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

Uma Arquitetura Baseada em Containers para Workflows de Bioinformática em Nuvens Federadas

Tiago Henrique Costa Rodrigues Alves

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

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

Brasília
2017

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

AAL474a

Alves, Tiago Henrique Costa Rodrigues
Uma Arquitetura Baseada em Containers para Workflows de Bioinformática em Nuvens Federadas / Tiago Henrique Costa Rodrigues Alves; orientador Aletéia Patrícia Favacho de Araújo. -- Brasília, 2017.
93 p.

Dissertação (Mestrado - Mestrado em Informática) -- Universidade de Brasília, 2017.

1. Computação em Nuvem. 2. Nuvens Federadas. 3. Containers. 4. Docker. 5. Bioinformática. I. Patrícia Favacho de Araújo, Aletéia, orient. II. Título.

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

Uma Arquitetura Baseada em Containers para Workflows de Bioinformática em Nuvens Federadas

Tiago Henrique Costa Rodrigues Alves

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 Clicia Castro
IME/UERJ

Prof.ª Dr.ª Maria Emília Machado Teles
CIC/UnB

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

Brasília, 06 de Julho de 2017

Dedicatória

Dedico este trabalho ao meu filho Benjamin e a minha esposa Lise. Sem vocês eu não teria chegado até aqui.

Agradecimentos

Agradeço a minha esposa Lise pela compreensão, dedicação e incentivo empreendidos durante todos os semestres que culminaram com a entrega deste trabalho, e ao meu filho Benjamin, que durante os meses finais dessa jornada me trouxe a imensa alegria de ser pai.

A minha orientadora e professora Aletéia que não me permitiu esmorecer no momento mais difícil.

Ao meu amigo Luiz Serique por despertar o desejo de iniciar este trabalho e aos demais amigos do Tribunal de Justiça do Distrito Federal por compreenderem quando precisei me ausentar.

Aos colegas da UnB que me acompanharam durante cada semestre. Essa amizade foi fundamental no dia a dia.

E principalmente ao Autor e Consumador da minha fé.[Hb 12:1-3]

Resumo

Reproduzir experimentos de Bioinformática pode ser uma atividade dispendiosa. Os recursos computacionais necessários, muitas vezes, não estão disponíveis. Instalar e/ou compilar os softwares utilizados em experimentos de Bioinformática, gerenciar suas dependências e garantir a execução das versões corretas são atividades que podem consumir bastante tempo. Este trabalho propõe uma arquitetura baseada em *containers* para a execução de *workflows* de Bioinformática em nuvens federadas capaz de auxiliar pesquisadores na execução, distribuição e reprodução de experimentos científicos, e no provisionamento de recursos computacionais em diferentes provedores de nuvem.

Palavras-chave: computação em nuvem, nuvens federadas, *containers*, Bioinformática

Abstract

Playing Bioinformatics experiments can be a costly activity. The necessary computational resources are often not available. Installing or compiling software used in Bioinformatics experiments, managing their dependencies, and ensuring that the correct version is running are very time-consuming activities. This work suggests a containers based architecture for Bioinformatics workflows in federated clouds capable of assisting researchers in the execution, distribution and reproduction of scientific experiments, and in the provisioning of computational resources in different cloud providers.

Keywords: cloud computing, federated cloud, containers, Bioinformatics

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

De acordo com pesquisas realizadas em 2013 pela Gartner, empresa líder no segmento de pesquisas e consultoria em Tecnologia da Informação – *TI*, o uso da computação em nuvem cresceria ao ponto de se tornar o novo concentrador de gastos de Tecnologia da Informação até o ano de 2016 [26]. Já em 2015, a mesma empresa afirmou que, naquele mesmo ano, o gasto mundial em computação em nuvem, entregue sob o modelo de Infraestrutura como Serviço (*IaaS – Infraestructure-as-a-Service* [52]), cresceria 32,8% em comparação com o ano anterior [27]. No ano subsequente, mantendo o mesmo direcionamento, a Gartner disse que até 2020 a mudança para a nuvem afetará os gastos de TI em mais de um trilhão de dólares [29], e que o modelo híbrido[1] de utilização da nuvem será o mais comum [28].

Nesse sentido, os consumidores dos serviços de computação em nuvem entregues sob o modelo de *IaaS* devem estar atentos à escolha do provedor de nuvem. Ferry *et al.* [24] mencionam que a habilidade em executar e gerenciar sistemas multi-nuvem permite explorar as peculiaridades de cada solução em nuvem, e ainda otimizar a performance, a disponibilidade e o custo das aplicações. Além disso, os autores alertam que a diversidade de provedores resulta em soluções heterogêneas e, por vezes, incompatíveis. Fato que dificulta explorar todo o potencial da computação em nuvem, uma vez que impede a interoperabilidade e promove a dependência de fornecedor, bem como aumenta a complexidade de desenvolvimento e de administração dos sistemas multi-nuvem [24].

Seguindo nesse cenário, nota-se na literatura soluções e plataformas que buscam homogenizar a distribuição de aplicações e facilitar a sua interoperabilidade. Uma dessas soluções é a tecnologia de *Containers Linux*. De acordo com Di Tommaso *et al.* [18], *containers* permitem aplicações executarem em um empacotamento isolado e autocontido, capaz de ser eficientemente distribuído e executado de maneira portável através de uma ampla quantidade de plataformas computacionais. Assim, valendo-se das vanta-

[1] A computação em nuvem híbrida refere-se ao provisionamento, uso e gerenciamento de serviços coordenados numa composição de serviços em nuvens privadas e públicas [30].

gens quanto ao uso de *containers*, a plataforma Docker [21], solução baseada em *containers* que encapsula um software em um sistema de arquivos completo, contendo o necessário para sua execução, garante que o *container* executará da mesma maneira, independente do ambiente em que se está executando.

Ainda sob essa percepção, é possível encontrar na literatura pesquisas abordando o uso de *containers* na execução de *workflows* de Bioinformática. Em 2015, por exemplo, Zheng e Thain [79] propuseram um estudo de caso utilizando *containers* Docker e as ferramentas de *workflow MakeFlow*[2] e *Work Queue*[3]. No mesmo ano, Di Tommaso *et al.* [18] avaliaram o impacto da utilização de *containers* Docker na performance de *workflows* de genomas, e ainda Boettiger [8] abordou as vantagens da utilização de *containers* Docker na reprodutibilidade de pesquisas científicas.

Por outro lado, o BioNimbuZ [61], que é uma plataforma de federação de nuvens computacionais híbrida, capaz de executar aplicações e *workflows* de Bioinformática, de maneira transparente, flexível, eficiente e tolerante a falhas propõe uma abstração da complexidade necessária para gerenciar os diversos provedores de nuvens, a interoperabilidade entre eles, e as diferentes formas de contratação e cobrança.

Todavia, o BioNimbuZ não dispõe de nenhuma funcionalidade para gerenciar a instalação e a distribuição de aplicações de Bioinformática. Dessa forma, uma máquina virtual deve ser preparada previamente com todas as aplicações instaladas. Além disso, os *workflows* dos usuários podem utilizar diferentes versões da mesma aplicação, ou por vezes, um mesmo *workflow*, para efeito de comparação, deve ser executado em diferentes versões. Nesse aspecto, o BioNimbuZ não implementa funcionalidade capaz de gerenciar ou controlar versões de uma mesma aplicação.

Diante do exposto, com o intuito de homogenizar a distribuição e o uso das aplicações de Bioinformática, e melhorar a interoperabilidade entre provedores de nuvem sob o modelo de infraestrutura como serviço, este trabalho propõe uma arquitetura, baseada em *containers*, capaz de gerenciar *workflows* de Bioinformática em nuvens federadas.

1.1 Motivação

O uso de *containers* tem se mostrado sólido e crescente no encapsulamento e na distribuição de aplicações. Contudo, as principais plataformas capazes de gerenciar *containers* não estão preparadas para execução de *workflows* em nuvens federadas (ou ao menos em múltiplos *datacenters*). Além disso, mesmo o BioNimbuZ, que é capaz de utilizar nuvens federadas para a execução de *workflows*, não utiliza nenhum mecanismo para gerenciar e

[2]MakeFlow - http://ccl.cse.nd.edu/software/makeflow/
[3]Work Queue - http://ccl.cse.nd.edu/software/workqueue/

coordenar a distribuição e a instalação das diversas aplicações necessárias para a execução de *workflows* de Bioinformática.

Assim, a motivação deste trabalho é utilizar a tecnologia de *containers* para gerenciar e coordenar a distribuição, a instalação e a execução das diversas aplicações (e suas versões), necessárias para *workflows* de Bioinformática em ambientes de nuvens federadas.

1.2 Problema

Manter uma solução capaz de gerenciar um conjunto de aplicações necessárias para *workflows* de Bioinformática executados em um ambiente de nuvens federadas é uma atividade bastante dispendiosa. A forma tradicional de instalação e de execução de aplicações utiliza uma imagem de máquina virtual para distribuir um ambiente de execução padronizado. Contudo, essa estratégia leva aos seguintes problemas:

- Manter a compatibilidade entre as versões das aplicações de Bioinformática e suas dependências em uma mesma imagem muitas vezes não é possível, sendo necessário uma imagem para cada conjunto de aplicações compatíveis;
- Instalar ou atualizar uma nova aplicação exige a compilação e o envio de uma nova imagem da máquina para a nuvem;
- A mesma imagem pode não ser executada em provedores de nuvem diferentes, sendo necessário a geração de uma nova imagem de acordo com o provedor que se deseja executar as aplicações.

