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

BioNimbuZ 2 - Uma Plataforma de Federação de Nuvens em uma Arquitetura Orientada a Microsserviços

Felipe Lopes de Souza Mendes

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

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

Brasília
2018

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

BioNimbuZ 2 - Uma Plataforma de Federação de Nuvens em uma Arquitetura Orientada a Microsserviços

Felipe Lopes de Souza Mendes

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

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

Prof.ª Dr.ª Maria Clícia Castro
IME/UERJ

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

Prof. Dr. Bruno Macchiavello Espinoza
Coordenador do Programa de Pós-graduação em Informática

Brasília, 27 de Setembro de 2018

Dedicatória

Dedico este trabalho a todos os amigos e familiares queridos que nos últimos anos me escutaram pacientemente dizer a frase “não posso porque vou estudar”, sempre me apoiando e me dando forças.

Dedico em especial à minha esposa, que teve que aprender a dividir minha atenção com os estudos e nunca deixou de me apoiar e me incentivar.

Agradecimentos

Antes de tudo, agradeço a oportunidade oferecida pela empresa em que trabalho, de me oferecer todo o suporte possível para que eu pudesse alcançar essa nova etapa profissional em minha vida.

Agradeço verdadeiramente a todos os professores com os quais tive contato, seja por um semestre inteiro ou por poucas conversas, mas que me fizeram crescer não só como um profissional, mas também como pessoa. Cada um com sua personalidade, nos ensinam a cada dia como equilibrar a mente, com a grande quantidade de informações que sabemos que possuem, com a vida pessoal e profissional.

Sou muito grato especialmente à professora Aletéia por ter me aceitado como orientando, e pela qual tenho grande admiração, principalmente, por ter conhecido nos últimos anos a grande pessoa que é.

Agradeço também a todos os meus familiares, que sempre se preocuparam comigo e sentiram minha falta. Os familiares nos lembram constantemente da nossa importância em seus corações.

Agradeço a oportunidade de vida que tive nos últimos anos para conhecer as pessoas que conheci, e para fazer as amizades que fiz. Agradeço aos amigos do grupo de ex-orientandos da professora Aletéia, em especial ao Breno Moura, por sua solicitude ao atender os pedidos de ajuda feitos e por sua amizade.

Não poderia deixar de agradecer ao meu grande amigo Jefferson Gomes pelo apoio e ajuda que sempre me deu. Desses anos eu tiro uma lição importante, que desafios ficam menos difíceis quando se tem amigos ao seu lado.

Por fim, agradeço imensamente à minha querida esposa por escutar meus desabafos e minhas conquistas, me dar o apoio que eu precisei e nunca me deixar desistir dos meus sonhos.

Não espere por uma crise para descobrir o que é importante em sua vida. (Platão)

Resumo

A computação em nuvem é um modelo de computação caracterizado, principalmente, pela capacidade de provisionamento de recursos computacionais sob-demanda e na forma de serviços, com foco na economia de recursos. A federação de nuvens surge como resposta às grandes demandas por poder computacional exigidos por aplicações como, por exemplo, as das áreas da biologia ou da astronomia, e também por aplicações de *workflows* científicos. Nuvens que antes tinham recursos limitados podem agora buscar recursos externos ou, caso possuam recursos que não estejam sendo utilizados, podem alugá-los para aumentarem suas receitas. A abordagem de microsserviços, que tem sido utilizada por grandes empresas, como Netflix e Uber, define serviços que devem possuir pequenas responsabilidades, cada um rodando em seu próprio processo e que se comunicam por meio de mecanismos leves. Ela tem sido atualmente uma abordagem frequentemente utilizada para melhor distribuir sistemas monolíticos. Nesse sentido, a arquitetura apresentada neste trabalho teve como foco prover uma plataforma de federação de nuvens orientada a microsserviços eficiente para a execução de tarefas, e que pudesse integrar novas nuvens de maneira simplificada. Além disso, entre outros benefícios alcançados estão o de criar uma plataforma de federação para execução de aplicações genéricas, de tarifar diretamente seus usuários, de compartilhar credenciais e espaços de armazenamento e de efetuar o *download*/*upload* direto de arquivos para os espaços de armazenamento do usuário. Para validar a arquitetura apresentada, a mesma foi comparada com a plataforma de federação de nuvens BioNimbuZ, uma plataforma de federação de nuvens monolítica para a execução de *workflows* de Bioinformática. Os resultados mostraram uma redução de 70% no tempo para inicializar os componentes da plataforma e uma redução de consumo de recursos de CPU e de memória de 80% e 45%, respectivamente, durante a edição e criação dos *workflows*. Já durante a execução dos *workflows*, a redução do consumo de CPU foi de 93%, e de memória foi de 77%.

Palavras-chave: Nuvem Computacional, Federação de Nuvens, Microsserviços, *Workflows* Científicos, Arquitetura de Nuvens Federadas, BioNimbuZ, BioNimbuZ 2.

Abstract

Cloud computing is a computing model characterized primarily by the ability to provide computing resources on demand as a service, with a focus on the economy of resources. Cloud federations emerged as a response to the great demand for computational power by applications such as those in the fields of biology or astronomy, as well as by scientific workflows. Clouds that previously had limited resources can now procure external resources or, if they have resources that are not being used, can lease them to increase their revenues. The microservices approach, as used by large companies such as Netflix and Uber, defines services having small responsibilities, each running in its own process and communicating through lightweight mechanisms. It has become a popular approach to better distribute monolithic systems. In this sense, the architecture presented in this paper is focused on providing an efficient microservices-oriented cloud federation platform for the execution of tasks that can integrate new clouds in a simplified manner. Additional benefits include the creation of a federation platform that runs generic applications, enables direct charging of its users, shares credentials and storage spaces, and enables direct download and upload of files to the users' storage spaces. In order to validate the presented architecture, it was compared with BioNimbuZ, a monolithic cloud federation platform that executes Bioinformatics workflows. The results showed a reduction of 70% in the time to initialize the platform components and a reduction of 80% and 45% in the CPU and memory resource consumption respectively during the editing and creation of workflows. During execution of workflows, the reduction in CPU consumption was 93%, and memory was 77%.

Keywords: Cloud Computing, Cloud Federation, Microservices, Scientific Workflows, Cloud Federation Architecture, BioNimbuZ 2.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Há anos os cientistas de computação buscam soluções que permitem transformar os recursos computacionais, que estão em toda parte, seja em escritórios ou nos lares, em serviços que possam ser comparados aos serviços convencionais que permeiam a sociedade, tais como os de fornecimento de água e de luz [11].

A computação em nuvem surgiu a partir da necessidade de se consumirem recursos computacionais como serviços e sob-demanda, no qual é possível pagar apenas por aquilo que é utilizado [12]. Além disso, esse modelo computacional permite o acesso aos seus recursos por meio da Internet [13], não mais ocupando espaços em ambientes comerciais onde o metro quadrado pode custar valores altos.

As nuvens, contudo, apesar de transparecerem aos seus usuários o fornecimento ilimitado de recursos, com a premissa de serem "sob-demanda", fisicamente possuem recursos limitados. Assim, em determinados momentos, os recursos fornecidos pelos provedores de nuvem podem se esgotar, devido a uma alta demanda por estes.

Desta forma, aplicações que necessitam de grande quantidade de recursos computacionais e de armazenamento, como aplicações de *workflows* científicos, podem ter seu poder computacional limitado caso um provedor de nuvem não tenha mais recursos disponíveis.

Em resposta a esse problema, o modelo computacional de federação de nuvens propõe o estabelecimento de acordos com outros provedores de nuvens, de maneira a estender seus recursos para manter seus negócios e os de seus usuários. Por outro lado, os provedores de nuvem podem fornecer os seus recursos, que não estão sendo utilizados, para outros provedores e, assim, aumentarem suas receitas [2].

As plataformas de nuvens federadas devem transparecer aos seus usuários uma independência de provedores. Os usuários não devem se preocupar com detalhes de implementação de cada provedor, passando a ideia da existência de apenas um grande provedor de recursos computacionais. Nesse sentido, várias arquiteturas para a implantação de nuvens federadas têm sido publicadas na literatura [2, 7, 8, 14, 15, 16]. Assim, nesse cenário,

surgiu a plataforma BioNimbuZ [6, 17, 18, 19, 20, 21, 22], que implementa uma solução de nuvens federadas para a execução de *workflows* científicos. Esses *workflows* possuem tarefas capazes de consumir um alto poder computacional, levando horas ou mesmo dias para serem executadas [14]. Todavia, o BioNimbuZ apresenta uma arquitetura totalmente centralizada e monolítica, e isso não é adequado para um ambiente distribuído como é o caso das plataformas de federação de nuvens.

Assim, grandes empresas como Netflix [23], Uber [24] e SoudCloud [25] têm migrado seus serviços com arquiteturas monolíticas para arquiteturas de microsserviços em vista dos possíveis benefícios alcançados. Uma arquitetura monolítica é construída sob uma única aplicação, que mesmo apesar de possuir diferentes camadas que organizam a solução desenvolvida, como por exemplo uma aplicação web com o padrão MVC (*Model-View-Controller*), qualquer mudança no sistema, mesmo que pequena, necessita que todo o processo de implantação da nova versão seja realizado [26]. Além do mais, em arquiteturas monolíticas, como no caso do BioNimbuZ, alguns módulos que são utilizados apenas em determinados momentos, estão sempre operacionais, competindo com os recursos utilizados pelas tarefas em execução.

Nesse cenário, este trabalho propõe uma plataforma de federação de nuvens, com uma arquitetura paralela e distribuída, e que faz o uso do conceito de microsserviços para a execução de diferentes tarefas. Essa proposta foi motivada pela economia de recursos dos usuários, implantação simplificada e comunicação eficiente de seus componentes.

Assim, além de transparecer independência de provedores, com a arquitetura apresentada, pretende-se obter uma melhor utilização dos recursos computacionais e redução de custos para os usuários, que fornecerão suas credenciais para que sejam administradas pela plataforma de maneira mais eficiente. A arquitetura visa também a distribuição de seus serviços para uma melhor comunicação e redução de latência de rede, além de um monitoramento mais eficiente dos *workflows* executados.

1.1 Objetivos

O trabalho apresentado tem como objetivo principal fornecer uma arquitetura paralela e distribuída de federação de nuvens com a utilização de microsserviços que facilite a execução de tarefas para o usuário final, dando a transparência necessária para que ele não se preocupe com o fato de executar em uma plataforma de federação.

Para alcançar o objetivo principal deste trabalho, é necessário atingir os seguintes objetivos específicos:

- Definir uma estrutura de controle que possibilite um sistema multi-usuário e *multi-tenant*;

- Definir protocolos para a padronização de comunicação com diferentes plataformas de nuvem;
- Projetar componentes de maneira a dividir a comunicação em dois níveis: interno e externo às nuvens;
- Desenvolver *plugins* específicos de determinadas plataformas de nuvem que fornecerão mecanismos para o estabelecimento da federação, tais como o de provisionamento e o de descoberta de recursos;
- Projetar mecanismos para manutenção e controle de credenciais dos usuários.

1.2 Estrutura desta Dissertação

Este trabalho contém, além deste capítulo introdutório, mais cinco capítulos que contextualizam a arquitetura apresentada com as necessidades levantadas, e os objetivos alcançados.

No Capítulo 2 são descritas as definições e as características das plataformas de nuvem e seus propósitos. Também são apresentados os diferentes modelos de serviços que podem ser fornecidos aos seus usuários, e os modelos de implantação dessas plataformas. Em seguida, são detalhadas, ainda no Capítulo 2, as federações de nuvem, sendo estas uma resposta à necessidade de uma maior demanda de recursos computacionais das nuvens. Por fim, são apresentados os *workflows* científicos, que são aplicações que exigem grande poder computacional, adequadas a esse modelo de computação.

A arquitetura da plataforma BioNimbuZ é detalhada no Capítulo 3, pois foi o trabalho relacionado escolhido para ser usado na comparação de desempenho e usabilidade com a arquitetura proposta neste trabalho.

O Capítulo 4 apresenta a arquitetura de nuvens federadas proposta neste trabalho, a qual é capaz de executar serviços em diferentes nuvens. Neste capítulo também são apresentados os trabalhos relacionados ao universo de arquiteturas federadas.

Os resultados obtidos com a utilização da nova plataforma são detalhados no Capítulo 5. Em seguida, são comparados com a plataforma BioNimbuZ em sua versão anterior.

Por último, no Capítulo 6, são apresentadas as conclusões pertinentes ao trabalho apresentado e alguns trabalhos futuros.

Capítulo 2

Computação em Nuvem e Federação

Nas Seções 2.1 e 2.2 são apresentadas as características e as definições do modelo computacional das nuvens, e também são detalhadas as federações de nuvens, juntamente com seus benefícios e limitações. Na Seção 2.3 é apresentado o conceito de uma arquitetura orientada a microsserviços, o que é bastante explorado por este trabalho. Em seguida, na Seção 2.4 são apresentados os *workflows* científicos, aplicações que exigem grande quantidade de recursos computacionais, e que são adequadas ao ambiente de federação de nuvens.

2.1 Computação em Nuvem

A computação em nuvem surgiu como resposta à necessidade de se consumir recursos computacionais como serviços, obtidos principalmente pela utilização de tecnologias de virtualização. Nesse modelo, poder de processamento, de armazenamento ou de rede, são fornecidos por meio da Internet e sob demanda, permitindo aos seus consumidores pagarem apenas por aquilo que de fato utilizarem [11].

Nesse sentido, há na literatura diversas definições para o ambiente de nuvens, algumas delas são:

- Segundo NIST, *National Institute of Standards and Technology* [13], a computação em nuvem é um modelo que possibilita de forma ubíqua, conveniente e sob-demanda, acessar um *pool* compartilhado de recursos configuráveis, por meio de acesso à rede, e que podem ser rapidamente provisionados ou liberados com um mínimo esforço de gerenciamento ou de interação com os provedores do serviço;

- Buyya *et al.* [11] definem computação em nuvem como um tipo de sistema paralelo e distribuído, que consiste de uma coleção de computadores interconectados e virtualizados que são dinamicamente provisionados, e se apresentam como sendo

apenas um, baseando-se em SLAs (acrônimo em inglês para Acordos de Nível de Serviço) estabelecidos por meio de negociações entre o provedor do serviço e os seus consumidores;

- Armbrust *et al.* [27] afirmam que a computação em nuvem caracteriza-se tanto pelas aplicações fornecidas como serviços por meio da Internet quanto ao hardware e software utilizados nos *datacenters* que fornecem esses serviços;

- De acordo com Foster *et al.* [12], computação em nuvem é um paradigma de computação distribuída em larga-escala orientado à sua economia, em que são entregues sob-demanda para consumidores externos, por meio da Internet, um *pool* de recursos computacionais, de armazenamento, de plataformas e de serviços de modo abstraído, virtualizado, gerenciado e dinamicamente escalável.

NIST [13] também faz uma compilação de caraterísticas essenciais apresentadas nas diversas plataformas de nuvem, as quais são:

- ***Self-service***: um consumidor pode requisitar recursos computacionais quando desejar sem que seja necessária uma intervenção humana;

- **Amplo acesso à rede**: capacidades fornecidas por meio da Internet com interfaces padronizadas que permitem acesso por diferentes tipos de plataformas (por exemplo *smartphones, tablets* e estações de trabalho);

- ***Pooling* de recursos**: os recursos fornecidos pelos provedores aos seus consumidores se apresentam como um grande *pool*, podendo ser consumidos sob demanda e de forma dinâmica, sem a necessidade de conhecimento sobre onde os recursos estão instalados fisicamente;

- **Elasticidade rápida**: capacidade de aumento ou diminuição de consumo dos recursos de forma a atender a demanda, até mesmo de forma automática. Esta característica é de fato muito importante no universo da computação em nuvem, pois transparece aos seus consumidores uma capacidade virtualmente infinita de consumo de recursos;

- **Serviços medidos**: as nuvens devem medir o consumo de seus serviços de forma automática, provendo transparência desse consumo aos provedores de nuvens e seus consumidores.

2.1.1 Modelos de Serviço

Dentro do universo de nuvem computacional existem diferentes modelos de fornecimento de serviços. Para NIST [13], os modelos de serviços se enquadram em três tipos, os quais são:

- **SaaS (*Software-as-a-Service*)**: modelo de fornecimento de software como serviço, sem que seus usuários tenham conhecimento da infraestrutura envolvida para disponibilizá-lo;
- **PaaS (*Platform-as-a-Service*)**: são serviços que fornecem o ambiente necessário para desenvolver e acompanhar todo o ciclo de implementação de um sistema, desde seu desenvolvimento até sua implantação;
- **IaaS (*Infrastructure-as-a-Service*)**: serviços que fornecem toda a infraestrutura física necessária a uma organização, ajudando a reduzir a utilização de espaço, a manutenção de equipamentos, e mesmo o custo dentro das organizações.

Assim, na Figura 2.1 é possível ver a relação existente entre os três modelos, dispostos em camadas. As camadas superiores possuem dependência das camadas inferiores para que sejam disponibilizadas, cada camada podendo ser implantada em qualquer uma das camadas inferiores subjacentes.

Dessa forma, os serviços fornecidos pelo modelo SaaS podem ser implantados nos serviços dos modelos PaaS, IaaS, ou diretamente nos recursos de hardware. Os serviços fornecidos pelo PaaS, por sua vez, podem ser implantados diretamente nos serviços IaaS ou no hardware e, por último, os serviços IaaS, disponibilizados principalmente por meio de tecnologias de virtualização, são implantados diretamente nos recursos físicos de hardware.

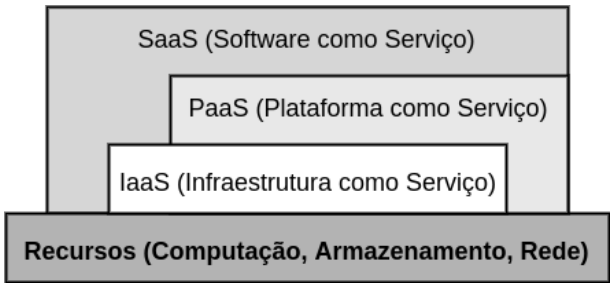

Figura 2.1: Arquitetura em Camadas da Nuvem [1].

2.1.2 Modelos de Implantação

Os modelos de implantação definem os diferentes tipos de nuvem de acordo com a abrangência de acesso por usuários ou organizações. Segundo [13, 28], os principais modelos de implantação são:

- **Nuvem Privada**: a infraestrutura é provisionada apenas para uma determinada organização, somente ela e seus membros têm o acesso aos recursos;
- **Nuvem Comunitária**: constituída por múltiplas organizações com interesses em comum, e que compartilham o provisionamento de recursos da infraestrutura estabelecida;
- **Nuvem Pública**: qualquer organização ou consumidor (público geral), é capaz de acessar e provisionar recursos da nuvem em questão;
- **Nuvem Híbrida**: é formada por uma combinação de dois ou mais modelos diferentes, anteriormente citados, obtido por tecnologias ou padrões que permitem a portabilidade de dados ou aplicações.

2.2 Federação de Nuvens

Em vista da necessidade de maior poder computacional exigido por algumas aplicações, muitas delas do meio científico, a federação de nuvens é um modelo computacional que permite que provedores de nuvem possam estender e aumentar seus recursos com a integração de outras nuvens, por meio de acordos estabelecidos. Dessa forma, os provedores obtêm ganhos com a economia de escala, com o uso eficiente de seus recursos e também com o aumento de suas capacidades [2].

Esta seção contém algumas definições de termos muito utilizados pela literatura e que ainda estão em grande evolução. Alguns termos são facilmente confundidos com outros e quase se sobrepõem. Assim, faz-se necessário definir claramente os seguintes conceitos:

- **Federação de Nuvens:** as nuvens têm participação voluntária no estabelecimento de conexão com outras nuvens para buscar e alocar recursos, estendendo seus recursos de forma automática sem a interação com mediadores [1, 29];
- ***Multi-cloud*:** relação na qual muitas nuvens são utilizadas para fornecimento de recursos, sem que estas tenham conhecimento entre si. As nuvens neste contexto não têm participação voluntária no estabelecimento de conexão com outras nuvens. Ou seja, há um terceiro ator responsável por gerenciar os recursos dessas diferentes nuvens, seja um usuário alocando recursos diretamente em uma nuvem, ou seja por meio de um *cloud-broker*, que é um agente intermediário entre o usuário final e as nuvens [1, 29, 30];
- ***Intercloud*:** qualquer relação existente entre nuvens, incluindo as duas anteriores, de forma que os recursos de determinada nuvem possam ser estendidos, estabelecendo uma verdadeira nuvem-de-nuvens [1, 2, 30].

Assim, na Figura 2.2 são apresentadas as diversas interações existentes dentro do universo da computação em nuvem.

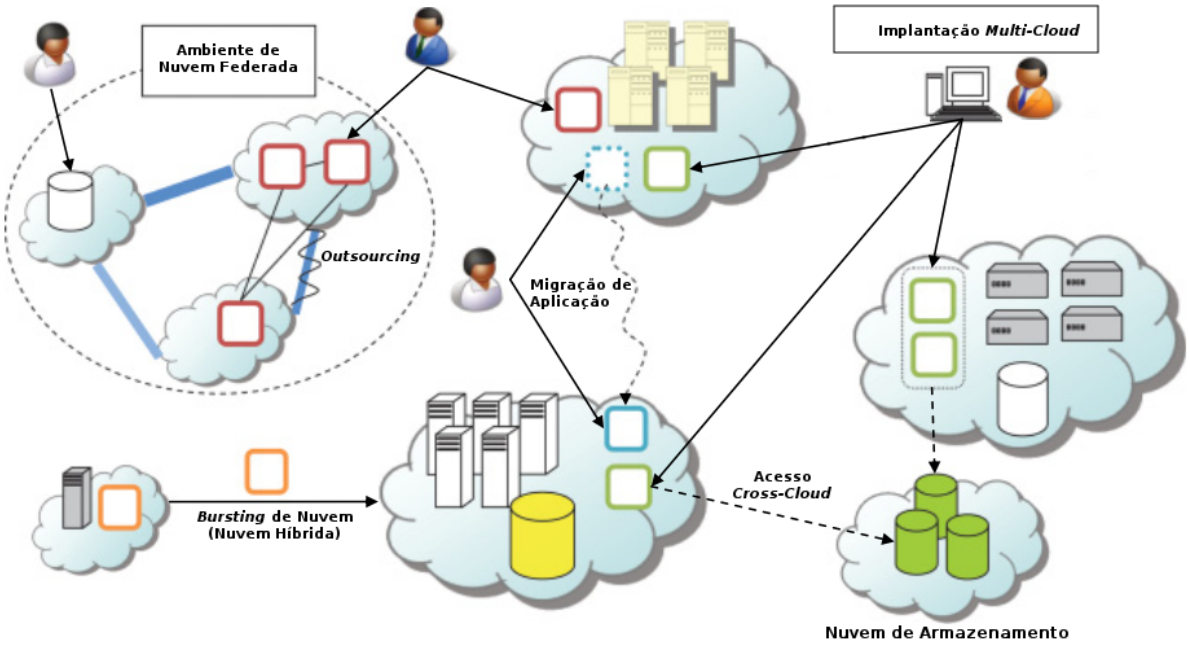

Figura 2.2: Diferentes Interações entre Nuvens, proposta por Toosi *et al.* [1].

Em uma federação de nuvens, é necessário que exista uma cooperação entre, pelo menos, duas diferentes nuvens, de forma que os seguintes comportamentos sejam notados entre elas [31]:

- ***Outsourcing*:** o provedor de nuvem, ao perceber que seus recursos estão se esgotando, busca recursos extras em outros provedores com recursos disponíveis. Essa estratégia é importante para que não haja negação de serviço por parte do provedor, evitando assim a perda de consumidores. Possivelmente, dependendo da estratégia de negócio a ser adotada, o provedor cobrará um preço diferenciado para cobrir com os custos extras;
- ***Insourcing*:** do outro lado, o provedor que possui recursos sobressalentes, ou seja, que não estão sendo muito demandados, para evitar um alto custo com sua infraestrutura, fornece acesso a estes recursos aos provedores externos.

Na Figura 2.3 é possível visualizar como os recursos são alocados em uma federação de nuvens. Uma Nuvem Local adquire recursos de uma Nuvem Estrangeira A ou B

(*outsorcing*), e estas por sua vez fornecem recursos disponíveis, para nuvens externas (*insorcing*).

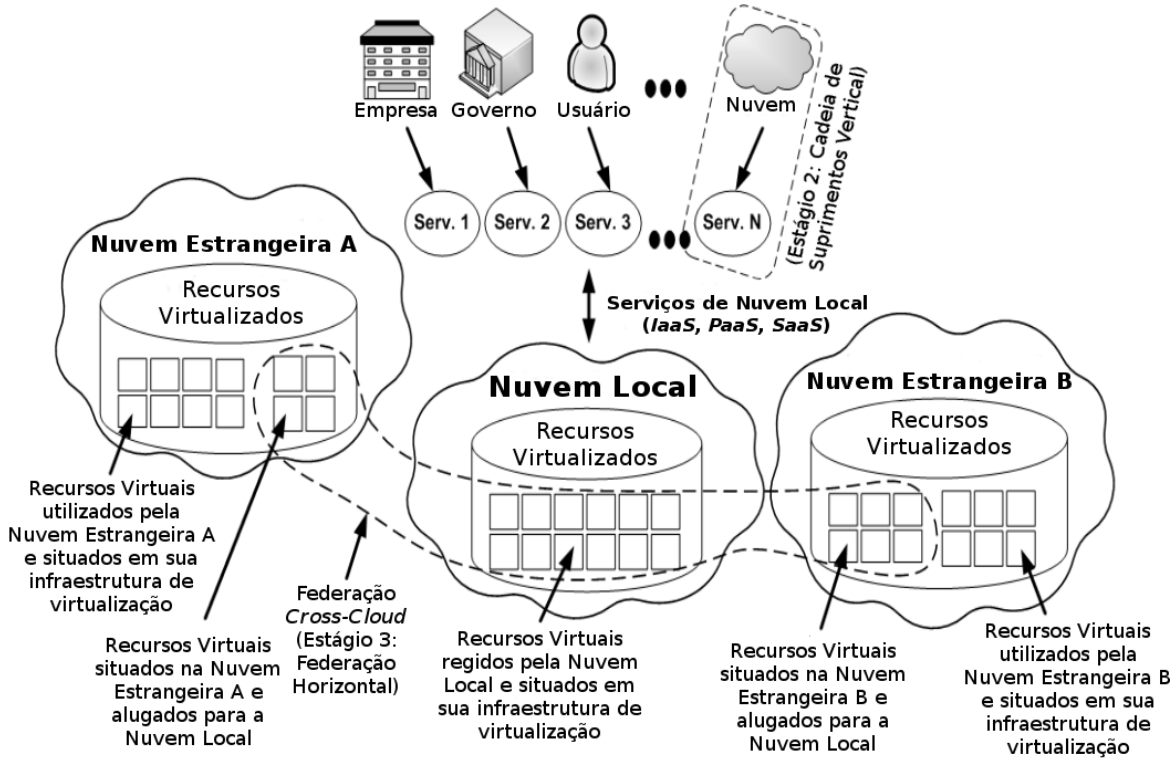

Figura 2.3: Arquitetura para Federação de Nuvens, proposta por Celesti *et al.* [2].

2.2.1 Estágios de Evolução

No processo de evolução da federação de nuvens, Celesti *et al.* [2] descrevem os três seguintes estágios necessários para o amadurecimento deste modelo:

1. **Monolítico:** empresas grandes fornecerão serviços de nuvem com tecnologia proprietária, serão como grandes ilhas de nuvens (Figura 2.4);

Figura 2.4: Estágio Monolítico [3].

2. **Cadeia Vertical de Suprimentos:** alguns serviços de nuvem exercerão influência no mercado, de tal forma que outros provedores menores também participarão desse mercado, fornecendo seus serviços (Figura 2.5);

Figura 2.5: Estágio Vertical [3].

3. **Federação Horizontal:** provedores de nuvem de qualquer tamanho farão acordos entre si para maiores ganhos: economia de escala, uso eficiente de seus ativos e o aumento de suas capacidades (Figura 2.6).

Figura 2.6: Estágio Horizontal [3].

No último estágio, da federação horizontal, as nuvens tornam-se uma verdadeira nuvem-de-nuvens (*cloud-of-clouds* [1]), uma grande nuvem composta de nuvens interconectadas, cada uma com suas próprias características servindo à diferentes necessidades [30].

2.2.2 Benefícios da Federação de Nuvens

Os seguintes benefícios podem ser obtidos por uma utilização mais ampla da federação de nuvens pelos provedores [1, 31]:

- **Economia de escala:** a infraestrutura das empresas que adotam federação de nuvens pode ser reduzida, visto que agora as empresas só contratam recursos extras em momentos de pico, reduzindo assim os custos com energia e espaço;
- **Melhor aproveitamento de recursos:** recursos subutilizados são alugados, gerando renda e reduzindo custos de manutenção de infraestrutura;
- **Extensão de recursos:** uma nuvem que esteja com os recursos esgotados, pode aumentar sua disponibilidade solicitando a outras nuvens, evitando assim a recusa de requisições de seus usuários;
- **Alta disponibilidade e baixa latência:** sistemas implantados em nuvens diferentes, e estas podendo estar situadas em regiões diversas pelo globo, permitem a redundância de informações e a aproximação dessas informações dos usuários, aumentando a interatividade do usuário com os sistemas (maior velocidade de resposta);

- **Questões legais:** dados e sistemas que dependem de questões legais específicas se beneficiam ao serem implantados em regiões diferentes.

2.2.3 Limitações da Federação de Nuvens

Existem diversos desafios dentro do tema federação de nuvens, pois uma completa federação depende de muitos fatores. Os problemas mais comuns observados são [1, 29, 32]:

- **Carência de padrões abertos:** cada provedor adota seu próprio padrão (por exemplo mecanismos de segurança e autenticação, comunicação, máquinas virtuais, armazenamento, etc.), dificultando o desenvolvimento de sistemas que fornecem soluções de federação de nuvens. Dessa forma, surge o problema chamado de *vendor lock-in*, no qual as empresas ou os desenvolvedores ficam presos à determinadas tecnologias, o que trava a migração de negócios para outras tecnologias por conta do preço proibitivo (atualização de funcionários também entra no custo);

- **Manutenção de SLAs:** os SLAs (acrônimo em inglês para Acordos de Nível de Serviço) diferem-se entre nuvens, e garantir que estejam de acordo entre todas as nuvens participantes é uma tarefa muito difícil. É mais fácil que grandes provedoras cubram custos de quebra de acordos para não perder seus clientes, o que se torna um problema para pequenas empresas, pois custear tais acordos pode inviabilizar seus negócios;

- **Dificuldade de monitoramento:** dificuldade em consolidar dados de monitoramento de recursos de uma federação com uma grande quantidade de nuvens, e em regiões diferentes. As nuvens em questão podem estar situadas em partes muito distantes no globo, aumentando assim a latência de rede da comunicação.

Biran *et al.* [4] propõem um novo paradigma para a federação de nuvens, de modo que as duas partes envolvidas, os Provedores de Serviços e os Provedores de Serviços de Nuvem, se beneficiem com a otimização do consumo dos recursos utilizados, tanto em melhor aproveitamento destes quanto em redução de consumo de energia. Além do mais, os Provedores de Serviços obterão outras vantagens, como a de evitar o *vendor lock-in*, a de obter maiores opções de preços e recursos no mercado, assim como a vantagem de melhorar o tempo de resposta para seus usuários em aplicações *on-line*.

