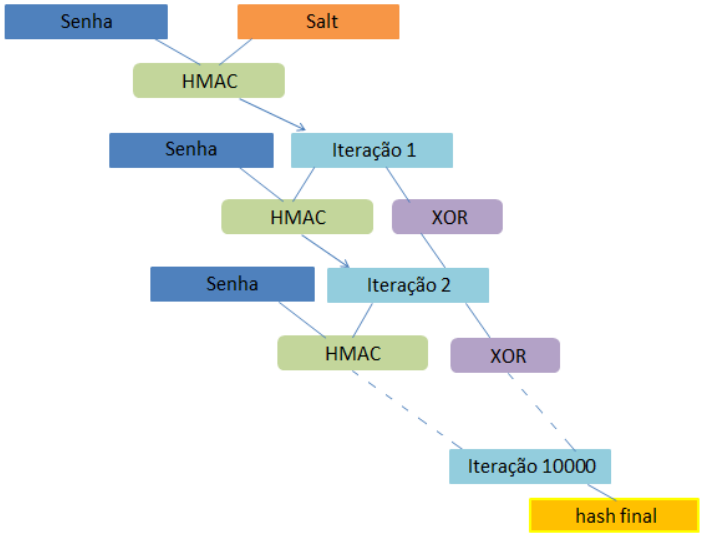

Figura 5.4: Esquema de Funcionamento do PBKDF2 para 10000 Iterações.

5.2 Autorização

A autorização, como apresentado no Capítulo 4, é o processo que diz se um usuário pode ter acesso a um determinado recurso ou não. Para implementar a autorização na plataforma BioNimbuZ, algumas classes foram criadas de forma a representar os atores mostrados na Figura 4.6 e também as entidades do sistema. As principais classes estão descritas a seguir nas próximas seções.

5.2.1 Gerente de Políticas

É a classe responsável por gerenciar todas as regras de acesso presentes no sistema. Uma regra de acesso é uma expressão, que relaciona os valores dos atributos de sujeito, de objeto e de sistema, para chegar a uma decisão de autorização. Os métodos desta classe realizam:

- A criação de regras: para que uma nova regra seja criada pelo serviço de segurança do BioNimbuZ é preciso seguir um padrão. A regra deve ter a forma

  $$x.nome = valor$$

  na qual *nome* representa o nome do atributo, *valor* representa o valor do atributo, e x deve ser substituído por u, o ou s. Ele será substituído por u caso o atributo seja de usuário, ou por o, caso o atributo seja de objeto, e por s, caso o atributo seja de sistema. Um exemplo é a regra

  $$u.idade = 22$$

  que representa os usuários que possuem idade igual a 22.

  Além disso, as regras podem ser combinadas, utilizando-se os operadores lógicos AND e OR, para formar regras mais complexas, como nas regras

  $$o.permissao = aberto\ AND\ o.extensao = java$$

$$u.cargo = analista\ OR\ u.diretoria = tecnologia$$

que representam os arquivos cuja permissão seja aberta e que tenham o atributo *extensao* igual a java, e os usuários que possuem cargo de analista ou que são da diretoria de tecnologia.

Algumas regras são padrões dentro do serviço de segurança, e não precisam ser criadas, podendo ser excluídas caso o administrador assim desejar. Essas são as regras que autorizam qualquer usuário a acessar arquivos com permissão aberta, e também os seus próprios arquivos, ou seja, arquivos cujo atributo proprietário seja ele mesmo.

As regras são escritas nesse padrão para facilitar o entendimento por parte dos usuários, porém são armazenadas no sistema em forma de consulta ao banco de dados em um repositório específico para regras. Cada regra contém um identificador único para sua identificação;

- A exclusão de regras: assim como é possível criar regras de autorização no serviço de segurança, também é possível excluí-las. Para isto, é necessário utilizar o método específico de exclusão, informando o identificador da regra que se deseja excluir. Quando uma regra é excluída, todas as associações referentes a ela também são excluídas, ou seja, as tomadas de decisão que levavam em consideração esta regra, não irão mais considerá-la;

- A atribuição de regras: as regras recém criadas precisam ser atribuídas aos objetos e usuários para que possam realizar a sua tarefa. As regras envolvendo atributos de sistema não precisam ser atribuídas aos usuários e objetos, pois elas afetam a todos que estejam nas condições de ambiente do sistema.

  As regras que são padrões dentro do sistema são atribuídas aos objetos e usuários de forma automática, como por exemplo a regra que autoriza os usuários a acessarem os seus próprios arquivos.

É importante ressaltar que apenas o administrador possui acesso ao painel de gerenciamento, e só ele pode criar, excluir e atribuir regras no serviço de segurança do BioNimbuZ, enquanto que os usuários comuns possuem acesso à interface de usuário, e podem realizar o *upload*, o *download*, a listagem de arquivos, e também submeter, consultar e cancelar *jobs*.

5.2.2 Ponto de Decisão de Políticas

Classe responsável por realizar a tomada de decisão quanto à autorização de acesso dos usuários aos objetos. Toda requisição de acesso, a qualquer tipo de objeto do sistema, deve passar antes pela análise dos métodos desta classe.

O primeiro passo para a tomada de decisão é a identificação de quais regras estão associadas aos objetos que o usuário deseja acessar, consultando o repositório de regras. O PDP (*Policy Decision Point*) então, entra em contato com os gerentes dos atributos de usuários, de objetos e de sistema, a fim de obter os valores dos atributos que estão presentes nas regras. Se os valores dos atributos corresponderem aos que estão presentes nas regras, o acesso é autorizado, caso contrário, o acesso é negado.

Existe um método desta classe que lista todos os objetos que um determinado usuário pode acessar, que é uma funcionalidade que a interface de usuário disponibiliza aos usuários do BioNimbuZ. Este método constrói uma lista na memória contendo os arquivos que o usuário pode acessar. Assim sendo, se o método for chamado duas vezes em sequência, a primeira vez será mais lenta do que a segunda, pois o tempo para calcular a autorização dos arquivos será maior, enquanto que na segunda chamada bastará percorrer a lista que já está montada.

5.2.3 Gerente de Atributos

A classe de gerente de atributos é uma classe abstrata que possui os métodos para cadastrar, excluir e alterar os valores de um atributo. Atualmente, três classes estendem esta classe, de forma a especializar o gerenciamento dos atributos, que são as classes dos atributos de usuário, de objeto e de sistema como representado na Figura 5.5. Todo atributo possui um nome, um valor e um identificador único.