Todavia, mesmo utilizando a tecnologia de *containers* para solucionar os problemas enumerados, ainda ficam evidentes os seguintes problemas:

- As plataformas que gerenciam *containers* não são capazes de executar *workflows* em nuvens federadas;
- Carência de *containers* preparados para trabalhar com aplicações de Bioinformática.

Diante disso, este trabalho pretende fazer uso da tecnologia de *containers* para tratar os problemas destacados anteriormente.

1.3 Objetivos

1.3.1 Principal

O objetivo principal deste trabalho é definir uma arquitetura capaz de utilizar a tecnologia de *containers* para gerenciar a distribuição, a instalação e a execução de aplicações de Bioinformática, encapsuladas sob a coordenação de uma plataforma de nuvem federada.

1.3.2 Específicos

Para cumprir o objetivo principal, este trabalho tem os seguintes objetivos específicos:

- Definir uma arquitetura para execução de *workflows* de Bioinformática em nuvens federadas utilizando *containers*;
- Avaliar a arquitetura da plataforma de federação BioNimbuZ, que será usada como estudo de caso;
- Executar um *workflow* real de Bioinformática na arquitetura proposta;
- Analisar o desempenho do uso de *containers* em nuvem federada versus o uso em uma única nuvem.

1.4 Estrutura do Trabalho

Este trabalho está dividido, além deste capítulo, em mais seis. No Capítulo 2 são apresentados os conceitos e as classificações de computação em nuvem, as definições de federação de nuvens e de Bioinformática.

O Capítulo 3 detalha a arquitetura da plataforma de federação de nuvens BioNimbuZ e descreve as principais limitações desta plataforma.

No Capítulo 4 são exibidos os conceitos e as característica de *containers* juntamente com a evolução desta tecnologia. Além disso, também são enumeradas as principais plataformas de orquestração de *containers* e suas diferenças. No mesmo capítulo são expostas as motivações na escolha da tecnologia de *containers* para compor a arquitetura proposta.

No Capítulo 5 é apresentada a arquitetura proposta neste trabalho, baseado em *containers*, para gerenciar a distribuição, a instalação e a execução de aplicações de Bioinformática em um ambiente de nuvens federadas. Além disso, o mesmo capítulo aborda trabalhos relacionados, comparando-os com a arquitetura proposta.

O Capítulo 6 exibe os resultados obtidos através da execução de um *workflow* de Bioinformática na arquitetura proposta neste trabalho.

Por último, a conclusão e os trabalhos futuros são apresentados no Capítulo 7.

Capítulo 2

Computação em Nuvem, Nuvens Federadas e Bioinformática

Neste capítulo são apresentadas as definições, as características e os modelos de serviço e de implantação de computação em nuvem. Além disso, é abordada a definição de nuvens federadas, bem como cenários nos quais tal abordagem pode ser necessária. Por fim, são apresentados alguns conceitos de Bioinformática utilizados neste trabalho.

2.1 Computação em Nuvem

Em 2008, Vaquero *et al.* [73] definiram nuvem como um grande conjunto de recursos virtualizados acessíveis e de fácil utilização (tais como hardware, plataformas de desenvolvimento e/ou serviços). Esses recursos podem ser dinamicamente reconfigurados para se ajustarem a uma carga variável (escalável), permitindo uma utilização otimizada de recursos. O conjunto de recursos é tipicamente explorado sob o modelo de pagamento pelo uso (*pay-per-use*), no qual garantias são oferecidas pelos provedores de infraestrutura sob Acordos de Nível de Serviço (*Service Level Agreements* - SLA) customizáveis.

No mesmo ano, Armbrust *et al.* [5] mencionaram que a computação em nuvem faz referência tanto às aplicações entregues como serviço na Internet, quanto ao hardware e ao software que proveem esses serviços nos *datacenters*. Além disso, Buyya *et al.* [10] definiram nuvem como um tipo de sistema paralelo e distribuído que consiste em uma coleção interconectada de computadores virtualizados, e que são dinamicamente provisionados e apresentados como um ou mais recursos computacionais unificados, baseados em um Acordo de Nível de Serviço – SLA, por meio de uma negociação entre o provedor do serviço e o consumidor.

Por outro lado, em 2011, Mell e Grance [52] definiram computação em nuvem como sendo um modelo que permite, através da rede, o acesso ubíquo, conveniente e sob de-

manda a um conjunto compartilhado de recursos computacionais configuráveis (por exemplo rede, servidores, armazenamento, aplicações e serviços), capaz de ser rapidamente provisionado ou liberado, com o mínimo de esforço de gerenciamento ou a interação do provedor do serviço.

Diante das diversas definições apresentadas, este trabalho usa a definição de Mell e Grance [52] para computação em nuvem, e o termo nuvem é utilizado de forma genérica para definir um provedor de serviços sob o modelo de computação em nuvem ou do próprio modelo em si.

2.1.1 Características

Vaquero *et al.* [73] enumeraram um conjunto amplo de características presentes na computação em nuvem, contudo, enfatizaram apenas três delas em sua definição: escalabilidade, modelo de pagamento pelo uso e virtualização. Essas características podem ser subentendidas naquelas propostas por Mell e Grance [52], a saber:

- *Auto atendimento sob demanda* - O consumidor pode provisionar unilateralmente recursos computacionais de acordo com sua necessidade, sem depender da iteração humana. Neste escopo, a virtualização tem papel primordial, segmentando recursos físicos em virtuais, tais como os utilizados pelos Monitores de Máquinas Virtuais (*Virtual Machines Monitors - VMM*) ou Hipervisores (*Hypervisors*), e as Redes definidas por *Software* (*Software defined Networks - SDN*);

- *Amplo acesso a rede* - Recursos estão disponíveis através da rede e são acessados por mecanismos padronizados por clientes heterogêneos (por exemplo *smartphones*, *tablets*, computadores e estações de trabalho);

- *Compartilhamento de recursos* - Os recursos computacionais do provedor são reunidos para servir múltiplos consumidores utilizando um modelo multi-inquilino (*multi-tenancy*), com diferentes recursos físicos ou virtuais, sendo dinamicamente atribuídos e reatribuídos conforme a demanda do consumidor. Assim sendo, tem-se o conceito de independência de localização, no qual o cliente não sabe com exatidão a localização do recurso, sendo possível apenas uma escolha de forma ampla, tais como país, região e *datacenter*;

- *Elasticidade rápida* - Recursos podem ser elasticamente provisionados e liberados, em alguns casos automaticamente, escalando rapidamente conforme a demanda. Para o consumidor, os recursos disponíveis aparentam ser inesgotáveis. Nesse cenário, é importante mencionar uma diferença sutil entre os termos escalabilidade e elasticidade. Segundo Tost [71], a escalabilidade costuma ser usada para descrever

os mecanismos e os procedimentos aplicados a um sistema com mudanças previsíveis, quase lineares, de um volume de carga de trabalho em um período de tempo. Isso significa que pode haver um planejamento de capacidade adequado com antecedência. Os ajustes podem ser feitos manualmente ou com base em estatísticas. Por outro lado, a elasticidade é usada em casos em que a mudança nas cargas de trabalho ocorrem rapidamente, com aumentos ou quedas drásticos em um período curto de tempo, com pouco ou nenhum aviso prévio;

- *Serviço mensurável* - Os provedores automaticamente controlam e otimizam o uso dos recursos, medindo-os de forma apropriada quanto ao tipo de serviço (por exemplo armazenamento, processamento, largura de banda, número de contas de usuários). A utilização dos recursos pode ser monitorada, controlada e reportada, de forma transparente para o provedor e também ao consumidor do serviço utilizado. Neste sentido, o modelo de cobrança adotado na grande maioria das vezes é o pagamento pelo uso (*pay-per-use* ou *pay-as-you-go*), no qual o consumidor tem a percepção clara de quanto cada serviço custa e de como ele é medido, pagando conforme a variação das suas necessidades de utilização.

Diante do exposto, é importante destacar que este trabalho considera, necessariamente, que uma nuvem deve entregar as cinco características essenciais aqui enumeradas.

2.1.2 Modelos de Serviço

Atualmente, são observadas diversas terminologias sob o sufixo "como Serviço". Mell e Grance [52] propuseram três modelos: Software como Serviço (*Software as a Service - SaaS*), Plataforma como Serviço (*Platform as a Service - PaaS*) e Infraestrutura como Serviço (*Infraestrutura as a Service - IaaS*). Armbrust *et al.* [5], sob uma visão diferente, consideram *PaaS* e *IaaS* de forma conjunta, mas reconhecem *SaaS* como um modelo de consenso geral. Neste trabalho são utilizadas as definições de modelos de serviço propostas por Mell e Grance [52], a qual identifica os três modelos:

- Software como Serviço (*SaaS*) – O provedor entrega ao consumidor um aplicativo que é executado em uma infraestrutura na nuvem. O consumidor não gerencia ou controla a camada de infraestrutura (rede, servidores, sistemas operacionais, armazenamento) responsável pela execução do aplicativo;
- Plataforma como Serviço (*PaaS*) – O consumidor implanta um aplicativo por ele desenvolvido ou adquirido no ambiente do provedor. Da mesma forma que o modelo *SaaS*, o consumidor não gerencia ou controla a camada de infraestrutura, contudo,

é dado o controle ao aplicativo implantado e às configurações do ambiente que o aplicativo é executado (plataforma);

- Infraestrutura como Serviço (*IaaS*) – O consumidor é capaz de provisionar processamento, armazenamento, redes e outros recursos computacionais fundamentais no qual pode executar *softwares* arbitrários (sistemas operacionais e aplicativos).