Assim, chamado SoS, ou Sistema-de-Sistemas, caracteriza-se por um conjunto de sistemas necessários para que se possa disponibilizar outros sistemas, ou seja, por meio desse conjunto de sistemas é possível implantar outros sistemas em diferentes provedores de nuvens.

Na Figura 2.7 o Plano de Controle estabelece mecanismos para que os Provedores de Serviços obtenham recursos dos Provedores de Serviços de Nuvens, criando federações. O Coordenador de Nuvens se responsabiliza por manter registros de preços e de tipos de recursos das diferentes nuvens disponíveis. O *Broker* consulta os dados de preços e de recursos ao ser solicitado pelos Provedores de Serviços que sejam alocados os recursos.

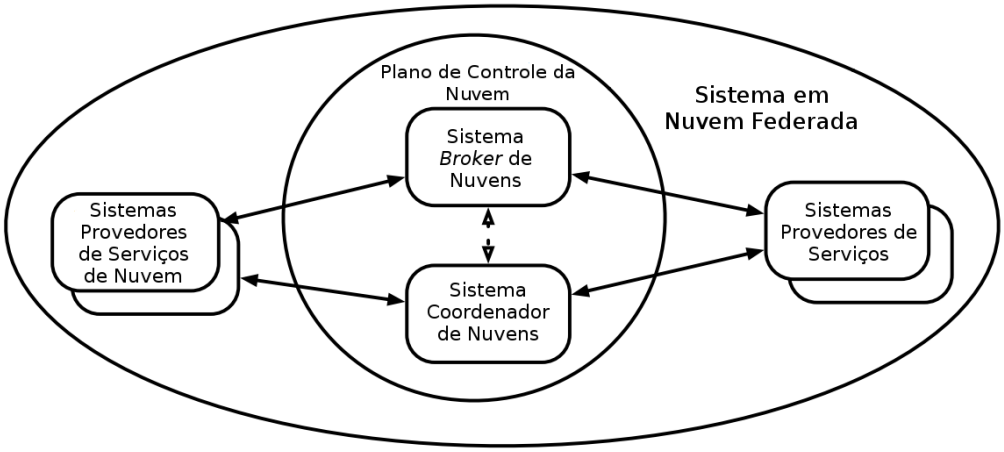

Figura 2.7: Sistema-de-Sistemas [4].

A arquitetura apresentada neste trabalho pode ser considerada um SoS, como proposto por Biran *et al.* [4], na qual há uma plataforma em que se possa executar outros sistemas em um ambiente de federação de nuvens e, dessa forma, tanto os provedores de nuvens quanto os usuários da plataforma, que serão os provedores de serviços, podem obter benefícios com sua utilização, como o de evitar o *vendor lock-in* e a otimização do uso de recursos computacionais. Assim, a arquitetura apresentada é vista em detalhes no Capítulo 4.

2.3 Microsserviços

Grandes empresas atualmente, como Netflix [23], Uber [24] e SoudCloud [25], têm investido na evolução dos seus sistemas para que os mesmos deixem de ser aplicações monolíticas e passem a ser aplicações distribuídas com a utilização da abordagem de microsserviços.

Os microsserviços são serviços que possuem pequenas responsabilidades, cada um executando em seu próprio processo [26] e que se comunicam por meio de mecanismos leves, geralmente, uma API sobre o protocolo HTTP (*Hypertext Transfer Protocol*).

Alguns dos benefícios apontados por [33] podem ser obtidos ao ser utilizada a abordagem de microsserviços, que são eles:

- **Tecnologias heterogêneas:** um sistema utilizando diversos serviços que trabalham de maneira colaborativa entre si, permite que cada serviço adote a tecnologia que melhor atinja suas necessidades, desprendendo-se de uma tecnologia que tenha que abranger todo o sistema, como por exemplo com o uso de linguagens, de *frameworks* e de bancos de dados;

- **Otimização para substituição:** sistemas muito grandes e que possuem muito tempo de vida, podem possuir tecnologias não muito mais utilizadas e que tornam sua migração dificultosa. Assim, a medida que tecnologias mais eficientes ou com maior utilização e suporte da comunidade são criadas e ou adotadas, as pequenas partes do sistema podem ser mais facilmente migradas, sem que todo o sistema seja comprometido;

- **Resiliência:** falhas podem ser melhor isoladas de tal forma que, caso aconteçam, não sejam propagadas para todo o sistema, permanecendo apenas no serviço cujo problema ocorreu;

- **Facilidade de implantação:** por permitir um sistema mais resiliente devido à independência dos diferentes serviços e a não propagação de falhas no sistema como um todo, essa qualidade acarreta uma maior facilidade de implantação de novas versões, com correções de erros ou novas funcionalidades;

- **Escalabilidade:** em um sistema monolítico, para que o fornecimento de seus serviços seja aumentado ou reduzido, todo o sistema é considerado, necessitando aumentar os serviços que não são necessários ou reduzindo aqueles que são importantes. Já em um ambiente de microsserviços, esses podem ser aumentados ou reduzidos de forma independente conforme a demanda;

- **Alinhamento organizacional:** grandes sistemas tendem a ter maiores problemas de alinhamento entre equipes, principalmente, se essas equipes são distribuídas. Com a utilização de microsserviços, cada módulo do sistema pode ser atribuído a pequenas equipes, facilitando também a distribuição de equipes dentro de uma organização;

- **Compatibilidade:** os microsserviços permitem uma maior reutilização de seus serviços, de tal maneira que diferentes plataformas *mobile*, aplicativos *desktop* ou web, ou ainda outros serviços, possam utilizá-los.

Todavia, ainda segundo Newman [33], um ambiente de microsserviços também tem suas dificuldades, assim como as encontradas em qualquer sistema distribuído, tais como

prover integração entre os componentes, implantar novas versões, realizar testes, monitorar de maneira efetiva e tratar questões de segurança. Assim, sem a adoção de técnicas que abordem essas questões, os benefícios apontados não são obtidos.

Neste trabalho, como será visto nas Seções 4.3 e 4.5, as decisões arquiteturais envolvidas para a utilização de microsserviços são melhor detalhadas, tais como a de redução da dependência das mudanças dos protocolos dos provedores de nuvem, para o funcionamento global do sistema, e também da comunicação com mecanismos de autenticação e autorização de forma distribuída entre os microsserviços.

Assim, este trabalho propõe uma arquitetura para uma uma plataforma de federação de nuvens, visando alcançar os benefícios apontados pela utilização da abordagem de microsserviços. Essa arquitetura será descrita em detalhes no Capítulo 4.

2.4 *Workflows* Científicos

Como apresentado anteriormente, as aplicações que exigem grande quantidade de recursos computacionais e de armazenamento são adequadas para o ambiente de nuvem computacional e/ou federação de nuvens. Nesse contexto, a execução de *workflows* científicos geralmente utiliza aplicações robustas que necessitam dos benefícios oferecidos por tais ambientes de computação.

Um *workflow* científico é dado pela necessidade de executar um conjunto de tarefas, compostas por um conjunto de dados de entrada/saída, que tem sua execução ordenada a fim de alcançar um determinado objetivo [34, 35]. Em outras palavras, um *workflow* refere-se à automação de processos, de tal maneira que documentos, informações ou tarefas são distribuídas entre os componentes do *workflow*, segundo um conjunto pré-definido de regras, com o objetivo de alcançar o resultado desejado de um determinado negócio.

Ainda que um *workflow* possa ser executado de forma manual, na prática e em sua grande maioria, são estruturados em um contexto de sistema de informação, que provê apoio para a execução automática dos processos. Tais sistemas são conhecidos como Sistemas Gerenciadores de *Workflows* Científicos (SGWfC) [36].

Uma das características dos *workflows* científicos é que eles tendem a ser modificados com frequência, e envolvem a análise de grande volume de dados [37]. Cientistas são exploradores por natureza e comumente exploram seus dados de pesquisas na forma de tentativa-e-erro. Assim, os *workflows* devem ser iniciados, pausados, interrompidos, revisados e re-executados sempre que for necessário.

Outra característica importante é a de que, por serem construídos por cientistas de diferentes áreas, que embora possuam proficiência própria, muitas vezes não são especialistas em computação, o que pode alterar significativamente a forma como a interface

de tais sistemas é projetada. A Seção 2.4.1 descreve as estruturas básica dos *workflows* e como as tarefas podem ser organizadas para a formação dos mesmos.

2.4.1 Estruturas Básicas

Os *workflows* podem definir uma infinidade de combinações entre as tarefas e seus dados de entrada e saída, e estes dados, por sua vez, podem ser entradas ou saídas de novas tarefas. Ainda assim, os *workflows* são constituídos por algumas estruturas básicas, como pode ser visto na Figura 2.8.

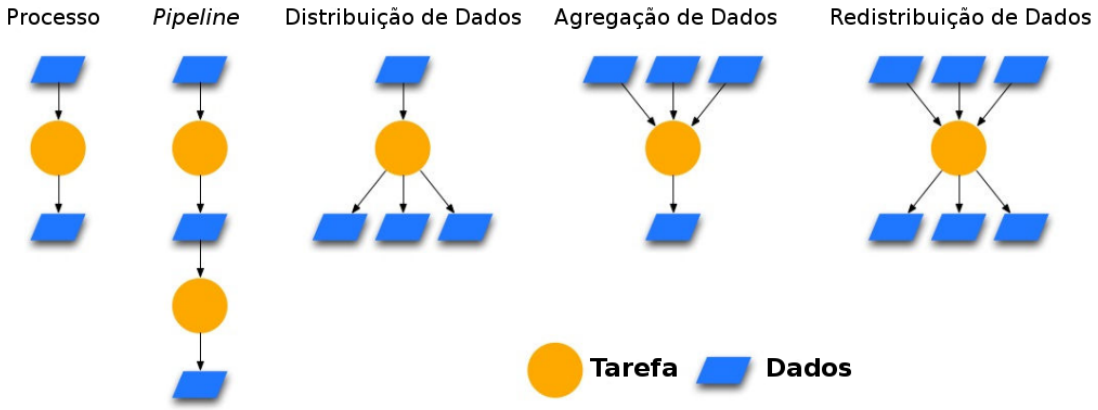

Figura 2.8: Estruturas Básicas de um *Workflow* [5].

A primeira estrutura básica é o Processo, no qual uma simples entrada para uma tarefa gera uma respectiva saída. Na segunda, chamada de *Pipeline*, os Processos anteriormente definidos são dispostos em sequência para produzir sua estrutura.

As últimas estruturas são observadas em *workflows* que possuem algum tipo de paralelismo. Sendo que na Distribuição de Dados, uma entrada é fornecida, gerando ao final muitas saídas. Em contrapartida, na Agregação de Dados, muitas entradas são fornecidas, gerando ao final apenas uma saída.

O último caso, a Redistribuição de Dados, é uma combinação das últimas duas estruturas definidas, na qual muitas entradas são fornecidas e muitas saídas são geradas.

Na Figura 2.9 são apresentados dois exemplos reais de *workflows* científicos, sendo que o da esquerda é utilizado pelos cientistas da área da Astronomia com o Montage[1], e o *workflow* da direita por cientistas da área da Biologia com o SIPHT [38].

[1] `Montage,http://montage.ipac.caltech.edu/`

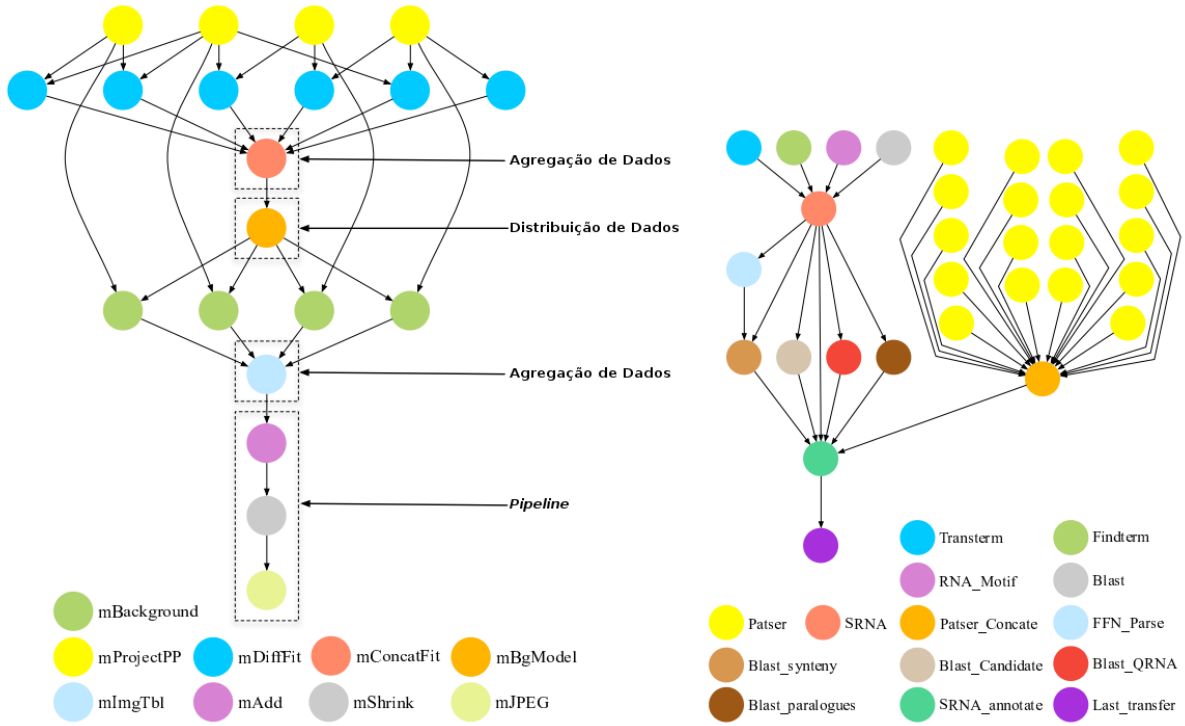

Figura 2.9: *Workflows* Científicos [5], à esquerda um *Workflow* Utilizado no Montage; e à direita Utilizado no SIPHT.

Portanto, essas estruturas, aplicadas ao contexto das nuvens, podem significar a execução de múltiplas tarefas ou máquinas virtuais em paralelo, e armazenamento de dados de entrada e saída em diferentes nuvens.

2.5 Considerações Finais

O modelo de computação em nuvem oferece computação sob-demanda como serviços, e ainda encontra-se em grande evolução. Todavia, ainda que as nuvens já estejam sendo bastante utilizadas por diferentes organizações e o público em geral, elas ainda não têm padrões universais de APIs e arquiteturas para que as federações sejam disponibilizadas mais facilmente. Para cada federação a ser realizada, são necessárias implementações de *plugins*, que são adaptações para os diversos cenários das diferentes tecnologias encontradas nos provedores de nuvem.

Assim, Barril *et al.* [29] afirmam que será necessário algo a mais no ramo da federação para que ela seja impulsionada, e padrões universais sejam criados. Esse algo a mais segundo eles, vem com o grande crescimento da *IoT* (acrônimo em inglês para Internet-

das-Coisas), no qual todas as "coisas" estarão conectadas na rede, e uma grande demanda por recursos computacionais surgirá em breve.

Órgãos como *IEEE Standards Association's*[2] criarão padrões que os provedores de nuvem poderão seguir para padronizarem suas interfaces, além de suas implementações privadas, até que esses padrões sejam universais. Assim, os consumidores poderão facilmente migrar para outras nuvens que suportem esses padrões, sempre que visualizarem um melhor custo/benefício.

As plataformas de federação de nuvem têm como objetivo tratar as questões levantadas pelo modelo de federação de nuvens, ainda que não existam padrões para a implementação de suas arquiteturas. Com isso, a utilização de microsserviços pelos provedores de nuvem em vista dos benefícios que podem ser alcançados e, aliados a definições de padrões de comunicação para acesso de seus recursos, pode ser uma resposta às dificuldades apontadas para a criação de plataformas de federação de nuvem.

Nesse cenário, foi desenvolvida uma arquitetura centralizada e monolítica de uma plataforma de federação de nuvens chamada BioNimbuZ, que é vista em detalhes no Capítulo 3.

[2] `IEEE-SA,https://standards.ieee.org`

Capítulo 3

Plataforma BioNimbuZ

As seções seguintes detalham o funcionamento e os mecanismos utilizados pela plataforma de federação de nuvens para a execução de *workflows* científicos, chamada BioNimbuZ, que foi utilizada como estudo de caso neste trabalho.

3.1 Visão Geral

Inicialmente proposto por Saldanha *et al.* [17, 39], o BioNimbuZ é uma plataforma de federação de nuvens projetada para a execução de *workflows* científicos de Bioinformática. Atualmente, a plataforma conta com diversas evoluções em sua arquitetura e serviços internos, e uma das principais mudanças foi a utilização de mecanismos de comunicação mais eficientes [6, 18, 19, 20, 21, 22]. Anteriormente, a plataforma contava com comunicação P2P (*peer-to-peer*) para a comunicação distribuída com o núcleo do sistema. Agora, a plataforma utiliza tecnologias para a coordenação de tarefas, manutenção e sincronização de informações em ambiente distribuído [40], fornecidas pelas ferramentas ZooKeeper[1] e de serialização de dados e Chamada de Procedimento Remoto (*Remote Procedure Call* - RPC) com o Avro[2].

Dentre as mudanças importantes, a plataforma fornece algoritmos de escalonamento de tarefas que identificam as melhores máquinas disponíveis, em relação ao poder computacional e à quantidade de memória, e que também identificam as máquinas menos utilizadas (para balanceamento de carga), na execução das tarefas dos *workflows*, de forma que o tempo de execução ou o custo total seja reduzido [22, 41]. Também são implementadas políticas de armazenamento que consideram localidade de dados de entrada, custo, latência de rede e a utilização de recursos gratuitos ou não [19, 40, 42].

[1] Apache ZooKeeper, `https://zookeeper.apache.org/`
[2] Apache Avro, `https://avro.apache.org/`

Além disso, a plataforma atualmente implementa dois controladores, um de elasticidade, responsável por aumentar e diminuir a quantidade ou a capacidade de determinado recurso [18] e também, o controlador de Acordos de Nível de Serviço (*Service Level Agreement* - SLA), em que acordos são feitos com os usuários para que estes tenham salvaguarda em caso de não cumprimento desses acordos pelo provedor de nuvem [6].

3.2 Arquitetura do BioNimbuZ

A plataforma BioNimbuZ divide-se em quatro camadas [14], as quais comunicam-se entre si para a devida execução de *workflows* em ambiente federado, e gerenciamento de recursos utilizados. As camadas que compõem o BioNimbuZ são: a Camada de Aplicação, a Camada de Integração, a Camada de Núcleo e a Camada de Infraestrutura, conforme apresentadas na Figura 3.1, e detalhadas a seguir:

- **Camada de Aplicação:** interface web que implementa um sistema gerenciador de *workflows* científicos, por meio da qual os usuários interagem com a plataforma, criando e acompanhando a execução de seus *workflows*, e fazendo o *upload/download* de seus arquivos de entrada e de saída;

- **Camada de Integração:** camada responsável pela integração entre a Camada de Aplicação e a Camada de Núcleo, utilizando *Web Services* implementados de acordo com a arquitetura REST (*REpresentional State Transfer*) [43], que pode ser utilizado como uma alternativa ao RPC [44];

- **Camada de Núcleo:** esta camada pode ser considerada a camada mais importante da plataforma por conter a maior parte dos serviços dos quais é composta. Por isso, ela será melhor detalhada na Seção 3.2.1;

- **Camada de Infraestrutura:** nesta camada estão localizados os recursos computacionais utilizados nas diversas plataformas de nuvem. A criação dos recursos nas diferentes nuvens é feito por meio dos *plugins*, que são interfaces adaptadas aos diferentes tipos de nuvem, de forma que a criação desses recursos seja transparente em relação à nuvem utilizada.

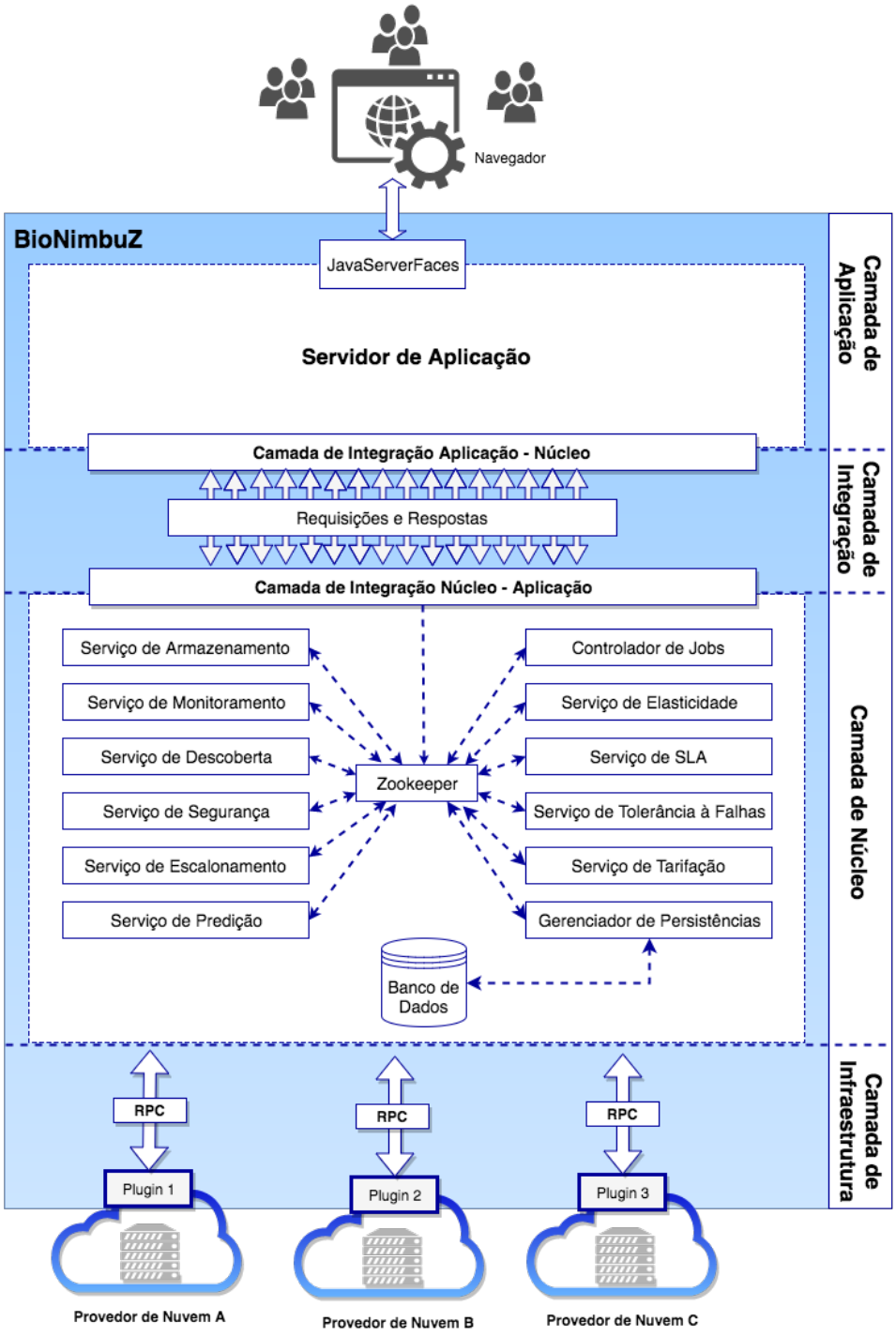

Figura 3.1: Arquitetura da Plataforma BioNimbuZ [6].

3.2.1 Camada de Núcleo

Como citado anteriormente, esta é a camada mais importante da plataforma BioNimbuZ, pois por meio dela a plataforma implementa os diversos tipos de serviços necessários para a execução dos *workflows* científicos no ambiente de nuvem federada.

A Camada de Núcleo, formada por diversos serviços e controladores, estabelece comunicação de forma distribuída com as máquinas que executam as tarefas, com a utilização dos mecanismos fornecidos pelo ZooKeeper e pelo Avro. Assim, os principais serviços são:

- **Serviço de Armazenamento:** incumbido de implementar estratégias de armazenamento de arquivos utilizados pelos *workflows*, sejam eles arquivos de entrada ou arquivos de saída. Para isto, o serviço analisa fatores como a latência de rede, largura de banda, localização geográfica entre os provedores, número de núcleos de processamento, carga de trabalho, custo de armazenamento e capacidade de armazenamento disponível em cada recurso virtual. O Serviço de Armazenamento trabalha com duas estratégias de armazenamento de arquivos relacionados a execução de *workflows*. A primeira refere-se aos próprios recursos virtuais utilizados na execução do *workflow*, já a segunda refere-se a utilização de serviços de armazenamento fornecidos pelos provedores de nuvem (*e.g.*, *Buckets*[3][4]);

- **Serviço de Monitoramento:** este serviço é responsável por acompanhar a execução de todos os *workflows* submetidos à plataforma. O acompanhamento realizado, tem como foco verificar o *status* de execução de tarefas nos recursos virtuais utilizados para a execução do *workflow*. A verificação é feita por meio de consultas ao serviço centralizado do ZooKeeper. Além disso, o Serviço de Monitoramento permite a recuperação de dados utilizados por cada instância do BioNimbuZ;

- **Serviço de Descoberta:** identifica e mantém informações a respeito dos provedores participantes da federação. Dentre as informações, destacam-se a capacidade de armazenamento, de processamento e de latência de rede, relacionados aos recursos virtuais disponíveis na federação. Detalhes sobre parâmetros de execução, arquivos de entrada e arquivos de saída, também são mantidos por este serviço. Todo o controle aqui realizado utiliza funcionalidades providas pelo serviço do ZooKeeper, que fornece uma visão global de todos os recursos existentes na federação;

- **Serviço de Segurança:** encarregado por garantir os princípios básicos da segurança da informação: a confidencialidade, a integridade e a disponibilidade de informações. Em outras palavras, este serviço é responsável por: (1) manter o sigilo de informações, impedindo que pessoas não autorizadas tenham acesso a informações sigilosas; (2) proteger as informações submetidas por usuários, bem como as informações geradas pelos *workflows*, de modo a impedir mudanças em seu conteúdo,

[3]Amazon S3 Buckets, `http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingBucket.html`
[4]Google Cloud Storage Buckets, `https://cloud.google.com/compute/docs/disks/gcs-buckets`

sejam elas acidentais, indevidas ou até mesmo intencionais (*e.g.*, alterações indevidas no conteúdo de arquivos de entrada/saída); e (3) garantir que as informações sempre estarão disponíveis para pessoas autorizadas a qualquer momento;

- **Serviço de Escalonamento:** este serviço é acionado pelo Controlador de *Jobs*, e tem como responsabilidade distribuir as tarefas que fazem parte do *workflow* submetido, nos provedores da federação. Além disso, este serviço também mantém o registro de todas as tarefas que já foram escalonadas. Métricas como latência, balanceamento de carga, tempo de espera e capacidade de processamento são consideradas no escalonamento aplicado. A política de escalonamento utilizada atualmente pela plataforma é a C99 [22], na qual seu algoritmo tem como objetivo ser *any-time* e incremental, com rápida convergência para boas soluções de escalonamento;

- **Serviço de Predição:** tem como objetivo auxiliar o usuário final na escolha dos recursos necessários à execução de seu *workflow*, com estimativas de tempo, de custo e de recursos computacionais. Para isto, o usuário deve preencher, na Camada de Aplicação, um modelo com informações de seu *workflow* para que o serviço efetue as estimativas mencionadas. Este serviço é constituído de quatro fases, são elas: (1) coleta de informações, na qual o usuário informa os parâmetros desejados para a execução do *workflow* (por exemplo, capacidade desejada para processamento, memória e disco); (2) avaliação da base histórica de execução de *workflows*, para otimizar a acurácia no cálculo das estimativas, a base histórica contém informações de execuções passadas de todos os *workflows* já submetidos à plataforma BioNimbuZ; (3) aplicação do algoritmo GRASP [45], a fim de obter boas soluções em tempo viável, por meio da aplicação de meta-heurísticas baseadas em informações coletadas de fases anteriores; e por fim, (4) armazenar e informar ao usuário os dados das predições obtidas com a aplicação das três fases anteriores;

- **Controlador de *Jobs*:** é incumbido de receber as requisições da Camada de Aplicação, verificando as credenciais de acesso dos usuários por meio da utilização do Serviço de Segurança, para permitir ou não a execução de tarefas na plataforma BioNimbuZ. Além disso, o Controlador de *Jobs* é o responsável por gerenciar as solicitações dos usuários, bem como prover respostas para cada solicitação;

- **Serviço de Elasticidade:** responsável por ajustar automaticamente o número de instâncias de máquinas virtuais, reconfigurando parâmetros de utilização de processador e memória, a fim de otimizar a utilização da infraestrutura disponível. A elasticidade pode ser vertical, quando há um redimensionamento dos atributos de

capacidade de processador e memória, como também pode ser horizontal, quando o número de instâncias de máquinas virtuais é modificado para mais ou para menos;

- **Controlador de SLA:** encarregado por implementar o ciclo de vida de SLA, o qual é formado por seis atividades: (1) descoberta de provedores, (2) definição de SLA, (3) estabelecimento do acordo de serviço, (4) monitoramento de violação do acordo, (5) término de acordo e (6) aplicação de penalidades por violação do acordo. Para isso, o usuário deve configurar o modelo de SLA desejado. O modelo é constituído por requisitos funcionais, como número de núcleos do processador, tamanho de memória, tamanho de armazenamento e também por requisitos não funcionais, como custo máximo para a execução de um *workflow*, bem como a taxa mínima para transferência de dados durante a execução de um *workflow*. Para as requisições dos usuários, o Controlador de SLA verifica junto aos provedores de nuvem participantes da federação, se eles podem ou não suportar, naquele momento, os parâmetros configurados pelo usuário. Essa verificação é feita por meio da utilização dos *plugins*, responsáveis pela comunicação com os provedores;

- **Serviço de Tolerância à Falhas:** é responsável por monitorar os dados referentes às execuções dos *workflows*, ou seja, arquivos de entrada/saída, de modo a garantir que sempre existam no mínimo duas cópias de cada arquivo utilizado/gerado pela plataforma. Por meio da utilização do serviço centralizado ZooKeeper, o serviço é capaz de mapear os recursos que contém os arquivos duplicados, bem como identificar quando alguns dos recursos se tornam indisponíveis na federação. Ao identificar que um recurso virtual não está disponível ou que existam arquivos ainda não duplicados, o Serviço de Tolerância a Falhas inicia o processo de duplicação para evitar que arquivos necessários e arquivos gerados pela execução de *workflows* sejam perdidos;

- **Serviço de Tarifação:** é o serviço responsável por calcular o valor devido pelos usuários, 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 como o tempo de execução e a quantidade de máquinas virtuais alocadas para as tarefas que foram, e que estão sendo executadas pelo usuário. Em um ambiente de nuvem federada, a complexidade de tal serviço é maior do que em ambientes de modelo de nuvem única, uma vez que a infraestrutura de uma plataforma de nuvens federadas é garantida pela integração de recursos de diversos provedores de nuvem autônomos;

- **Gerenciador de Persistências:** fornece aos demais serviços, um canal de comunicação com o repositório de dados utilizado pela plataforma BioNimbuZ, que possibilita a consulta e a atualização de dados referentes aos *workflows* submetidos, e aos dados dos usuários cadastrados no sistema.