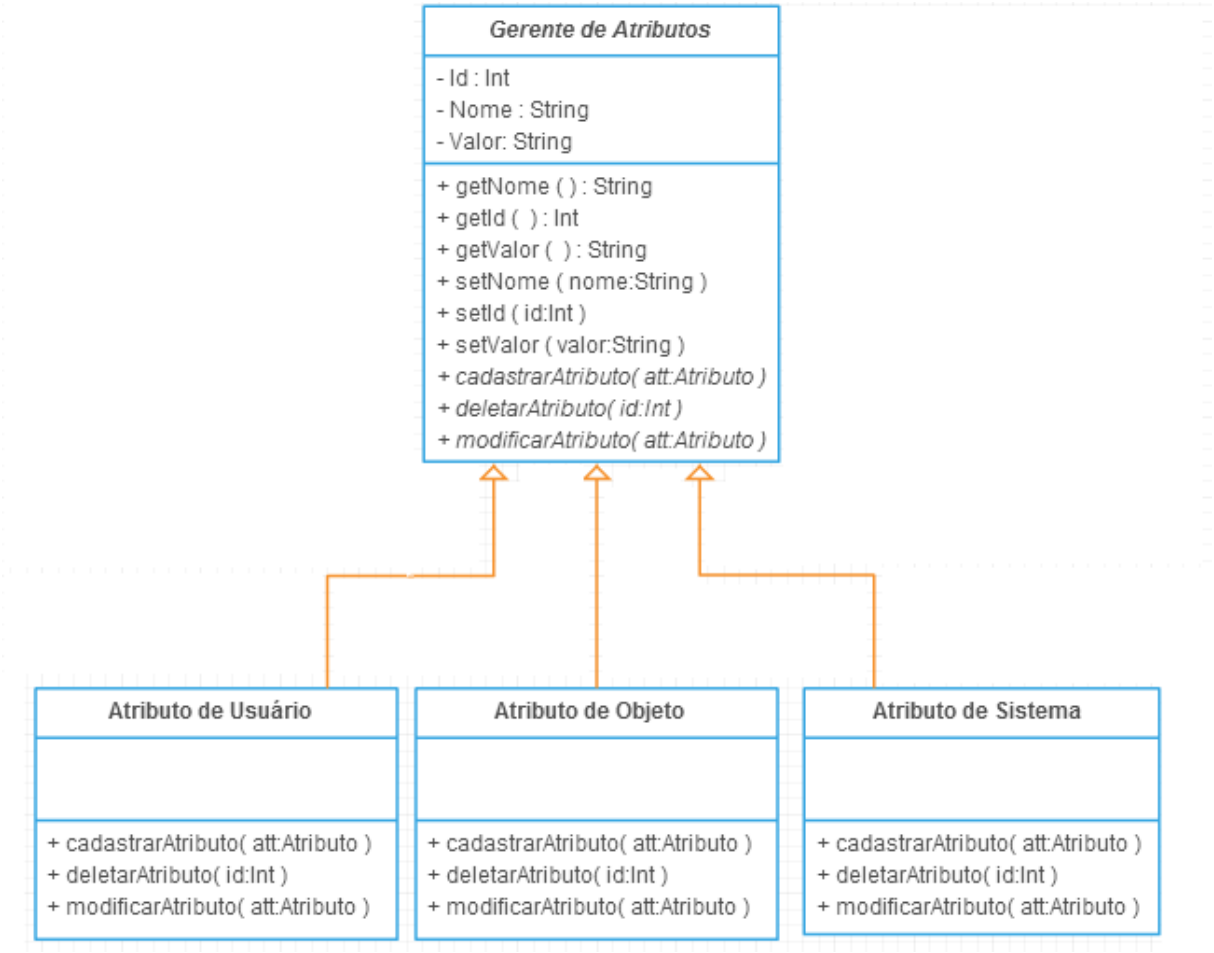

Figura 5.5: Diagrama de Classes dos Atributos.

Assim como ocorre com as regras de autorização, apenas o administrador pode criar novos atributo no sistema. Alguns atributos já são criados de forma automática, como por exemplo, extensão, tamanho, proprietários que são atributos de arquivo.

Para criar um novo tipo de atributo (além de atributos de usuário, de objeto e de sistema) é preciso apenas estender a classe gerente de atributos e implementar os métodos de gerenciamento que for preciso. Isto pode ser útil para criar novos modelos de controle de acesso, que levam em consideração outros tipos de atributos na decisão de autorização.

5.2.4 Entidades

As entidades são as classes que representam os usuários, os objetos e os atributos, e estão representadas na Figura 5.6. As classes de entidades possuem os métodos para a criação, a exclusão e a modificação das mesmas. É possível ainda, obter uma relação com todas as regras associadas àquela entidade, e uma relação de todos os usuários, os objetos e os sistemas cadastrados.

Para se obter uma relação de todas as regras associadas a uma determinada entidade é preciso consultar o repositório de regras. Utilizando o identificador de cada entidade, obtém-se todas as regras que estão associadas a ela.

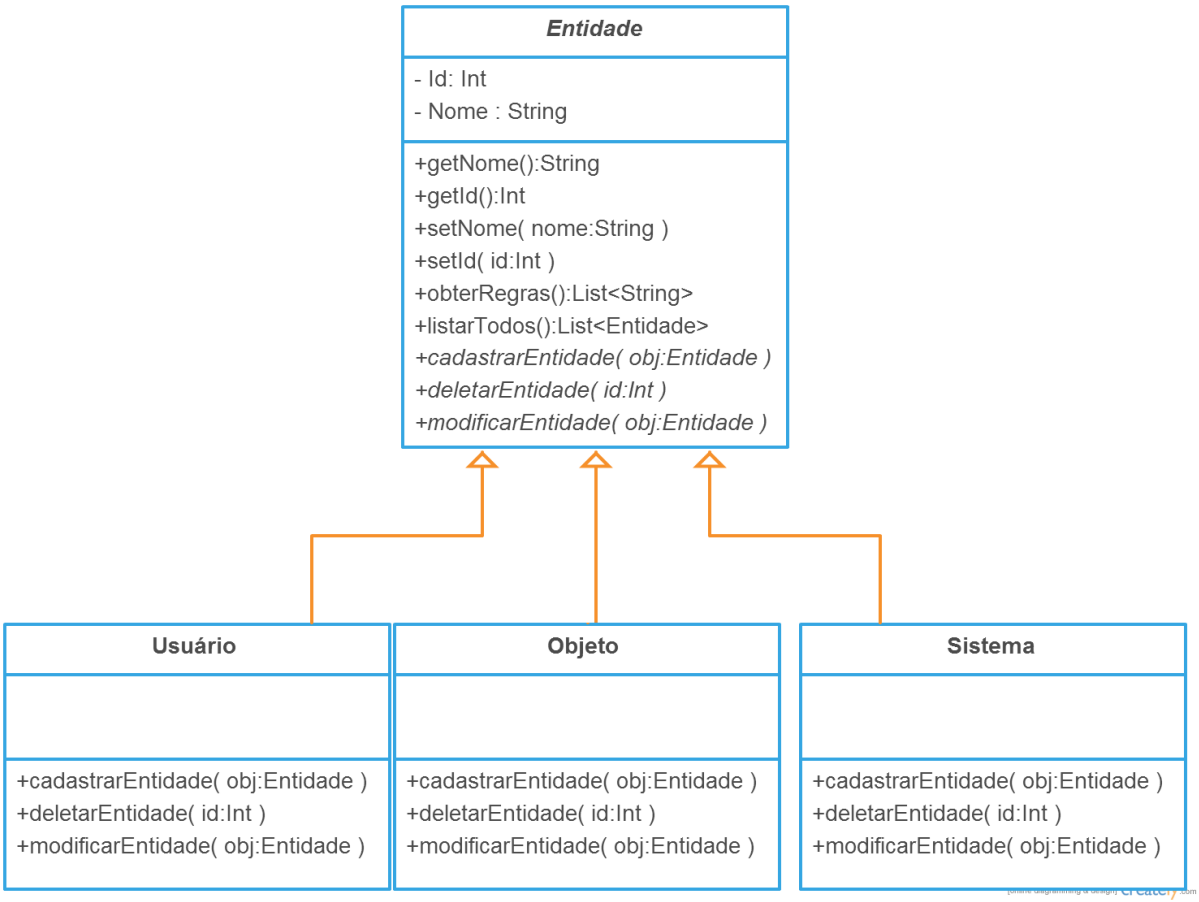

Figura 5.6: Diagrama de Classes das Entidades.

Atualmente, são consideradas apenas três entidades (usuário, objeto e sistema) na plataforma BioNimbuZ. Porém, novas entidades podem ser acrescentadas posteriormente, sendo necessário apenas criar uma classe da nova entidade, estender a classe entidade e implementar os métodos de gerenciamento necessários.

5.3 Integração com o BioNimbuZ

Para implementar o controle de acesso ABAC no BioNimbuZ, algumas modificações nas camadas de aplicação e de núcleo da plataforma tiveram que ser feitas. A primeira modificação feita foi na camada de aplicação com a interface do usuário, que antes permitia o acesso direto dos usuários aos objetos do sistema. Agora, foi adicionado o serviço de segurança que contém a autenticação e a autorização, como apresentado na Figura 5.7, que mostra o funcionamento antes e depois.

Uma outra modificação precisou ser feita no serviço de armazenamento, para que os atributos dos arquivos que foram armazenados pudessem ser coletados e armazenados pelo gerente de atributos.

Outra modificação necessária foi a implantação de um banco de dados para armazenar as informações que o modelo ABAC utiliza. Dentre as informações que devem ser armazenadas estão os nomes dos usuários e as senhas reforçadas, como mostrado na Seção 5.1, para a realização da autenticação. Além disso, é necessário armazenar para a autorização os atributos, os valores dos atributos e as regras de acesso.

Nesse contexto, foi utilizado como sistema de gerenciamento de banco de dados (SGBD) o MySQL [75], por ser livre e ser facilmente manuseável na linguagem de programação java. As principais tabelas no banco de dados que representam os repositórios do ABAC são descritas a seguir.