Em todos os modelos, o provedor deve garantir que a utilização de recursos por um consumidor não impacte negativamente (performance, acesso, segurança) no uso de outro.

2.1.3 Modelos de Implantação

Os modelos de implantação propostos por Mell e Grance [52] são bem aceitos e simples. Sendo assim, é o modelo adotado neste trabalho. Neste modelo as nuvens são classificadas em quatro tipos, as quais são:

- Nuvem Privada - A infraestrutura da nuvem é utilizada por uma única organização. A organização pode ou não ser a proprietária dos recursos físicos, sendo que estes podem estar disponíveis em uma localização remota. Além disso, o gerenciamento e a operação podem ser realizados pela organização que utiliza esse modelo de implantação ou podem ser terceirizados;

- Nuvem Comunitária - A infraestrutura da nuvem é utilizada exclusivamente por uma comunidade de organizações que divide seus objetivos em comum (por exemplo requisitos de segurança e missão). Além disso, o gerenciamento e a operação podem ser realizados por uma, ou mais organizações da comunidade, ou por um terceiro. Os recursos físicos, bem como sua localização, obedecem o mesmo conceito da nuvem privada;

- Nuvem Pública - A infraestrutura da nuvem é aberta para utilização do público em geral. Organizações acadêmicas, comerciais, de governo ou uma combinação entre elas podem ser a proprietária dos recursos físicos, além de gerenciá-los e operá-los. Os recursos estão localizados nas dependências do provedor de nuvem;

- Nuvem Híbrida - A infraestrutura da nuvem é composta por dois ou mais modelos distintos de implantação (privado, comunitário ou público) que são utilizados de maneira combinada, de forma que um provedor não necessite ter conhecimento da existência do outro.

2.1.4 Vantagens da Computação em Nuvem

Do ponto de vista econômico, Armbrust *et al.* [5] enumeraram três cenários favoráveis para a utilização da computação em nuvem:

- Quando a demanda por um serviço varia com o tempo – Não é necessária a alocação fixa dos recursos baseados no pico de utilização;
- Quando a demanda por um serviço não é conhecida com antecedência – Os recursos necessários para sustentar o serviço podem ser alocados de acordo com a necessidade;
- Quando organizações desejam realizar análises em lote mais rapidamente – A ideia é que mil máquinas em execução por uma hora tenham o mesmo custo que uma máquina executando por mil horas.

Além disso, ao comparar computação em nuvem, sob o modelo de implantação de nuvem pública, com *datacenters* convencionais, o mesmo autor destacou as vantagens daquele em relação a este, conforme apresentado na Tabela 2.1.

Tabela 2.1: Comparação entre Nuvens Públicas e *Datacenters* convencionais.

.

Vantagens	**Nuvem Pública**	***Datacenter***
Aparenta ter recursos computacionais sob demanda infinitos	Sim	Não
Elimina compromisso financeiro antecipado	Sim	Não
Capacidade de pagamento pelo uso dos recursos computacionais de acordo com a necessidade	Sim	Não
Economia de escala devido a *datacenters* muito grandes	Sim	Geralmente não
Maior utilização ao multiplexar a carga de trabalho de diferentes organizações	Sim	Depende do tamanho da organização
Operação simplificada e aumento da utilização através de virtualização de recursos	Sim	Não

Na mesma linha, Zhang *et al.* [78] também elencaram como vantagens da computação em nuvem a ausência da antecipação de investimentos com infraestrutura, a redução do custo operacional devido a economia de escala e a facilidade de acesso (operação).

Dessa forma, foram adicionadas a essa lista: uma alta escalabilidade e a redução dos riscos de negócio (terceirização para um provedor que tem maior conhecimento técnico em hardware) e das despesas de manutenção (redução da equipe necessária para manter equipamentos).

2.2 Nuvens Federadas

Apesar das vantagens da computação em nuvem, Armbrust *et al.* [5] observaram algumas preocupações e obstáculos no que tange a sua efetiva utilização. As principais observações foram a continuidade do negócio, a disponibilidade do serviço e o aprisionamento dos dados no provedor (*data lock-in*). Contudo, como oportunidades para contornar tais barreiras, os mesmos autores sugeriram a utilização de múltiplos provedores de nuvem e o uso de APIs (*Application Program Interface*) padronizadas entre eles.

Neste escopo, surgiram algumas terminologias para arquiteturas envolvendo mais de um provedor de nuvem: nuvens federadas [75], multi-nuvem [24] e inter-nuvem [32] [37]. Assim, esta seção traz definições e características desses termos, unificando-os. Além disso, são apresentados cenários e aplicações no qual o uso de mais de um provedor de nuvem se faz desejado.

2.2.1 Definições e Características

Para Kurze *et al.* [46], nuvens federadas compreendem serviços de diferentes provedores agregados em um único conjunto de recursos que suportam três características básicas de interoperabilidade – migração de recursos, redundância de recursos e a combinação de recursos. Por outro lado, Villegas *et al.* [75] usaram o termo federação como uma rede colaborativa de nuvens que facilita o compartilhamento de recursos com diferentes camadas ou modelos de serviço, a fim de alcançar maior escalabilidade dinâmica e utilização eficaz de recurso durante o provisionamento em um pico de demanda.

Além disso, outros autores tem mencionado termos diferentes para conceitos que se assemelham, em alguns aspectos, à ideia de federação. Um desses termos é conhecido por inter-nuvem, o qual é definido como um modelo de nuvem que, com a finalidade de garantir a qualidade do serviço, tais como o desempenho e a disponibilidade de cada serviço, permite realocação sob demanda de recursos e a transferência de carga de trabalho através de uma interoperabilidade de sistemas de nuvem de diferentes provedores [32]. Uma simplificação desse termo é citado por Grozev e Buyya [37] como "a nuvem das nuvens". E de forma genérica, é observada a utilização do termo multi-nuvem para descrever a capacidade de realizar atividades coordenadas em mais de um provedor de nuvem [24].

Neste trabalho o termo federação é usado para unificar os conceitos de nuvens federadas, federação de nuvens, inter-nuvem e multi-nuvem sob a seguinte definição: federação é uma rede colaborativa de nuvens capaz de fornecer um conjunto unificado de recursos de diferentes provedores acessados de maneira padronizada.

2.2.2 Vantagens da Federação de Nuvens

Mesmo com a aparente ideia de recursos ilimitados, os provedores de nuvem impõem limites quanto a utilização de seus recursos. É o caso da Amazon AWS [4], da Microsoft Azure [54] e do Google App Engines [33]. Neste cenário, a federação de nuvens pode solucionar esse problema de limitação ao unificar os recursos entregues através dos diferentes provedores de nuvem. Assim, as principais vantagens de uma federação de nuvens são [36]:

- Publicação de serviços globalizados – Aplicativos acessados mundialmente podem servir seu conteúdo em regiões mais próximas do solicitante;
- Disponibilidade, redundância e tolerância a falhas – A escolha pode considerar o grau de confiança em relação à tolerância a falhas do provedor;
- Redução de custos – Pode-se escolher os provedores mais econômicos;
- Qualidade dos serviços – Pode-se escolher os provedores que apresentam critérios específicos de qualidade de serviço desejado (por exemplo latência e redundância de dados);
- Independência quanto ao fornecedor – Livre escolha de provedores e possíveis trocas e alterações.

Nesse sentido, ao reunir recursos computacionais de diferentes provedores, uma federação de nuvens além de oferecer as vantagens já citadas, é capaz de solucionar as três principais preocupações levantadas por Armbrust *et al.* [5] quanto a utilização de um único provedor de nuvem, os quais são: a continuidade do negócio, a disponibilidade do serviço e o aprisionamento dos dados no provedor (*data lock-in*).

2.3 Bioinformática

O termo Bioinformática é definido pelo *Oxford Dictionary of English* como a ciência de coletar e analisar dados biológicos complexos, como códigos genéticos. Luscombe *et al.* [51] mencionam que o termo Bioinformática é muitas vezes utilizado como a aplicação de

técnicas computacionais para compreender e organizar a informação associada às macromoléculas biológicas. Os mesmos autores reconhecem os avanços tecnológicos, destacando que as melhorias no que tange aos processadores (CPU), ao armazenamento em disco e a Internet permitiram cálculos mais rápidos, o melhor armazenamento da informação e revolucionaram os métodos para acessar e trocar dados. Além disso, destacam três objetivos da Bioinformática [51]:

- Organizar os dados permitindo que os pesquisadores acessem as informações existentes e apresentem novas entradas à medida que são produzidas;
- Desenvolver ferramentas e recursos que ajudem na análise de dados (por exemplo, na análise de uma determinada proteína que foi sequenciada);
- Utilizar as ferramentas desenvolvidas para analisar os dados e interpretar os resultados de uma maneira biologicamente significativa.

Para De Paula [57], a Bioinformática surgiu da necessidade de utilização de ferramentas computacionais para as soluções de problemas da Biologia, tanto de processamento quanto de armazenamento e recuperação de dados. Na próxima seção, e ainda sob a ótica de Bioinformática, são apresentados conceitos e utilização de métodos e ferramentas de Bioinformática na composição de *workflows*.

2.3.1 *Workflows*

Raicu *et al.* [58] mencionam que o termo *workflow* foi inicialmente utilizado para indicar o sequenciamento de tarefas em processos de negócio e que o mesmo termo muitas vezes é usado para representar qualquer computação no qual o controle e os dados são repassados de uma tarefa para outra.