Apesar de fornecer uma gama de serviços, o BioNimbuZ necessita de mecanismos mais eficientes para a execução de tarefas em um ambiente de nuvens federadas, como os apresentados por este trabalho.

Primeiramente, sua Camada de Núcleo é possuidora de todos os serviços, inclusive os serviços de execução, o que a torna uma aplicação mal distribuída e monolítica, levando em consideração que alguns serviços não serão efetivamente utilizados em determinados momentos (por exemplo, quando somente executando uma tarefa).

Os *plugins* de nuvens utilizados pela plataforma estão integrados à Camada de Núcleo, e também concorrem com os recursos utilizados pelos seus serviços. A utilização de microsserviços proposta por este trabalho possibilita o desacoplamento dessa camada, tanto para evitar concorrência do uso dos recursos quanto para permitir que mudanças sejam implantadas mais rapidamente, sendo necessário que apenas os *plugins* sejam reiniciados, e não todas as máquinas do ambiente de federação.

Outro problema decorrente de sua arquitetura é o fato de que todos os componentes necessitam estar conectados ao ambiente ZooKeeper, mesmo que estejam em nuvens diferentes. Assim, sincronizar informações com uma grande quantidade de recursos em um ambiente distribuído de nuvens, torna-se um problema quando esses recursos constantemente atualizam essas informações. Desta forma, faz-se necessário reduzir a quantidade de informações que trafegam entre as nuvens, para que explorem mais o tráfego dentro das nuvens, o que é apresentado por este trabalho com a criação dos Coordenadores e Executores.

3.3 Considerações Finais

Este capítulo apresentou a plataforma de federação de nuvens para execução de *workflows* científicos de informática, o BioNimbuZ.

Em vista dos problemas apontados, e levando em consideração a importância que a plataforma apresenta para a área da Bioinformática, ela foi utilizada como estudo de caso para a implementação e análise da arquitetura apresentada por este trabalho.

A arquitetura de federação de nuvens hierárquica e distribuída a ser apresentada tem o objetivo de facilitar e tornar transparente a integração entre nuvens diferentes, e é detalhada no Capítulo 4.

Capítulo 4

BioNimbuZ 2: Federação de Nuvens Sob Microsserviços

Neste capítulo é detalhada a arquitetura hierárquica e distribuída orientada a microsserviços proposta neste trabalho para o ambiente de federação de nuvens. Na Seção 4.1 é apresentada uma visão geral sobre os componentes que constituem a arquitetura da plataforma de federação de nuvens proposta. Nas Seções 4.2, 4.3, 4.4 e 4.5 são detalhadas suas camadas e as decisões arquiteturais envolvidas para seu desenvolvimento. Assim, na Seção 4.6 é descrito o fluxo de execução principal da plataforma para que se possa executar uma tarefa. Em seguida, na Seção 4.7 é detalhado o modelo de tolerância a falhas implementado no BioNimbuZ 2. Por fim, na Seção 4.8 são comparados os trabalhos mais recentes da literatura, relacionados a arquiteturas de federação de nuvens, com a arquitetura proposta neste trabalho.

4.1 Visão Geral

A arquitetura proposta neste trabalho foi projetada de maneira hierárquica e distribuída, a fim de facilitar a distribuição dos seus componentes em diferentes provedores de nuvem (veja a Figura 4.1). Além disso, ela possui camadas bem divididas, o que permite maior facilidade de implantação de seu ambiente e de integração de seus componentes na federação de nuvens. Outro fator que contribui para essa facilidade é a criação dos *Plugins* de nuvem como microsserviços [26], que são totalmente independentes do funcionamento global da plataforma, e podem ser encerrados e iniciados sempre que for necessário, garantindo maior tolerância a falhas para a plataforma de federação.

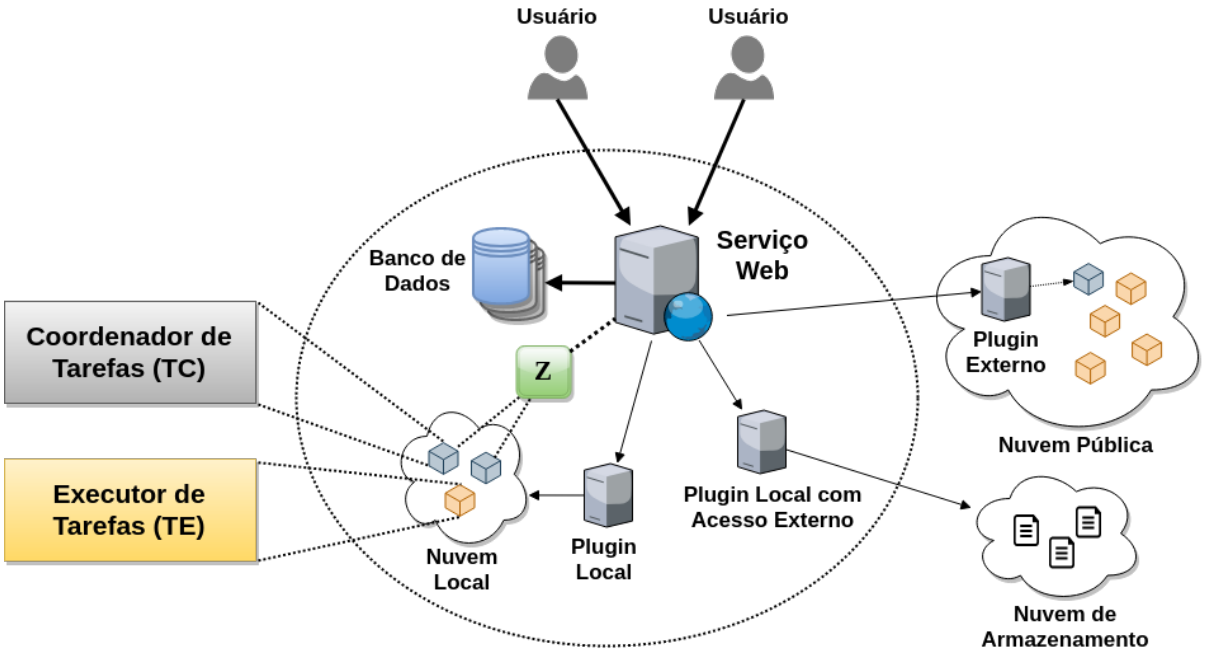

Figura 4.1: Distribuição dos Componentes da Arquitetura Proposta.

Além disso, a comunicação entre os componentes é feita em dois níveis, uma em alto nível e a outra em baixo nível. A primeira, em alto nível, é feita entre as nuvens, permitindo uma troca de informação mais objetiva e direta. A outra, em mais baixo nível, é feita internamente em cada nuvem componente da federação, trafegando informações de monitoramento das tarefas em execução, que podem executar em paralelo. Desta forma, a arquitetura proposta possibilita uma atualização de informações em tempo real aos seus múltiplos usuários.

Outra característica importante da arquitetura apresentada é a possibilidade de utilização de credenciais de nuvens de seus usuários, e o seu compartilhamento com grupos de usuários, permitindo que organizações, como as universidades por exemplo, criem grupos de estudantes com uma mesma credencial. Assim, a tarifação é feita diretamente pelos provedores de nuvem, sem que a plataforma de federação tenha a responsabilidade sobre o controle e o processo de cobrança de cada usuário. Além do mais, seus usuários podem acessar a plataforma de maneira simplificada com o serviço web disponibilizado.

Para isso, a arquitetura proposta foi estruturada em quatro camadas, as quais são: a Camada de Aplicação, que fornece mecanismos para que os usuários possam interagir com a plataforma por meio do Serviço Web; a Camada de Federação, com mecanismos facilmente acopláveis à plataforma e que permitem a criação da federação de nuvens; a Camada de Coordenação, implementada pelo Coordenador de Tarefas (*Task Coordinator* – TC), e que é responsável por coordenar o fluxo e o monitoramento de execução das

tarefas; e a Camada de Execução, implementada pelo Executor de Tarefas (*Task Executor* – TE), e é responsável por efetivamente processar as tarefas e monitorar seus estados e o consumo de recursos (por exemplo, CPU e memória).

É importante ressaltar que na arquitetura proposta por este trabalho, cada camada consiste de controladores, serviços e mecanismos, os quais são distribuídos de maneira a obter um melhor aproveitamento de recursos computacionais e de rede. Assim, a arquitetura proposta foi usada para a implementação da nova versão da plataforma BioNimbuZ, chamada de BioNimbuZ 2 (ou BioNimbuZ versão 2).

Na Figura 4.2 é possível ter uma visão geral das camadas da arquitetura apresentada, e implementadas no BioNimbuZ 2. As próximas seções descrevem cada uma dessas camadas em detalhes.

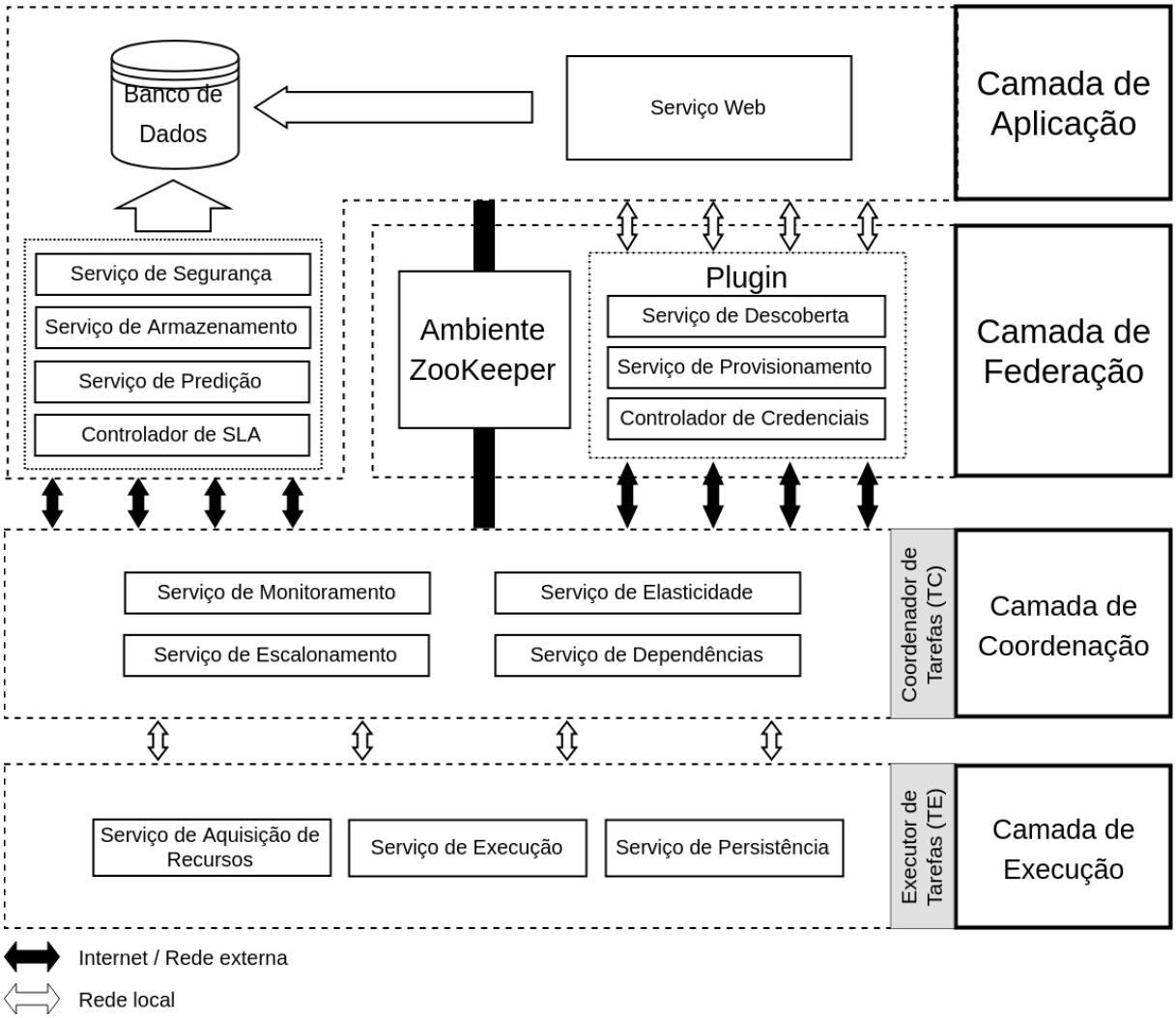

Figura 4.2: Arquitetura Proposta para Plataforma de Nuvem Federada Orientada a Microsserviços – BioNimbuZ 2.

4.2 Camada de Aplicação

Esta camada é responsável por disponibilizar a interação dos usuários com o sistema, por meio de interface web. A interface permite aos usuários acessarem a plataforma e manterem suas credenciais de provedores de nuvem. Também é possível aos usuários executarem aplicações e criarem dependências entre elas (por exemplo, as dos *workflows* científicos), e enviarem dados de entrada/saída para as nuvens cadastradas.

Assim, os dados fornecidos (registros, credenciais de nuvem, tarefas, dependências, metadados dos dados de entrada/saída, etc.) são mantidos em banco de dados relacional PostgreSQL[1] em sua versão 9.5, e acessados pelos módulos que compõem esta camada. Os serviços e os controladores desta camada são os seguintes:

- **Serviço Web:** este serviço é o responsável por fornecer a interação dos usuários com a plataforma por meio de páginas web. Com este serviço os usuários fazem o acesso à plataforma e podem desenhar seus *workflows*, efetuar o *upload*/*download* de arquivos que os *worflows* usam como entrada ou que geram como saída, e monitorar os *status* de execução;

- **Serviço de Segurança:** responsável por armazenar os dados de autenticação dos usuários na plataforma, e as credenciais dos serviços de nuvem disponíveis (a ser melhor detalhado na Seção 4.2.3). Este serviço também viabiliza o acesso de múltiplos usuários à plataforma, garantindo que os dados estejam isolados entre os usuários e os seus grupos (credenciais, *workflows*, resultados, etc.);

- **Serviço de Armazenamento:** este serviço é responsável por tomar decisões de armazenamento dos arquivos de entrada e de saída das tarefas executadas. As decisões de armazenamento consideram questões como o custo e a capacidade, e são tomadas com base nos dados levantados pelos *Plugins* e o Serviço de Descoberta, que serão vistos na Seção 4.3;

- **Serviço de Predição:** as plataformas de nuvem consideradas na predição para a execução das tarefas serão apenas as relacionadas às credenciais fornecidas pelos usuários, não necessitando considerar todas as plataformas suportadas pelo sistema. Assim, se o usuário cadastrar apenas a credencial do provedor de nuvem da Google, o *Google Cloud Platform*[2], sua aplicação somente poderá executar nesse provedor. Todavia, se o usuário fornecer credenciais do provedor de nuvem da Google e da Amazon, o *Amazon Web Services*[3]; a plataforma de federação, juntamente com as

[1] PostgreSQL, `https://www.postgresql.org/`
[2] *Google Cloud Platform*, `https://cloud.google.com/`
[3] *Amazon Web Services*, `https://aws.amazon.com/`

decisões deste serviço, poderá distribuir a execução das tarefas do usuário em ambos os provedores de nuvem;

- **Controlador de SLA:** mantém o controle dos SLAs criados pelos usuários para a execução das tarefas, analisando constantemente as informações de monitoramento para garantir que os acordos estabelecidos não sejam violados. Este controlador está sempre atualizado em relação às informações de monitoramento coletadas pelos Coordenadores, que são descritos na Seção 4.4.

A Camada de Aplicação foi desenvolvida utilizando o *framework* de código aberto Playframework em sua versão para Java [46], que fornece um ambiente de desenvolvimento sobre uma arquitetura MVC (*Model View Controller*), totalmente RESTful [43]. Além disso, por ser uma arquitetura *stateless*, ou seja, que não armazena estado de seus usuários, esta camada também utiliza o padrão de *tokens* JWT (*JSON*[4] *Web Tokens*) [47] para autorizar o acesso aos usuários.

Além do mais, o ambiente de desenvolvimento provido pelo *framework* Playframework, faz com que o Serviço Web, na Camada de Aplicação, seja totalmente independente de bibliotecas, oferecendo aos desenvolvedores livre escolha. Assim sendo, é importante ressaltar a escolha da biblioteca de código aberto Bootstrap [48] para o desenvolvimento de páginas web responsivas, ou seja, que automaticamente se adaptam à tela do usuário, seja acessando um computador pessoal ou um *smart-phone*.

Nas próximas sub-seções são detalhados os tipos de usuários que podem acessar a plataforma de federação, os *menus* exibidos e também as decisões arquiteturais envolvidas para a implementação desta camada.

4.2.1 Usuários da Plataforma

A plataforma de federação desenvolvida com a arquitetura proposta possui dois tipos de usuários, um usuário do tipo Administrador e um usuário do tipo Cliente. O usuário Administrador é mais avançado, no sentido de possuir mais poderes de acesso e de poder criar outros usuários.

Por outro lado, o usuário do tipo Cliente é mais simples, ou seja, não precisa lidar com questões de configuração da plataforma de federação de nuvens.

Após o devido acesso do usuário ao sistema por meio da Tela de *Login* (veja a Figura 4.3), se o usuário for reconhecido como um usuário do tipo Cliente, a Tela Principal, vista na Figura 4.4, é exibida.

[4]JSON, `http://www.json.org/`

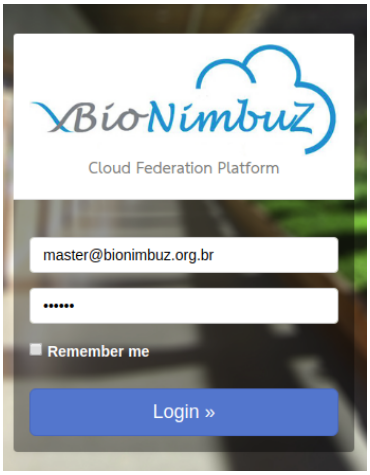

Figura 4.3: Tela de *Login*.

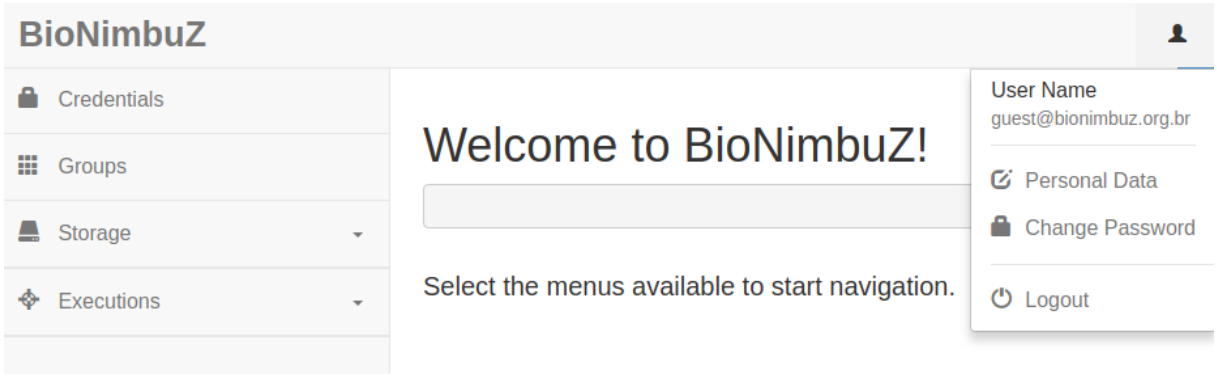

Figura 4.4: Tela Principal do Usuário Cliente.

Caso o usuário do sistema seja identificado como sendo do tipo Administrador, a Tela Principal exibida para ele é mostrada na Figura 4.5.

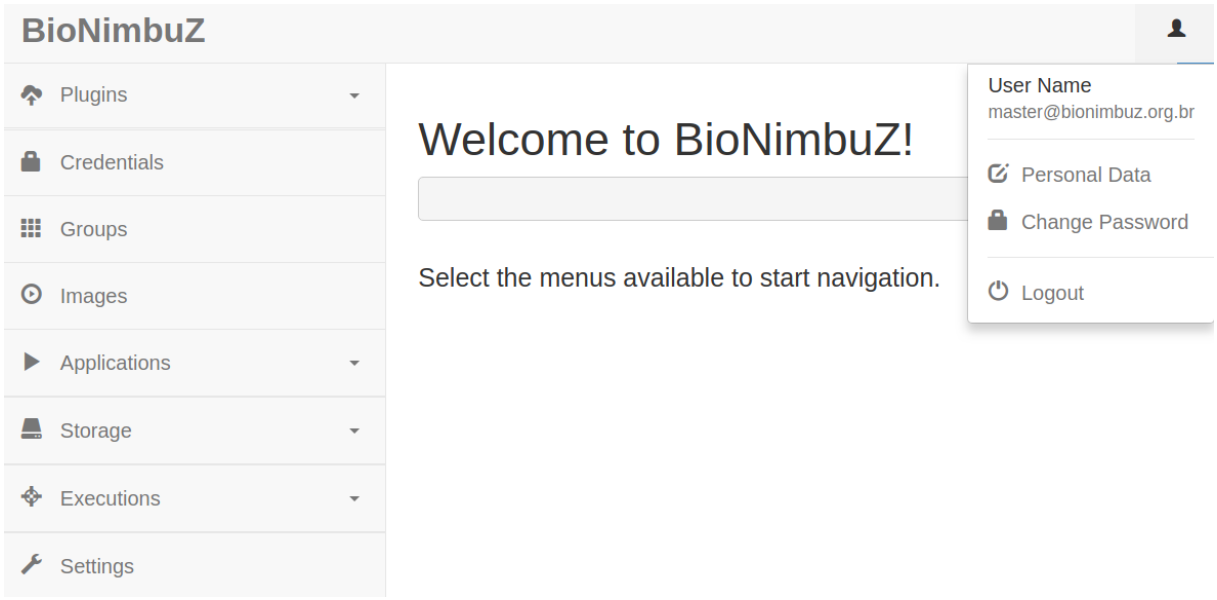

Figura 4.5: Tela Principal do Usuário Administrador.

Como pode ser notado na Figura 4.5, para o usuário Administrador, além das funcionalidades de usuário Cliente, também são exibidas mais funcionalidades, que requerem maior conhecimento sobre a arquitetura da plataforma e sobre os provedores de nuvem que compõem a federação. Assim, os *menus* que compõem a tela principal mostrada na Figura 4.5 são os seguintes:

- ***Plugins*:** *menu* composto por outros dois *sub-menus*, no qual os *Plugins*, que fornecem o ambiente de federação de nuvens à plataforma, podem ser adicionados. Seus *sub-menus* são os seguintes:
  - *Administration*: *sub-menu* por meio do qual os *Plugins* de nuvem podem ser adicionados;
  - *Price Tables*: neste *sub-menu* o *status* das tabelas de preços dos *Plugins* podem ser consultados.
- ***Credentials*:** neste *menu* os usuários do sistema podem adicionar suas credenciais de nuvem a serem utilizadas pela plataforma para a criação de suas tarefas;
- ***Groups*:** com a funcionalidade oferecida por este *menu* os usuários podem configurar seus grupos de usuários, possibilitando o compartilhamento de credenciais e de espaços de armazenamento;
- ***Images*:** neste *menu* são buscadas e adicionadas as imagens de sistemas operacionais disponibilizadas pelas plataformas de nuvem;

- ***Applications*:** as aplicações que podem ser criadas neste *menu* possuem as informações de execução das tarefas dos usuários, tais como os *scripts* de inicialização e de execução, e as imagens de sistema operacional que são utilizadas na criação da instância na nuvem escolhida. Este *menu* é composto de outros dois *sub-menus*, os quais são:
  - *Coordinator*: este *sub-menu* permite a configuração de uma única aplicação padrão, que é utilizada para a coordenação das tarefas em execução;
  - *Executors*: com este *sub-menu* o usuário Administrador pode adicionar diferentes aplicações, que permitem executar as tarefas dos usuários.
- ***Storage*:** as funcionalidades de armazenamento de arquivos dos usuários são fornecidas acessando os seguintes *sub-menus*:
  - *Spaces*: os espaços são locais de armazenamento disponibilizados em determinadas regiões de uma nuvem, e neles são mantidos os arquivos dos usuários;
  - *Files*: este *sub-menu* permite que os usuários façam o *upload*/*download* de arquivos em um espaço existente.
- ***Executions*:** este *menu* contém a principal funcionalidade do sistema, que necessita que os *menus* anteriores tenham sido devidamente acessados e configurados, e é composto pelos seguintes *sub-menus*:
  - *Instances*: as instâncias representam uma instância de VM em execução na nuvem selecionada, que é criada utilizando as credenciais do usuário ou do grupo de usuários que estão compartilhando uma credencial;
  - *Workflows*: neste *sub-menu* os usuários podem, por meio de interface gráfica, desenhar seu fluxo de execução de tarefas, que é um conjunto de instâncias de VM, e as dependências existentes entre elas, para que a plataforma as execute de forma adequada.
- ***Settings*:** neste *menu* é possibilitado ao usuário alterar as configurações globais utilizadas pela plataforma.

A plataforma também permite que os usuários alterem suas informações pessoais básicas acessando os *menus* "*Personal Data*" e "*Change Password*" no canto superior direito das Figuras 4.4 e 4.5.

4.2.2 *Menu*: *Plugins*

Como apresentado na tela principal do usuário Administrador (Figura 4.5), há o *menu* *Plugins*, o qual indica os provedores de nuvem em funcionamento. Assim, neste trabalho, os *Plugins* são os microsserviços que fornecem acesso padronizado às diferentes nuvens integrantes da federação, e são melhor detalhados na Seção 4.3. Este *menu* possui outros dois *sub-menus*, sendo eles: *Administration* e *Price Tables*.

Administration

Ao acessar este *menu*, são exibidos os *Plugins* de nuvem já cadastrados, e também é permitido ao usuário cadastrar novos *Plugins*. Por segurança, e em vista da manipulação de *tokens* de autorização dos usuários pelos *Plugins*, os microsserviços são adicionados na plataforma de maneira manual pelos usuários administradores. A inserção manual evita que qualquer *Plugin* malicioso que implemente o protocolo de comunicação faça parte da federação.

No cadastro de *Plugins*, como visto na Figura 4.6, é necessário apenas que o usuário informe o endereço do *Plugin*, e ao clicar no botão "*Search*", todas as outras informações são recuperadas.

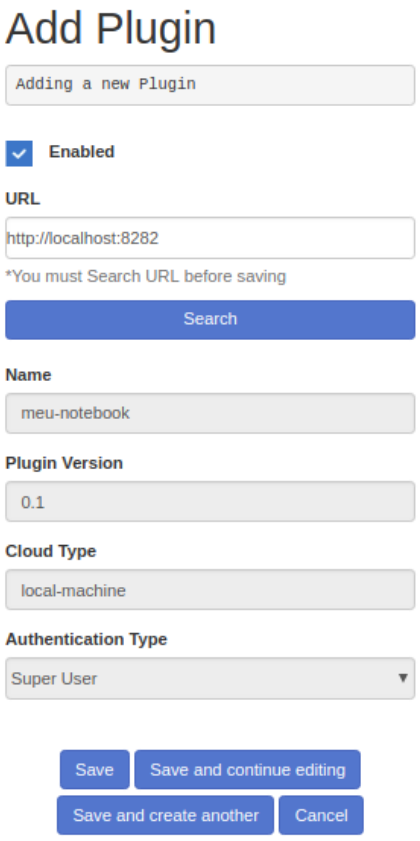

Figura 4.6: Tela para Adicionar Novos *Plugins*.

Após o estabelecimento da comunicação com o microsserviço, todas as futuras comunicações são feitas com conexão segura utilizando o protocolo HTTPS (HTTP sobre TLS) [49].

Price Tables

Esta opção exibe informações sobre a sincronização das tabelas de preço feitas pelo sistema. A cada hora o sistema verifica se houve alguma mudança nas tabelas de preço mantidas por cada um dos *Plugins*. Todavia, como pode ser visto na Figura 4.7, é possível solicitar uma sincronização forçada a qualquer momento, clicando no botão "*Force Synchronization*", para verificar se houve alguma atualização de tabela de preço para os *Plugins* existentes.

Para verificar se uma tabela de preço foi atualizada por um *Plugin*, não é necessário efetuar o *download* de toda a tabela de preço, pois os *Plugins* mantêm a informação da data da tabela de preço em uma URL de consulta específica, que não contém toda a tabela, reduzindo o tempo de resposta e o fluxo de informações. Essa URL de consulta é melhor detalhada na Sub-seção 4.3.3.

Como também é possível notar na Figura 4.7, para o *Plugin* do *Google Cloud Platform*, quando há alguma alteração no formato da tabela de preços, acusada pelo número de versão, essa informação é mostrada para o usuário, que precisa então atualizar seu *Plugin* para fazer o correto processamento da nova tabela de preços.

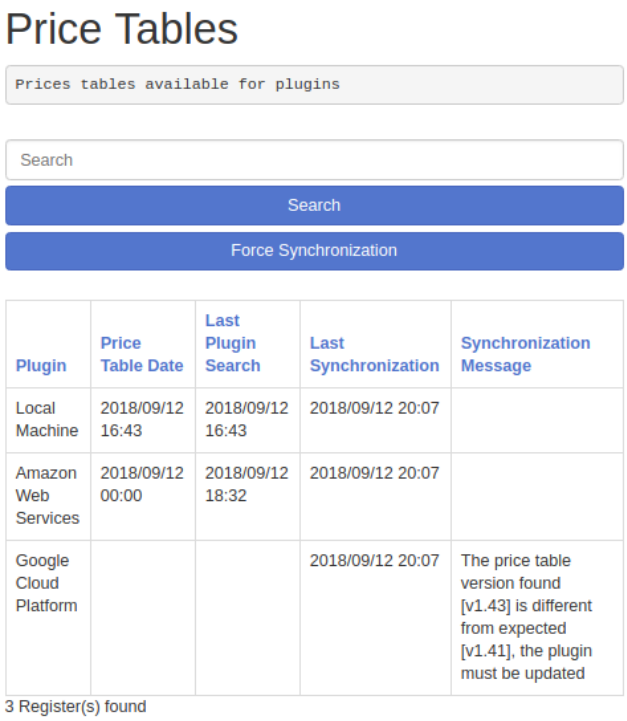

Figura 4.7: Tela do *Menu* de Tabelas de Preço.

4.2.3 *Menu*: *Credentials*

Na tela principal, exibida tanto para o usuário Cliente (na Figura 4.4), quanto para o usuário Administrador (na Figura 4.5), há o *menu Credentials*. A manutenção de credenciais é motivada pela necessidade de evitar a tarifação indireta, ou seja, a plataforma não faz a cobrança de seus usuários para efetuar um posterior repasse para o provedor de nuvem.

Assim, o usuário interessado em utilizar a plataforma de federação cadastra suas credenciais e o sistema faz a utilização destas para o provisionamento de recursos em nuvens externas, caso seja necessário, com as devidas garantias feitas por acordos prévios de SLAs na criação de suas tarefas. Desta forma, o usuário não necessita fazer qualquer tipo de pagamento para a utilização da plataforma de federação BioNimbuZ 2.