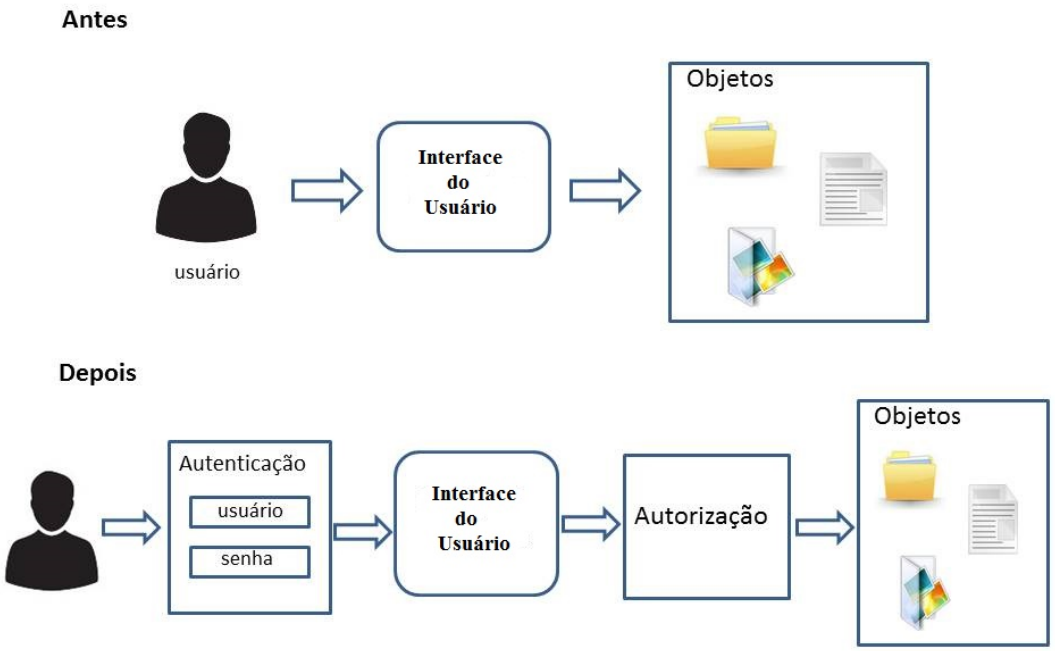

Figura 5.7: Modificação da Interface do Usuário.

5.3.1 Repositório de Atributos

Os atributos de usuários e os atributos de objetos são armazenados em tabelas específicas para cada um deles, sendo que as tabelas possuem apenas os campos do identificador e o nome dos atributos. Os atributos de sistema não possuem seus valores armazenados em uma tabela específica no banco, pois são condições de ambiente, como data e horário, e possuem valores que podem se modificar constantemente, e são armazenados em variáveis globais quando requisitados.

Cada usuário possui um valor para cada atributo de usuário, e da mesma forma cada objeto possui um valor para os atributos de objetos. No banco de dados, para representar essa relação, foi criada uma nova tabela que contém o identificador do usuário, o identificador do atributo e o valor correspondente do atributo daquele usuário, estas tabelas estão representadas na Figura 5.8.

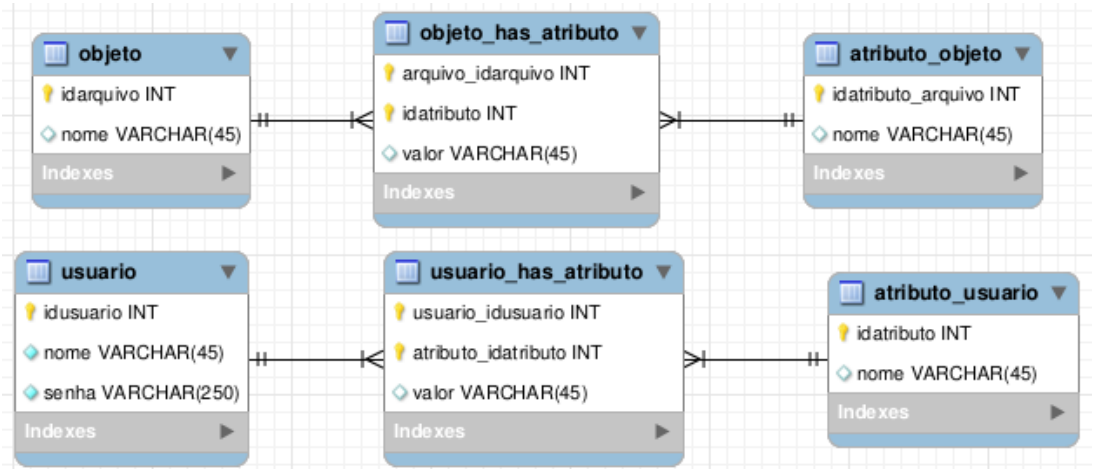

Figura 5.8: Tabelas das Entidades e dos Atributos.

Os nomes dos objetos, dos usuários e dos atributos são armazenados em forma de texto, enquanto que os identificadores são números inteiros. É importante ressaltar que no campo da senha, na tabela que contém as informações dos usuários, não é armazenado o valor da senha diretamente, mas sim da senha reforçada através do algoritmo PBKDF2 [73].

5.3.2 Repositório de Regras

As regras do ABAC são expressões lógicas que envolvem os atributos e seus valores, e resultam na decisão de autorização de acesso. Estas regras possuem um padrão que foi apresentado na Subseção 5.2.1. Assim sendo, foi criada uma tabela no banco de dados para armazenar a regra escrita neste padrão. É preciso porém, converter a regra escrita neste padrão para uma consulta ao banco de dados, necessária para a tomada de decisão. Dessa forma, foi criada uma tabela que armazena o valor da regra já convertida para uma consulta e também o identificador da regra na qual ela está relacionada, ambas as tabelas estão representadas na Figura 5.9.

Existem várias formas de converter as regras em uma consulta. A regra

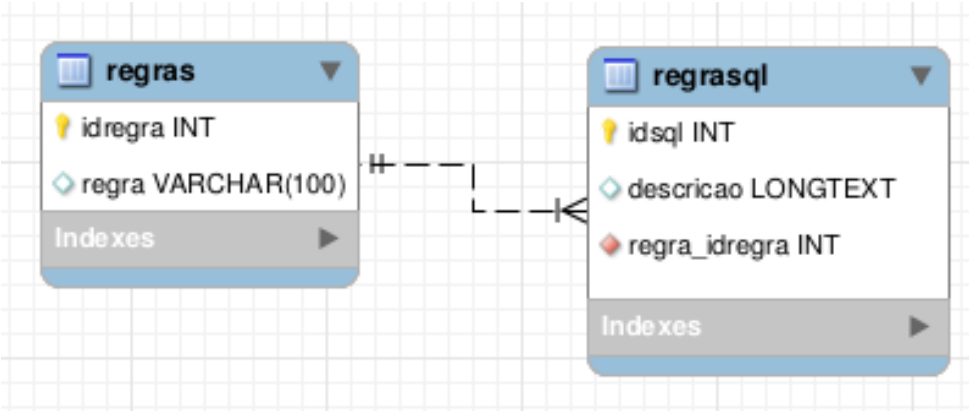

Figura 5.9: Tabelas do Repositório de Regras.

$$u.cargo = gerente$$

por exemplo, pode ser convertida para a consulta em SQL, que retorna o nome de todos os usuários que possuem o valor do atributo *cargo* igual a *gerente*, conforme indicado abaixo.

```
SELECT u.nome FROM usuario AS u,usuario_has_atributo AS f,
atributo_usuario AS a WHERE u.idusuario = f.usuario_idusuario
AND a.idatributo = f.atributo_idatributo
AND a.nome = 'cargo' AND valor = 'gerente';
```

5.3.3 Otimizações

Para garantir a eficiência do modelo algumas otimizações foram feitas, principalmente, na avaliação das regras e na geração da lista de objetos permitidos para cada usuário.

Nas avaliações das regras foram tomadas medidas para que em alguns casos não seja preciso avaliar por completo a expressão. Assim, em expressões que contém apenas o operador lógico OR, sempre que uma das partes da regra for verdadeira, a expressão toda é considerada verdadeira, evitando assim que o restante da regra seja avaliada de forma desnecessária. Da mesma forma, em expressões contendo apenas o operador lógico AND, sempre que uma das partes da expressão é falsa, toda a expressão é considerada imediatamente falsa.

Quanto a lista de objetos que o usuário pode acessar, ela é gerada uma vez e é mantida em memória para futuras consultas. Ela continua a mesma enquanto não houver alguma ação que a modifique, como por exemplo a introdução de uma nova regra ou a alteração do valor de algum atributo, o que resulta na necessidade de gerar novamente essa tabela e carregá-la em memória.

5.3.4 Funcionamento

Após a criação das classes necessárias para o modelo de controle de acesso ABAC, a criação dos repositórios e a integração, o funcionamento da plataforma BioNimbuZ foi alterado. A Figura 5.10 apresenta a nova organização interna.