Contribuindo para a definição de *workflows*, Bharathi *et al.* [7] enumeram cinco estruturas básicas presentes em um *workflow*. Essas estruturas podem ser visualizadas na Figura 2.2, e são descritas a seguir:

- Processo – é a estrutura mais simples, ocorre quando um *job* processa dados de entrada produzindo uma saída;
- *Pipeline* – formado pelo sequenciamento de processos, a saída de um processo é a entrada do processo subsequente;
- Distribuição – essa estrutura pode trabalhar de duas maneiras: produzindo dados que serão processados por múltiplos *jobs*, ou dividindo dados de entrada muito grandes que poderão ser utilizados em outras estruturas;

- Agregação – combina as saídas de diferentes *jobs* agregando-as em uma única saída;
- Redistribuição – representa um ponto de sincronização da perspectiva do processamento de dados.

Além disso, no cenário de Bioinformática, De Paula [57] atribui os projetos de Bioinformática ao processamento de dados biológicos relacionados aos processos de sequenciamento e explica que esses processos consistem na tarefa de descobrir, para um determinado organismo, qual é a sequência de bases que forma cada fragmento de DNA[1] ou RNA[2] que está sendo investigado. A partir desses fragmentos, diversas etapas de processamento computacionais podem ser executadas formando-se um *workflow*.

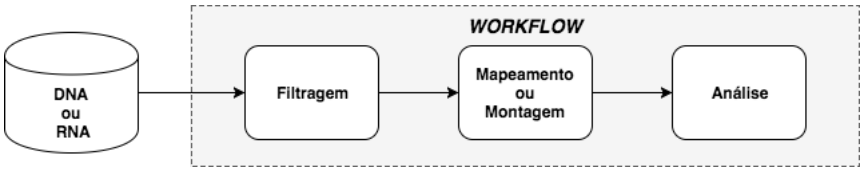

Figura 2.1: Exemplo de *Workflow* Apresentado por De Paula [57].

Assim, os *workflows* de Bioinformática extraem, a partir da transformação da entrada de fragmentos de DNA ou RNA, informações como funções biológicas e a localização dentro da célula. De Paula [57] exemplifica, como mostrado na Figura 2.1, um *workflow* dividido em três etapas: filtragem, mapeamento ou montagem e análise. O autor explica que a partir do sequenciamento de porções de DNA ou RNA (também conhecidas como *reads*) é realizada a etapa de filtragem – que vai incluir no restante do *workflow* apenas as *reads* com boa qualidade. Os arquivos da etapa de filtragem consomem, de maneira geral, grande quantidade de espaço em disco (da grandeza de *Gigabytes*). O mapeamento tenta localizar, em um genoma[3] de referência as *reads* filtradas. Se não há genoma de referência, a etapa de montagem tenta agrupar as *reads* que possivelmente vieram de uma mesma porção do genoma (ou transcriptoma), formando sequências maiores. Na etapa de análise, diferentes resultados podem ser encontrados, por exemplo, a busca pela expressão diferencial genética entre humanos suscetíveis e não suscetíveis à infecção de um determinado vírus.

[1]DNA - *Deoxyribonucleic acid* ou ácido desoxirribonucléico.
[2]RNA - *Ribonucleic acid* ou ácido ribonucléico
[3]Genoma - é o conjunto completo de DNA de um organismo, incluindo todos os seus genes. Cada genoma contém todas as informações necessárias para construir e manter esse organismo [55].

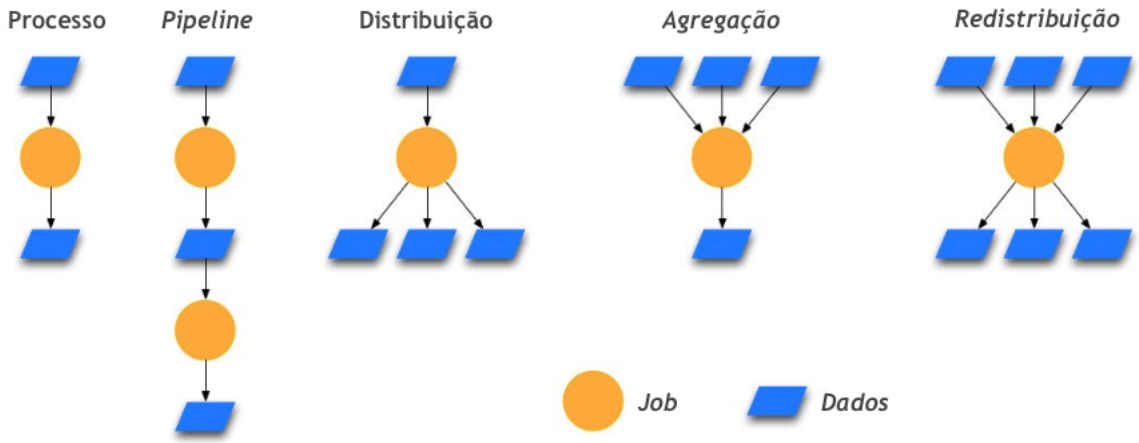

Figura 2.2: Estruturas Básicas de um *Workflow* [7].

2.4 Considerações Finais

Como visto no decorrer deste capítulo, recursos entregues através de provedores de nuvem apresentam vantagens e desvantagens. Armbrust *et al.* [5], sugeriram alguns métodos para superar as desvantagens quanto a utilização de provedores de nuvem. Além disso, o termo federação de nuvens foi cunhado como uma estratégia de agrupamento de provedores com o intuito de diminuir a dependência caracterizada pela utilização de um único provedor. Ademais, foram definidos os termos Bioinformática e *workflows*. Assim, no próximo capítulo será apresentada a arquitetura do BioNimbuz [49], que é uma plataforma de federação de nuvens computacionais híbrida, capaz de executar aplicações e *workflows* de Bioinformática.

Capítulo 3

Plataforma de Federação BioNimbuZ

Neste capítulo será apresentada a plataforma de federação de nuvem do BioNimbuZ [49], seus principais componentes e serviços. Além disso, serão expostas as principais limitações dessa plataforma.

3.1 Visão Geral

O BioNimbuZ é uma plataforma para federação de nuvens híbridas, que foi proposta originalmente por Saldanha [62], e que tem sido aprimorada constantemente em outros trabalhos [16] [43] [49] [59] [61] [74]. Ele foi desenvolvido para suprir a demanda de plataformas de nuvens federadas, visto que a utilização de nuvens de forma isolada já não atende, em muito casos, às necessidades de processamento, de armazenamento, entre outros, na execução das aplicações de Bioinformática.

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

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

Originalmente, toda a comunicação existente no BioNimbuZ era realizada por meio de uma rede *Peer-to-Peer* (P2P) [65]. Porém, para alcançar os objetivos desejados de escalabilidade e de flexibilidade, percebeu-se a necessidade de alterar a forma de comunicação entre os componentes da arquitetura do BioNimbuZ, pois a utilização de uma rede

de comunicação *Peer-to-Peer* (P2P) não estava mais suprindo as necessidades nestes dois quesitos. É importante ressaltar que os outros objetivos propostos por Saldanha [62], tais como obter uma arquitetura tolerante a falhas, com grande poder de processamento e armazenamento, e que suportasse diversos provedores de infraestrutura, foram mantidos e melhorados.

Assim sendo, no trabalho [49] foi proposta a utilização de Chamada de Procedimento Remoto (RPC) [47] para realizar a comunicação de forma transparente, pois ela permite que haja chamadas de procedimentos, que estão localizados em outras máquinas, sem que o usuário perceba [64]. E para auxiliar na organização e na coordenação do BioNimbuZ, foi utilizado um serviço voltado à sistemas distribuídos chamado Apache ZooKeeper [67]. A seguir, será explicado, de forma detalhada, o funcionamento do Zookeeper e do Avro[68], que é o sistema de RPC, também da Fundação Apache.

3.1.1 Apache Avro

O Apache Avro [68] é um sistema de RPC e de serialização de dados desenvolvido pela Fundação Apache [69]. Algumas vantagens deste sistema são o fato de ser livre, a utilização de mais de um protocolo de transporte de dados em rede, e o suporte a mais de um formato de serialização de dados. Quanto ao formato dos dados, existe o suporte aos dados binários e aos dados no formato JSON [17]. Em relação aos protocolos pode-se utilizar tanto o protocolo HTTP [25] como um protocolo próprio do Avro.

O Avro foi criado para ser utilizado com um grande volume de dados, e possui algumas características [68] definidas pela própria Fundação Apache, como uma rica estrutura de dados com tipos primitivos, um formato de dados compacto, rápido e binário e a integração de forma simples com diversas linguagens de programação.

O Avro foi escolhido como *middleware* de chamada remota de procedimento para o BioNimbuZ, pois ele é livre, flexível e possui integração com várias linguagens.

3.1.2 Apache Zookeeper

O Zookeeper [67] é um serviço de coordenação de sistemas distribuídos, criado pela Fundação Apache, para ser de fácil manuseio. Ele utiliza um modelo de dados que simula uma estrutura de diretórios, e tem como finalidade facilitar a criação e a gestão de sistemas distribuídos, que podem ser de alta complexidade e de difícil coordenação e manutenção.

No Zookeeper são utilizados espaços de nomes chamados de *znodes*, que são organizados de forma hierárquica, assim como ocorre nos sistemas de arquivos. Cada *znode* tem a capacidade de armazenar no máximo 1 Megabyte (MB) de informação, e são identificados pelo seu caminho na estrutura. Neles podem ser armazenadas informações que facilitem

o controle do sistema distribuído, tais como metadados, caminhos, dados de configuração e endereços [49].