As credenciais adicionadas pelos usuários podem ser associadas a um grupo de usuários. Desta forma, a ideia de grupos não somente é usada para o compartilhamento de dados, mas também de credenciais.

O compartilhamento de credenciais se mostra útil, por exemplo, em equipes de cientistas de universidades que recebem créditos de nuvens externas, e que são compartilhados entre os vários cientistas e alunos. Assim, na Figura 4.8 podem ser vistas as relações possíveis entre três usuários e três credenciais. O primeiro dos usuários, o Usuário 1, possui as Credenciais A e B, sendo esta última compartilhada com um grupo de usuários contendo os Usuários 2 e 3. Assim, o Usuário 2, não possuidor de nenhuma credencial, pode utilizar a Credencial B. Quanto ao Usuário 3, este está incluso no mesmo grupo do Usuário 2, mas não compartilha sua Credencial C com o grupo, assim, ninguém mais além dele poderá utilizá-la.

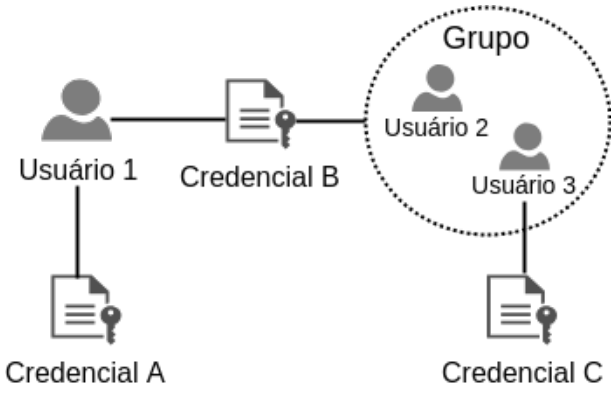

Figura 4.8: Grupos de Usuários e Compartilhamento de Credenciais.

O ambiente proposto por Kanewala *et al.* [50] aborda a questão do armazenamento seguro de credenciais de usuários para a sua utilização pelos chamados *Gateways*, que são fornecedores de mecanismos que reduzem a complexidade de acesso às diferentes camadas

do sistema, como por exemplo, o provisionamento de recursos de determinado provedor de nuvem sem a interação direta dos usuários. Além disso, são utilizados em ambientes que necessitam realizar o compartilhamento de credenciais entre grupos de usuários, o que se aplica à arquitetura apresentada neste trabalho.

Contudo, algumas medidas básicas devem ser estabelecidas antes de qualquer implementação do ambiente proposto por Kanewala *et al.*. Primeiramente, deve ser garantido que os arquivos do banco de dados estejam devidamente protegidos com os mecanismos de controle de acesso fornecidos pelo sistema operacional, e com os mecanismos de autenticação e autorização necessários para seu acesso. Em segundo lugar, deve ser garantido que a comunicação entre o Portal Web e o *Gateway*, que neste trabalho são fornecidos pela Camada de Aplicação e os *Plugins*, seja realizada com uma conexão segura TLS (*Transport Layer Security*) [49].

Assim sendo, na plataforma de federação BioNimbuZ 2, desenvolvida neste trabalho, são armazenados os seguintes tipos de credenciais:

- **Baseadas em delegação:** *tokens* de autorização gerados por mecanismos de autenticação única, *single sign-on*, fornecidos por padrões abertos como OAuth2[5], OpenID[6] ou SAML[7];
- **Baseadas em chaves *Secure Shell* (SSH):** a criação de chaves deixa de ser tarefa do usuário e passa a ser gerada pelo sistema responsável pelo seu armazenamento. Assim, solicitada a criação de uma chave SSH, o sistema faz a criação do par de chave pública/privada, e faz o armazenamento da chave privada em banco, e retorna para o usuário apenas a sua chave pública;
- **Credencial simples:** caso não seja possível a utilização de credenciais mais seguras, as credenciais são armazenadas puramente como usuário/senha para posterior utilização.

Os três tipos de credenciais citados são armazenados no banco de dados de forma criptografada por meio do algoritmo de chave simétrica AES (*Advanced Encryption Standard*) [51]. A chave do algoritmo é gerada com a utilização de uma palavra-passe, que obriga o sistema utilizador da chave a conhecer essa palavra-passe para criptografar ou descriptografar dados. Com isso, é possível favorecer a segurança para o armazenamento de dados sensíveis no ambiente apresentado. Desta forma, os sistemas que utilizarem a chave do AES, precisam ser informados sobre a palavra-passe, para então armazenar essa informação apenas em memória por medida de segurança.

[5]OAuth2, `https://oauth.net/2/`
[6]OpenID, `http://openid.net/`
[7]SAML, `http://xml.coverpages.org/saml.html`

O ambiente proposto por Kanewala *et al.* se difere da arquitetura apresentada por este trabalho por possuir *Gateways* que centralizam diversas funcionalidades. Assim sendo, neste trabalho estas funcionalidades são oferecidas de maneira distribuída na Camada de Aplicação e ainda, difere-se nas formas de armazenamento das credenciais, de provisionamento dos recursos computacionais e de fornecimento de mecanismos de autenticação específicos para cada provedor de nuvem.

Logo, acessando esse *menu*, os usuários podem cadastrar suas credenciais, de acordo com o fornecido pela nuvem utilizada (veja a Figura 4.9). No caso do GCP (acrônimo em inglês para *Google Cloud Platform*), é permitido criar a chamada "*Service Account Key*", que é uma chave em formato JSON que contém as credenciais do usuário. Já na AWS (acrônimo em inglês para *Amazon Web Services*), é permitido criar uma "*Access key ID*", e no momento de sua criação pode ser feito o *download* do arquivo no formato CSV, que também é aceito pela plataforma.

Após solicitar o armazenamento da chave, o registro criado é então armazenado de maneira criptografada no banco de dados, com um algoritmo de chave simétrica AES de 256 bits, para que possa ser recuperado posteriormente.

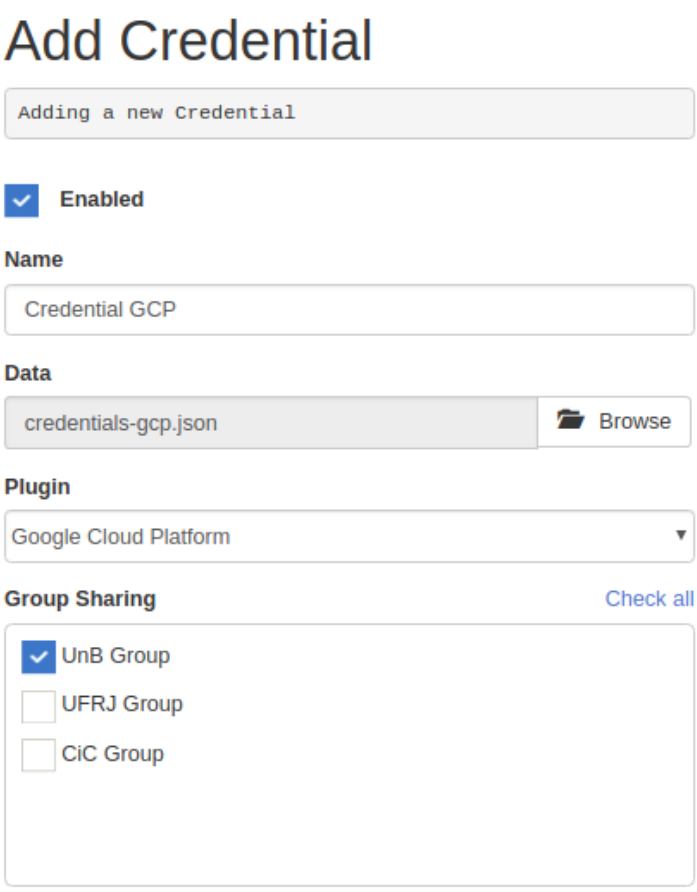

Figura 4.9: Tela para Adicionar Novas Credenciais.

Como pode ser visto na Figura 4.9, a credencial pode ser compartilhada com grupos de usuários. Os outros usuários participantes do grupo não podem recuperar a chave, somente o usuário que a criou. Os grupos de usuários são melhor detalhados na Subseção 4.2.4.

4.2.4 *Menu*: *Groups*

O conceito de grupos de usuários na arquitetura apresentada é fundamental para prover a característica de aplicações que necessitam do compartilhamento de resultados entre outros usuários (resultados de dados científicos, por exemplo). Desta forma, os dados são compartilhados somente entre membros de uma determinada equipe.

No *menu Groups*, os usuários podem criar grupos de pessoas com as quais deseja compartilhar credenciais para a execução de tarefas e o compartilhamento de arquivos. A Figura 4.10 apresenta a tela de um grupo previamente criado, caso o usuário seja o dono do grupo, ele pode convidar outros usuários informando o email destes, e também pode torná-los dono do grupo com a opção "*Make Owner*". Essa opção se faz necessária caso haja a necessidade de delegar a função de administrador do grupo para outros usuários.

Por fim, o usuário também tem a opção de abandonar o grupo clicando no botão "*Leave*" ou remover os usuários participantes, e todas os arquivos e as credenciais deixam de ser compartilhadas com o grupo.

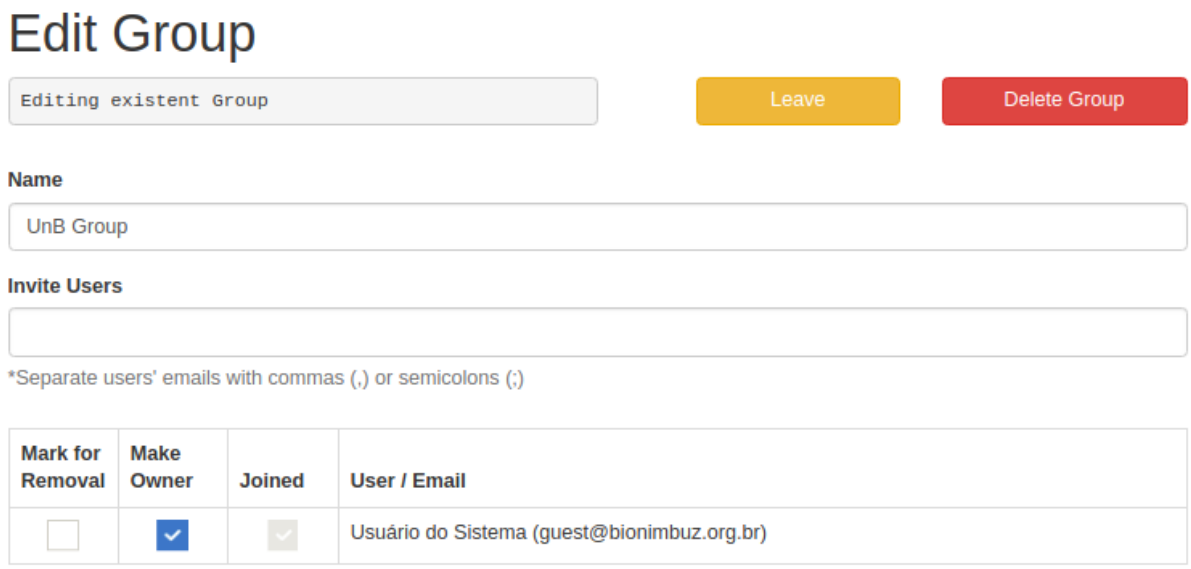

Figura 4.10: Tela para Editar Grupos de Usuários.

4.2.5 *Menu*: *Images*

Neste *menu* o usuário adiciona as imagens dos sistemas operacionais da nuvem selecionada. Com isso, uma imagem de sistema operacional, juntamente com o tipo de instância – que especifica a quantidade de núcleos e memória – permitem a criação de uma instância de máquina virtual na nuvem.

Na Figura 4.11, ao selecionar o *Plugin* de nuvem desejado, automaticamente o sistema entra em contato com o respectivo *Plugin* para solicitar as imagens públicas disponíveis.

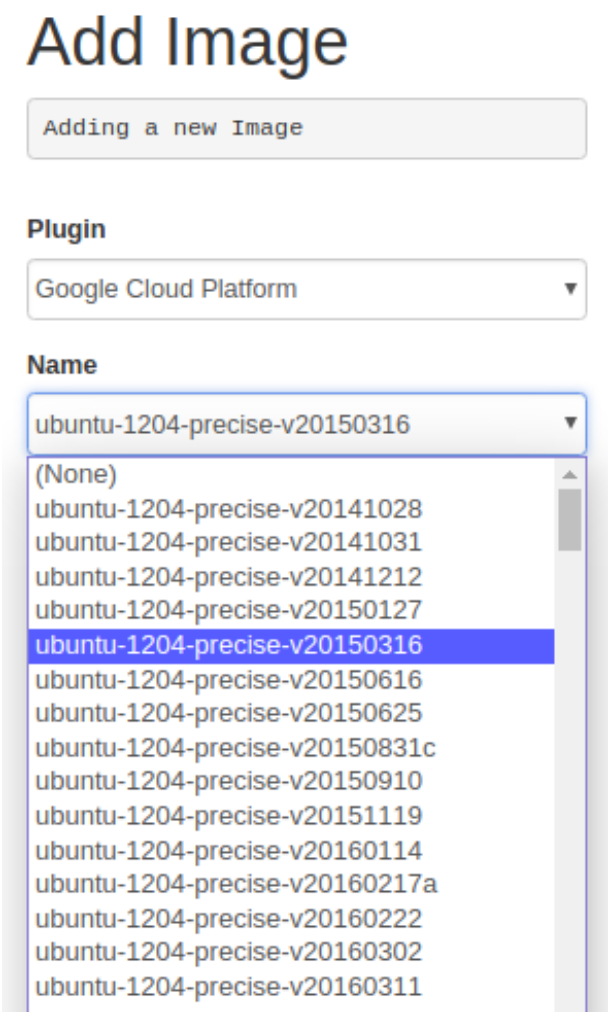

Figura 4.11: Tela para Adicionar Novas Imagens.

4.2.6 *Menu*: *Aplications*

Este *menu* permite a criação das aplicações usadas pelos usuários, que são utilizadas para a execução dos seus *workflows*, e é composto por dois *sub-menus*, um para a configuração do Coordenador de Tarefas, e o outro para a criação dos Executores de Tarefas que são melhor detalhados na Seção 4.5. Nessa criação, o usuário associa *scripts* de execução às

imagens das nuvens previamente cadastradas no *Menu Images*. Com isso, é importante ressaltar que nesse momento são criadas as aplicações e não as instâncias de nuvem, ou seja, são apenas *templates* usados quando as instâncias de VM são criadas.

O Coordenador de Tarefas, nesse momento de cadastro, difere-se do Executor de Tarefas apenas pelo fato de que só pode existir um no sistema, que são sempre usados para coordenarem os *workflows* dos usuários. Já os executores podem existir quantos forem necessários para o cadastro de aplicativos diversos. A tela vista na Figura 4.12 exibe os campos necessários para a criação de um Executor, que são eles:

- ***Name*:** o nome da aplicação a ser utilizada pelo usuário que faz a criação dos *workflows* de tarefas;
- ***Startup Script*:** este campo foi determinante para suportar a utilização de qualquer aplicação pela plataforma, pois explora uma característica de grandes nuvens como a AWS e o GCP, de executar *scripts* de inicialização antes que a VM torne-se operacional. Assim que a instância de VM é criada na nuvem, o *script* definido neste campo é executado como super usuário, permitindo por exemplo a utilização de gerenciadores de pacote como o Apt no Ubuntu (*apt-get install bowtie*). Assim sendo, os processos Executores possibilitam a utilização de qualquer aplicação, desde que seja definida corretamente sua instalação neste campo. Com isso, como mostrado na Figura 4.12, neste campo é feita a instalação do Java 8 que é a dependência do microsserviço de execução de tarefas, detalhado na Seção 4.5, e em seguida sua devida execução;
- ***Enable Execution Script*:** habilitando esta opção, o campo *Execution Script* é exibido para edição;
- ***Execution Script*:** este campo, de forma análoga ao *Startup Script*, é utilizado pelo microsserviço de execução de tarefas, e é executado assim que o mesmo se inicia na instância da VM criada;
- ***Command Line*:** este campo reflete o comando que é executado pelo microsserviço de execução de tarefas na instância da VM criada. Quando a opção *Enable Execution Script* está selecionada, o microsserviço de execução cria um *script* com o nome "*application.sh*" com o conteúdo do campo "*Execution Script*", e caso não esteja selecionada, é possível informar um comando já existente na máquina. Além do comando a ser executado, é possível a utilização dos seguintes marcadores: {i} para arquivos de entrada, {o} para arquivos de saída, e {a} quando a aplicação possibilitar informar argumentos extras. Apenas o marcador {a} pode ser adicionado uma única vez, assim as demais utilizações são ignoradas;

- ***TCP/UDP Port Rules for Firewall*:** as portas definidas nestes campos são liberadas para entrada para a instância criada. O trabalho de abertura de portas é feito pelo *Plugin* de nuvem ao criar a instância, ou seja, se liberada a porta 80 para TCP, a instância pode então receber requisições HTTP que por padrão escutam nessa porta, permitindo a utilização de servidores HTTP como o Apache;

- ***Images*:** neste campo são associadas as imagens a serem utilizadas quando a instância for criada na nuvem. Apenas uma imagem por nuvem pode ser associada ao executor, para que este campo fique transparente ao usuário, e o mesmo não precise informar ao criar seu *workflow*.

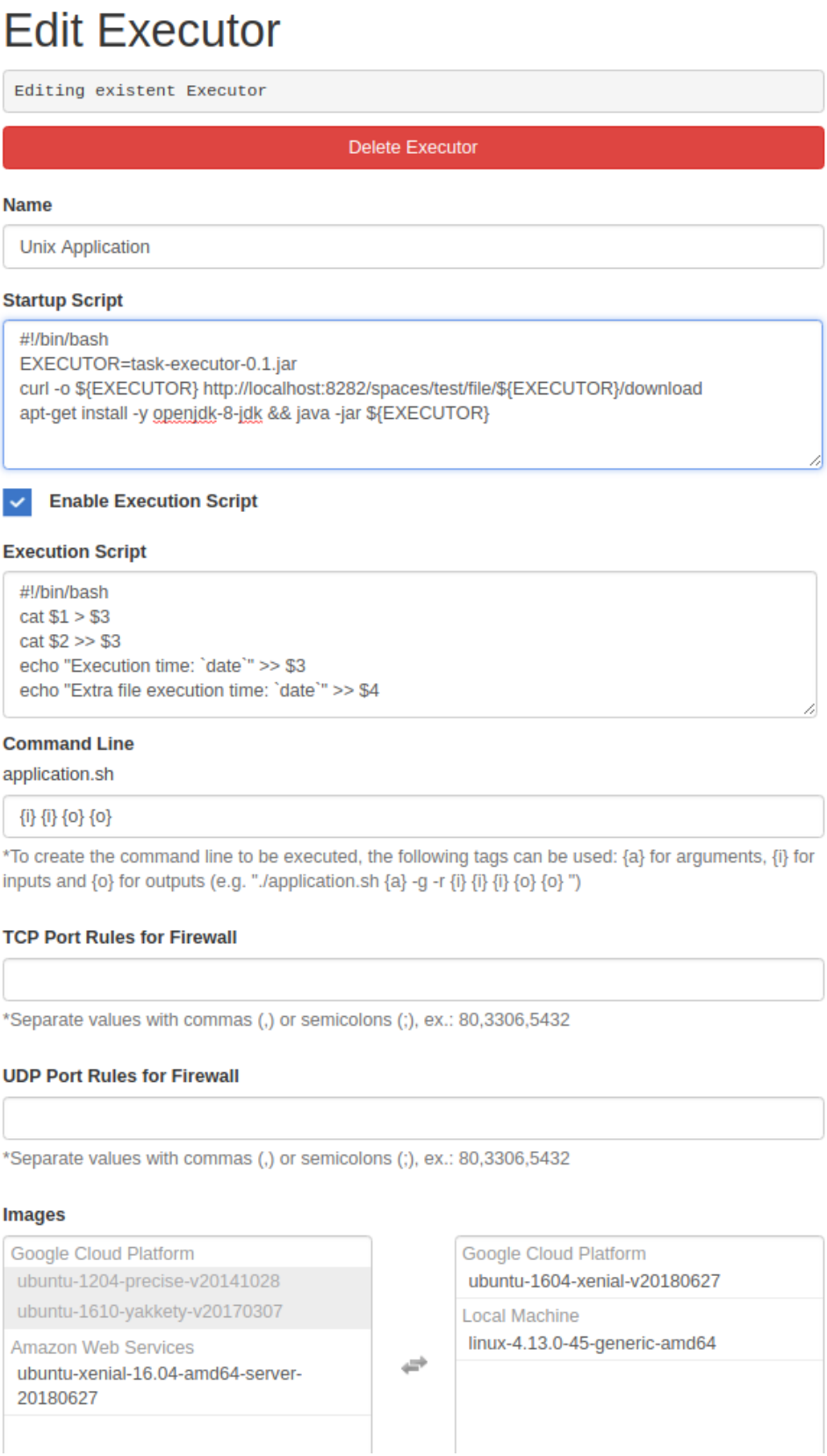

Figura 4.12: Tela para Editar o Executor de Tarefas.

4.2.7 *Menu*: *Storage*

Este *menu* contém outros dois *sub-menus*, e permite armazenar os arquivos a serem utilizados pelas aplicações dos usuários, sendo eles os seguintes:

Spaces

Os *Spaces* são abstrações de dispositivos de armazenamentos criados na nuvem, nos quais os arquivos dos usuários são mantidos. Assim como as credenciais, eles também podem ser compartilhados com grupos de usuários, permitindo a estes usuários fazerem apenas *upload*/*download* de arquivos, e não a manutenção destes espaços.

Na Figura 4.13, na criação de um espaço, ao selecionar o *Plugin* de nuvem desejado, as regiões possíveis de armazenamento e seus respectivos preços são disponibilizadas para a seleção.

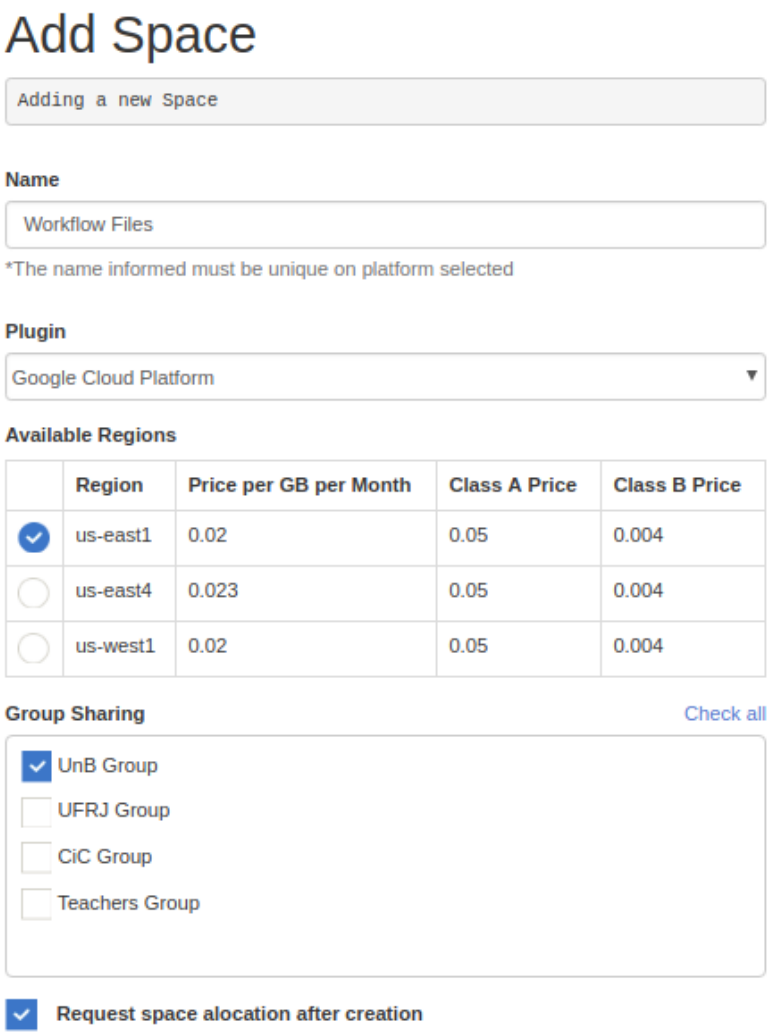

Figura 4.13: Tela para Adicionar um Espaço.

Files

Na arquitetura proposta por este trabalho, o mecanismo de armazenamento de arquivos foi desenvolvido de maneira a evitar que o arquivo enviado para o espaço de armazenamento na nuvem trafegue de forma indireta, ou seja, não há necessidade de enviar o arquivo para o Serviço Web, para que depois seja enviado para a nuvem na qual o espaço foi criado. Dessa forma, o usuário faz o envio direto para o seu espaço de armazenamento, evitando também que a infraestrutura destinada à Camada de Aplicação armazene arquivos dos usuários.

Assim, o armazenamento de um arquivo no espaço selecionado, que foi previamente criado no *Menu Spaces*, segue o processo descrito na Figura 4.14. Primeiramente em (1), o usuário solicita o *download* ou o *upload* de um arquivo, por meio do navegador, à Camada de Aplicação em (2), que é a camada que tem acesso às credenciais de nuvem dos usuários, solicita à nuvem possuidora do recurso, um *token* de autorização para a operação com tempo de vida curto em (3), e somente em (4), após autorizada a operação, utilizando o *token* previamente adquirido, o navegador do usuário faz o *upload* diretamente para a nuvem sem que tenha que enviar para a Camada de Aplicação. Em caso de *download* em (5), o navegador faz a requisição do arquivo diretamente à nuvem utilizando o *token* de autorização adquirido no passo (3).

Esse processo de *upload*/*download* de arquivos com aquisição de *tokens* temporários também é utilizado pelos microsserviços de execução de tarefas, descritos na Seção 4.5.

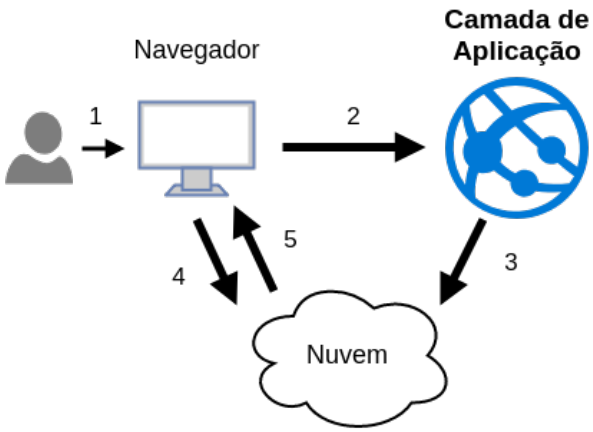

Figura 4.14: Processo de *Upload*/*Download* de Arquivos.

Na Figura 4.15 é possível ver a tela de *upload* de arquivos no espaço previamente criado. A opção de "*Upload File*" quando desabilitada, somente permite a utilização do campo "Public Url", utilizado para *download* direto sem utilização de *tokens* de autorização.

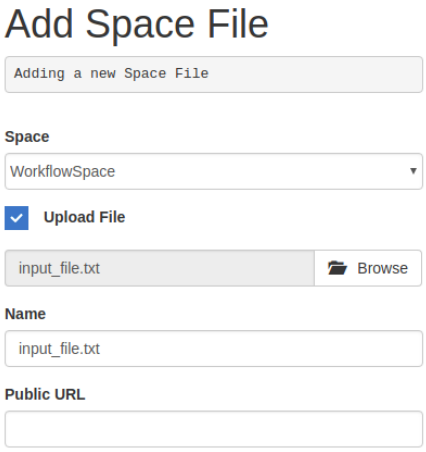

Figura 4.15: *Upload* de Arquivo no Espaço.

Por outro lado, na Figura 4.16 é possível ver o processo de *download* de arquivo do espaço. Um comportamento importante adicionado ao processo de *upload*/*download* de arquivos foi a criação dos Nomes Virtuais, como visto no campo não editável exibido por "Virtual Name". O Nome Virtual é utilizado para nomear o arquivo na nuvem onde encontra-se armazenado, e o nome informado pelo usuário é armazenado em banco de dados.

Dessa forma, a criação de diretórios é facilmente abstraída pela Camada de Aplicação, necessitando apenas que os diretórios sejam criados em banco de dados e não na nuvem, agregando os arquivos com seus nomes virtuais. Além do mais, o Nome Virtual dificulta buscas não autorizadas nos arquivos dos usuários e associações com seus *workflows*.

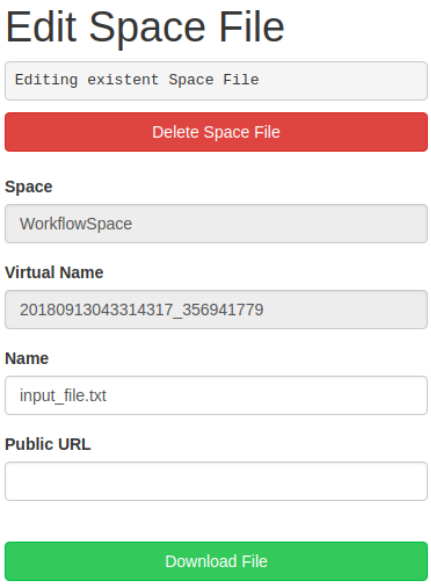

Figura 4.16: *Download* de Arquivo do Espaço.

4.2.8 *Menu*: *Executions*

Pode-ser dizer que este *menu* contém a principal funcionalidade do sistema, que é a de execução de aplicações e de *workflows*. Ele contém os dois seguintes *sub-menus*: *Instances* e *Workflows*.

Instances

Este *sub-menu* possibilita aos usuários a criação de aplicações em uma determinada nuvem. Desta forma, ao solicitar a execução de uma aplicação, a Camada de Aplicação solicita ao *Plugin* da nuvem selecionada a criação de uma VM, que em seguida faz a devida execução da aplicação.

Uma instância representa uma instância de VM em execução na nuvem selecionada pelo usuário, e que executa a aplicação associada à ela, como pode ser visto na Figura 4.17.

Figura 4.17: Campo para a Seleção de um Executor.