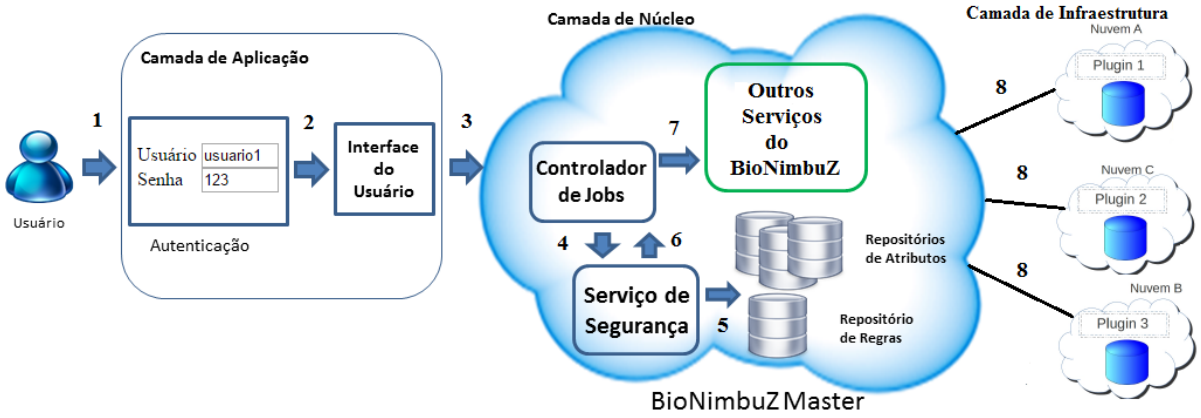

Figura 5.10: Nova Organização Interna do BioNimbuZ.

Dessa forma, o primeiro contato do usuário é com a camada de aplicação (passo 1). Nesta camada, o usuário deve realizar a autenticação, para só então ter acesso a interface do usuário e poder acessar os serviços do BioNimbuZ (passo 2). Todas as requisições que são feitas chegam ao controlador de *jobs* (passo 3), que pertence a camada de núcleo e atua como o ponto de aplicação de políticas (*Policy Enforcement Point* - PEP), presente na arquitetura da Figura 4.6. O controlador de *jobs* porém, não toma decisões de autorização, ele apenas aplica as decisões que foram tomadas pelo ponto de decisão de políticas (*Policy Decision Point* - PDP), que se encontra no serviço de segurança (passo 4). Para isto, o serviço de segurança consulta os repositórios de regras e de atributos (passo 5), e comunica a decisão para o controlador de *jobs* (passo 6), que aplica a decisão, liberando ou não o acesso aos outros serviços da plataforma BioNimbuZ (passo 7).

Assim, como pode ser visto, o serviço de segurança implementado neste trabalho garante que o acesso aos recursos da nuvem federada seja dado somente aos usuários autorizados.

Dessa forma, com o modelo implementado foram realizados testes, de integridade e eficiência, os quais são apresentados no próximo capítulo.

Capítulo 6

Resultados

Neste capítulo serão apresentados os resultados obtidos nos testes de segurança e de desempenho da política de segurança proposta neste trabalho. Para isso, serão mostrados o ambiente no qual os testes foram realizados e uma descrição detalhada dos atributos, das regras e dos demais elementos utilizados.

6.1 Experimentos

Os testes realizados neste trabalho tinham dois objetivos. O primeiro objetivo foi validar a política de controle de acesso baseada em atributos implementada neste trabalho, ou seja, determinar se ela está de fato restringindo o acesso aos recursos somente aos usuários autorizados. E o segundo objetivo é determinar quanto tempo o algoritmo demora para tomar uma decisão de autorização, visto que em ambientes de nuvens federadas o tempo é um requisito importante. Para isso, os testes foram divididos em três grupos, os quais são apresentados nas Subseções 6.1.2, 6.1.3 e 6.1.4.

6.1.1 Ambiente de Execução

Para a realização dos testes foram utilizadas duas nuvens na federação, as quais tinham as seguintes configurações:

- Uma nuvem privada, localizada na Universidade de Brasília (UnB), composta por 3 computadores com a configuração de 8 GB de memória primária, processador Intel(R) Core(TM) i7-3770 com frequência de 1.6 GHz, com oito núcleos e 2 TB de memória secundária;
- Uma nuvem pública na Amazon [27], com três máquinas com processador Intel Xeon E5-2676 com frequência 2,4 GHz, com quatro núcleos, e com 8 GB de memória primária e 100 GB de memória secundária.

Nestas nuvens estavam executando o BioNimbuZ *master*, o servidor Zookeeper e o sistema gerenciador de banco de dados MySQL, contendo o banco de dados utilizado pelo BioNimbuZ. Essa configuração da nuvem utilizada está representada na Figura 6.1.

É importante ressaltar que um dos objetivos é determinar o tempo que o algoritmo demora para tomar uma decisão de autorização, e para isto, dois testes foram realizados.

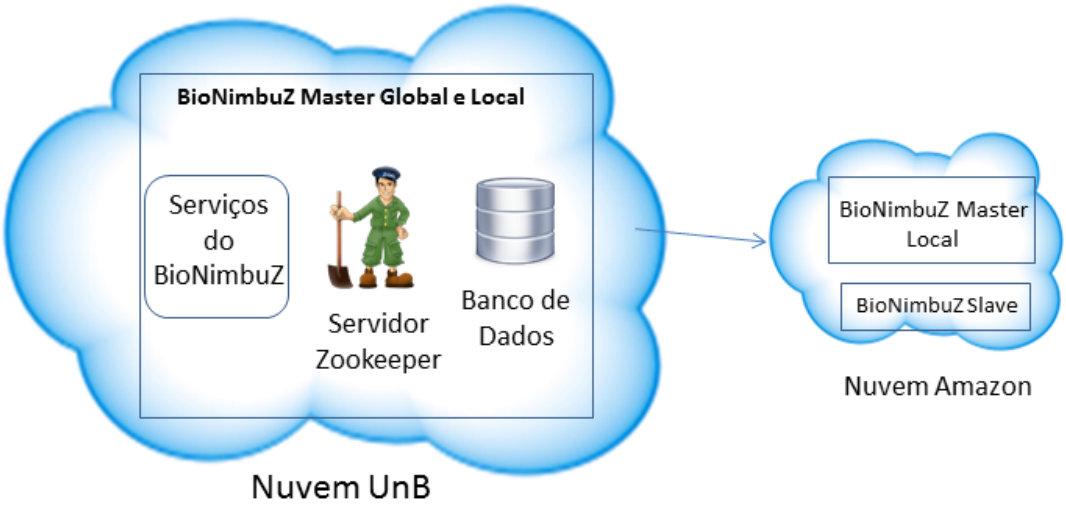

Figura 6.1: Configuração da Nuvem da UnB.

O primeiro contendo apenas a nuvem da UnB, com o BioNimbuZ *master* executando nela. E o segundo contendo as duas nuvens, e o BioNimbuZ *master* global executando na nuvem da Amazon. Dessa forma, os tempos de transmissão da rede não foram considerados no primeiro teste, apenas no segundo.

6.1.2 Validação do Controle de Acesso

Para determinar se o controle de acesso está realizando a sua função de garantir que os recursos sejam acessados apenas por usuários autorizados, foi verificado se as regras de autorização estavam sendo cumpridas. Para isso, foram criados arquivos de testes com valores de atributos escolhidos aleatoriamente e foram criadas regras de acesso que envolviam tais atributos. Além disso, foram criados usuários que possuiam valores de atributos variados, que tentavam acessar os diversos arquivos criados anteriormente.

Nestes testes, os usuários acessavam os arquivos utilizando o comando para listar todos os arquivos autorizados para aquele usuário, disponível na interface do usuário, porém o funcionamento é o mesmo para os outros comandos disponíveis, como o de *download* e o de *upload*, por exemplo.

Na Tabela 6.1 são apresentados cinco usuários, os atributos que foram considerados e os valores dos atributos para cada usuário. Os valores foram distribuídos aleatoriamente e representam usuários criados exclusivamente para os testes. É importante lembrar que com a flexibilidade do modelo é possível criar diversos outros atributos.

Além disso, foram criados 100 arquivos com diferentes valores de atributos. Os atributos considerados e seus respectivos valores foram:

- Tamanho: 10KB, 20KB, 50KB e 100KB;
- Proprietário: usuario1, usuario2, usuario3, usuario4 e usuario5;
- Extensão: txt, html, java, pdf e exe;
- Classificação: aberto, restrito.

Tabela 6.1: Usuários Criados para os Testes.

Usuários	Universidade	Curso	Laboratório
`usuario1`	UnB	Computação	lab3
`usuario2`	USP	Biologia	lab2
`usuario3`	UnB	Física	lab1
`usuario4`	USP	Física	lab3
`usuario5`	UnB	Computação	lab3

O modelo ABAC utiliza ainda, além dos atributos de usuários e de objetos, os atributos de sistema, e neste contexto, foram utilizados a data e a hora local.