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

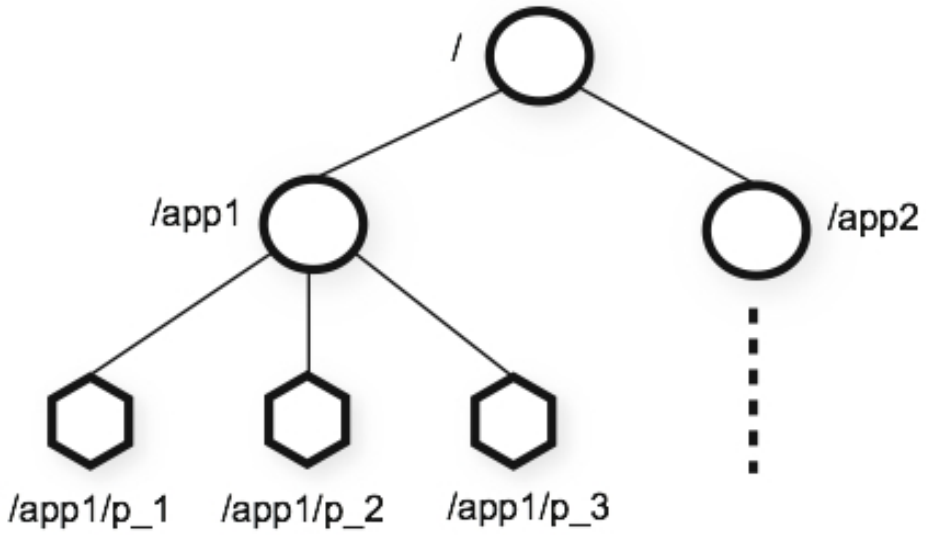

Figura 3.1: Estrutura Hierárquica dos *Znodes* [67].

O Zookeeper também suporta o conceito de *watchers*, que funcionam como observadores de mudanças, e enviam alertas sobre as alterações ocorridas em algum dos *znodes*. Este conceito é útil para sistemas distribuídos, pois permitem o monitoramento constante do sistema, deixando que *watchers* avisem quando um provedor estiver fora do ar, por exemplo, ou então quando um novo recurso estiver disponível.

3.2 Arquitetura do BioNimbuZ

O BioNimbuZ faz uso de uma arquitetura hierárquica distribuída, conforme apresentada na Figura 3.2. Ela representa a interação entre as quatro camadas principais: Aplicação, Integração, Núcleo e Infraestrutura.

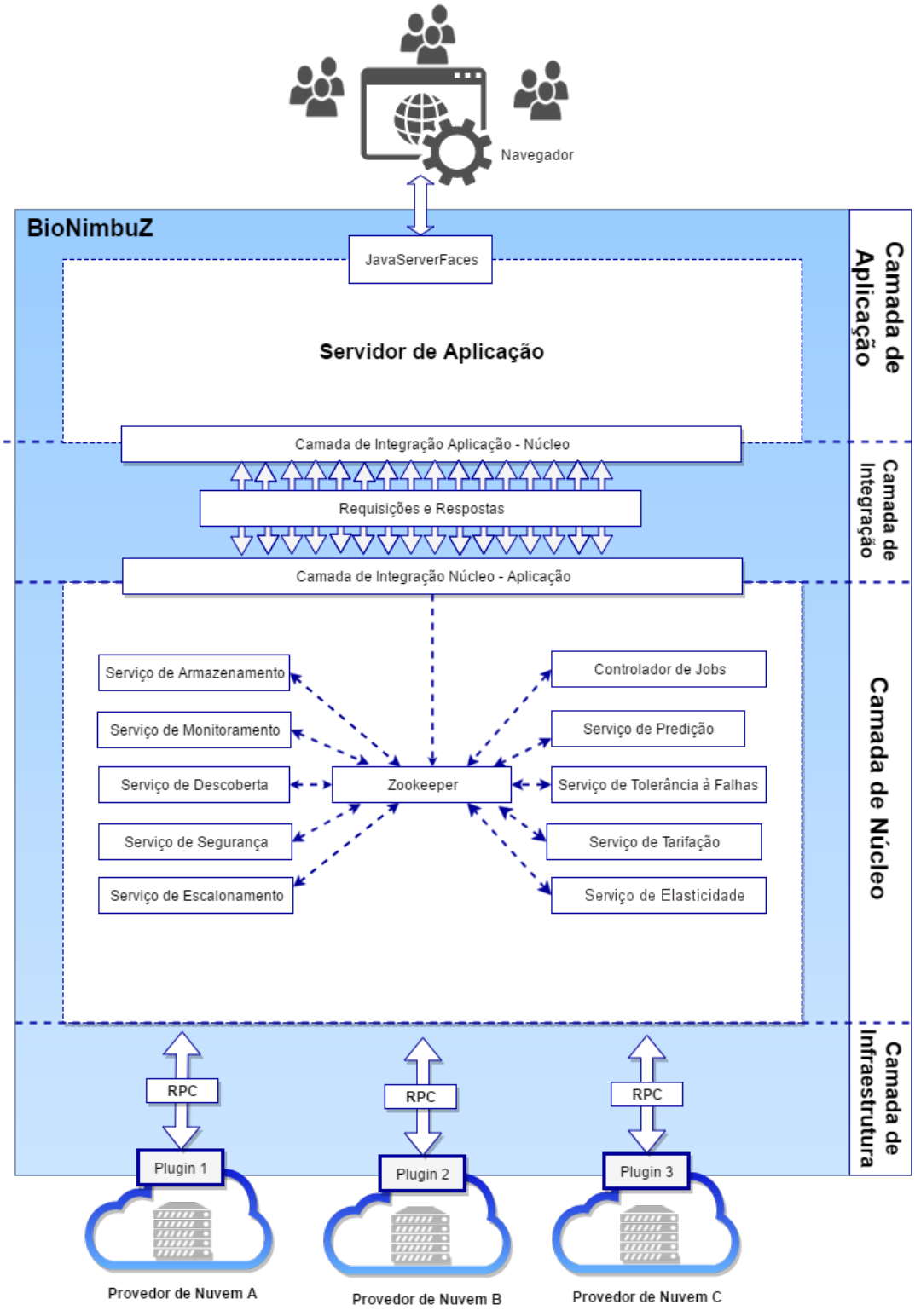

Figura 3.2: Arquitetura da Plataforma de Federação BioNimbuZ.

A Camada de Aplicação é composta pela aplicação web e implementa o sistema gerenciador de *workflows* científicos e utiliza a Camada de Integração para enviar requisições e

receber respostas da Camada de Núcleo.

A Camada de Integração é responsável por integrar as Camadas de Aplicação e Núcleo. Ela realiza a troca de mensagens entre essas camadas utilizando *webservices*, disparando requisições da Camada de Aplicação para o Núcleo e recebendo respostas do Núcleo para a Aplicação.

A Camada de Núcleo é responsável pelo armazenamento de arquivos, escalonamento de *jobs*, execução de *workflows*, descoberta dos provedores e o gerenciamento de possíveis falhas. Além do Controlador de *Jobs*, possui os seguintes serviços: Armazenamento, Monitoramento, Descoberta, Segurança, Escalonamento, Predição, Tolerância à Falhas, Tarifação e Elasticidade.

- **Controlador de *Jobs***: é quem recebe a requisição que vem da aplicação e envia para a camada de núcleo, verificando as credenciais dos usuários através do serviço de segurança, para permitir ou não a execução das tarefas. É responsável por gerenciar os vários pedidos e garantir que os resultados sejam entregues de forma correta para cada uma das requisições, e mantê-los para que possam ser consultados posteriormente;

- **Serviço de Armazenamento**: responsável pela estratégia de armazenamento dos arquivos que são utilizados ou mantidos pelas aplicações. O armazenamento deve ocorrer de forma eficiente, para que as aplicações possam utilizar os arquivos com o menor custo possível. Este custo é calculado utilizando-se algumas métricas, tais como, latência de rede, distância entre os provedores e capacidade de armazenamento. Outra estratégia adotada no armazenamento dos arquivos é a replicação em mais de um provedor, para tentar garantir que os arquivos estejam disponíveis quando as aplicações forem utilizá-los;

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

- **Serviço de Descoberta**: é o serviço que identifica e mantém informações a respeito dos provedores de nuvem que estão na federação, como capacidade de armazenamento, processamento e latência de rede, e também mantém detalhes sobre parâmetros de execução e arquivos de entrada e de saída. Para obter estas informações a respeito dos provedores, o Zookeeper é consultado, pois todos os provedores presentes na federação possuem *znodes* que armazenam seus dados. Sempre que

uma modificação é realizada, como a saída ou entrada de algum provedor, os *watchers* alertam os outros serviços, mantendo assim todos os participantes da federação atualizados;

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

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

  Para realizar a distribuição de tarefas na federação, algumas métricas são levadas em consideração, como latência, balanceamento de carga, tempo de espera e capacidade de processamento, entre outras, visando atender o que foi determinado no acordo de SLA;

- **Serviço de Predição**: auxilia o usuário na escolha de um ambiente computacional que execute seus *workflows* científicos de forma transparente, com estimativas de tempo, custo financeiro e recursos a serem utilizados. O usuário preenche um *template* com as limitações de custo financeiro e de tempo de execução, para que o serviço estime o custo, o tempo e os melhores recursos computacionais que se enquadram nas variáveis informadas;

- **Serviço de Tolerância a Falhas**: tem como objetivo garantir que todos os principais serviços estejam sempre disponíveis, e em caso de falhas, possa ser feito uma recuperação. Essa recuperação exige que todos os objetos que forem afetados pela falha voltem ao estado anterior;

- **Serviço de Tarifação**: é o serviço responsável por calcular o quanto os usuários devem pagar pela utilização dos serviços oferecidos na plataforma BioNimbuZ. Para que isto seja possível, este serviço se mantém em constante contato com o serviço de

monitoramento, para obter informações, tais como tempo de execução e quantidade de máquinas virtuais alocadas para as tarefas que foram executadas;

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

Por fim, a Camada de Infraestrutura é composta pelos plugins que traduzem as requisições do Núcleo do BioNimbuZ para os provedores de serviço utilizados, e que compõem sua federação de nuvens.