Ao selecionar um Executor, são exibidas duas abas, que configuram os parâmetros de execução da instância na nuvem, as quais são:

- ***Application Arguments***:

  Nesta aba, como visto na Figura 4.18, são informados os parâmetros de execução da aplicação. A quantidade de *Inputs* e *Outputs* disponibilizados para que o usuário configure é relativa a quantidade de marcadores {i} e {o} exibidos no campo "*Command Line*", respectivamente. O campo não editável "*Command Line Result*" exibe o comando que é executado pelo microsserviço de execução de tarefas;

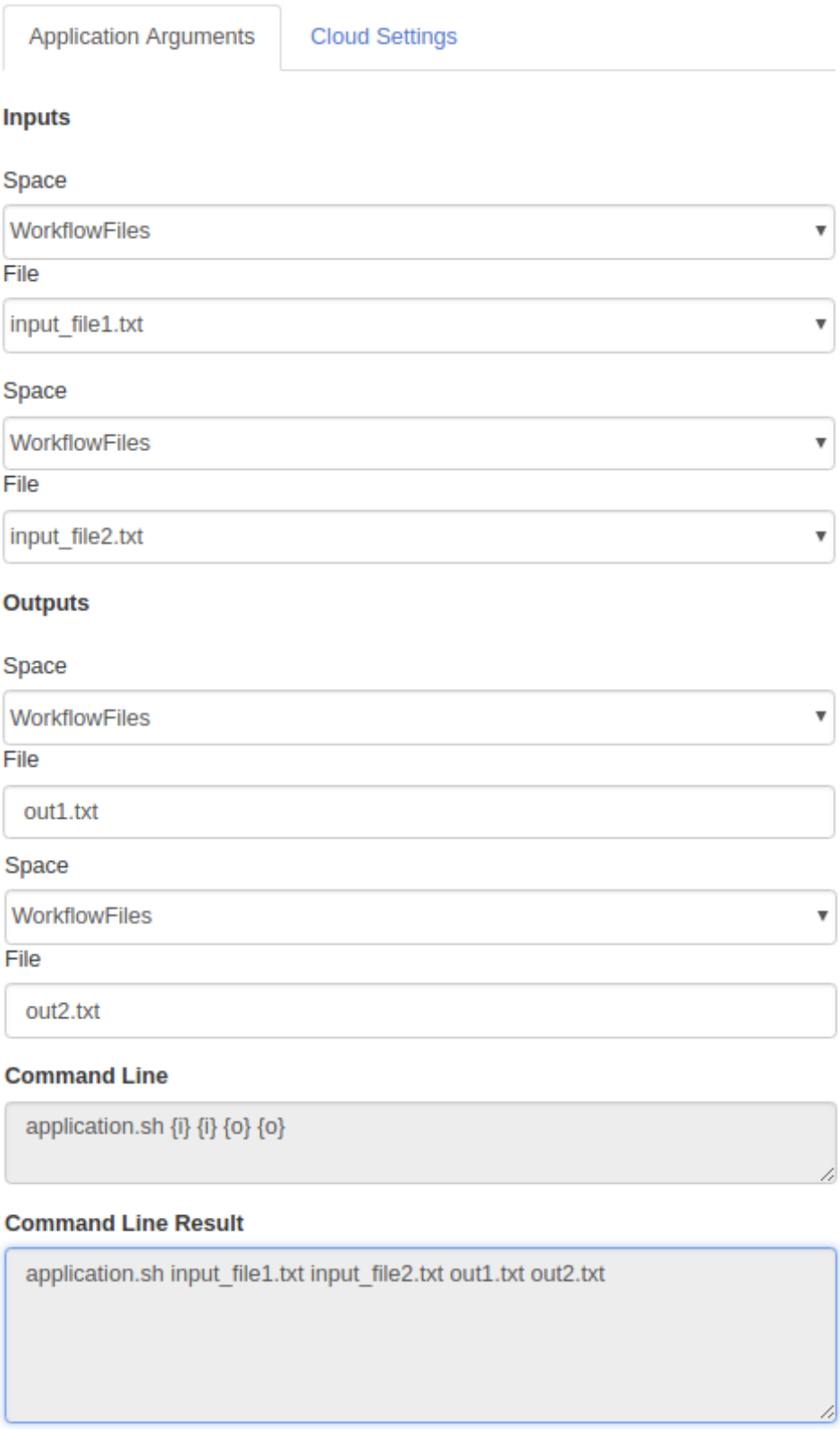

Figura 4.18: Tela de Configuração de Argumentos da Aplicação.

- ***Cloud Settings*:**

  A última configuração a ser feita antes que a instância seja criada refere-se as configurações de nuvem, permitindo ao usuário escolher o local de execução e o tipo de instância a ser utilizada, com diferentes quantidades de núcleos e de memória. Como pode ser visto na Figura 4.19, o usuário informa no campo "*Credential Usage*", a ordem de preferência de utilização das suas credenciais de nuvem. Assim, se por exemplo escolher "*Shared First*", entre todas as suas credenciais, incluindo as que estão compartilhadas, a tentativa de criação da instância é feita utilizando-se, preferencialmente, as credenciais compartilhadas, e caso não o consiga, por último são utilizadas suas próprias credenciais.

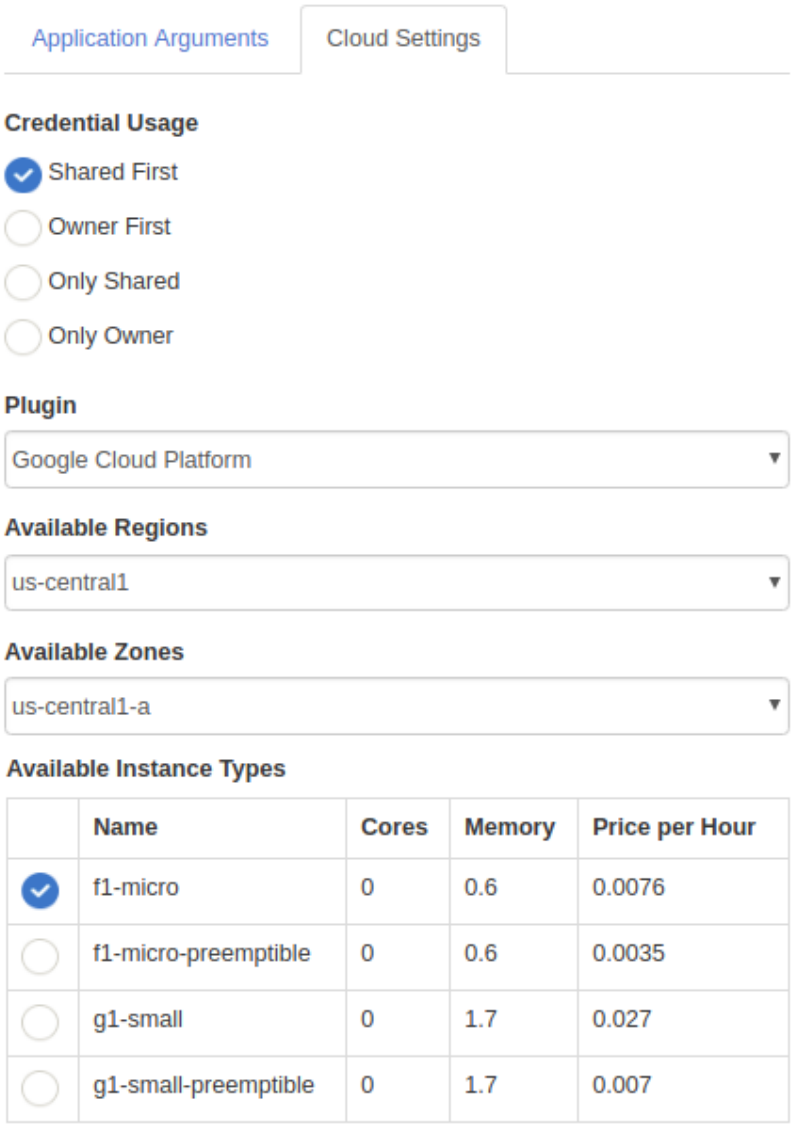

Figura 4.19: Configuração da Nuvem Utilizada.

Workflows

A opção de criação de *workflows* permite ao usuário criar dependências entre a execução de diferentes instâncias, podendo utilizar as saídas de tarefas executadas anteriormente como entradas de tarefas posteriores.

Na Figura 4.20 é possível ver um exemplo de *workflow* que contém duas tarefas que são executadas em paralelo partindo de "*Start*", as quais são: a "*Unix Application*" e a "MacOS Application". Dessa forma, a tarefa "*Output Parser CSV*" aguarda até que todas as tarefas anteriores sejam executadas para então iniciar a nova tarefa.

Ainda na Figura 4.20, é possível notar a caixa de ferramentas utilizada para a edição do *workflow*. Em (1), é permitido arrastar e soltar um novo nó dentro do modelo; em (2) e (3) é possível aumentar e diminuir o *zoom* aplicado ao modelo; em (4) o usuário pode analisar, sempre que necessário, se o modelo possui erros de conexão, verificando a existência de nós que não alcançam o fim, ou nós que nunca são iniciados por não ter o nó de início como seu ascendente, e ainda se existe algum ciclo entre os nós como visto na Figura 4.21; em (5), o usuário pode excluir o nó criado ou a conexão feita entre os nós; por último, em (6), ao selecionar um nó, clicando nessa opção, o usuário pode configurar os parâmetros de execução para a instância.

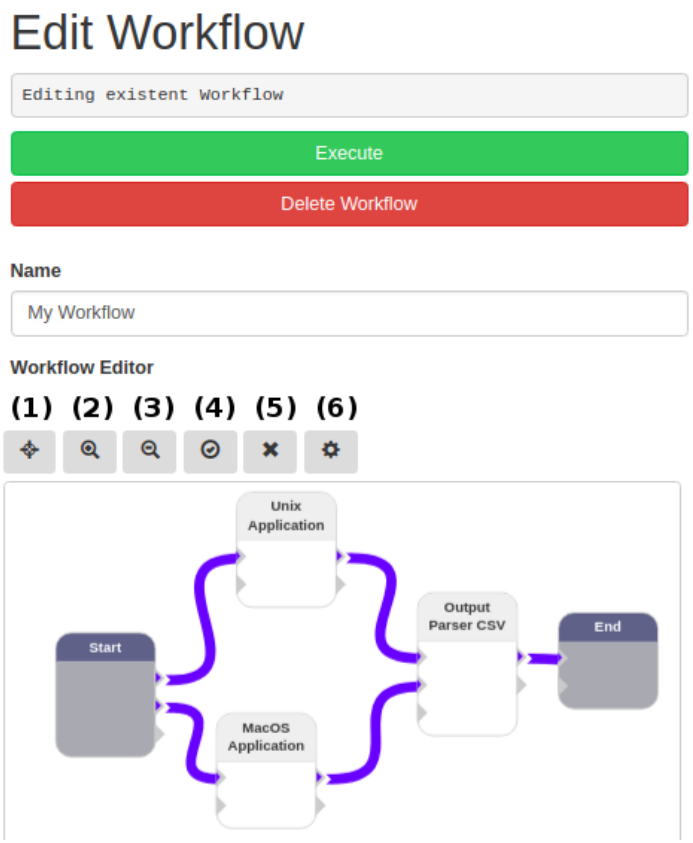

Figura 4.20: Tela para a Edição de *Workflow*.

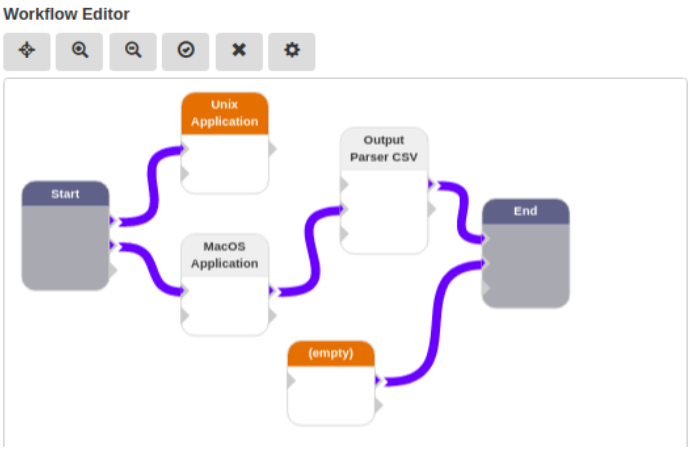

Figura 4.21: *Workflow* com Desconexões.

Assim sendo, na Figura 4.22, é possível ver um *workflow* em execução onde a tarefa "*Unix Application*" finalizou sua execução com sucesso e "*MacOS Application*" não executou com sucesso, por tanto a tarefa "Output Parser CSV" não será executada e o *workflow* será parado, solicitando que as tarefas em execução, no caso a tarefa "*Network Analyzer*", também parem de executar. Toda a análise de execução é feita pelo módulo de coordenação de tarefas, o qual analisa os grafos de dependências assim que cada tarefa é finalizada, e solicita a execução das próximas tarefas, ou finaliza as existentes.

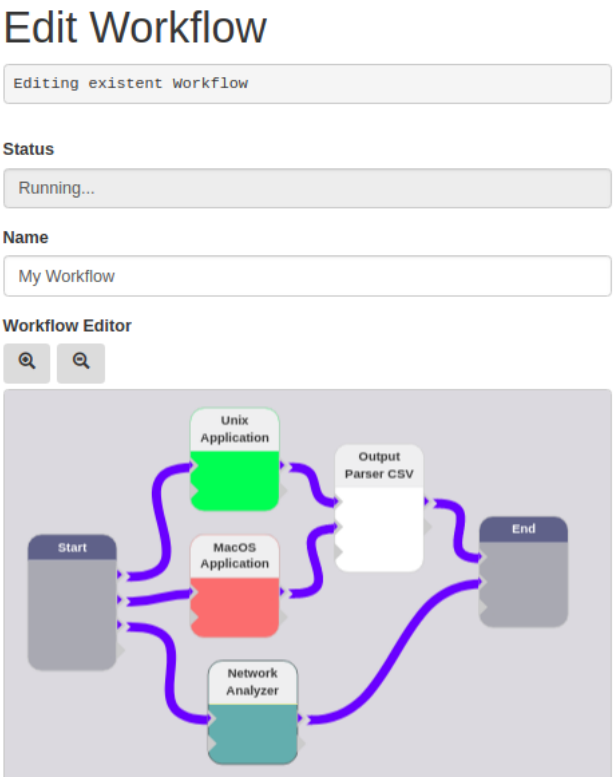

Figura 4.22: *Workflow* em Execução.

4.2.9 *Menu*: *Settings*

Por fim, no *menu Settings*, o usuário informa as configurações globais do sistema. A principal das configurações é o endereço de acesso externo à Camada de Aplicação, que é utilizado para a comunicação dos Coordenadores e dos Executores, que são vistos nas Seções 4.5 e 4.4.

4.3 Camada de Federação

A Camada de Federação é responsável por garantir a integração dos diferentes componentes do sistema que podem estar situados em diversas nuvens, formando uma federação de nuvens. A federação é obtida pela utilização do ambiente ZooKeeper [52] e, principalmente, pelos *Plugins* que implementam as funcionalidades inerentes a determinado provedor de nuvem.

Com o ambiente fornecido pelo ZooKeeper, são mantidos dados de forma distribuída entre os Coordenadores e a Camada de Aplicação, de maneira que possam ser consultados e acompanhados em tempo real. Tais dados são, por exemplo, o estado de execução das tarefas, os endereços das máquinas monitoradas pelos Coordenadores, consumo de recursos e os metadados dos arquivos de entrada/saída das tarefas. Por outro lado, a criação dos *Plugins* de federação é necessária para que a plataforma se comunique com os diferentes provedores de nuvem participantes da federação.

Os *Plugins* são implementados de acordo com a abordagem de microsserviços, e implementam funcionalidades básicas e padronizadas de comunicação, de maneira que não seja necessário que a Camada de Aplicação tenha que ser interrompida para a atualização e a execução deles. Dessa forma, todas as responsabilidades de comunicação intrínsecas a determinada nuvem são transferidas para o microsserviço implementado para aquela nuvem. Logo, os *Plugins* não são executados dentro da nuvem para a qual são especializados, evitando assim o consumo de recursos das credenciais cadastradas. Assim, os *Plugins* possuem os seguintes serviços e controladores:

- **Serviço de Descoberta:** esse serviço compila informações de tabelas de preços, e tipos de recursos de computação e de armazenamento de determinada plataforma de nuvem. As informações estão sempre em memória para uma consulta mais eficiente pela Camada de Aplicação, visto que esses dados não são sensíveis aos usuários;
- **Serviço de Provisionamento:** neste serviço é fornecido à Camada de Aplicação a possibilidade de alocar e desalocar máquinas de determinada nuvem. Também é responsável por provisionar espaços para *upload*/*download* de arquivos de entrada/saída das tarefas definidas na Camada de Aplicação;

- **Controlador de Credenciais:** este controlador interpreta as informações de credenciais enviadas pela Camada de Aplicação por meio dos arquivos JSON, na forma de chave-valor, e traduz para o mecanismo de segurança utilizado pela nuvem sob seu controle.

A Camada de Federação com os *Plugins* foi desenvolvida utilizando o *framework Spring Boot* [53], que permite criar serviços web em Java que são auto-contidos, ou seja, o pacote final gerado, do tipo *.jar, já contém o servidor web necessário para atender as requisições web solicitadas por seus usuários, sem a necessidade de ser integrado em outros ambientes, como por exemplo o Apache Tomcat[8].

Para sua execução é necessário apenas executar o pacote final com o comando "java -jar pacote.jar", o que facilita sua integração em ambientes de *containers*, como por exemplo o Kubernetes[9].

Além disso, o ambiente fornecido pelo *Spring Boot* [53] oferece recursos que facilitam a construção de APIs com arquitetura de comunicação REST [43], e troca de JSONs, que foi usado neste trabalho.

A Figura 4.23 demonstra a estrutura de comunicação entre os componentes da arquitetura para estabelecimento da federação, com a Camada de Aplicação se comunicando com os *Plugins* por meio de pacotes JSON, e estes sendo responsáveis por provisionar recursos dos diferentes provedores de nuvem.

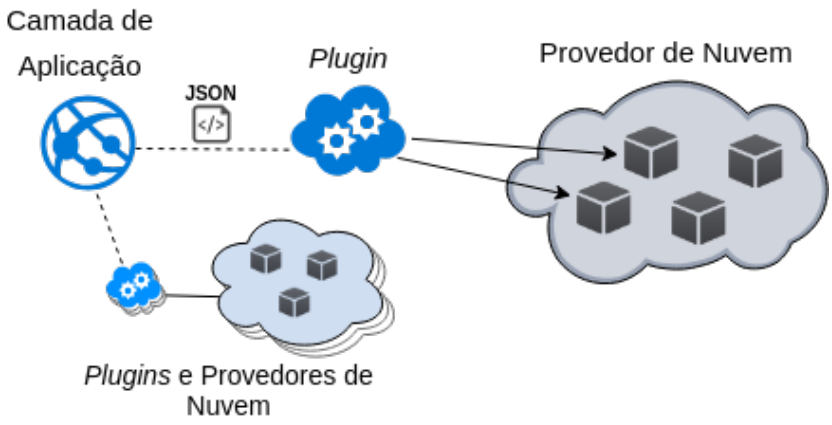

Figura 4.23: Estrutura de Comunicação com os Microsserviços.

Nas próximas sub-seções são melhor detalhadas as interfaces REST que devem ser fornecidas por um *Plugin*, para que o mesmo possa ser integrado à federação. Além disso, para algumas APIs é especificada a necessidade de utilização dos seguintes cabeçalhos:

[8]Apache Tomcat, `http://tomcat.apache.org/`
[9]Kubernetes, `https://kubernetes.io/`

- **APIVersion**: este cabeçalho é enviado para que o *Plugin* possa fazer o controle de retrocompatibilidade com versões antigas;
- **Authorization**: cabeçalho que contém informações de autenticação relativas ao usuário, para adquirir o recurso desejado [54]. Em algumas nuvens esse valor contém os *tokens* de autorização para a criação das instâncias e para o *upload*/*download* de arquivos;
- **AuthorizationId**: este cabeçalho, quando necessário, é utilizado para identificar o proprietário do recurso requisitado.

4.3.1 Informações

Esta URL é utilizada para estabelecer um primeiro contato com o *Plugin*, e retornar suas informações básicas para o armazenamento pela Camada de Aplicação.

Como pode ser visto no Conteúdo de Resposta da Tabela 4.1, o *Plugin*, quando cabível, retorna o escopo do recurso requisitado (campos *instanceWriteScope*, *instanceReadScope*, *storageWriteScope*, *storageReadScope*), e são utilizados para requisitar os *tokens* de autorização.

Tabela 4.1: Informações do *Plugin*.

Método HTTP e Caminho	`GET: /info`
Conteúdo de Resposta	`{ "message":"OK", "content":{ "name":"Google Cloud Platform", "apiVersion":"0.1", "pluginVersion":"0.1", "cloudType":"google-compute-engine", "authType":"AUTH_BEARER_TOKEN", "instanceWriteScope": "https://www.googleapis.com/auth/compute", "instanceReadScope": "https://www.googleapis.com/auth/compute.readonly", "storageWriteScope": "https://www.googleapis.com/auth/devstorage.read_write", "storageReadScope": "https://www.googleapis.com/auth/devstorage.read_only" } }`

4.3.2 Imagens

Nesta URL são retornadas as imagens dos sistemas operacionais que estão disponíveis na nuvem para a criação da instância, como pode ser visto na Tabela 4.2.

Tabela 4.2: Imagens de Sistemas Operacionais.

Método HTTP e Caminho	`GET: /images`
Cabeçalhos	`APIVersion, Authorization, AuthorizationId`
Conteúdo de Resposta	`{ "message":"OK", "content":[{ "id":"15508054221909398824", "name":"ubuntu-1204-precise-v20141028", "url":"https://www.googleapis.com/compute/v1/projects/ubuntu-os-cloud/global/images/ubuntu-1204-precise-v20141028" }, { "id":"7734210992432622498", "name":"ubuntu-minimal-1804-bionic-v20180814", "url":"https://www.googleapis.com/compute/v1/projects/ubuntu-os-cloud/global/images/ubuntu-minimal-1804-bionic-v20180814" }] }`

4.3.3 Tabela de Preços

Verificação de *Status*

A URL de *status* é utilizada para verificar de maneira mais rápida se houve atualizações na tabela de preço, consumindo menos tráfego de rede, como pode ser visto na Tabela 4.3.

Tabela 4.3: *Status* da Tabela de Preço.

Método HTTP e Caminho	`GET: /pricing/status`
Cabeçalhos	`APIVersion`
Conteúdo de Resposta	`{ "message":"OK", "content":{ "lastSearch":"Sep 13, 2018 8:12:25 PM", "status":"OK", "errorMessage":"OK" } }`

Recuperação da Tabela Completa

Consultando esta URL é retornada toda a tabela de preço, juntamente com seu *status*, contendo todos os tipos de máquina e sua disponibilidade nas diferentes regiões (campo *listInstancePricing*). Também na mesma resposta, e quando cabível, o *Plugin* pode retornar os preços dos dispositivos de armazenamento e suas regiões (campo *listStoragePricing*), como visto na Tabela 4.4.

Tabela 4.4: Conteúdo da Tabela de Preço.

Método HTTP e Caminho	`GET: /pricing`
Cabeçalhos	`APIVersion`
Conteúdo de Resposta	`{ "message":"OK", "content":{ "status":{ "lastSearch":"Sep 13, 2018 8:12:25 PM", "status":"OK", "errorMessage":"OK" }, "price":{ "lastUpdate":"Sep 12, 2018 12:00:00 AM", "listInstancePricing":{ "f1-micro":{ "name":"f1-micro", "cores":0, "memory":0.6, "listRegionPricing":{ "asia-southeast":0.0092, "northamerica-northeast1":0.0084 } }, "n1-standard-4":{"...": "..."} }, "listStoragePricing":{ "southamerica-east1":{ "region":"southamerica-east1", "price":0.035, "classAPrice":0.05 }, "us-west1":{"...": "..."} } } } }`

4.3.4 Armazenamento

Criando Espaços

Para fazer uma requisição à URL, é necessário enviar um objeto JSON contendo o nome do espaço e sua região, como visto na Tabela 4.5.

Tabela 4.5: Criação de Espaços.

Método HTTP e Caminho	`POST: /storage/spaces`
Cabeçalhos	`APIVersion, Authorization, AuthorizationId`
Conteúdo de Requisição	`{ "name":"bionimbuz_workflow_us_central1", "region":"us-central1" }`
Conteúdo de Resposta	`{ "message":"OK", "content":{ "name":"bionimbuz_workflow_us_central1", "region":"us-central1" } }`

Removendo Espaços

Diferentemente da criação do espaço, nesta URL é necessário apenas enviar o nome do espaço para sua exclusão, conforme apresentado na Tabela 4.6.

Tabela 4.6: Remoção de Espaços.

Método HTTP e Caminho	`DELETE: /storage/spaces/{name}`
Variáveis do Caminho	• *name* : nome do espaço a ser removido.
Cabeçalhos	`APIVersion, Authorization, AuthorizationId`
Conteúdo de Resposta	`{ "message":"OK", "content":true }`

Solicitando URL para *Upload* de Arquivo

Requisitando esta URL, o *Plugin* deve ser capaz de construir a URL específica do provedor de nuvem para que o arquivo seja enviado por meio de HTTP, veja a Tabela 4.7.

Tabela 4.7: *Upload* de Arquivo.

Método HTTP e Caminho	`GET: /storage/spaces/{name}/files/{file}/upload/url`
Variáveis do Caminho	• *name*: nome do espaço que deverá armazenar o arquivo; • *file*: nome do arquivo a ser enviado.
Cabeçalhos	`APIVersion`
Conteúdo de Resposta	`{ "message":"OK", "content":{ "fileName":"20180902161910883_662067847", "spaceName":"bionimbuz_workflow_us_central1", "url":"https://www.googleapis.com/upload/storage/v1/b/bionimbuz_workflow_us_central1/o?uploadType\u003dmedia\u0026name\u003d20180902161910883_662067847", "method":"POST" } }`

Solicitando URL para *Download* de Arquivo

Da mesma maneira que o *upload*, o *Plugin* deve ser capaz de construir a URL específica do provedor de nuvem para que se possa efetuar o *download* do arquivo. Assim sendo, tanto a URL de *upload* quanto a de *download* são utilizadas indiretamente como mostradas na Figura 4.14, e uma requisição de autorização prévia deve ser realizada para acessar o recurso, veja a Tabela 4.8.

Tabela 4.8: *Download* de Arquivo.

Método HTTP e Caminho	`GET: /storage/spaces/{name}/files/{file}/download/url`
Variáveis do Caminho	• *name*: nome do espaço que contém o arquivo; • *file*: nome do arquivo requisitado.
Cabeçalhos	`APIVersion`
Conteúdo de Resposta	`{ "message":"OK", "content":{ "fileName":"20180902161910883_662067847", "spaceName":"bionimbuz_workflow_us_central1", "url":"https://www.googleapis.com/storage/v1/b/bionimbuz_workflow_us_central1/o/20180902161910883_662067847?alt\u003dmedia", "method":"GET" } }`

4.3.5 Computação

Solicitando as Zonas de Computação Disponíveis para uma Região

Alguns recursos de computação necessitam, além de definirem a região de criação, também precisam definir a sua zona, e essa informação, em algumas nuvens, só pode ser recuperada sob-demanda. Assim, esta URL retorna as zonas de determinada região quando solicitada, conforme apresentado na Tabela 4.9.

Tabela 4.9: Listagem de Zonas de uma Região.

Método HTTP e Caminho	`GET: /computing/regions/{name}/zones`
Variáveis do Caminho	• *name*: nome da região solicitada.
Cabeçalhos	`APIVersion, Authorization, AuthorizationId`
Conteúdo de Resposta	`{ "message":"OK", "content":[ { "name":"us-east1-b" }, { "name":"us-east1-c" } ] }`

Criando uma Instância na Nuvem

Como pode ser visto no Conteúdo de Requisição da URL na Tabela 4.10, para a criação de uma instância é basicamente necessário fornecer as informações do tipo a ser utilizado, e a imagem do sistema operacional. Além disso, também pode ser requisitada a liberação de portas TCP e UDP, que deve ser implementada pelo *Plugin* de nuvem.

A sua resposta deve conter, principalmente, o IP externo atribuído à instância criada, para acesso dos coordenadores de tarefa; e o nome da instância que identifica a máquina na nuvem, para que a mesma possa ser removida pela Camada de Aplicação quando necessário.

Tabela 4.10: Criação de Instâncias.

Método HTTP e Caminho	`POST: /computing/instances`
Cabeçalhos	`APIVersion, Authorization, AuthorizationId`
Conteúdo de Requisição	`{ "startupScript":"apt-get update \u0026\u0026 apt-get install -y apache2 \u0026\u0026 hostname \u003e /var/www/index.html", "region":"us-east1", "zone":"us-east1-b", "imageUrl":"https://www.googleapis.com/compute/v1/projects/ubuntu-os-cloud/global/images/ubuntu-1604-xenial-v20170919", "type":"f1-micro", "firewallTcpPorts":[8080, 80], "firewallUdpPorts": [] }`
Conteúdo de Resposta	`{ "message":"OK", "content":{ "id":"5047256435314307634", "name":"bionimbuz-instance-1", "machineType":"f1-micro", "creationDate":"Sep 13, 2018 10:20:30 PM", "internalIp":"10.142.0.4", "externalIp":"35.185.47.153", "startupScript":"apt-get update \u0026\u0026 apt-get install -y apache2 \u0026\u0026 hostname \u003e /var/www/index.html", "region":"us-east1", "zone":"us-east1-b", "imageUrl":"https://www.googleapis.com/compute/v1/projects/ubuntu-os-cloud/global/images/ubuntu-1604-xenial-v20170919", "type":"f1-micro", "firewallTcpPorts":[8080, 80] } }`

Requisitando Informações de uma Instância Existente

Esta URL deve retornar as informações básicas de uma instância já existente e em execução, conforme mostrado na Tabela 4.11.

Tabela 4.11: Informações de uma Instância.

Método HTTP e Caminho	`GET:` `/computing/regions/{region}/zones/{zone}/instances/{name}`
Variáveis do Caminho	• *region*: região onde a instância foi criada; • *zone*: nome da zona onde a instância foi criada. • *name*: nome da região solicitada.
Cabeçalhos	`APIVersion, Authorization, AuthorizationId`
Conteúdo de Resposta	`{ "message":"OK", "content":{ "id":"230034877195923222", "name":"bionimbuz-instance-1", "machineType":"f1-micro", "creationDate":"Sep 13, 2018 10:42:18 PM", "internalIp":"10.142.0.3", "externalIp":"35.231.64.248" } }`

Removendo uma Instância Existente

Esta URL deve finalizar e remover a instância em execução na nuvem, conforme indicado na Tabela 4.12.

Tabela 4.12: Remoção de Instâncias.

Método HTTP e Caminho	`DELETE:` `/computing/regions/{region}/zones/{zone}/instances/{name}`
Variáveis do Caminho	• *region*: região onde a instância foi criada; • *zone*: nome da zona onde a instância foi criada; • *name*: nome que identifica a instância na nuvem.
Cabeçalhos	`APIVersion, Authorization, AuthorizationId`
Conteúdo de Resposta	`{ "message":"OK", "content":true }`

Listando Todas as Instâncias Existentes

Ao acessar esta URL, todas as informações das instâncias de todas as regiões devem ser retornadas, veja a Tabela 4.13.

Tabela 4.13: Listagem de Instâncias Existentes.

Método HTTP e Caminho	`GET: /computing/instances`
Cabeçalhos	`APIVersion, Authorization, AuthorizationId`
Conteúdo de Resposta	`{ "message":"OK", "content":[{ "id":"230034877195923222", "name":"bionimbuz-instance-1", "machineType":"f1-micro", "creationDate":"Sep 13, 2018 10:42:18 PM", "internalIp":"10.142.0.3", "externalIp":"35.231.64.248" }, { "id":"9014774058867666727", "name":"bionimbuz-instance-2", "machineType":"f1-micro", "creationDate":"Sep 13, 2018 10:41:29 PM", "internalIp":"10.142.0.2", "externalIp":"104.196.166.31" }] }`

4.4 Camada de Coordenação

Esta camada e os seus serviços são implementados pelos TCs (*Task Coordinators* – os Coordenadores de Tarefas, vistos na Figura 4.1), e a arquitetura foi concebida de forma que sua utilização fosse mais eficiente.

Os TCs são responsáveis por monitorar e coordenar o fluxo das tarefas criadas pelos usuários, bem como as dependências existentes entre si relativas a nuvem em que estão executando.