Assim, após criar os usuários e os arquivos com seus respectivos valores de atributos, foram criadas regras de autorização. As regras são expressões que envolvem os valores dos atributos e resultam em uma decisão de autorização. As seguintes regras foram consideradas:

- Todos os usuários podem acessar arquivos classificados como abertos;
- Todos os usuários podem acessar arquivos que eles mesmos são os proprietários;
- Usuários do mesmo laboratório, do mesmo curso e da mesma universidade podem ver arquivos um do outro;
- Arquivos secretos só podem ser acessados até as 18:00 hrs;
- Arquivos maiores do que 20KB, ou arquivos com extensão *exe* só podem ser acessados depois das 18:00.

As regras foram implantadas aos poucos nos testes, ou seja, primeiro foram feitos testes apenas com uma regra, depois com duas regras e assim por diante, até que todas as regras fossem testadas. Em cada uma destas etapas, todos os usuários utilizavam o comando para listar os objetos que tinha permissão de acesso, e a saída desses comandos foi redirecionada para arquivos.

Os arquivos com as saídas foram analisados e mostraram que as regras de autorização foram cumpridas, ou seja, os usuários só conseguiram acessar os arquivos que estavam autorizados pelas regras previamente definidas. Assim sendo, o primeiro objetivo dos testes foi cumprido.

6.1.3 Tempo de Resposta Na Nuvem Local

O objetivo do teste apresentado nesta seção foi descobrir o tempo que o controle de acesso leva para tomar uma decisão de autorização, pois não adianta o sistema controlar o acesso dos usuários de forma correta se o tempo de resposta for muito longo, tornando a utilização do modelo proposto em nuvens federadas muito cara, pois o custo é determinado pelo tempo de utilização dos recursos. Assim sendo, foram realizados testes para determinar o tempo que o ABAC implementado leva para tomar a decisão de acesso em diferentes condições.

Para a realização dos testes de tempo de resposta foram utilizados os mesmos atributos definidos para o teste do controle de acesso, apresentados na Seção 6.1.2.

As regras foram modificadas para que utilizassem apenas um atributo por vez, depois três atributos por vez, e por fim utilizassem os cinco atributos. Além disso, foram utilizados quatro conjuntos de arquivos. O primeiro conjunto continha cem arquivos, o segundo continha mil arquivos, o terceiro continha dez mil arquivos e no último conjunto haviam cem mil arquivos. Os atributos que foram atribuídos a cada arquivo não seguiram um padrão, ou seja, tentou-se realizar o maior número possível de combinações diferentes entre os valores dos atributos, para que os resultados não tivessem um viés pré-definido.

No gráfico mostrado na Figura 6.2 são apresentados o tempo médio de resposta em milissegundos, de dez execuções, para as diferentes quantidades de arquivos, utilizando-se apenas uma regra de acesso e diferentes quantidades de atributos.

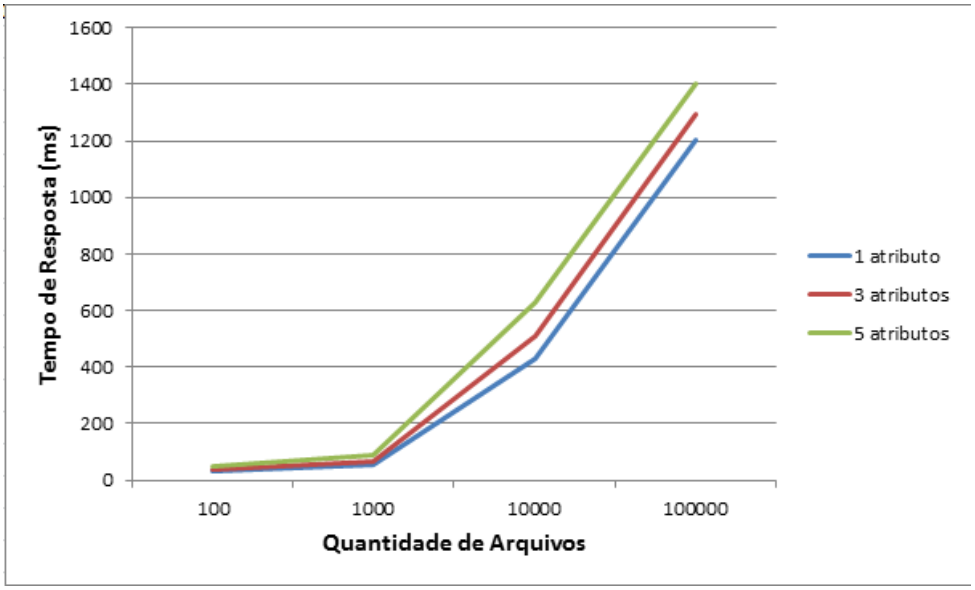

Figura 6.2: Tempo de Resposta Utilizando Uma Regra.

Nesse teste, o tempo é praticamente o mesmo nas três quantidade de atributos, quando se tem até 1000 arquivos, e isto gera uma distorção no gráfico, pois é muito rápido em relação aos dois conjuntos de arquivos seguintes. Após isso, existe um crescimento não linear do tempo, e isso se deve à forma como a consulta é feita no banco de dados, que envolve a quantidade de dados da tabela dos arquivos e nas tabelas dos atributos. É possível perceber que os tempos de resposta são bem parecidos para as diferentes quantidades de atributos, pois apenas uma regra está sendo considerada. Neste caso, a quantidade de atributos não interfere muito no tempo de resposta, situação que não se repete quando se utilizam três regras de acesso, como mostrada na Figura 6.3.

Quando são utilizadas três regras de acesso, o número de atributos interfere consideravelmente no tempo total, chegando a mais de dois segundos para cinco atributos. Nesse caso, é possível notar uma diferença maior nos tempos entre um e três atributos, do que entre três e cinco atributos, e a principal hipótese para que isto tenha acontecido é a de que as otimizações que foram mostradas na Subseção 5.3.3 tenham causado esta distorção, pois algumas regras são avaliadas por completo enquanto que outras podem ser decididas antes do fim.

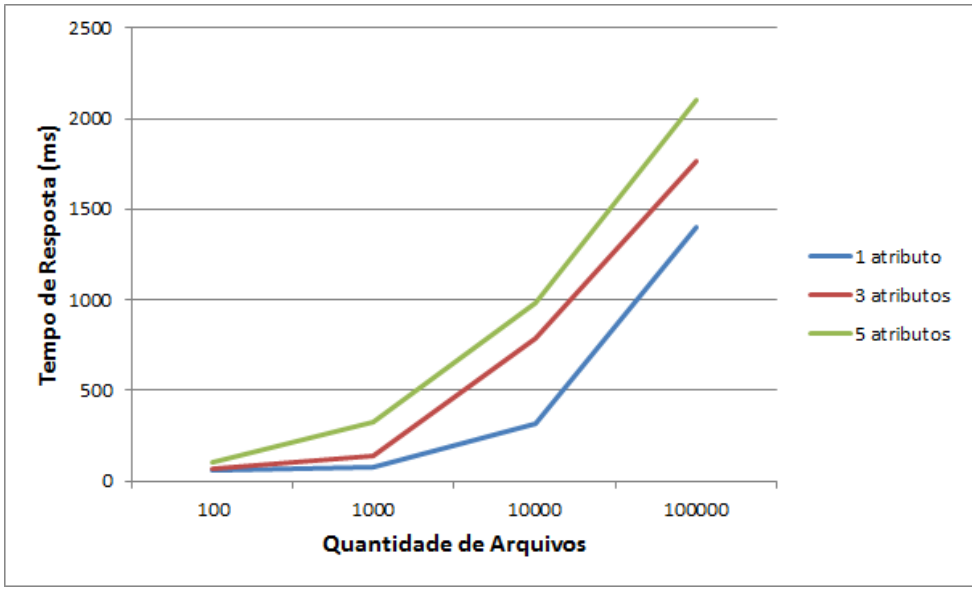

Figura 6.3: Tempo de Resposta Utilizando Três Regras.

A diferença nos tempos aumenta também quando são utilizadas cinco regras de acesso, que é apresentado na Figura 6.4. As diferenças de tempo quando se utilizam cinco atributos com as cinco regras é muito perceptível e maior do que nos testes anteriores, e neste caso, além das consultas ao banco de dados, é possível que as estruturas de dados utilizadas para armazenar os resultados possam interferir nos tempos.