3.3 Limitações

Atualmente, o BioNimbuZ não dispõe de nenhuma funcionalidade para gerenciar a instalação e a distribuição de aplicações de Bioinformática. Dessa forma, uma imagem de máquina virtual deve ser preparada previamente com todas as aplicações instaladas para então ser disponibilizada nos diversos provedores de nuvem gerenciados pelo BioNimbuZ.

Além disso, para diferentes provedores de nuvem, novas imagens de máquina virtual específicas devem ser geradas e disponibilizadas. Ainda assim, vários provedores de nuvem não oferecem opção para o envio de imagens de máquina virtual pelo cliente, sendo necessário a instalação manual de aplicações. A própria aplicação responsável pelo BioNimbuZ não dispõe de um mecanismo de instalação bem definido, utilizando diferentes procedimentos para a implantação de todos os seus componentes.

Outro ponto no tocante a aplicações de Bioinformática é a necessidade de se executar, para efeito de comparação de um mesmo *workflow*, diferentes versões da mesma aplicação. Nesse sentido, gerenciar as dependências e possíveis conflitos de bibliotecas de software no BioNimbuZ torna-se uma tarefa bastante dispendiosa.

Quanto as questões de arquitetura, no modelo atual, nota-se um elevado nível de coesão, uma vez que cada serviço desenhado possui uma responsabilidade bem definida e atua em um escopo limitado. Contudo, os serviços e seus *plugins* são fortemente acoplados, criando dependências desnecessárias. Além disso, todos os serviços e *plugins* estão incorporados em uma única aplicação monolítica, resultando na necessidade de se execu-

tar essa aplicação em todos os nós gerenciados pelo BioNimbuZ. Essas características são indesejáveis uma vez que:

- Nem todos os serviços são necessários em todos os recursos computacionais gerenciados pelo BioNimbuZ;
- Alterar ou adicionar novas funcionalidades a um serviço ou *plugin*, ou incluir novos serviços e/ou *plugins*, requer que a aplicação seja atualizada em todos os nós;
- Os serviços possuem necessidades diferentes quanto a escalabilidade, a segurança e a tolerância a falhas. Tratar esses detalhes individualmente pode ser mais simples do que atendê-los como um todo.

Assim sendo, diante das limitações expostas nesta seção, este trabalho propõe uma arquitetura, usando a tecnologia de *containers* na distribuição e na instalação de aplicações, para a implementação de um ambiente de federação de nuvens capaz de executar *workflows* de Bioinformática.

3.4 Considerações Finais

Neste capítulo foi apresentada a plataforma de federação BioNimbuZ. Além disso, foram enumerados os principais serviços dessa plataforma e suas limitações. No próximo capítulo será detalhada a tecnologia de *containers*, para que no Capítulo 5 seja apresentado a proposta de arquitetura deste trabalho.

Capítulo 4

Containers Linux

A tecnologia envolvida em *containers*, mais especificamente para sistemas operacionais Linux, tem recebido constante melhorias. Neste capítulo são abordados a definição, a história, a evolução e o cenário atual de *containers* Linux. Além disso, é apresentado um comparativo entre as tecnologias de virtualização e de *containers*, bem como o Docker [21] e as principais plataformas de orquestração de *containers*.

4.1 Definição

A definição do termo *container* está bastante vinculada a história de como essa tecnologia evoluiu nos últimos anos. Di Tommaso *et al.* [18] mencionam que *containers* permitem que aplicações executem em um empacotamento isolado e auto-contido, capaz de ser eficientemente distribuído e executado de maneira portável através de uma ampla quantidade de plataformas computacionais. Para a empresa RedHat [60], *containers* mantém aplicações e seus componentes juntos, combinando um isolamento leve de aplicações com um método de implantação baseado em imagens. Contudo, atualmente o termo *container* está fortemente associado ao produto Docker [21], da empresa de mesmo nome.

No cenário científico, já existem trabalhos que comparam a execução de aplicações em máquinas virtuais com *containers* Docker [23] [42], e também o impacto de *containers* na execuções de *workflows* [18]. Dessa forma, nas próximas seções deste capítulo, são apresentadas as principais diferenças entre máquinas virtuais e *containers*, a história e a evolução da tecnologia, bem como os detalhes da plataforma Docker, e como essa tecnologia influenciou a criação de outras plataformas (conhecidas como plataformas de orquestração).

4.2 História, Evolução e Atualidade

Em sistemas operacionais Linux, a iniciativa de se implementar uma tecnologia de *containers* surgiu com o projeto VServer, desenvolvido por Jacques Gélinas em 2001 [40]. Em sua versão inicial, o autor descreveu o projeto como sendo capaz de "executar vários servidores Linux, com propósitos diversos, em uma única máquina, com um alto nível de independência e segurança". Este foi o primeiro esforço em separar o espaço de usuário em unidades distintas, de tal forma que cada uma delas se assemelhasse a um servidor real para os processos ali contidos. Contudo, uma característica, ou limitação, do Linux VServer era a necessidade de aplicar um *patch* no *kernel* do Linux.

Em 2006, as adaptações propostas por Menage [53] ao mecanismo cpuset[1], já presente no *kernel* do Linux, elevaram a tecnologia de *containers* a um outro nível de utilização, uma vez que tais mudanças eram pouco intrusivas e – em se tratando de performance, complexidade e futuras compatibilidades – de baixo impacto. Posteriormente, o trabalho iniciado por Menage [53] resultou na implementação do *cgroups* no *kernel* do Linux. Essa funcionalidade permitiu reunir grupos de processos de forma a assegurar que cada grupo recebesse uma parte da memória, da CPU e dos recursos de E/S, e a evitar a monopolização dos recursos por qualquer um desses *containers*.

Em 2008, outra importante adição ao modelo de *containers* foi a implementação de *kernel namespaces* – permitindo que um processo tenha um conjunto de usuários, inclusive com privilégio de administrador, restrito ao contexto de seu *container*. Neste mesmo ano, foi criado o projeto *Linux Containers - LXC* [11], uma implementação de ferramentas para *cgroups* e *namespaces* que melhoraram a experiência do usuário quanto a utilização de *containers*. No quesito segurança, com o lançamento da versão 1.0 do LXC em 2014, foram implementadas extensões de segurança para SELinux[2] e Seccomp [14].

Em 2013, foi lançado o projeto de código aberto Docker [21]. Este projeto foi e, sem dúvida, continua sendo o responsável pela grande atenção que a tecnologia de *containers* tem recebido na atualidade. Por esse motivo, neste trabalho, o Docker é utilizado como plataforma de execução de *containers*.

Assim, atualmente *containers* são vistos como a tecnologia que mantém aplicações e seus componentes de tempo de execução juntos, combinando um isolamento leve de aplicações com um método de implantação baseado em imagens [60]. Além disso, *containers* introduzem autonomia para as aplicações, empacotando-as com suas bibliotecas e

[1] cpuset - um pseudo sistema de arquivos que atua como interface para o mecanismo de controle de CPU do *kernel*.

[2] SELinux - implementação da arquitetura *Mandatory Access Control - MAC* para o sistema operacional Linux - https://selinuxproject.org/

outras dependências binárias. Dessa forma, um *container* é executado da mesma maneira, independente do ambiente de execução.

Na Seção 4.4 são detalhadas as características principais do Docker e os avanços introduzidos por essa plataforma à tecnologia de *containers*. Antes disso, faz-se necessário entender as diferenças entre *containers* e virtualização, que são apresentadas na seção 4.3.

4.3 *Containers* e Virtualização

De acordo com o projeto Docker, *containers* e máquinas virtuais possuem isolamento de recursos e benefícios de alocação similares, contudo, uma diferente abordagem arquitetural permite que *containers* sejam mais portáveis e eficientes [21].

Essas diferenças podem ser melhor percebidas na Figura 4.1, que ilustra as camadas presentes entre os recursos físicos de um computador e as aplicações que ali estão sendo executadas. As máquinas virtuais (imagem da esquerda na Figura 4.1) incluem a aplicação e suas dependências – bibliotecas e arquivos binários – além do sistema operacional visitante. Os *containers* (imagem da direita na Figura 4.1), por outro lado, compartilham o *kernel* com outros *containers*, sendo executados em processos isolados no espaço de usuário (*user namespace*) em um mesmo sistema operacional.

Figura 4.1: Comparação entre Máquinas Virtuais e *Containers* [21].

Assim sendo, *containers* podem ser mais eficientes pois são executados diretamente no sistema operacional hospedeiro, eliminando a necessidade de um sistema operacional visitante e toda a memória que seria consumida por ele. Além disso, em ambientes nos quais o uso de virtualização se faz necessário, *containers* podem ser executados em máquinas virtuais. Neste cenário, uma estratégia possível é a alocação de menos máquinas

virtuais do que tradicionalmente se utiliza, sendo que cada máquina virtual pode passar a gerenciar uma quantidade maior de recursos físicos com a tecnologia de *containers* garantindo o isolamento entre processos.