Para isso, a partir do momento em que as tarefas são criadas e suas execuções solicitadas, um TC é criado na nuvem de destino, para que fique mais próximo das tarefas monitoradas (menor latência de comunicação). Os TCs são criados com as credenciais dos usuários ou do grupo de usuários (credenciais compartilhadas), que criam suas tarefas.

Assim, o usuário deve estar ciente de que para utilizar a plataforma, o custo total para a execução de suas tarefas (alocação de máquinas virtuais e espaços de armazenamento) também inclui o custo para a execução de seus Coordenadores. Esse é um custo extra para o usuário, mas que fornece benefícios como a execução mais eficiente de suas tarefas com serviços de provisionamento e de tolerância a falhas, providos pelo uso da plataforma de federação.

Os TCs solicitam a execução de uma determinada tarefa a um TE (*Task Executor* – os Executores de Tarefas, vistos na Figura 4.1) e analisam, em tempo real, as tarefas que devem ser executadas, para que de forma autônoma, decidam se eles devem ser encerrados ou não, para evitar consumo desnecessário de recursos. O TE, descrito na Seção 4.5, procede com a execução da tarefa em três passos, fazendo o *download* de arquivos, executando a aplicação solicitada e, por último, efetuando o *upload* de arquivos gerados.

Como exemplo desse comportamento, a Figura 4.24 exibe uma sequência de tarefas dependentes a serem executadas nas nuvens A, B e C. As Tarefas 1, 2 e 3 são iniciadas paralelamente em nuvens diferentes (Nuvens A e B). Para a criação dessas tarefas, foram criados anteriormente os TCs das Nuvens A e B. Na Nuvem C, o TC só é criado quando a Tarefa 3 for encerrada, pois só nesse momento é criada a tarefa 5 na Nuvem C.

A partir dos passos 5 e 6, toda a execução da sequência de tarefas é feita na Nuvem B, portando, os Coordenadores em execução nas Nuvens A e C tornam-se desnecessários, e então são encerrados. Outro momento em que os Coordenadores podem ser encerrados para economia de custos pode ser observado na execução da Tarefa 5. Essa tarefa pode levar dias para que seja encerrada, e então o Coordenador da Nuvem B é encerrado por falta de tarefas a serem executadas, pois a Tarefa 7, a ser executada na Nuvem B, depende da Tarefa 5.

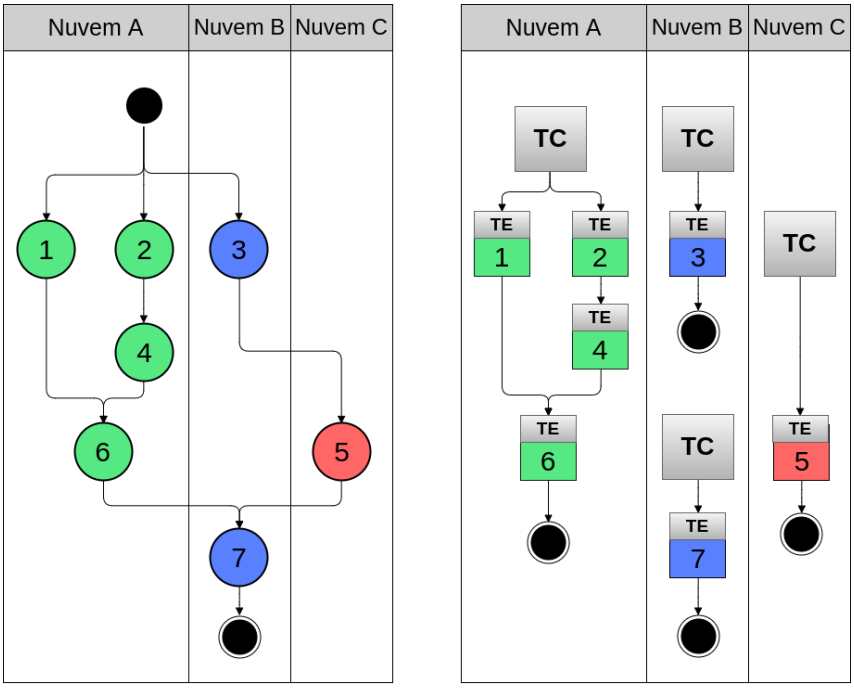

Figura 4.24: Execução de Tarefas na Plataforma de Federação Implementada com a Arquitetura Proposta. À esquerda, execução percebida pelo usuário, e à Direita o comportamento real de execução com a plataforma implementada.

Dessa forma, os serviços disponibilizados pela Camada de Coordenação juntamente com os TCs são:

- **Serviço de Monitoramento:** o serviço de monitoramento está sempre em contato com os Executores de Tarefas (os quais são detalhados na Seção 4.5), coletando os dados de execução levantados por estes. Este serviço também monitora o consumo de recursos utilizados por seu Coordenador, e o consumo utilizado pelos Executores de Tarefas. Esse monitoramento não é utilizado para a tarifação dos usuários, mas para o monitoramento e garantia dos SLAs. A tarifação dos usuários é dada pelas nuvens usadas na execução;

- **Serviço de Dependências:** este serviço verifica as demandas das tarefas. Assim, ele monitora as dependências das tarefas, ou seja, quais serão as próximas tarefas a serem executadas, e se determinado lote de tarefas foi concluído ou não. Ele também verifica se o TC necessita ser encerrado para economizar gastos;

- **Serviço de Elasticidade:** com o auxílio do Serviço de Monitoramento, este serviço analisa as informações coletadas pelos Executores de Tarefas. Desta forma, é capaz de decidir quando aumentar ou reduzir recursos (quantidade de máquinas virtuais, tamanho de memória, número de CPUs, etc.);

- **Serviço de Escalonamento:** responsável por distribuir as tarefas que devem ser executadas na nuvem em que o TC está executando, e solicitar o início da execução da tarefa ou o lote delas.

4.5 Camada de Execução

Esta camada é responsável pela execução das tarefas dos usuários em uma VM, e mantém comunicação constante com os Coordenadores informando os eventos da execução. Assim, os TEs (ou Executores de Tarefas), que são os processos que implementam esta camada, utilizam três serviços, que são eles:

- **Serviço de Aquisição de Recursos**: este serviço é responsável por obter os arquivos de entrada a serem utilizados durante o processo de execução;
- **Serviço de Execução**: serviço responsável por manter o ciclo de vida das tarefas e o monitoramento das mesmas;
- **Serviço de Persistência**: após finalizada com sucesso a execução da tarefa, este serviço é iniciado para efetuar o *upload* dos arquivos gerados para a nuvem especificada durante o processo de criação da tarefa.

Esta camada é implementada utilizando a abordagem de microsserviços com *Spring Boot* (citado na Seção 4.3), e disponibiliza as API REST para Iniciar Execução, Solicitar *Status* Completo de Execução e Solicitar *Status* do Serviço, os quais são descritos em detalhes nas Seções 4.5.1, 4.5.2 e 4.5.3.

4.5.1 Iniciar Execução

O primeiro acesso feito por um TC se faz por meio da URL vista na Tabela 4.14, que envia os parâmetros para a devida execução da tarefa. Assim sendo, recebido o primeiro contato para o início de execução, todo o processo é realizado em três fases, sendo elas:

- ***Download*:** esta fase é realizada pelo Serviço de Obtenção de Recursos que, primeiramente, faz a solicitação dos *tokens* de autorização ao TC para cada um dos arquivos de entrada contidos no campo "*listInputPathsWithExtension*", para em seguida efetuar o *download* diretamente da nuvem na qual o arquivo está armazenado. Os *downloads* são feitos em paralelo para otimizar a utilização de rede e de CPU;
- **Execução:** após o *download* de todos os arquivos de entrada serem realizados com sucesso, é iniciado o Serviço de Execução, que prepara a linha de comando contida

no campo "*commandLine*", fazendo a substituição dos marcadores de argumentos de linha (de *inputs* {i}, de *outputs* {o} e de argumentos {o}) para dar início ao processo de execução do *script* definido no campo "*executionScript*";

- ***Upload*:** por fim, no Serviço de Persistência e, de maneira análoga ao processo de *download*, para cada arquivo de *output* que o usuário desejou salvar na nuvem, informados no campo "*listOutputPathsWithExtension*", é feita a solicitação de *tokens* de autorização para cada arquivo a ser enviado. O processo de envio de múltiplos arquivos também é feito em paralelo.

Após cada mudança de fase é gerada uma notificação para o TC, e caso uma notificação já esteja em andamento, o TE detecta e evita que notificações repetidas sejam enviadas. Dessa forma, o TE mantém comunicação constante com os TCs, enviando dados de monitoramento, tais como consumo de CPU e de memória, e sua fase de execução.

Tabela 4.14: Iniciar Processo de Execução de Tarefa.

Método HTTP e Caminho	`POST: /execution/start`
Cabeçalhos	`APIVersion`
Conteúdo de Requisição	`{ "commandLine":"{i} {i} {i} {o} {o} {o}", "executionScript":"#!/bin/bash\n\ncat $1 \u003e $3\ncat $2 \u003e\u003e $3\n\necho \"Execution time: `date`\" \u003e\u003e $3\n\necho \"Extra file execution time: `date`\" \u003e\u003e $4", "args":"-a -b -c content", "refreshStatusUrl":"http://localhost:8080/external/status/refresh", "listInputPathsWithExtension":[{ "left":"http://localhost:8080/external/file/download/1", "right":"txt" }], "listOutputPathsWithExtension":[{ "left":"http://localhost:8080/external/file/upload/3", "right":"txt" }], "secureFileAccess":{ "token":"eyJhbGciOiJIUzI1NiJ9. eyJzdWIiOiIxQG1hY2hpbmUiLCJleHAiOjE1MzY5MDkyNzZ9. wJTREucqCaaRvJrAZ78vywQsz-Z2YPxae_GgXpbnq4U", "refreshTokenUrl":"http://localhost:8080/external/token/refresh" } }`
Conteúdo de Resposta	`{ "message":"OK", "content":true }`

4.5.2 Solicitação de *Status* Completo de Execução

Esta URL disponibiliza o acesso às informações de progresso de execução, como visto na Tabela 4.15.

Tabela 4.15: Solicitação de *Status* Completo.

Método HTTP e Caminho	`GET: /execution/status`
Cabeçalhos	`APIVersion`
Conteúdo de Resposta	`{ "message":"OK", "content":{ "status":"RUNNING", "phase":"DOWNLOADING", "downloadStatus":{ "total":1, "progress":0.0, "succeded":[ ], "failed":{ } }, "uploadStatus":{ "total":1, "progress":0.0, "succeded":[ ], "failed":{ } }, "errorMessage":"", "hasError":false } }`

4.5.3 Solicitação de *Status* do Serviço

Esta URL permite verificar de maneira mais leve, dada a pequena quantidade de informações contida no Conteúdo de Resposta, visto na Tabela 4.16, o estado de execução do serviço, permitindo, assim, ser utilizado por mecanismos de tolerância a falhas.

Tabela 4.16: Solicitação de *Status*.

Método HTTP e Caminho	`GET: /status`
Cabeçalhos	`APIVersion`
Conteúdo de Resposta	`{ "message":"OK", "content":"RUNNING" }`

4.6 Fluxo de Execução Principal

Para um melhor entendimento da integração entre as camadas da arquitetura apresentada, esta seção descreve a execução das tarefas no ambiente federado. Assim, o fluxo de execução é realizado de acordo com os seguintes passos:

1. **Acesso Web:** os usuários, inicialmente, acessam a plataforma web por meio da Internet, e efetuam seu *login*;

2. **Cadastro de Credenciais de Nuvem:** após o *login*, os usuários devem inserir suas credenciais das nuvens a serem utilizadas pelos *Plugins* das nuvens cadastradas, para que seja possível a execução de suas tarefas na federação;

3. **Criação de Tarefas:** após o cadastro das credenciais das nuvens suportadas, os usuários podem dar início a criação de suas tarefas e suas dependências. Além disso, podem efetuar o *upload* de arquivos e a criação de grupos de usuários. Antes do cadastro das credenciais não é possível efetuar a criação de tarefas e o *upload* de arquivos, pois esse processo consome recursos, que são custeados pelos próprios usuários. Após a criação das tarefas e com o auxílio do Serviço de Predição, ou a escolha direta de máquinas feita pelos usuários, são definidos os passos para a execução das tarefas com as dependências existentes entre si, bem como as nuvens a serem utilizadas. As tarefas criadas são armazenadas em banco de dados para posteriores consultas, e também são armazenadas no ambiente ZooKeeper, que só mantém essas informações enquanto as tarefas estiverem em execução;

4. **Coordenação de Tarefas:** definidas as nuvens iniciais que vão executar as tarefas, a Camada de Aplicação solicita aos *Plugins* a criação dos TCs. Estes recebem informações para ingressarem no ambiente ZooKeeper e se registrarem. Desta forma, ao iniciarem, eles consultam as tarefas pendentes a serem executadas, e solicitam a

criação dos respectivos TEs. Esse passo é realizado de maneira transparente para o usuário;

5. **Monitoramento de Tarefas:** os TCs estão sempre sendo atualizados pelos TEs em determinado intervalo de tempo. Esse mecanismo é necessário para detectar quedas, e para que o Serviço de Elasticidade identifique a necessidade de aumento ou redução de recursos;

6. **Término de Execução:** por fim, a Camada de Aplicação é notificada quando há o término de alguma tarefa por meio de eventos em tempo real, gerados pelo ambiente do ZooKeeper. Os dados de execução das tarefas (tempo total, localização dos arquivos de resultados, etc.) são armazenados em banco de dados para posterior consulta.

Dessa forma, a Figura 4.25 ilustra a relação existente entre os componentes integrantes da arquitetura apresentada neste trabalho. Assim, como pode ser observado em (1), os usuários acessam a plataforma pelo navegador e solicitam seu ingresso. Depois que possuem acesso, são capazes de cadastrar suas credenciais de nuvem, suas tarefas e seus grupos em banco de dados (2). Em seguida, a Camada de Aplicação solicita em (3) que o *Plugin* da nuvem, na qual a tarefa será executada, faça o provisionamento dos recursos necessários em (4). Depois que as máquinas são iniciadas, a Camada de Aplicação envia informações em (5) para conexão com o ambiente ZooKeeper para os Coordenadores, que fazem a conexão com o ambiente em (6). Por fim, em (7) é estabelecida a comunicação dos Executores com seus Coordenadores. Caso um lote de tarefas seja executado em nuvens diferentes, os Coordenadores das nuvens estabelecem comunicação em (8) para sincronização do fluxo de execução.

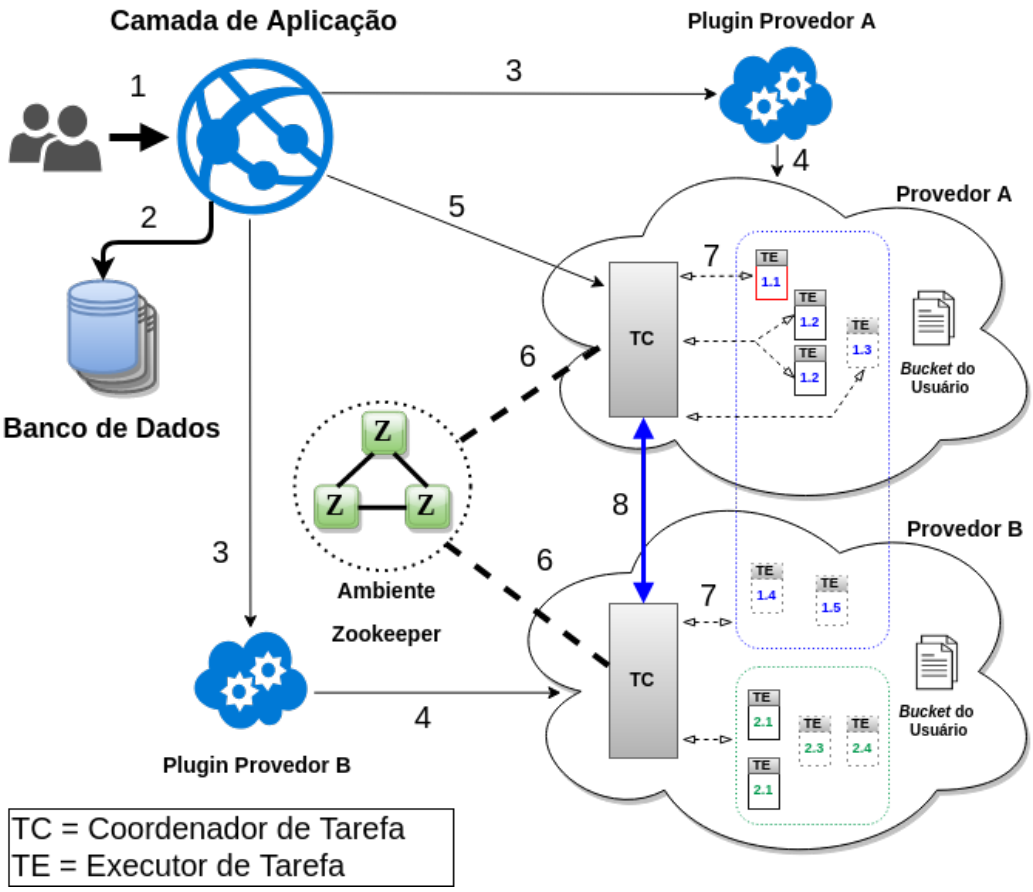

Figura 4.25: Comunicação entre os Componentes da Arquitetura Proposta.

Como pode ser notado na Figura 4.25, o processo de execução dos *workflows* dos usuários é realizado com a utilização das diferentes camadas, distribuídas em vários ambientes e máquinas, e qualquer falha durante este processo é crucial para a continuidade de sua correta operação. Logo, é evidente que a arquitetura proposta necessita de um mecanismo capaz de garantir que o sistema opere corretamente mesmo sob a presença de falhas, ou seja, um mecanismo que permita a arquitetura ser tolerante a falhas.

Diante disto, um modelo de tolerância a falhas multi-estratégico, descrito na Seção 4.7 e proposto por Gomes [55], foi integrado em todas as camadas da arquitetura de federação de nuvens proposta neste trabalho.

4.7 Serviço de Tolerância a Falhas

Conforme apresentado no Capítulo 3, o BioNimbuZ possui um serviço responsável pela "tolerância a falhas" em sua arquitetura. Todavia, o serviço existente apenas implementa o princípio básico da tolerância a falhas (*i.e.*, redundância de dados), e ainda o aplica apenas para os arquivos de saída relacionados à execução dos *workflows*. Em outras

palavras, o serviço de tolerância a falhas do BioNimbuZ, em sua versão anterior, deixa de fora o controle da própria plataforma, ou seja, se uma falha por queda ocorrer na Camada de Aplicação, por exemplo, todo o serviço se torna indisponível.

Gomes [55] propôs um modelo de tolerância a falhas multi-estratégico, que permite a adoção de estratégias de tolerância a falhas conforme a necessidade, e que opera de maneira integrável a sistemas, sejam eles distribuídos ou não. Assim, o modelo proposto por Gomes [55] é implementado como biblioteca, e isso facilita para que o mesmo seja integrado a outros sistemas. Além disto, o modelo possibilita a detecção e a recuperação de falhas de forma automática e parametrizável.

Para isto, o modelo implementa políticas proativas e reativas de tolerância a falhas, disponibilizando as técnicas de tolerância a falhas: rejuvenescimento de *software*, repetição de tarefas, reenvio de tarefas e replicação. A parametrização proposta no modelo visa flexibilizar para os usuários finais o nível de tolerância a falhas desejado, possibilitando que os mesmos otimizem, de acordo com a sua necessidade, os custos necessários para tolerar falhas em seu modelo de negócio.

Observadas estas características, este modelo de tolerância a falhas foi integrado em todas as quatro camadas da arquitetura do BioNimbuZ 2, apresentada neste trabalho, fornecendo um nível de tolerância a falhas parametrizável para toda a plataforma. A Seção 4.7.1 detalha as políticas e as técnicas disponíveis no modelo de tolerância a falhas, bem como sua integração ao trabalho proposto.

4.7.1 Integração do Modelo de Tolerância a Falhas

Antes de citar como foi realizada a integração entre o modelo de tolerância a falhas e a arquitetura apresentada, é importante citar que dentre as políticas e as diferentes técnicas existentes na área de tolerância a falhas citadas por Gomes [55], seu modelo implementa as que seguem:

1. **Política Proativa**: seu princípio é evitar a recorrência de falhas, erros e defeitos, tomando medidas preventivas de maneira proativa. As medidas são tomadas com base no estudo de indicadores de falhas e previsão de falhas subjacentes. O segundo passo é aplicar proativamente medidas corretivas em tempo de implementação, alterando o código ou substituindo os componentes propensos à falha. Para a concepção desta política tolerante a falhas, a seguinte técnica foi adotada:

    - **Rejuvenescimento de Software** (*Software Rejuvenation*): consiste na reinicialização periódica do sistema, a fim de trazê-lo novamente a um estado de operação limpo. Para esta técnica, suposições são feitas a respeito de quando o sistema falharia se nenhuma medida fosse tomada, e o tempo de reinicialização

é dado de acordo com esta suposição. É importante ressaltar que intervalos curtos implicam em maior custo, e intervalos longos implicam em maior chance de que falhas ocorram.

2. **Política Reativa**: seu princípio é reduzir o impacto de falhas que efetivamente já ocorreram, fazendo com que o sistema passe de um estado instável para um estado estável, para que então volte a operar e atender suas requisições normalmente. As técnicas utilizadas foram:

   - **Repetição ou Tentar Novamente** (*Retry*): consiste em executar novamente a tarefa que falhou, no mesmo recurso computacional. Esta é a técnica mais simples para tolerância a falhas;
   - **Reenvio de Tarefas** (*Task Resubmission*): consiste em reenviar a tarefa que apresentou falha para o mesmo ou para um outro recurso computacional. Isto ocorre em tempo de execução, sem interromper o fluxo de trabalho do sistema;
   - **Replicação** (*Replication*): consiste na criação ou na manutenção de cópias do sistema em recursos diferentes. Dois tipos de replicação foram observados na literatura: (1) a replicação ativa, na qual ambos os recursos, principal e secundário, executam a operação independentemente uma da outra; e (2) a replicação passiva, na qual somente o recurso principal executa a operação e se espera que o mesmo produza o resultado, enquanto os recursos secundários realizam a operação apenas em casos de falha do recurso principal [56]. Esta técnica adiciona redundância ao sistema e, com ela, espera-se que ao menos uma das réplicas finalize sua tarefa com sucesso.

Com isso, tornou-se possível configurar técnicas de tolerância a falhas diferentes para cada uma das quatro camadas existentes. Para a Camada de Aplicação e para a Camada de Federação, que consistem em sistemas com a característica de permanecer disponível 24 horas por dia, foi configurado o nível de tolerância a falhas com as técnicas de repetição e rejuvenescimento de software, com intervalo de reinicialização periódica a cada 24 horas. Já para a Camada de Coordenação, que consiste em um sistema que é executado sob demanda em um determinado provedor de nuvem, foi configurado apenas a técnica de repetição. Por fim, a Camada de Execução, que também consiste em um sistema executado sob demanda, mas em diversos provedores de nuvem, foi configurado o nível de tolerância a falhas para utilizar a técnica de reenvio de tarefas e replicação.

A Figura 4.26 ilustra como foi realizada a integração do modelo de tolerância a falhas proposto por Gomes [55] com a nova arquitetura do BioNimbuZ 2.

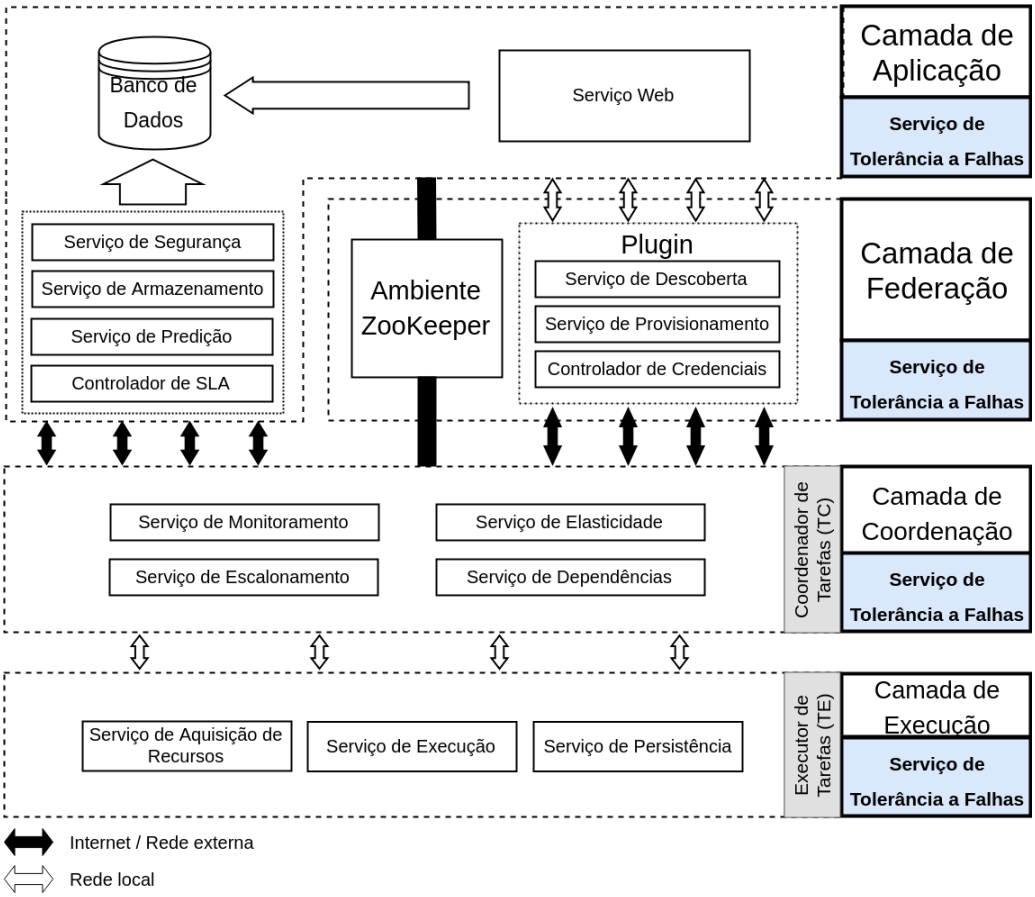

Figura 4.26: Arquitetura do BioNimbuZ 2 Integrada ao Modelo de Tolerância a Falhas.

Dessa forma, o modelo de tolerância a falhas utilizado fornece a implementação de tolerância a falhas de forma flexível, eficiente e otimizada, pois permite que seja usada a técnica de tolerância a falhas mais adequada de acordo com a necessidade.

A Figura 4.27 exemplifica a integração da tolerância a falhas na arquitetura desenvolvida neste trabalho com o fluxo de execução principal do BioNimbuZ 2, citado anteriormente na Seção 4.6, e ilustrado na Figura 4.25.

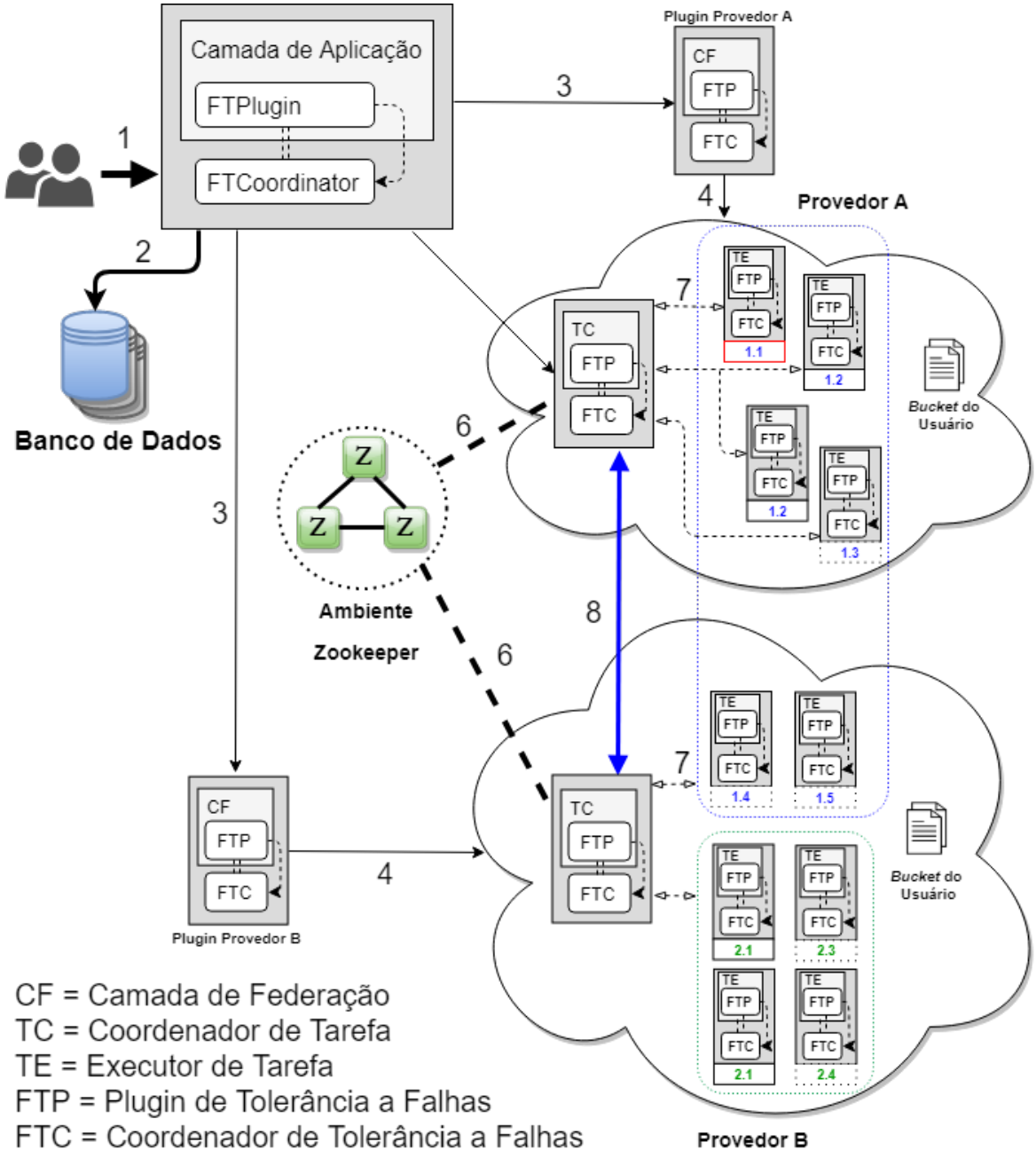

Figura 4.27: Comunicação entre os Diferentes Componentes e o Modelo de Tolerância a Falhas do BioNimbuZ 2.

Desta forma, é possível observar como é na prática a integração do modelo de tolerância a falhas nas quatro camadas do BioNimbuZ 2. Contudo, é importante ressaltar que a parametrização do nível de tolerância a falhas para os usuários finais apenas se aplica na Camada de Execução, pois esta é a camada responsável por executar as tarefas criadas na interface gráfica, na qual é permitido que os próprios usuários configurem o nível de tolerância a falhas desejado para cada tarefa. As demais camadas são parametrizadas de

maneira sistêmica, ou seja, apenas os desenvolvedores do BioNimbuZ têm acesso à sua configuração.