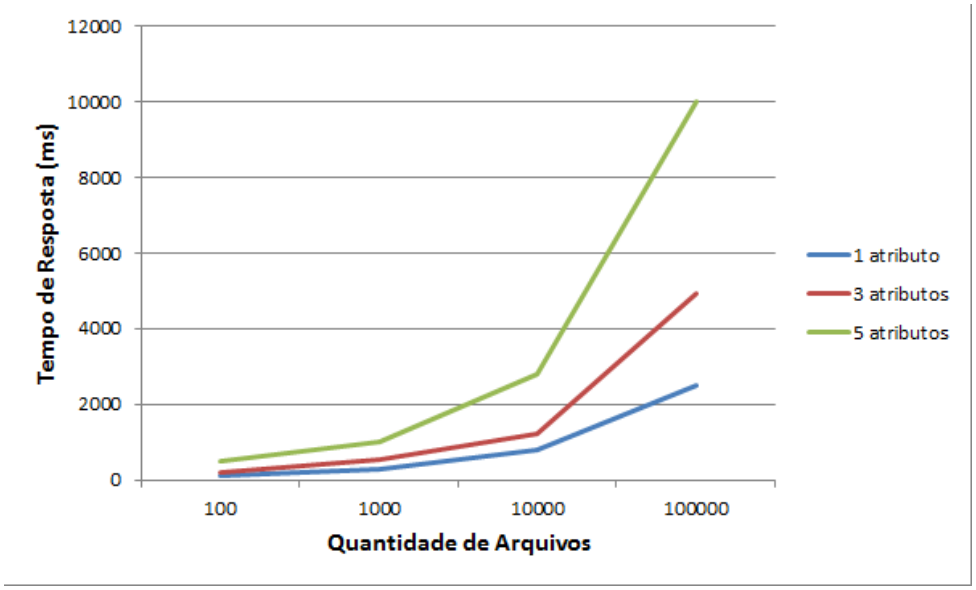

Figura 6.4: Tempo de Resposta Utilizando Cinco Regras.

Em todos os casos, como esperado, o tempo sempre foi menor quando apenas um atributo está sendo utilizado, e aumenta a medida em que se aumenta a quantidade de atributos. Isto acontece devido ao aumento da quantidade de informação que o ABAC deve considerar para tomar a decisão de autorização.

Estes tempos porém, não são percebidos de imediato pelo usuário, visto que em uma consulta aos objetos que ele está autorizado a acessar, os resultados aparecerão em páginas com quantidade bem menores do que cem mil arquivos, sendo geralmente cinquenta arquivos no máximo por vez, o que utiliza um tempo menor do que 0,5 segundo de espera. Neste cenário, os resultados apresentados para os tempos de resposta do controle de acesso ABAC se mostraram viáveis para a utilização em nuvens federadas.

6.1.4 Tempo de Resposta na Federação

Nos testes apresentados anteriormente, os tempos de transmissão da rede foram desconsiderados, para que se soubesse o tempo gasto pelo algoritmo para tomar uma decisão de autorização. Esses tempos porém, não refletem totalmente um caso de uso real, pois é preciso, muitas vezes, utilizar mais de uma nuvem para se obter uma federação. Assim sendo, foram realizados testes nas mesmas condições apresentadas na Subseção 6.1.3, mas com o BioNimbuZ *master* executando na nuvem da Amazon. O primeiro teste utilizou apenas uma regra e os resultados obtidos estão representados na Figura 6.5.

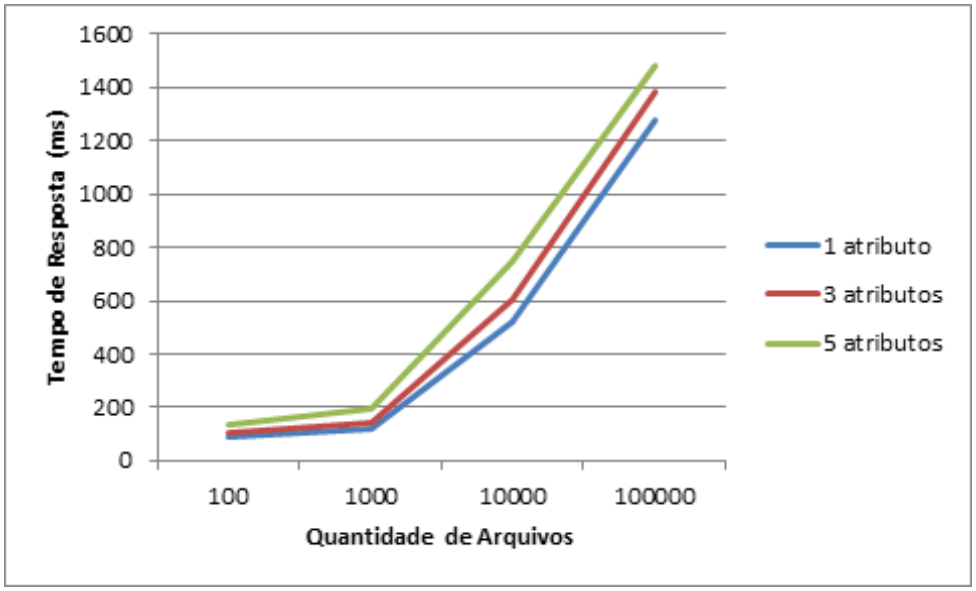

Figura 6.5: Tempo de Resposta Na Federação Utilizando Uma Regra.

Os aspectos dos resultados são bem parecidos com aqueles obtidos na Subseção 6.1.3. Nos dois primeiros conjuntos de arquivos o tempo de resposta é bem parecido, mesmo utilizando quantidades de atributos diferentes. As variações nos tempos são mais perceptíveis nos dois últimos conjuntos, quando o tempo cresce de forma não linear. E como o previsto, os tempos de resposta aumentaram um pouco em relação aos testes em uma única nuvem, e isso se deve ao tempo de transmissão na rede.

Os testes envolvendo três regras de acesso foram os que obtiveram uma maior diferença nos resultados, como o mostrado na Figura 6.6. Neste teste, a linha que representa a utilização de cinco atributos encontra-se bem mais distante da linha que representa três atributos. Esse resultado se deve a soma de dois fatores principais. O primeiro refere-se às otimizações que foram mostradas na Subseção 5.3.3 e o segundo é a utilização de

uma nuvem que possui máquinas com configurações diferentes das apresentadas nos testes anteriores.

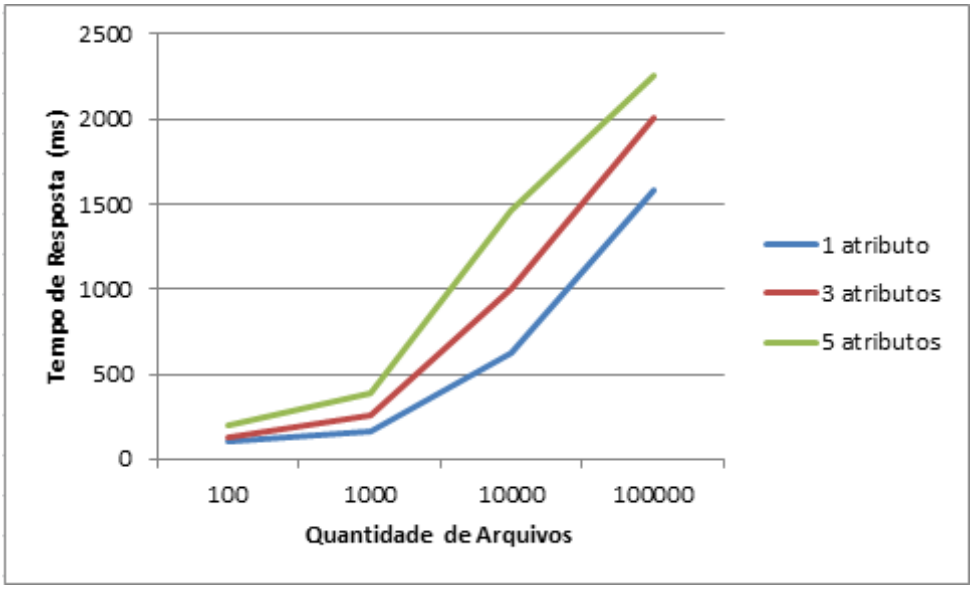

Figura 6.6: Tempo de Resposta Na Federação Utilizando Três Regras.

Também foram testados os tempos de resposta utilizando-se cinco regras de acesso e variando entre um, três e cinco atributos, e os resultados obtidos estão apresentados na Figura 6.7. Assim como nos testes utilizando uma e três regras, a principal diferença destes resultados para os resultados obtidos com apenas uma nuvem é um aumento no tempo de resposta, mas que de uma forma geral se mostra compatível com o resultado obtido anteriormente.