Para exemplificar melhor, imagine que uma organização possui um único computador e precise executar as aplicações A, B e C nos sistemas operacionais Microsoft Windows e Linux. A Tabela 4.1 mostra a necessidade de memória para cada aplicação e para os sistemas operacionais indicados.

Tabela 4.1: Exemplo de Consumo de Memória para Sistemas Operacionais e Aplicações.
.

Descrição	**Memória**
Sistema Operacional Microsoft Windows	1 GB
Sistema Operacional Linux	1 GB
Aplicação "A"em Microsoft Windows	2 GB
Aplicação "A"em Linux	2 GB
Aplicação "B"em Microsoft Windows	6 GB
Aplicação "B"em Linux	6 GB
Aplicação "C"em Linux	4 GB

Em uma estratégia de virtualização, para cada ambiente que se deseja isolar uma nova máquina virtual deve ser criada e um sistema operacional visitante deve ser destinado a essa máquina virtual. A Tabela 4.2 exibe uma alocação de máquinas virtuais utilizadas para acomodar cada aplicação e o consumo de memória estimado para cada uma delas.

Tabela 4.2: Exemplo do Consumo de Memória de Aplicações Executando em Máquinas Virtuais.
.

Descrição	**Quant.**	**Mem. S.O.**	**Mem. Aplic.**	**Mem. Total**
Máquina Virtual Aplicação "A"Windows	2	1 GB	2 GB	6 GB
Máquina Virtual Aplicação "B"Windows	1	1 GB	6 GB	7 GB
Máquina Virtual Aplicação "A"Linux	2	1 GB	2 GB	6 GB
Máquina Virtual Aplicação "B"Linux	1	1 GB	6 GB	7 GB
Máquina Virtual Aplicação "C"Linux	1	1 GB	4 GB	5 GB
Sistema Operacional Hospedeiro	1	2 GB	-	2 GB
Total		**7 GB**		**33 GB**

Nesse mesmo ambiente, ao inserir uma abordagem baseada em *containers*, como sugerido na Tabela 4.3, as aplicações que executam em Linux podem ser agrupadas em uma mesma máquina virtual, recebendo o mesmo grau de isolamento e de segurança que se busca na técnica de virtualização. Desta maneira, ao extinguir a necessidade de máquinas virtuais para a realização de isolamento de recursos e processos, as abordagens baseadas em *containers* tornam-se mais eficientes, considerando a utilização de memória total.

Tabela 4.3: Exemplo do Consumo de Memória de Aplicações Executando em *Containers*.

Descrição		**Quant.**	**Mem. S.O.**	**Mem. Aplic.**	**Mem. Total**
Máquina Virtual Aplicação "A"Windows		2	1 GB	2 GB	6 GB
Máquina Virtual Aplicação "B"Windows		1	1 GB	6 GB	7 GB
Máquina Virtual Linux		1	1 GB	-	1 GB
Containers	Aplicação "A"	2	-	2 GB	4 GB
	Aplicação "B"	1	-	6 GB	6 GB
	Aplicação "C"	1	-	4 GB	4 GB
Sistema Operacional Hospedeiro		1	2 GB	-	2 GB
Total			**5 GB**		**30 GB**

As demais características da abordagem baseada em *containers*, especialmente as introduzidas pelo Docker, serão tratadas na seção subsequente.

4.4 Plataforma Docker

Como mencionado na Seção 4.2, o Docker foi escolhido para ser a plataforma de execução de *containers* neste trabalho. Existem outras implementações capazes de executar *containers*, tais como o Rkt[3], o LXC[4] e o OpenVZ[5], contudo, nenhuma delas entrega todos os recursos oferecidos pelo Docker.

Inicialmente, o Docker utilizava o LXC [11] como mecanismo padrão de execução de *containers*. Em 2014, a partir da versão 0.9, uma implementação própria – *libcontainer* – foi introduzida como padrão. Em 2015, com o aumento do interesse pela tecnologia de *containers* por grandes empresas, tais como Google, Cisco Systems, RedHat e IBM, foi criado o *Open Container Initiative - OCI* [50], um consórcio de empresas que tem por foco a criação de especificações para o formato de imagens de *containers* e seus mecanismos de execução. Mais uma vez o projeto Docker teve papel fundamental nesta iniciativa, ao

[3]Rkt - https://coreos.com/rkt/docs/latest/
[4]LXC - https://linuxcontainers.org/
[5]OpenVZ - https://github.com/OpenVZ

doar o código fonte do *libcontainer* e contribuir com o primeiro rascunho da especificação de *containers* OCI [50].

Assim, nota-se que o Docker reinventou a tecnologia de *containers* ao incluir os seguintes conceitos e funcionalidades [21]:

- Portabilidade – Definiu um formato para imagens de *containers* capaz de ser executado sem alterações em máquinas com a instalação do Docker. Anteriormente, o LXC oferecia apenas isolamento de processos – característica necessária a portabilidade, mas não suficiente por si só. Até então os *containers* precisavam ser configurados (rede, armazenamento, *logs*) de acordo com as especifidades de cada ambiente de execução. Neste aspecto, o Docker criou uma abstração às configurações de máquina, garantindo que o mesmo *container* seja executado sem mudanças;
- Foco na Aplicação – Otimizou a implantação de aplicações em vez de tratar *containers* como máquinas virtuais leves;
- Automatização - Ofereceu ferramentas automatizadas de construção, de empacotamento e de dependências de *containers*;
- Versionamento - Possibilitou o controle, o gerenciamento e a comparação das versões de um *container*. Semelhante a ferramenta de controle de versões *GIT*[6], o Docker implementou o envio/recebimento incremental para novas versões de um mesmo *container*;
- Reuso de Componentes - Qualquer *container* pode ser usado como base para a criação de um outro. Dessa forma, pode-se, por exemplo, preparar um *container* com ferramentas de desenvolvimento para ser utilizado como base para outros mais especializados;
- Compartilhamento - Implementou um repositório público de *containers* no qual indivíduos e empresas podem distribuir seus *containers* uns aos outros;
- Integração via API - Distribuiu todas as funcionalidades de sua arquitetura via interface de programação de aplicações (do inglês *Application Programming Interface - API*), possibilitando a extensão de suas funcionalidades por outros desenvolvedores.

A partir da visão de *containers* como arcabouço de execução de aplicações introduzido pelo Docker, foram desenvolvidas diferentes plataformas de orquestração de *containers*. Dentre essas plataformas, que tem por foco comum o gerenciamento de grandes quantidades de *containers* distribuídos através de diversos recursos computacionais visualizados

[6]GIT - https://git-scm.com/

como um único ambiente, destacam-se o Google Kubernetes [34], e o Docker Swarm [20]. Outras plataformas, não focadas exclusivamente em *containers* como o Apache Mesos [70] e o Nomad [39], são capazes de orquestrar a execução de *containers* além de outras aplicações. Na próxima seção essas quatro plataformas serão exploradas.

4.5 Plataformas de Orquestração de *Containers*

De acordo com a Gartner Inc. [6], empresa do segmento de pesquisa e consultoria em Tecnologia da Informação, o ritmo acelerado em que a tecnologia de *containers* está sendo adotada nos *datacenters* em breve afetará a estratégia de entrega de soluções de muitas organizações de infraestrutura e operações. A mesma consultoria cita o Docker como principal plataforma de execução de *container*, expõe a necessidade de orquestrar múltiplos *containers* através de servidores e enumera três produtos capazes de preencher esses requisitos: Google Kubernetes [34], Apache Mesos [70] e Docker Swarm [20] [77].

Nesse sentido, uma plataforma de orquestração de *containers* gerencia e coordena, de maneira simples e eficiente, uma rede de recursos computacionais independentes, que executam uma plataforma de execução de *container*. Em linhas gerais, uma plataforma de orquestração preocupa-se com dois ou mais recursos computacionais, enquanto uma plataforma de execução está focada em um único recurso computacional.

Assim sendo, nas próximas seções, além das plataformas já listadas (Google Kubernetes, Apache Mesos, Docker Swarm) será descrita a plataforma Nomad [39]. De maneira sucinta, cabe aqui mencionar a plataforma CoreOS Fleet [15], que foi lançada em 2014 mas deixará de ser suportada em fevereiro de 2018. Essa plataforma era bastante utilizada em conjunto com o sistema operacional CoreOS Container Linux [15], principal sistema operacional especializado em *containers* da atualidade.

4.5.1 Docker Swarm

O Docker Swarm [20] é uma plataforma de orquestração de *containers* capaz de gerenciar e coordenar uma rede de recursos computacionais independentes, onde cada um desses recursos executa o Docker como plataforma de execução de *containers*. Na Figura 4.2 pode ser vista a plataforma de orquestração Docker Swarm coordenando três instâncias do Docker (plataforma de execução) em recursos independentes.