Assim, é possível perceber que no cenário apresentado nas Figuras 4.26 e 4.27, os níveis de tolerância a falhas foram consideravelmente aumentados, tanto para a plataforma BioNimbuZ 2 quanto para as tarefas que constituem os *workflows*, pois agora é possível parametrizar a técnica de tolerância a falhas mais adequada para cada componente da plataforma.

Finalmente, a Seção 4.8 apresenta os principais trabalhos relacionados encontrados na literatura.

4.8 Trabalhos Relacionados

Nesta seção são detalhadas as plataformas de nuvens federadas que são mais relevantes na literatura recente, e que foram utilizados como trabalhos relacionados para o desenvolvimento deste trabalho.

4.8.1 InterCloud

Buyya *et al.* [7] apresentam o InterCloud, um *framework* arquitetural de federação de nuvens utilitário [11]. O ambiente proposto sugere a criação de um Mercado de Nuvens, no qual são centralizadas as informações das nuvens e dos usuários interessados em vender ou comprar serviços. O ambiente do InterCloud é composto por três componentes (apresentado na Figura 4.28), que são eles:

- ***Broker* de Nuvem:** este componente encontra-se no domínio dos usuários participantes do ambiente de federação. Por meio dele, os usuários são capazes de solicitarem os recursos que melhor se enquadram às suas necessidade de QoS (*Quality of Service*). Para isso, o *Broker* de Nuvem estabelece comunicação com o Mercado de Nuvens, que busca a nuvem mais adequada às necessidades do usuário. Em seguida, após requisitada a nuvem, o *Broker* de Nuvem faz a negociação direta com os Coordenadores de Nuvem, que fornecem os recursos disponíveis da nuvem responsável por gerenciar;

- **Coordenador de Nuvem:** as nuvens participantes da federação devem fornecer este componente para que seus recursos sejam gerenciados e expostos para acesso externo (pelos *Brokers* de Nuvem). Com isso, os Coordenadores devem implementar funcionalidades básicas de gerenciamento de recursos, tais como a de provisionamento, de agendamento, de virtualização, de monitoramento e de descoberta. Além

disso, eles devem manter informações atualizadas sobre políticas de preços e regras de SLA no ambiente do Mercado de Nuvens, de tal maneira que a decisão de escolha de nuvens não seja baseada em dados antigos, o que pode ser um problema para a economia de custos do usuário;

- **Mercado de Nuvens (CEx):** o CEx, do inglês *Cloud Exchange*, é o componentes fundamental desse ambiente, que registra as informações das nuvens integrantes da federação para que sejam acessadas pelos *Brokers* de Nuvem. Esse componente realiza a verificação de requisitos dos usuários, de acordo com os preços e as políticas das nuvens, para a escolha da nuvem a ser utilizada. Ainda, como é um componente que fornece um mercado de nuvens, ele deve possuir três serviços básicos. O primeiro, de Diretório, registra as necessidades tanto de usuários quanto de provedores de nuvem, para consumo e fornecimento de recursos. O segundo serviço, de Leiloeiro (no inglês *Auctioneer*), registra as propostas dos participantes, para determinar o que melhor cumpre com os requisitos disponíveis. Por último, o serviço de Banco, viabiliza as transações financeiras necessárias para estabelecer os acordos entre os participantes.

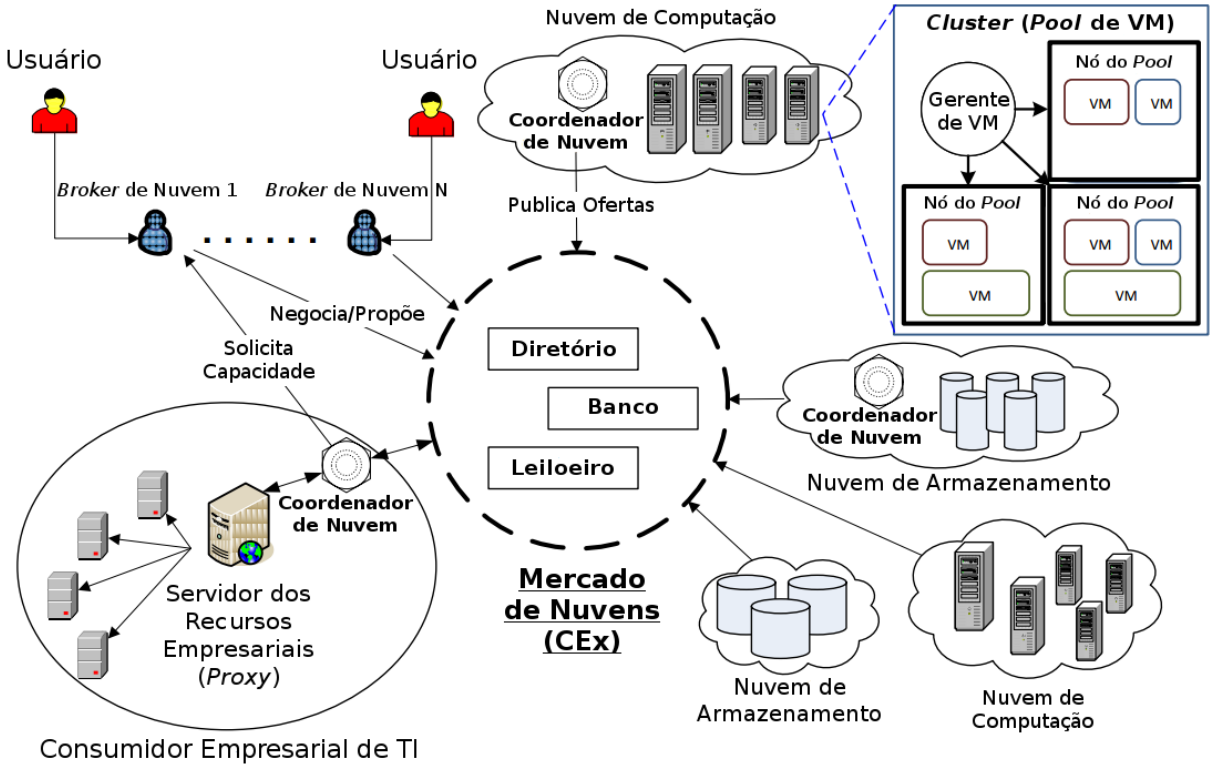

Figura 4.28: Ambiente de Federação InterCloud [7].

Com o ambiente do InterCloud é necessário que seja criado um mercado de nuvens, o qual deve ser mantido por terceiros confiáveis para que sejam imparciais quanto aos participantes. Além disso, esse ambiente depende da voluntariedade das plataformas de nuvem implementarem mecanismos de comunicação únicos aliados a novos protocolos de negociação para participarem de um mercado global de nuvens.

Desta forma, a nova arquitetura apresentada cria os *Plugins*, que estabelecem uma camada de acesso aos recursos dos provedores de nuvem, reduzindo a dependência dos serviços que devem ser implementados pelos Coordenadores de Nuvens, vistos no ambiente do InterCloud. Assim, os *Plugins* são implementados como microsserviços, e podem ser facilmente escaláveis, sem necessitar que outros serviços sejam também iniciados junto com ele, como é o caso dos Coordenadores do InterCloud.

4.8.2 CCFM

Celesti *et al.* [2] propõem o Gerente de Federação *Cross-Cloud*–CCFM, que deve ser implantado em um Gerente de Nuvem para o estabelecimento de federações com nuvens estrangeiras, que deve ser realizado de acordo com as três fases seguintes:

1. **Descoberta:** primeiramente é feito um mapeamento das nuvens disponíveis que se possam estabelecer acordos, utilizando para esse mapeamento, comunicação *peer-to-peer* entre os Agentes de Descoberta das diferentes nuvens;

2. **Verificação:** em seguida, são analisados os requisitos das nuvens estrangeiras, de tal maneira que cumpram com os requisitos desejados (em relação aos recursos disponíveis, políticas, mecanismos de autenticação suportados, etc.);

3. **Autenticação:** por fim, escolhida a melhor nuvem que satisfaça os requisitos desejados, as duas nuvens, a local e a estrangeira, iniciam o processo de autenticação para estabelecer uma interação segura.

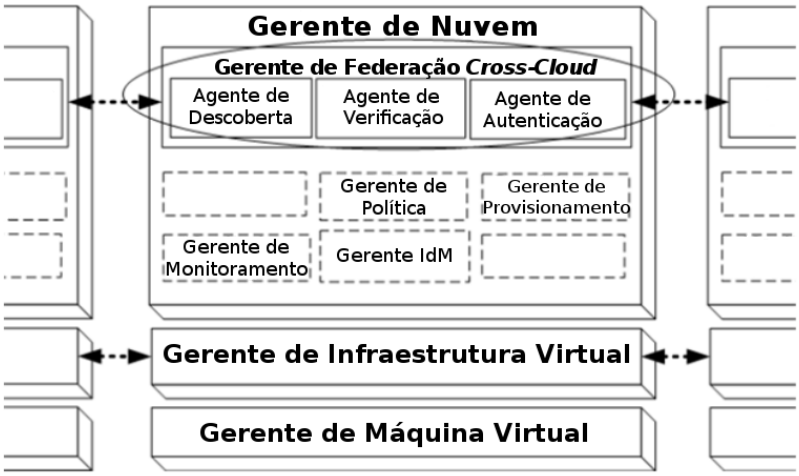

Figura 4.29: Arquitetura do CCFM [2].

Dessa maneira, nota-se que CCFM possui uma arquitetura centralizada em que todos os serviços são disponibilizados no mesmo componente, tanto as funcionalidades específicas de cada nuvem quanto as funcionalidades de monitoramento do sistema, implantados dentro do Gerente de Nuvem.

Assim, para evitar uma gerência centralizada, na nova arquitetura, as particularidades de cada nuvem são implementadas de maneira independente do sistema, com interfaces bem definidas para serem utilizados pela Camada de Aplicação. Além do mais, com os Executores de Tarefas, que além de manterem o ciclo de vida das tarefas é possível monitorar os recursos utilizados pelas mesmas, dividindo as responsabilidades de monitoramento de recursos pelos componentes do sistema.

4.8.3 ICFF

Demchenko *et al.* [57] descrevem o *framework* arquitetural ICAF (do inglês *Intercloud Architecture Framework*), com o objetivo de oferecer soluções para serviços que são baseados em infraestruturas de múltiplos provedores de nuvem em domínios variados, de tal maneira que os recursos utilizados das diferentes nuvens sejam vistos como um único *pool* de recursos.

Para que os objetivos do *framework* sejam alcançados, são necessários os seguintes componentes:

- **Modelo de Serviços de Nuvem Multinível (CSM):** esse componente define mecanismos de interação vertical entre as diferentes camadas de serviços fornecidos por um provedor de nuvem;

- **Plano de Controle e Gerenciamento Intercloud (ICCMP):** o papel desse componente é o de prover interfaces lógicas e funcionais padronizadas para as diferentes camadas de serviços dos provedores de nuvem (SaaS, PaaS e IaaS), para que as camadas situadas em diferentes domínios possam estabelecer comunicação entre si e assim, caso seja necessário, realizar o controle e o gerenciamento de camadas inferiores desses outros domínios;

- ***Framework* de Operação Intercloud (ICOF):** nesse componente são definidas funcionalidades para o suporte de operação de infraestrutura em múltiplos provedores de nuvem, incluindo *workflows* de negócio, gerenciamento de SLAs e tarifação. Além disso, são definidos os papéis e os atores responsáveis pela operação, gerenciamento e de propriedade dos recursos utilizados;

- ***Framework* de Segurança Intercloud (ICSF):** são definidos nesse componente, um conjunto de funcionalidades para o gerenciamento de identidade e de confiança, controle de acesso e comunicação segura no ambiente de múltiplos provedores de nuvem, garantindo uma federação de nuvens segura em diversos aspectos [58];

- ***Framework* de Federação Intercloud (ICFF):** com o componente ICFF, é viabilizada a federação de nuvens em diferentes domínios administrativos, permitindo aos seus usuários acessarem os recursos de múltiplos domínios, sem a necessidade de obter identidades específicas para cada um [8].

O componente ICFF é composto pelos *Brokers*, os *Gateways*, e o FedIDP (*Federated Identity Provider*). Este último sendo o principal componente, pois ele viabiliza a implementação de mecanismos seguros de comunicação para provisionamento de recursos, no qual devem ser implementados protocolos padronizados de IDP, tais como OpenID, SAML, OAuth e Keystone[10]. Desta forma, ele viabiliza a autenticação e a autorização facilitada entre os diferentes domínios de federação.

Os *Brokers* são responsáveis por fazer a negociação direta com os provedores de nuvem e solicitar seus recursos. Em seguida, as solicitações de recursos e as negociações são traduzidas para a linguagem específica da nuvem por meio dos *Gateways*.

Na Figura 4.30 são apresentados os módulos que são implementados pelo componente ICFF do *framework* ICAF.

[10]OpenStack Keystone, `https://docs.openstack.org/keystone/`

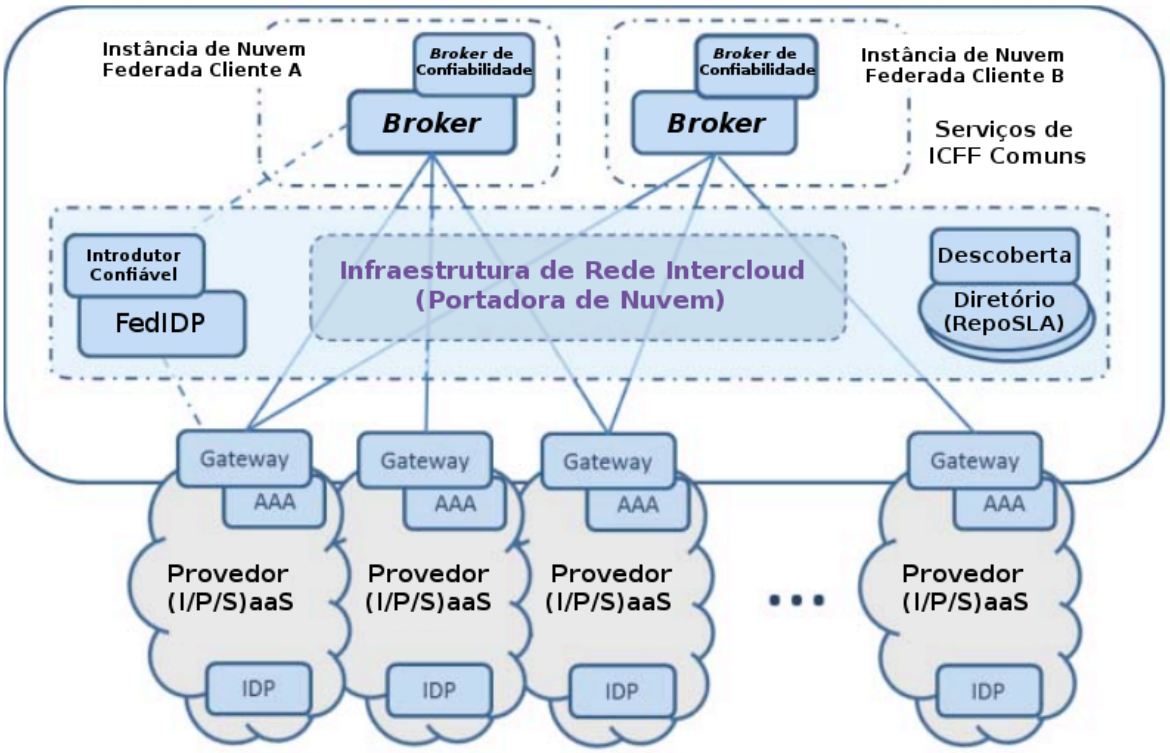

Figura 4.30: Infraestrutura do ICFF [8].

No trabalho proposto pelo ICFF, algumas funcionalidades intrínsecas à determinada nuvem estão divididas em ambientes separados, como a de Descoberta e a de *Gateway*, dificultando a sua manutenção em caso de atualização dos protocolos de determinada nuvem. Além disso, a função de Descoberta depende da voluntariedade de uma plataforma de nuvem em atualizar as informações dos recursos disponíveis, como tabelas de preço e tipos de recursos.

Com a arquitetura apresentada neste trabalho, os *Plugins* de nuvem, de forma ativa, criam uma camada de acesso para as nuvens participantes da federação, atualizando suas tabelas de preço e outras informações sobre os recursos disponíveis. Dessa forma, é alcançada uma transparência em relação a voluntariedade de um provedor de nuvem, por exemplo da Amazon, em atualizar essas informações.

Outra vantagem da utilização dos *Plugins*, é a sua unificação de funcionalidades específicas de um provedor de nuvem e sua disponibilização como microsserviços. Dessa forma, de maneira simplificada, é possível escalar seus serviços de acordo com a demanda, sem prejudicar o funcionamento global do sistema.

4.8.4 SciCumulus

Proposto inicialmente por Oliveira *et al.* [16] e, posteriormente, modificado por Viana *et al.* [59], Costa *et al.* [60] e Silva *et al.* [9], o *middleware* SciCumulus é um orquestrador de tarefas que permite a execução de *workflows* científicos. As tarefa são executadas em um ambiente distribuído de VMs, criado em um determinado provedor de nuvem.

O *middleware* é composto por três camadas, a *Starter*, que fornece um sistema gerenciador de *workflows* científicos (SWfMS), instalado localmente nas máquinas dos cientistas, onde podem criar as tarefas e os arquivos a serem enviados para a nuvem de destino.

A camada *Core* é iniciada pela camada *Starter* de maneira distribuída com MPI[11] nas próprias VMs em execução na nuvem, ou seja, cada VM possui essa camada, e é responsável por executar, monitorar e armazenar dados de proveniência dos *workflows*. Esta camada também é responsável pela criação e pela remoção de novas VMs de acordo com a demanda dos *workflows* de forma adaptativa.

A última camada refere-se à de Banco de Dados de Proveniência, por meio da qual são armazenados os dados de proveniência dos *workflows*. A Figura 4.31 apresenta as camadas da plataforma e a relação existente entre elas.

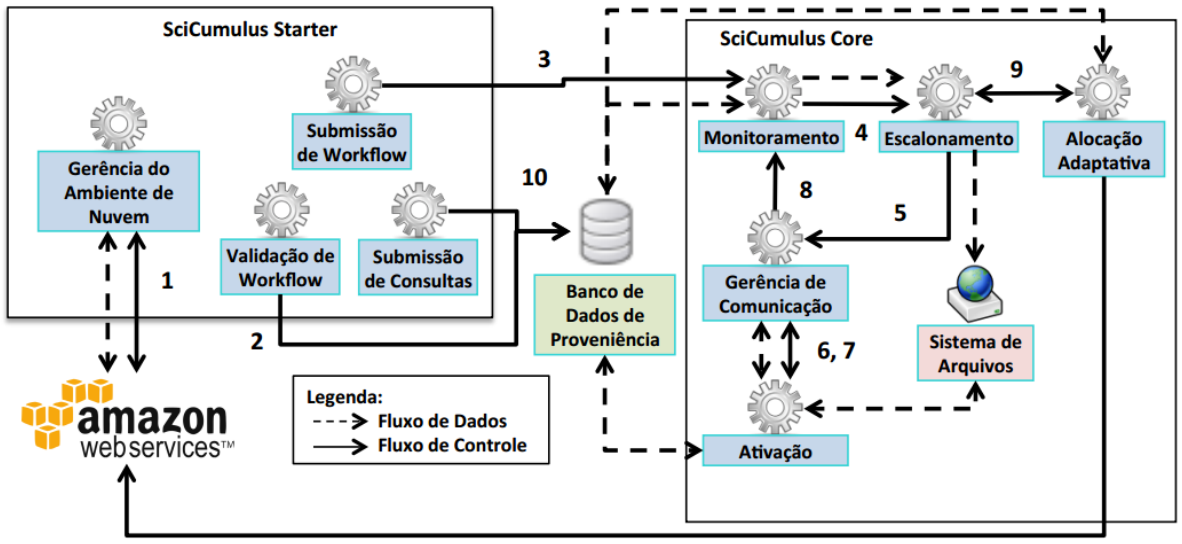

Figura 4.31: Arquitetura do SciCumulus [9].

Diferentemente do SciCumulus, este trabalho apresenta uma arquitetura federada em que várias nuvens podem fazer parte da federação de maneira simplificada. Além disso, o SciCumulus utiliza o MPI para o compartilhamento de informações entre todas as

[11] *Message Passing Interface*, `https://www.mpi-forum.org/`

máquinas em execução, o que pode ser um problema em um ambiente de nuvem federada caso as nuvens estejam situadas em diferentes partes do globo (alta latência de rede).

Ainda na nova arquitetura, são implementados os TCs, que centralizam informações de monitoramento das tarefas em execução, e estabelecem comunicação com outros Coordenadores em nuvens diferentes. Os Executores de Tarefa estão em constante comunicação com os seus respectivos Coordenadores, evitando que esses Executores realizem comunicação com outras nuvens, e explorando a comunicação na nuvem local. Assim sendo, a arquitetura apresentada tem a vantagem de dividir a comunicação dos componentes em dois níveis, a primeira no nível entre as nuvens, permitindo uma troca de informação mais objetiva e direta, com os Coordenadores. A outra, em mais baixo nível, é feita internamente em cada nuvem, com os Executores.

4.8.5 CometCloud

A plataforma, proposta por Kim e Prashar [15] e seguida por outras propostas de Montes *et al.* [10, 61], foi projetada para permitir uma grande heterogeneidade de plataformas de nuvem para criar uma federação dinâmica e sob-demanda, que executa tarefas de *workflows* científicos de diversas naturezas.

Como visto na Figura 4.32 (a), a plataforma é composta por três camadas. Na camada de Programação/interface (Gerenciador de *workflow* e Gerenciador de recursos), são fornecidos à plataforma os *workflows* definidos manualmente pelos cientistas por meio de arquivos XML, contendo a localização dos dados de entrada, as tarefas e as dependências entre si, políticas de escalonamento e outros objetivos e políticas. Nesta camada também são informados os recursos disponíveis, bem como as políticas e as limitações que regulam sua utilização (por exemplo, o intervalo do dia em que os recursos podem ser utilizados).

O trabalho de Wang *et al.* [62] realiza a integração do Kepler[12] nesta camada da plataforma, que é uma ferramenta instalável em máquina local, para criação dos *workflows* em ambiente gráfico.

Na camada de Gerenciamento autônomo são definidos os critérios para o provisionamento dos recursos na execução dos *workflows*. Para o atendimento desses critérios, essa camada monitora o andamento das execuções e as adapta de maneira que os acordos estabelecidos sejam cumpridos.

Por fim, a camada de Infraestrutura/federação garante mecanismos que estabelecem a federação dos diferentes tipo de infraestruturas e nuvens. Esta camada é subdividida em dois níveis, como visto na Figura 4.32 (b). Em um nível mais baixo, no espaço de execução compartilhado, os recursos disponíveis em uma nuvem local (M, W e R) são mapeados

[12]Kepler, `https://kepler-project.org/`

com a utilização de tabelas de *hash* distribuídas (*Distributed Hash Table* - DHT) com comunicação *peer-to-peer*, que são utilizados para *lookup* dos recursos. No nível mais alto, os diferentes *datacenters* são também mapeados com as DHTs, formando o espaço de gerenciamento de federação.

O espaço de execução compartilhado contém os recursos utilizados na execução dos *workflows*, e fazem o provisionamento de máquinas externas aos *datacenters* locais. Ainda na Figura 4.32 (b), é possível ver os diferentes tipos de nós e suas responsabilidades dentro da arquitetura proposta, que são os seguintes:

- **M:** nó *master*, responsável por gerenciar as tarefas dos *workflows*;
- **W e IW:** *workers* que executam as tarefas, sendo eles de dois tipos. Os seguros, que executam em ambiente protegido em rede local; e os inseguros, que são executados em ambientes externos (nuvens estrangeiras) e não fazem parte do espaço de execução compartilhado;
- **P e R:** por meio do *Proxy* seguro, o *Request Handler* se comunica com os IWs para coletar as informações geradas por estes.

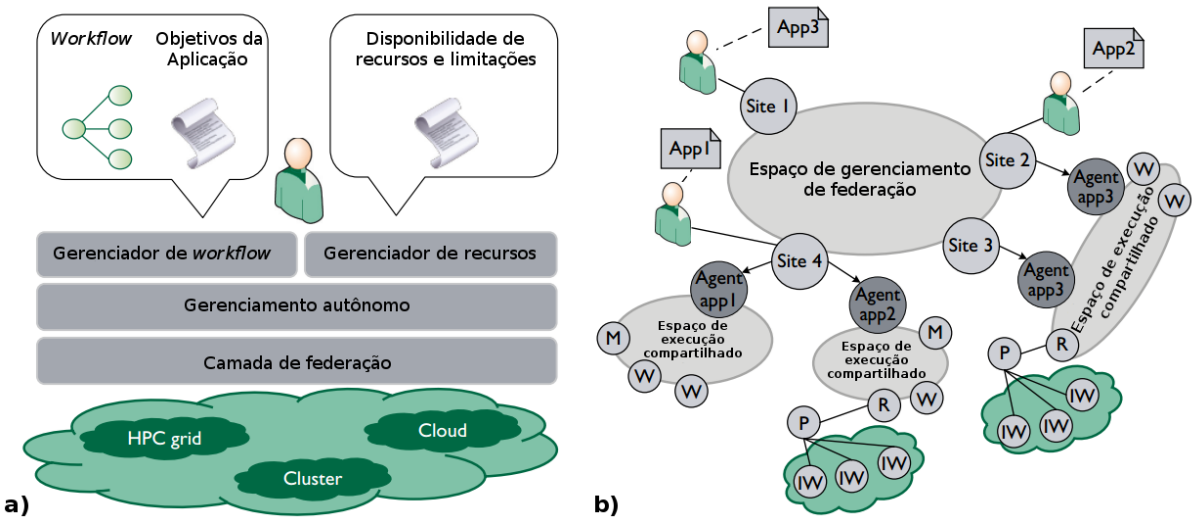

Figura 4.32: Arquitetura do CometCloud [10]. Em (a) as três camadas da arquitetura; e em (b) o modelo de coordenação utilizado.

Como pode ser observado, a plataforma CometCloud está dividida em duas camadas, uma camada em nível de nuvens, *grids* e *clusters*; e a outra camada em nível de máquinas dentro dos *datacenters*. Ainda, as máquinas dentro dos *datacenters* estão interconectadas

no modelo *peer-to-peer* para armazenamento de informações de descoberta, o que pode tornar-se um problema para a sincronização de informações em um ambiente com muitas máquinas. De forma diferente, na arquitetura proposta por este trabalho, os Coordenadores estão em comunicação direta apenas com os recursos que estão executando tarefas, e detectam quedas em tempo real. Assim, os recursos são liberados quando não mais utilizados, evitando desperdícios e poluição da rede local.

Diante dos trabalhos relacionados apresentados, a Tabela 4.17 mostra um comparativo entre os trabalhos relacionados e a arquitetura proposta, implementada no BioNimbuZ 2. Como pode ser observado, a arquitetura apresentada é a única que trabalha com *multi-tentant*, microsserviços e grupos de usuários, além de fornecimento de outras funcionalidades importantes como o serviço web, *plugins* desacoplados, serviços distribuídos e comunicação multi-nível.

No BioNimbuZ 2, são fornecidos os *Plugins* de federação como microsserviços, que permitem uma integração facilitada no ambiente de federação. Além disso, são fornecidos mecanismos para a utilização e o controle de credenciais de provedores de nuvem dos usuários, de maneira que os custos sejam repassados diretamente sem a intervenção da plataforma.

Outro ponto abordado pela nova arquitetura é o de prover uma melhor execução dos TCs, que têm responsabilidades semelhantes aos nós *Master* do CometCloud [15]. Todavia, neste trabalho, esses nós estabelecem contato com outros nós de coordenação, e detectam quando não estão sendo mais utilizados, procedendo com seu encerramento imediato para evitar consumo desnecessário de recursos.

Tabela 4.17: Trabalhos Relacionados.

Trabalho	**Multi-*tentant***	**Serviço Web**	**Micros-serviços**	**Plugins Desaco-plados**	**Grupos de Usuá-rios**	**Serviços Distri-buídos**	**Comuni-cação em Níveis**	**Tolerância a Falhas Multi-estratégica**
InterCloud [7]						X	X	
CCFM [2]						X		
ICFF [8]				X			X	
SciCumulus [16]				X		X	X	
CometCloud [15]				X		X	X	
BioNimbuZ [14]		X						
Arquitetura BioNimbuZ 2	X	X	X	X	X	X	X	X

4.9 Considerações Finais

Neste capítulo foi apresentada a arquitetura de federação de nuvens que fornece funcionalidades não observadas em outros trabalhos relacionados, principalmente, tendo como foco

a melhor utilização dos recursos dos usuários e a melhor implantação de seus componentes.

Dessa forma, é apresentado no Capítulo 5 a análise dos resultados obtidos com a implementação do BioNimbuZ 2, quando comparado com a versão anterior da plataforma.

Capítulo 5

Análise dos Resultados

Neste capítulo são detalhados os testes realizados e os resultados obtidos com a utilização da nova arquitetura do BioNimbuZ, em comparação com a sua versão anterior. No primeiro teste, detalhado na Seção 5.1.1, é comparado o tempo de inicialização dos componentes das plataformas. Já no segundo teste, visto na Seção 5.2, é comparado o consumo de recursos de CPU e de memória utilizados. Por último, na Seção 5.3 são apresentados os benefícios obtidos com a utilização do BioNimbuZ 2.

5.1 Estratégia de Execução dos Testes

Com os testes descritos neste capítulo pretende-se comparar quantitativamente o tempo de inicialização dos serviços, contado a partir do momento da sua solicitação de execução até o momento em que estão totalmente operacionais. Além disso, os testes analisam o consumo de recursos da máquina para que a plataforma esteja operacional.

Para a execução dos testes foi utilizada uma máquina com processador Intel® Core™ i7-6700HQ com frequência base de 2.60GHz, 4 núcleos físicos e 8 *threads*.

Durante a execução dos testes, e para facilitar a visualização e a comparação dos mesmos, são utilizados os termos BNZ-1, para o BioNimbuZ em sua primeira versão, e BNZ-2 para a versão implementada por este trabalho.

Além disso, e em vista das grandes mudanças implementadas na arquitetura, o BNZ-1 é comparado ao BNZ-2 em dois momentos. Em um primeiro momento, a ser chamado por "Edição", quando os serviços estão disponíveis para que os seus usuários possam criar e editar seus *workflows*, e em um segundo momento, chamado por "Execução", quando estes estão em execução.

A estratégia adotada se deve ao fato de que a Camada de Núcleo do BNZ-1 contém vários serviços que foram separados em novos serviços na arquitetura proposta, como por exemplo os serviços de execução de tarefas e de federação de nuvens, que são realiza-

dos pelos *Plugins*. Assim, o agrupamento dos serviços foi feito de maneira que fossem aproximadas as responsabilidades exercidas por cada plataforma, sendo eles os seguintes:

- **Momento de Edição:**
  - BNZ-1: Camadas de Aplicação e de Núcleo;
  - BNZ-2: Camadas de Aplicação e de Federação com os *Plugins* GCP e AWS;
- **Momento de Execução:**
  - BNZ-1: Camada de Núcleo;
  - BNZ-2: Executor de Tarefas.