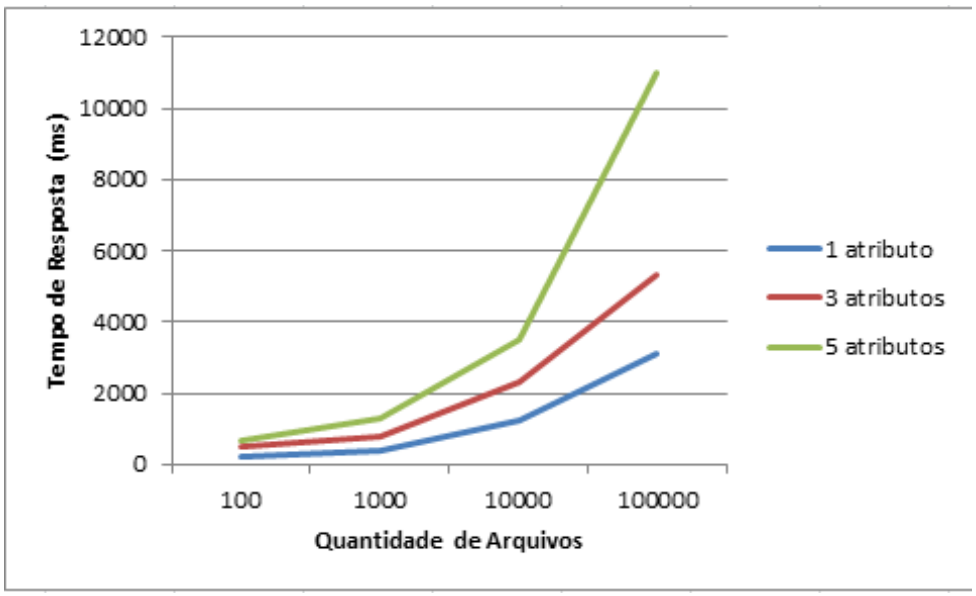

Figura 6.7: Tempo de Resposta Na Federação Utilizando Cinco Regras.

6.1.5 Considerações Finais

Neste capítulo foram apresentados os testes realizados no serviço de segurança da plataforma BioNimbuZ, que utiliza o modelo de controle de acesso baseado em atributos, proposto neste trabalho. Os testes tinham dois objetivos, sendo o primeiro o de determinar se o controle de acesso está restringindo corretamente o acesso aos objetos, e o segundo objetivava determinar o tempo que leva para que sejam tomadas as decisões de autorização.

Assim sendo, os dois objetivos foram atingidos neste trabalho, pois foi mostrado que o modelo de controle de acesso baseado em atributos, implementado no BioNimbuZ, está de fato realizando o que deveria, que é garantir que os recursos sejam acessados apenas por usuários autorizados, e ainda executando em tempo compatível com o que se espera de uma plataforma de nuvem federada.

Capítulo 7

Conclusão e Trabalhos Futuros

Neste trabalho foi proposto um modelo de controle de acesso para a plataforma BioNimbuZ. O modelo proposto foi baseado no ABAC, que utiliza as características, chamadas de atributos, dos atores envolvidos para tomar a decisão de autorização.

O controle de acesso foi a primeira implementação de um serviço de segurança no BioNimbuZ, e atua principalmente na confidencialidade, garantindo que apenas usuários com permissão possam acessar os recursos da plataforma. Para isto, o modelo proposto neste trabalho leva em consideração atributos de usuário, atributos de objetos e atributos do sistema, que são relacionados por meio de regras de acesso, as quais decidem se a autorização deve ser concedida ou não.

Na integração do controle de acesso ao BioNimbuZ foram feitas algumas alterações na versão atual da plataforma, principalmente no serviço de interação com o usuário e no serviço de armazenamento. Se antes qualquer usuário tinha total acesso aos serviços e recursos oferecidos na plataforma, agora é preciso realizar uma autenticação, o que já limita o acesso dos usuários, e para cada recurso que se queira acessar é preciso ser aprovado pelo processo de autorização. Além disso, foi preciso adicionar um banco de dados à plataforma, para que fosse possível armazenar os valores dos atributos utilizados no ABAC.

Os testes realizados mostraram que o modelo ABAC está, de fato, garantindo que os recursos só estejam sendo acessados por usuários que tenham permissão para isso e, além disso, mostraram que o tempo gasto pelo modelo para chegar a uma decisão de autorização não o torna inviável para utilização em nuvens federadas.

Como trabalhos futuros propõe-se a utilização de um banco de dados distribuído, para que a autorização e a autenticação possam ser feitas nos BioNimbuZ *masters* locais, e não apenas no *master* global como é feita atualmente. Deve ser feito também um estudo para verificar a viabilidade da implementação de algum outro padrão de autenticação e autorização, como por exemplo o OpenID [76], para que a confidencialidade continue sendo preservada.

Além disso, propõe-se a implementação de outros mecanismos de segurança, como a verificação de integridade dos arquivos na federação e a criptografia dos dados armazenados. Pode ser utilizado, por exemplo, uma verificação de *hashes*, de forma a descobrir se os arquivos que estão armazenados na federação estão integros.

Referências

[1] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009. viii, 1, 7, 9, 14, 15

[2] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008. vii, 1, 9, 11, 13, 14

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical report, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, 2009. 1, 9

[4] Antonio Celesti, Francesco Tusa, Massimo Villari, and Antonio Puliafito. How to enhance cloud architectures to enable cross-federation. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 337–345. IEEE, 2010. vii, 1, 16, 17, 18

[5] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo Neves Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Algorithms and architectures for parallel processing*, pages 13–31. Springer, 2010. vii, 1, 16, 17, 18, 19

[6] Hugo Vasconcelos Saldanha. Bionimbus: uma arquitetura de federação de nuvens computacionais híbrida para a execução de workflows de bioinformática. Master's thesis, Departamento de Ciência da Computação, Universidade de Brasília, 2012. 1, 16, 20

[7] Karin Bernsmed, Martin Gilje Jaatun, Per Hakon Meland, and Astrid Undheim. Thunder in the clouds: Security challenges and solutions for federated clouds. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 113–120. IEEE, 2012. 2

[8] Subashini Subashini and Veeraruna Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications*, 34(1):1–11, 2011. 2

[9] Joseph Donald Sloan. *High performance Linux clusters with OSCAR, Rocks, openMosix, and MPI*. Rodopi, 2009. vii, 4, 6, 7

[10] Gordon Edward Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998. 4

[11] Charles Mann. The end of Moore's law. *Technology Review*, 103(3):42–48, 2000. 4

[12] Nilton Alberto de Castilho Lages and José Marcos Silva Nogueira. *Introdução aos sistemas distribuídos*. Ed. da Unicamp/Papirus, 1986. 4, 5, 6, 13

[13] Marcos Pitanga. *Construindo supercomputadores com Linux*. Brasport, 2004. 4

[14] George Coulouris, Jean Dollimore, and Tim Kindberg. *Sistemas distribuidos*. Addison Wesley, 2001. 5

[15] Maarten Van Steen and Andrew Tanenbaum. *Distributed Systems: Principles and Paradigms*. Prentice Hall; 2 edition, 2006. 5, 21

[16] Randy Otte, Paul Patrick, and Mark Roy. *Understanding CORBA (Common Object Request Broker Architecture)*. Prentice-Hall, Inc., 1995. 6

[17] Oracle. Java Platform, Enterprise Edition. `http://www.oracle.com/technetwork/java/javaee/overview/index.html`, 1999. Acessado online em 25 de maio de 2015. 6

[18] Gregory Francis Pfister. *In search of clusters*. Prentice-Hall, Inc., 1998. 6

[19] Naidila Sadashiv and Saridi Madhava Dilip Kumar. Cluster, grid and cloud computing: A detailed comparison. In *Computer Science Education (ICCSE), 2011 6th International Conference on*, pages 477–482, Aug 2011. 6

[20] National Science Foundation. Open science grid. `http://www.opensciencegrid.org/`, 2013. Acessado online em 27 de abril de 2015. 7

[21] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001. vii, 8

[22] Luis Miguel Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008. vii, 9, 10

[23] Peter Mell and Tim Grance. The NIST definition of cloud computing. *National Institute of Standards and Technology*, 53(6):50, 2009. 9

[24] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm. What's inside the cloud? an architectural map of the cloud landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 23–31. IEEE Computer Society, 2009. 10

[25] Lamia Youseff, Maria Butrico, and Dilma Da Silva. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. IEEE, 2008. 10

[26] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A taxonomy and survey of cloud computing systems. In *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*, pages 44–51. Ieee, 2009. 11, 12

[27] Amazon Web Services LLC. Amazon elastic compute cloud (ec2). http://aws.amazon.com/pt/ec2/, 2015. Acessado online em 27 de abril de 2015. 11, 26, 51

[28] Amazon Web Services LLC. Amazon Simple Storage Service (Amazon S3). http://aws.amazon.com/s3/, 2015. Acessado online em 29 de maio de 2015. 11, 26

[29] Google. Google app engine. https://developers.google.com/appengine/?hl=pt-br, 2012. Acessado online em 27 de abril de 2015. 12

[30] Google. Google docs. https://docs.google.com/?hl=pt-BR, 2012. Acessado online em 27 de abril de 2015. 12

[31] Bill Nowicki. NFS: Network File System Protocol Specification , Request For Coments (RFC 1094). http://tools.ietf.org/pdf/rfc1094.pdf, 1989. Acessado online em 29 de maio de 2015. 14

[32] Thomas Bittman. The evolution of the cloud computing market. *Gartner Blog Network, http://blogs. gartner. com/thomas bittman/2008/11/03/theevolution-of-the-cloud-computing-market*, 2008. Acessado online em 20 de abril de 2015. 15

[33] Breno Rodrigues Moura and Deric Lima Bacelar. Política para armazenamento de arquivos no zoonimbus, 2013. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 20, 21, 25

[34] Gabriel Silva Souza de Oliveira. Acosched: um escalonador para o ambiente de nuvem federada Zoonimbus, 2013. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 20, 21, 24

[35] Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *Networking, IEEE/ACM Transactions on*, 11(1):17–32, 2003. 20

[36] Raj Srinivasan. RPC: Remote Procedure Call protocol Specification version 2 , Request For Coments (RFC 1831). http://tools.ietf.org/pdf/rfc1831.pdf, 1995. Acessado online em 04 de junho de 2015. 21

[37] The Apache Software Fundation. Apache zookeeper. http://zookeeper.apache.org/, 2015. Acessado online em 30 de março de 2015. vii, 21, 22

[38] The Apache Software Fundation. Apache fundation. http://apache.org/, 2015. Acessado online em 30 de março de 2015. 21

[39] The Apache Software Fundation. Apache avro. https://avro.apache.org/, 2015. Acessado online em 05 de junho de 2015. 21

[40] Douglas Crockford. The application/json Media Type for Javascript Object Notation (JSON) , Request For Coments (RFC 4627). http://tools.ietf.org/pdf/rfc4627.pdf, 1995. Acessado online em 05 de junho de 2015. 21

[41] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext Transfer Protocol–http/1.1 , Request For Coments (RFC 2616). http://tools.ietf.org/pdf/rfc2616.pdf, 1999. Acessado online em 05 de junho de 2015. 21

[42] The Apache Software Fundation. Apache avro documentation. http://avro.apache.org/docs/1.7.7/, 2015. Acessado online em 05 de junho de 2015. 21

[43] Barbara Guttman and Edward Roback. *An introduction to computer security: the NIST handbook*. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, 1995. 28

[44] Matt Bishop. *Introduction to computer security*. Addison-Wesley Professional, 2004. 28, 29

[45] Dimitrios Zissis and Dimitrios Lekkas. Addressing cloud computing security issues. *Future Generation computer systems*, 28(3):583–592, 2012. 29

[46] Jorge Rady de Almeida Junior. *Segurança em Sistemas Críticos e em Sistemas de Informação–Um estudo Comparativo*. PhD thesis, Tese de Livre Docência. Escola Politécnica da USP, 2003. 29

[47] Pierre Bieber and Frederic Cuppens. Expression of confidentiality policies with deontic logic. 1992. 30

[48] Douglas Robert Stinson. *Cryptography: theory and practice*. CRC press, 2005. 30

[49] David Elliott Bell and Leonard Joseph La Padula. Secure computer system: Unified exposition and multics interpretation. Technical report, DTIC Document, 1976. 30

[50] Butler Lampson. Protection. In *Proc. 5th Princeton Conf. on Information Sciences and Systems*, pages 18–24. Princeton, 1971. 30, 32

[51] André Thiago Souza da Silva and Hugo Vasconcelos Saldanha. Controle de acesso baseado em papéis na informatização de processos judiciais, 2006. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 30, 31, 34

[52] Bassem El Zant, Nahla El Zant, Nabil El Kadhi, and Maurice Gagnaire. Security of cloud federation. In *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*, pages 335–339. IEEE, 2013. 30

[53] David Ferraiolo, Ravi Sandhu, Serban Gavrila, Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001. 31

[54] David Ferraiolo, Rick Kuhn, and Ravi Sandhu. Rbac standard rationale: Comments on"a critique of the ansi standard on role-based access control". *Security & Privacy, IEEE*, 5(6):51–53, 2007. 31, 35

[55] Mairon de Araújo Belchior. Modelo de controle de acesso no projeto de aplicações na web semântica. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2011. vii, 31, 35

[56] Ravi Sandhu and Pierangela Samarati. Authentication, access control, and audit. *ACM Computing Surveys (CSUR)*, 28(1):241–243, 1996. 31, 35

[57] Luiz Otávio Botelho Lento, Joni da Silva Fraga, and Lau Cheuk Lung. A nova geração de modelos de controle de acesso em sistemas computacionais. *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 151–201, 2006. 32, 34

[58] David Elliott Bell and Leonard Joseph LaPadula. Secure computer systems: Mathematical foundations. Technical report, DTIC Document, 1973. 33, 34

[59] Romuald Thion. *Cyber warfare and cyber terrorism*, chapter Access Control Models. IGI Global, 2007. 34

[60] David Dana Clark and David R Wilson. A comparison of commercial and military computer security policies. In *2012 IEEE Symposium on Security and Privacy*, pages 184–184. IEEE Computer Society, 1987. 34

[61] Terry Mayfield, Eric Roskos, Stephen Welke, John Boone, and Catherine McDonald. Integrity in automated information systems. Technical report, Technical Report 79-91, National Computer Security Center, 1991. 35

[62] Ravi Sandhu and Pierangela Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40–48, 1994. vii, 35, 36

[63] Vincent Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (ABAC) definition and considerations. *NIST Special Publication*, 800:162, 2014. vii, 36, 38

[64] Eric Yuan and Jin Tong. Attributed based access control (abac) for web services. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005. vii, 36, 37, 39

[65] Robert Morris and Ken Thompson. Password security: A case history. *Communications of the ACM*, 22(11):594–597, 1979. 41

[66] Ronald Linn Rivest. The MD5 message-digest algorithm, Request For Coments (RFC 1321). `http://tools.ietf.org/pdf/rfc1321.pdf`, 1992. Acessado online em 05 de maio de 2015. 41

[67] Department Of Commerce Washington DC. Secure hash standard (FIPS PUB 180-1). *NIST, National Technical Information Service, Springfield, VA, USA*, 1995. 41

[68] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *Proceedings of the 24th annual international conference on Theory and Applications of Cryptographic Techniques*, pages 19–35. Springer-Verlag, 2005. 41

[69] Himanshu Kumar, Sudhanshu Kumar, Remya Joseph, Dhananjay Kumar, Sunil Kumar Shrinarayan Singh, Ajay Kumar, and Praveen Kumar. Rainbow table to crack password using MD5 hashing algorithm. In *Information & Communication Technologies (ICT), 2013 IEEE Conference on*, pages 433–439. IEEE, 2013. 41

[70] Andrew Zonenberg. Distributed hash cracker: A cross-platform GPU-accelerated password recovery system. *Rensselaer Polytechnic Institute*, 27:395–399, 2009. 42

[71] Tomosuke Murakami, Ryuta Kasahara, and Takamichi Saito. An implementation and its evaluation of password cracking tool parallelized on gpgpu. In *Communications and Information Technologies (ISCIT), 2010 International Symposium on*, pages 534–538. IEEE, 2010. 42

[72] Meltem Sönmez Turan, Elaine Barker, William Burr, and Lily Chen. Recommendation for password-based key derivation. *NIST special publication*, 800:132, 2010. 42

[73] Burt Kaliski. PKCS5: Password-based cryptography specification version, Request For Coments (RFC 2898). http://tools.ietf.org/pdf/rfc2898.pdf, 1999. Acessado online em 05 de maio de 2015. 42, 48

[74] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-hashing for message authentication, Request For Coments (RFC 2104). http://tools.ietf.org/pdf/rfc2104.pdf, 1997. Acessado online em 05 de maio de 2015. 42

[75] Oracle Corporation. MySQL. https://www.mysql.com/, 2015. Acessado online em 18 de junho de 2015. 47

[76] OpenID Foundation. OpenID. http://openid.net/, 2015. Acessado online em 23 de junho de 2015. 59