Após as comparações quantitativas que foram realizadas nas Seções 5.1.1 e 5.2, são realizadas as comparações qualitativas na Seção 5.3 de recursos disponibilizados pela nova arquitetura implementada para o BioNimbuZ 2, e não disponíveis em sua primeira versão.

5.1.1 Tempo de Inicialização

A medição de tempo de inicialização considerou apenas o momento de "Edição", pois a Camada de Núcleo do BNZ-1 necessita que um *template* de máquina seja criado previamente na nuvem, com todas as aplicações que a tarefa for utilizar. Assim sendo, medir o tempo para a criação de um *template* no BNZ-1 seria algo subjetivo, pois depende de quem o cria e se esse possui o conhecimento para tal, e em que nuvem deve ser criado.

Já para o Executor de Tarefas do BNZ-2, a plataforma de federação fornece uma interface única em que se possa criar aplicações que contém *scripts* de inicialização independentes de nuvem. Após a criação das aplicações do usuário, a execução dos *scripts* de inicialização, instalando o Java por exemplo, executa em poucos segundos. Desta forma, a versatilidade de execução de aplicações independentes de nuvens é uma vantagem do BNZ-2 em relação ao BNZ-1, pois o BNZ-2 cria uma federação nesse quesito.

Assim, como exibido na Tabela 5.1, considerando o tempo de inicialização das Camadas de Aplicação, e o total de todas as aplicações necessárias para fornecer a edição de *workflows* para seus usuários nas duas plataformas, observou-se uma redução de 73.31% e 67.33% no BNZ-2, respectivamente, mostrado na Figura 5.1.

Tabela 5.1: Tempo de Inicialização em Segundos.

	BNZ-1		BNZ-2		
	Aplicação	**Núcleo**	**Aplicação**	**Plugin AWS**	**Plugin GCP**
	20,90	14,94	5,58	2,96	3,17
Total	35,84		11,71		

	Aplicação	**Total**
BNZ-1	20,90	35,84
BNZ-2	5,58	11,71
Redução	73,31%	67,33%

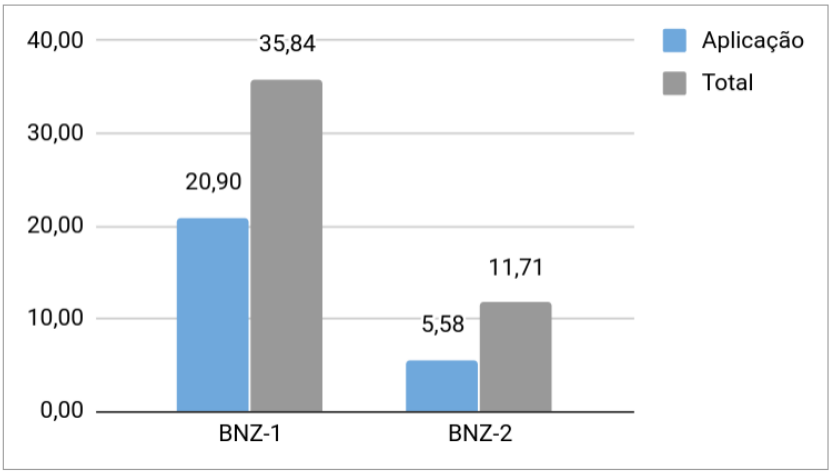

Figura 5.1: Tempo de Inicialização em Segundos.

5.2 Consumo de Recursos

Quanto ao consumo de recursos para o momento de "Edição", o melhor resultado obtido está relacionado ao consumo de CPU, obtendo uma redução de aproximadamente 80% do total de consumo utilizado pelo BNZ-1, conforme apresentado na Tabela 5.2 e na Figura 5.2.

É fundamental observar que os ganhos obtidos foram comparados com serviços que apresentam responsabilidades compatíveis. Assim sendo, os ganhos obtidos com a utilização da abordagem de microsserviços nos *Plugins* podem ser comparados apenas qualitativamente.

Tabela 5.2: Percentual de Consumo de Recursos para Edição.

	BNZ-1		BNZ-2		
	Aplicação	**Núcleo**	**Aplicação**	**Plugin AWS**	**Plugin GCP**
CPU	1,7%	4,7%	0,7%	0,3%	0,3%
Memória	9,5%	16,8%	6,5%	4,7%	3,4%

Total CPU	6,40%	1,30%
Total Memória	26,30%	14,60%

		Aplicação	**Total**
BNZ-1	**CPU**	1,70%	6,40%
	Memória	9,50%	26,30%
BNZ-2	**CPU**	0,7%	1,30%
	Memória	6,50%	14,60%
Redução	**CPU**	58,82%	79,69%
	Memória	32%	44,49%

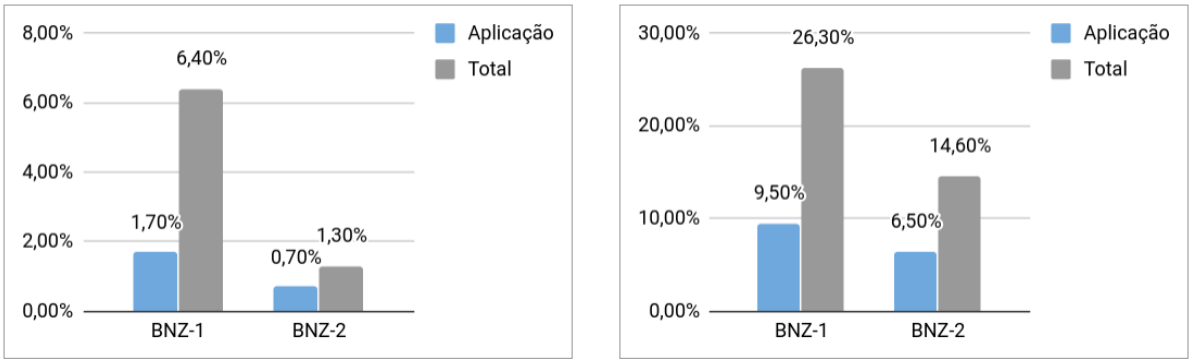

Figura 5.2: Percentual de Consumo de Recursos para Edição, à Esquerda, o Consumo de CPU e à Direita, o Consumo de Memória.

Por último, no teste realizado em relação ao momento de "Execução", exibido na Tabela 5.3, é possível visualizar os ganhos reais com o desacoplamento dos serviços atribuídos à Camada de Núcleo do BNZ-1, obtendo uma redução de consumo de CPU de aproximadamente 15 vezes, e de memória de aproximadamente 4 vezes, conforme mostrado na Figura 5.3.

Tabela 5.3: Percentual de Consumo de Recursos para Execução.

	CPU	Memória
BNZ-1	4,7%	16,8%
BNZ-2	0,3%	3,8%
Redução	93,62%	77,38%

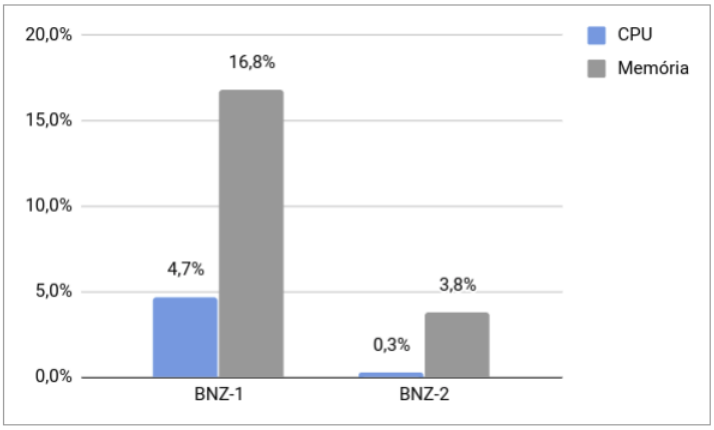

Figura 5.3: Percentual de Consumo de Recursos para Execução.

5.3 Benefícios Obtidos

Nesta seção são detalhados os benefícios obtidos com a utilização do BioNimbuZ 2, que em razão de grandes mudanças em sua arquitetura original, podem apenas ser comparadas com o BioNimbuZ em sua primeira versão de forma qualitativa, ou seja, não é possível fazer um paralelo direto para que ambos sejam comparados de maneira precisa com números.

5.3.1 Configuração de *Templates* de Aplicações

A configuração de *templates* de aplicações, como anteriormente descrito na Seção 5.1.1, permite a customização de aplicações para qualquer tipo de propósito, tornando o BNZ-2 uma plataforma genérica de execução de tarefas.

No BNZ-1 é necessário que haja um desenvolvimento direto na plataforma, para que a mesma dê suporte a novas aplicações, dificultando bastante a execução de diferentes *workflows*. Em outras palavras, o BNZ-1 permite atualmente que apenas um *workflow* seja executado por vez, com uma combinação máxima de 4 tarefas. Além disso, para realizar a adição de novos *workflows* é necessário o conhecimento de programação para tal, ou seja, o desejo de um usuário executar um novo *workflow* agrega, além da necessidade de profundo

conhecimento de como a plataforma foi construída, a necessidade de conhecimentos de programação.

5.3.2 Tarifação Direta

O BNZ-1 necessita ter uma credencial para cada nuvem utilizada, e os gastos de seus usuários deveriam ser repassados para os seus usuários, o que não ocorria, pois não havia essa funcionalidade.

Com a utilização direta das credenciais dos usuários, o BNZ-2 não necessita lidar com questões de tarifação indireta, e nem com mecanismos de repasse de custos. Assim, é garantido que cada usuário paga unicamente pelos recursos consumidos, independente dos provedores usados na execução da tarefa submetida.

5.3.3 Compartilhamento de Credenciais e Espaços

Anteriormente com a plataforma BNZ-1 não era possível compartilhar os resultados obtidos com as execuções dos *workflows*, algo extremamente necessário entre cientistas, principalmente, os que participam de um mesmo projeto.

Com a nova plataforma, os usuários têm acesso aos arquivos que estão compartilhados com os grupos em que participa, e também podem utilizar credenciais compartilhadas para dar início a execução de tarefas, comportamento comum em ambientes nos quais se recebe uma verba para um projeto que precisa compartilhar entre cientistas e alunos de uma universidade.

5.3.4 *Upload/Download* Direto de Arquivos

Na plataforma anterior, os usuários necessitavam enviar seus arquivos primeiramente para o BNZ-1, e este se encarregava de enviar os arquivos para as nuvens participantes da federação. O BNZ-2 possibilita aos seus usuários o envio direto de arquivos para a nuvem selecionada, sem o *overhead* de envio para o BNZ-1. Assim, o usuário faz uso dos seus serviços de armazenamento, utilizando suas próprias credenciais de nuvem.

5.3.5 Protocolo para Federação de Nuvens

As interfaces criadas para os *Plugins* de federação no BNZ-2 permitem que novos *Plugins* sejam acoplados à plataforma de maneira transparente aos seus usuários, o que não era possível no BNZ-1, visto que para adicionar um novo provedor de nuvem, era necessário atualizar toda a plataforma do BNZ-1 em nível de código.

5.4 Considerações Finais

Neste capítulo, foram detalhados os testes realizados com a plataforma de federação BioNimbuZ em sua primeira versão, e comparados com os testes realizados no BioNimbuZ 2, desenvolvido por este trabalho.

Além disso, foram exibidos os benefícios obtidos com a utilização da nova plataforma, que em vista das grandes mudanças em relação à sua primeira versão, puderam ser comparados apenas qualitativamente.

Para finalizar, o Capítulo 6 apresenta as conclusões deste trabalho e os trabalhos futuros considerados.

Capítulo 6

Considerações Finais

A computação em nuvem, em constante evolução, é um modelo com o qual é possível adquirir recursos computacionais como serviço e sob-demanda. Esses serviços são obtidos, principalmente, por meio de tecnologias de virtualização, e permitem um rápido provisionamento aos seus consumidores.

Os provedores desses serviços transparecem aos seus usuários a independência de localização e um *pool* infinito de seus recursos, mas na prática, aplicações que requerem um grande poder computacional podem esgotar seus recursos. Para que os provedores de nuvem mantenham seus negócios, e os negócios de seus clientes, tem-se adotado os mecanismos oferecidos pelo modelo de federação de nuvens.

Outra questão importante abordada pelo modelo de federação é o de resolver o *vendor lock-in*, em que os usuários se tornam dependentes de uma determinada plataforma de nuvem, em vista da carência de utilização de padrões abertos pelos provedores. Além disso, a federação deve transparecer aos seus usuários a independência de provedores de nuvem, ou seja, sempre que quiserem um recurso, a plataforma de federação deve oferecer o recurso mais adequado ao seu usuário, independentemente do provedor de nuvem utilizado.

Com o modelo de federação de nuvens são estabelecidos acordos entre as diferentes nuvens integrantes da federação, possibilitando a uma nuvem alugar seus recursos em caso de subutilização, ou alugar mais recursos externos caso não possua mais recursos disponíveis aos seus usuários. Todavia, o modelo de federação precisa de padrões abertos, a fim de garantir a integração entre diferentes provedores.

Assim sendo, este trabalho apresentou uma arquitetura de federação orientada a microsserviços para a execução de diferentes tarefas, oferecendo um ambiente de implantação simplificado e eficiente, em que os diversos provedores de nuvens possam participar da federação sem afetar o funcionamento global da aplicação com a utilização de microsserviços de *Plugins*, os quais podem ser escalados, evoluídos, inseridos ou removidos sem afetarem

a plataforma como um todo. Além do mais, são abordadas outras duas questões, a de execução e comunicação eficiente de tarefas em ambiente federado, e a de manutenção e utilização de credenciais de provedores de nuvem pela plataforma, de maneira a simplificar os mecanismos e as políticas de tarifação.

Quanto a facilitação de políticas de tarifação, os usuários podem utilizar suas credenciais e pagar somente por aquilo que utilizarem, sem a necessidade de que a plataforma de federação mantenha uma credencial própria para a execução de seus serviços e, posteriormente, faça a devida cobrança aos seus usuários.

A arquitetura desenvolvida foi implementada na plataforma BioNimbuZ, dando origem ao BioNimbuZ 2, com seu projeto disponibilizado no GitHub[1], que é um site que permite gerenciar e armazenar códigos-fonte. Os resultados mostraram que com a utilização da nova plataforma, houve ganhos com a redução de tempo de inicialização e de consumo de CPU e memória dos componentes que foram desacoplados da arquitetura inicial do BioNimbuZ.

Além disso, outros benefícios foram alcançados com o desenvolvimento da nova plataforma de federação, tais como o de permitir a execução de *workflows* e de aplicações com diversos fins, e não somente as de Bioinformática; o de tarifar diretamente seus usuários que agora utilizam suas próprias credenciais e mantidas pela plataforma; o benefício do compartilhamento das credenciais e de espaços de armazenamento de forma segura com outros usuários; o de realizar o *upload* e o *download* direto de seus arquivos, sem trafegar pelos servidores da plataforma do BioNimbuZ 2; e por fim, o benefício de disponibilização de um protocolo simples utilizado pela plataforma para que novos *Plugins* de nuvens sejam acoplados de tal maneira que não afetem a execução geral da plataforma.

6.1 Trabalhos Futuros

Como visto no trabalho apresentado, existem muitos desafios para a implementação de uma plataforma de federação de nuvens, entre eles a falta de padrões abertos para sua implementação, a utilização de sistemas distribuídos e questões de segurança e de monitoramento de recursos, o que exige uma gama de soluções e tecnologias para alcançar esse objetivo.

Neste trabalho utilizou-se o conceito de microsserviços, o que possibilitou entre outras características, migrar um sistema monolítico para um sistema com pequenos serviços com diferentes responsabilidades, cada um com seu processo, permitindo assim, a utilização de diferentes tecnologias, conforme a que melhor atenda às suas necessidades.

[1]BioNimbuZ 2, `https://github.com/bionimbuz/BionimbuzWeb`

Em trabalhos futuros, deve ser levada em consideração a utilização de bancos de dados não relacionais, os chamados NoSQL [63], em conjunto com a solução relacional atualmente desenvolvida, ou mesmo se é possível fazer uma migração total para um banco não relacional, o que permitirá uma melhor distribuição do banco de dados utilizado.

Além disso, deve ser considerada a utilização de ambientes de desenvolvimento mais desacoplados de *back-end*, e que facilitem a utilização dos microsserviços, como fornecidos, por exemplo, pelos diversos *frameworks* e bibliotecas JavaScript existentes, como o Angular [64], Vue [65] e o Ember [66].

Por fim, para a plataforma BioNimbuZ 2, devem ser implementados novos *Plugins* de nuvens para um maior suporte de nuvens à federação, como por exemplo a plataforma de nuvem da Microsoft, a Azure [67], e a plataforma de nuvem de armazenamento S3 da Amazon [68]. Assim como a implementação de *Plugins* para trabalhar com nuvens privadas, tais como CloudStack [69] e OpenStack [70].

Referências

[1] Toosi, Adel Nadjaran, Rodrigo N Calheiros e Rajkumar Buyya: *Interconnected cloud computing environments: Challenges, taxonomy, and survey.* ACM Computing Surveys (CSUR), 47(1):7, 2014. xi, 6, 7, 8, 10, 11

[2] Celesti, A., F. Tusa, M. Villari e A. Puliafito: *How to enhance cloud architectures to enable cross-federation.* Em *2010 IEEE 3rd International Conference on Cloud Computing*, páginas 337–345, July 2010. xi, xii, 1, 7, 9, 76, 77, 83

[3] Bittman, Thomas: *The evolution of the cloud computing market. gartner blog network*, 2008. xi, 9, 10

[4] Biran, Yahav, George Collins, Syed Azam e Joel Dubow: *Federated cloud computing as system of systems.* Em *Computing, Networking and Communications (ICNC), 2017 International Conference on*, páginas 711–718. IEEE, 2017. xi, 11, 12

[5] Bharathi, S., A. Chervenak, E. Deelman, G. Mehta, M. H. Su e K. Vahi: *Characterization of scientific workflows.* Em *2008 Third Workshop on Workflows in Support of Large-Scale Science*, páginas 1–10, Nov 2008. xi, 15, 16

[6] Moura, B. R.: *Arquitetura de um controlador de sla para ambiente de nuvens federadas.* Tese de Mestrado, Departamento de Ciência de Computação, Universidade de Brasília, 2017. `http://repositorio.unb.br/handle/10482/24192`. xi, 2, 18, 19, 20

[7] Buyya, Rajkumar, Rajiv Ranjan e Rodrigo Calheiros: *Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services.* Algorithms and architectures for parallel processing, páginas 13–31, 2010. xii, 1, 74, 75, 83

[8] Demchenko, Yuri, Canh Ngo, Cees de Laat e Craig Lee: *Federated access control in heterogeneous intercloud environment: Basic models and architecture patterns.* Em *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, páginas 439–445. IEEE, 2014. xii, 1, 78, 79, 83

[9] Silva, Vítor, D Oliveira e Marta Mattoso: *Scicumulus 2.0: Um sistema de gerência de workflows científicos para nuvens orientado a fluxo de dados.* Sessão de Demos do XXIX Simpósio Brasileiro de Banco de Dados, 2014. xii, 80

[10] Diaz-Montes, Javier, Moustafa AbdelBaky, Mengsong Zou e Manish Parashar: *Cometcloud: Enabling software-defined federations for end-to-end application workflows.* IEEE Internet Computing, 19(1):69–73, 2015. xii, 81, 82

[11] Buyya, Rajkumar, Chee Shin Yeo, Srikumar Venugopal, James Broberg e 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. 1, 4, 74

[12] Foster, I., Y. Zhao, I. Raicu e S. Lu: *Cloud computing and grid computing 360-degree compared.* Em *2008 Grid Computing Environments Workshop*, páginas 1–10, Nov 2008. 1, 5

[13] Mell, Peter e Tim Grance: *The nist definition of cloud computing.* National Institute of Standards and Technology, 53(6):50, 2009. 1, 4, 5, 6

[14] Rosa, Michel, Breno Moura, Guilherme Vergara, Lucas Santos, Edward Ribeiro, Maristela Holanda, Maria Emília Walter e Aletéia Araújo: *Bionimbuz: A federated cloud platform for bioinformatics applications.* Em *Bioinformatics and Biomedicine (BIBM), 2016 IEEE International Conference on*, páginas 548–555. IEEE, 2016. 1, 2, 19, 83

[15] Kim, Hyunjoo e Manish Parashar: *Cometcloud: An autonomic cloud engine.* Cloud Computing: Principles and Paradigms, páginas 275–297, 2011. 1, 81, 83

[16] Oliveira, D. de, E. Ogasawara, F. Baião e M. Mattoso: *Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows.* Em *2010 IEEE 3rd International Conference on Cloud Computing*, páginas 378–385, July 2010. 1, 80, 83

[17] Saldanha, H. V.: *Bionimbus: uma arquitetura de federação de nuvens computacionais híbrida para a execução de workflows de bioinformática.* Tese de Mestrado, Departamento de Ciência de Computação, Universidade de Brasília, 2012. `http://repositorio.unb.br/handle/10482/12046`. 2, 18

[18] Vergara, G. F.: *Arquitetura de um controlador de elasticidade para nuvens federadas.* Tese de Mestrado, Departamento de Ciência de Computação, Universidade de Brasília, 2017. `http://repositorio.unb.br/handle/10482/23700`. 2, 18, 19

[19] Azevedo, D. R. e T. B. Freitas Júnior: *Biocirrus: uma nova política de armazenamento para a plataforma BioNimbuZ de nuvem federada.* `http://bdm.unb.br/handle/10483/13199`, 2015. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 2, 18

[20] Costa, H. H. P. M.: *Controle de acesso na plataforma de nuvem federada BioNimbuZ.* `http://bdm.unb.br/handle/10483/11141`, 2015. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 2, 18

[21] Ramos, V. A.: *Um sistema gerenciador de workflows científicos para a plataforma de nuvens federadas BioNimbuZ.* `http://bdm.unb.br/handle/10483/13145`, 2016. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 2, 18

[22] Barreiros Júnior, W. O.: *Escalonador de tarefas para o plataforma de nuvens federadas BioNimbuZ usando beam search iterativo multiobjetivo.* `http://bdm.unb.br/handle/10483/13146`, 2016. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 2, 18, 22

[23] Baraiya, V e V Singh: *Netflix conductor: A microservices orchestrator*. Netflix, 12:59, 2016. 2, 12

[24] Haddad, Einas: *Service-oriented architecture: Scaling the uber engineering codebase as we grow*, 2015. 2, 12

[25] Käppler, Matthias: *Inside a soundcloud microservice (accessed on 11/09/2018). 2017*. URL: https://developers.soundcloud.com/blog/inside-a-soundcloud-microservice, 2017. 2, 12

[26] Fowler, Martin e James Lewis: *Microservices (acessado em 11/09/2018)*. URL: http://martinfowler. com/articles/microservice s. html, 2014. 2, 12, 25

[27] Armbrust, Michael, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica e Matei Zaharia: *A view of cloud computing*. Commun. ACM, 53(4):50–58, abril 2010, ISSN 0001-0782. `http://doi.acm.org/10.1145/1721654.1721672`. 5

[28] Rimal, B. P., E. Choi e I. Lumb: *A taxonomy and survey of cloud computing systems*. Em *2009 Fifth International Joint Conference on INC, IMS and IDC*, páginas 44–51, Aug 2009. 6

[29] Barril, J. F. H., J. Ruyter e Qing Tan: *A view on internet of things driving cloud federation*. Em *2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, páginas 221–226, July 2016. 7, 11, 16

[30] Kogias, D. G., M. G. Xevgenis e C. Z. Patrikakis: *Cloud federation and the evolution of cloud computing*. Computer, 49(11):96–99, Nov 2016, ISSN 0018-9162. 7, 10

[31] Goiri, I., J. Guitart e J. Torres: *Characterizing cloud federation for enhancing providers' profit*. Em *2010 IEEE 3rd International Conference on Cloud Computing*, páginas 123–130, July 2010. 8, 10

[32] Kurze, Tobias, Markus Klems, David Bermbach, Alexander Lenk, Stefan Tai e Marcel Kunze: *Cloud federation*. Em *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011)*, volume 1971548541. Citeseer, 2011. 11

[33] Newman, Sam: *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015. 12, 13

[34] Raicu, I., I. T. Foster e Yong Zhao: *Many-task computing for grids and supercomputers*. Em *2008 Workshop on Many-Task Computing on Grids and Supercomputers*, páginas 1–11, Nov 2008. 14

[35] Mattoso, Marta, Jonas Dias, Flavio Costa, Daniel de Oliveira e Eduardo Ogasawara: *Experiences in using provenance to optimize the parallel execution of scientific workflows steered by users*. Em *Workshop of Provenance Analytics*, 2014. 14

[36] Hollingsworth, David, Fujitsu Services e United Kingdom: *The workflow reference model: 10 years on*. Em *Fujitsu Services, UK; Technical Committee Chair of WfMC*, páginas 295–312, 2004. 14

[37] Taylor, Ian J., Ewa Deelman, Dennis B. Gannon e Matthew Shields: *Workflows for e-Science: Scientific Workflows for Grids*. Springer Publishing Company, Incorporated, 2014, ISBN 1849966192, 9781849966191. 14

[38] Teonadi, Hidayat, Miron Livny, Jonathan Livny e Matthew K Waldor: *High-throughput, kingdom-wide prediction and annotation of bacterial non-coding rnas*. 2008. 15

[39] Saldanha, Hugo, Edward Ribeiro, Carlos Borges, Aletéia Araújo, Ricardo Gallon, Maristela Holanda, Maria Emília Walter, Roberto Togawa e João Carlos Setubal: *Towards a hybrid federated cloud platform to efficiently execute bioinformatics workflows*. Em *BioInformatics*. InTech, 2012. 18

[40] Lima, Deric, Breno Moura, Gabriel Oliveira, Edward Ribeiro, Aleteia Araujo, Maristela Holanda, Roberto Togawa e Maria Emilia Walter: *A storage policy for a hybrid federated cloud platform: a case study for bioinformatics*. Em *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, páginas 738–747. IEEE, 2014. 18

[41] Oliveira, Gabriel SS de, Edward Ribeiro, Diogo A Ferreira, Aletéia PF Araújo, Maristela T Holanda e Maria Emilia MT Walter: *Acosched: A scheduling algorithm in a federated cloud infrastructure for bioinformatics applications*. Em *Bioinformatics and Biomedicine (BIBM), 2013 IEEE International Conference on*, páginas 8–14. IEEE, 2013. 18

[42] Gallon, Ricardo, Maristela Holanda, Aletéia Araújo e Maria E Walter: *Storage policy for genomic data in hybrid federated clouds*. Em *Brazilian Symposium on Bioinformatics*, páginas 107–114. Springer, 2014. 18

[43] Leonard Richardson, Sam Ruby, David Heinemeier Hansson: *Restful Web Services*. O'Reilly Media, 1ª edição, 2007, ISBN 0596529260,9780596529260. 19, 29, 52

[44] Feng, Xinyang, Jianjing Shen e Ying Fan: *Rest: An alternative to rpc for web services architecture*. Em *2009 First International Conference on Future Information Networks*, páginas 7–10, Oct 2009. 19

[45] Feo, T. A. e M. G. C. Resende: *Greedy randomized adaptive search procedures*. Journal of Global Optimization, 6(2):109–133, 1995, ISSN 1573-2916. `http://dx.doi.org/10.1007/BF01096763`. 22

[46] Lightbend: *Playframework*. `https://www.playframework.com/`. 29

[47] Jones, Michael, John Bradley e Nat Sakimura: *Json web token (jwt)*. Relatório Técnico, 2015. RFC 7519. 29

[48] Spurlock, Jake: *Bootstrap: Responsive Web Development*. " O'Reilly Media, Inc.", 2013. 29

[49] Dierks, Tim: *The transport layer security (tls) protocol version 1.2*. 2008. 34, 36

[50] Kanewala, Thejaka Amila, Suresh Marru, Jim Basney e Marlon Pierce: *A credential store for multi-tenant science gateways*. Em *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, páginas 445–454. IEEE, 2014. 35

[51] Daemen, Joan e Vincent Rijmen: *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013. 36

[52] Junqueira, Flavio e Benjamin Reed: *ZooKeeper: distributed process coordination*. " O'Reilly Media, Inc.", 2013. 51

[53] Pivotal Software: *Spring boot*. `https://spring.io/projects/spring-boot`. 52

[54] Berners-Lee, Tim, Roy Fielding e H Frystyk: *Hypertext transfer protocol—http/1.0, may 1996*. Relatório Técnico, 2005. RFC 1945. 53

[55] Gomes, J. C.: *Modelo multi-estratégico de tolerância a falhas para ambiente de nuvem federada*. Tese de Mestrado, Departamento de Ciência de Computação, Universidade de Brasília, 2018. `http://repositorio.unb.br/handle/`. 69, 70, 71

[56] Hasan, Moin e Major Singh Goraya: *Fault tolerance in cloud computing environment: A systematic survey*. Computers in Industry, 99:156 – 172, 2018, ISSN 0166-3615. `http://www.sciencedirect.com/science/article/pii/S0166361517304438`. 71

[57] Demchenko, Yuri, Canh Ngo, Cees De Laat, Joan Antoni Garcia-Espin, Sergi Figuerola, Juan Rodriguez, Luis M Contreras, Giada Landi e Nicola Ciulli: *Intercloud architecture framework for heterogeneous cloud based infrastructure services provisioning on-demand*. Em *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, páginas 777–784. IEEE, 2013. 77

[58] Demchenko, Yuri, Fatih Turkmen, Cees de Laat e Mathias Slawik: *Defining intercloud security framework and architecture components for multi-cloud data intensive applications*. Em *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, páginas 945–952. IEEE Press, 2017. 78

[59] Viana, Vitor, Daniel de Oliveira, Eduardo S Ogasawara e Marta Mattoso: *Scicumulus-ecm: Um serviço de custos para a execução de workflows científicos em nuvens computacionais*. Em *SBBD (Short Papers)*, páginas 1–8, 2011. 80

[60] Costa, F., D. d. Oliveira, K. Ocaña, E. Ogasawara, J. Dias e M. Mattoso: *Handling failures in parallel scientific workflows using clouds*. Em *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, páginas 129–139, Nov 2012. 80

[61] Montes, Javier Diaz, Mengsong Zou, Rahul Singh, Shu Tao e Manish Parashar: *Data-driven workflows in multi-cloud marketplaces*. Em *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, páginas 168–175. IEEE, 2014. 81

[62] Wang, Jianwu, Moustafa Abdelbaky, Javier Diaz-Montes, Shweta Purawat, Manish Parashar e Ilkay Altintas: *Kepler+ cometcloud: dynamic scientific workflow execution on federated cloud resources*. Procedia Computer Science, 80:700–711, 2016. 81

[63] Pokorny, Jaroslav: *Nosql databases: a step to database scalability in web environment*. International Journal of Web Information Systems, 9(1):69–82, 2013. 94

[64] Angular: *One framework. mobile & desktop.* `https://angular.io/`. 94

[65] Vue: *The progressive javascript framework.* `https://vuejs.org/`. 94

[66] Tilde Inc.: *Ember.* `https://www.emberjs.com/`. 94

[67] Microsoft: *Azure*. `https://azure.microsoft.com/`. 94

[68] Amazon: *S3*. `https://aws.amazon.com/pt/s3/`. 94

[69] Apache: *Cloudstack*. `https://cloudstack.apache.org/`. 94

[70] Rackspace: *Openstack*. `https://www.openstack.org/`. 94