Quando executada em modo "*swarm*", a plataforma Docker Swarm torna-se um *cluster* nativo para o Docker. Nesse modo, todos os componentes necessários para a inicialização, o monitoramento e a orquestração de um *cluster* Docker já encontram-se embutidos no próprio Docker. Além disso, o Docker Swarm mantém a compatibilidade com todos as demais ferramentas e API's disponibilizadas pela plataforma Docker. Uma dessas

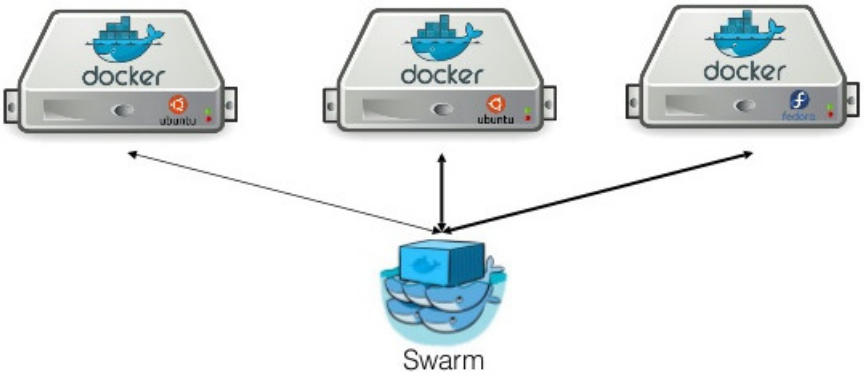

Figura 4.2: Plataforma de Orquestração de *Containers* Docker Swarm.

ferramentas é o Docker Compose [22], que é capaz de criar um conjunto de *containers*, volumes de dados, redes definidas por software e segredos (definição dada a proteção aplicada a informações sensíveis, como senhas e chaves privadas) por meio de um arquivo de configuração. Outras características do Docker Swarm são:

- Gerenciamento integrado ao Docker – A partir de uma instalação Docker é possível iniciar um *cluster* Docker Swarm, não sendo necessário nenhum *software* adicional de orquestração para gerenciar o *cluster*;
- Elasticidade horizontal – As aplicações em execução podem ter suas instâncias aumentadas ou diminuídas dinamicamente;
- Auto-recuperação – Na ocorrência de falhas de uma máquina, as aplicações podem ser direcionadas para outra. O mesmo comportamento ocorre quando um *container* falha;
- Descobrimento de serviços – Para cada aplicação em execução o Swarm é capaz de adicionar uma entrada em um serviço de DNS[7] interno. Dessa forma, as aplicações podem referenciar os *containers* uma das outras por nome ao invés de IP's;
- Balanceamento de carga – Ao utilizar o descobrimento de serviços e a elasticidade horizontal, o Docker Swarm é capaz de balancear a carga entre instâncias de uma mesma aplicação;
- Segurança – Por padrão, cada nó de um *cluster* Swarm utiliza autenticação mútua baseada em TLS - *Transport Layer Security* [63].

Apesar da quantidade de funcionalidades e da flexibilidade do Docker Swarm, algumas funcionalidades não estão presentes nessa plataforma, dentre elas:

[7]DNS - Domain Name System

- Federação – O Docker Swarm não é capaz de coordenar um *cluster* formado por recursos disponibilizados em diferentes regiões geográficas. A baixa latência de rede é algo fundamental na construção e funcionamento de um *cluster* Swarm;
- Execução de *Jobs* – Inicialmente, o Docker Swarm é utilizado para execução de serviços e não *jobs*. A diferença de um serviço para um *job* é que para um serviço, o desejável é que se esteja em execução permanente, já um *job* deve possuir um início e um fim.;
- Execução de *workflows* – O Docker Swarm não tem nenhum mecanismo para a execução de *jobs* em sequência.

4.5.2 Google Kubernetes

O Google Kubernetes [34] é uma plataforma de código aberto para a implantação, o redimensionamento (*scaling*) e o gerenciamento automatizado de aplicações em *containers*. Assim como o Docker Swarm, o Google Kubernetes apresenta as seguintes funcionalidades:

- Escalonamento - distribui automaticamente *containers* de acordo com filtros e requisitos de utilização de recursos sem sacrificar a disponibilidade. Combina cargas de trabalho críticas e de melhor esforço para elevar a economia de recursos;
- Auto-recuperação - reinicia *containers* na ocorrência de falhas, substitui e reescalona *containers* em caso de falha na máquina e finaliza *containers* que não respondem as verificações de disponibilidade definidas pelos usuários;
- Elasticidade horizontal - aumenta e diminui horizontalmente instâncias de uma aplicação (manualmente ou automaticamente baseado na utilização de CPU);
- Descobrimento de serviços e balanceamento de carga - sem a necessidade de modificar as aplicações. O Kubernetes atribui um endereço de IP único por *container* e também uma única entrada DNS para um conjunto de *containers*, balanceando a carga entre eles;
- Reversões *rollbacks* e implantações *rollouts* automáticas - O Kubernetes progressivamente implanta mudanças na aplicação, ou suas configurações, enquanto monitora a disponibilidade garantindo a existência de instâncias no decorrer do processo. Em caso de falha, o Kubernetes é capaz de reverter as mudanças;
- Gerenciamento de segredos e configurações de aplicações;

- Gerenciamento de serviços de armazenamento - Disponibiliza sistemas de armazenamento local, em provedores de nuvem públicos (Google Cloud Plataform[8], Amazon Web Services [2]) e em serviços de rede (NFS, iSCSI, Gluster[9], Ceph[10], Cinder[11] ou Flocker[12]).

O Kubernetes é capaz de coordenar *clusters* distribuídos em diferentes *datacenters*. Contudo, para isso, necessita que serviços externos de DNS e balanceamento de carga sejam previamente configurados. Cabe observar que durante a elaboração deste trabalho, foi verificado, em janeiro de 2017, que a documentação existente para a criação de uma federação baseada em Kubernetes citava apenas procedimentos que dependiam do provedor de nuvem da Google para serem realizados. Ao revisitar a documentação, verificou-se uma atualização, que passou a constar procedimentos genéricos para execução em outras nuvens ou em um ambiente local. Contudo, apesar de suportar a execução de *jobs* em um *cluster*, o Kubernetes não suporta a execução de *jobs* e *workflows* em sua implementação de federação.

4.5.3 Apache Mesos

O Apache Mesos [70] é uma plataforma capaz de abstrair CPU, memória, armazenamento e outros recursos computacionais em máquinas físicas ou virtuais, permitindo que sistemas distribuídos elásticos e tolerante a falhas possam ser facilmente construídos, e efetivamente executados. Nesse sentido, é importante ajuizar que o Apache Mesos não é exclusivo para orquestração de *containers*, tornando-o flexível e versátil também para ser utilizado por outras tecnologias alheias a *containers*. Dentre diversos sistemas que podem ser executados sob a gerência do Mesos, destacam-se o Hadoop[13], o Marathon[14], o Spark[15], o Storm[16], o Chronos[17], o Cassandra[18], e o ElasticSearch[19]. Outras características principais do Apache Mesos são:

- Escalabilidade linear – capaz de executar em mais de dez mil nós;
- Suporte a *containers* - executa imagens baseadas em Docker e AppC[20];

[8]Google Cloud Platform - https://cloud.google.com/
[9]Gluster - https://www.gluster.org/
[10]Ceph - http://ceph.com/
[11]Openstack Cinder - https://wiki.openstack.org/wiki/Cinder
[12]Flocker - https://clusterhq.com/flocker/introduction/
[13]Apache Hadoop - http://hadoop.apache.org/
[14]Marathon - https://mesosphere.github.io/marathon/
[15]Apache Spark - http://spark.apache.org/
[16]Apache Storm - http://storm.apache.org/
[17]Chronos - https://mesos.github.io/chronos/
[18]Apache Cassandra - http://cassandra.apache.org/
[19]ElasticSearch - https://www.elastic.co/products/elasticsearch
[20]AppC – especificação para imagens de *containers* desenvolvida pela *OCI – Open Container Initiative*

- Alta disponibilidade – utilizando agentes e a replicação de *masters* do Apache Zookeeper [67]. Além de atualizações não disruptivas;
- Suporte a múltiplos sistemas operacionais – Linux, OSX e Windows. Independência quanto ao provedor de nuvem.

Além disso, através da aplicação Chronos (que executa sob o Apache Mesos) é possível realizar a execução de *jobs* e *workflows*. Contudo, o Apache Mesos não possui suporte a federação. Até a conclusão deste trabalho apenas o banco de dados distribuído Cassandra fornecia documentação para ser executado em múltiplos *datacenters*, valendo-se da plataforma Apache Mesos.

4.5.4 Nomad

O Nomad é uma plataforma para gerenciamento de *clusters* de máquinas e execução de aplicações. Ele abstrai o conjunto de máquinas de um *cluster* e o local de execução de aplicações. Dessa forma, permite que usuários declarem apenas o que se deseja executar, deixando a decisão de como e onde executar a cargo da plataforma Nomad [39]. As principais características dessa plataforma são:

- Suporte ao Docker – *Jobs* submetidos podem utilizar o *driver* Docker para executar suas cargas de trabalho. O Nomad aplica as restrições especificadas pelo usuário, assegurando que o *job* seja executado apenas na região, *datacenter* e máquina corretos;
- Operação simplificada – O Nomad é distribuído como um único arquivo binário que é utilizado tanto para máquinas com o papel de cliente como no papel de servidor. Não requer serviços externos de coordenação e armazenamento;
- *Multi-datacenter* e multi-região – O modelo de infraestrutura do Nomad agrupa *datacenters* em regiões. As regiões podem ser federadas, permitindo o escalonamente de *jobs* de maneira global;
- Flexibilidade – Suporte a diferentes *drivers* permitindo execuções em *containers*, virtualizadas ou autônomas (*standalone*). Além disso, é capaz de executar em sistemas operacionais Linux, Windows, BSD ou OSX;
- Escalável – O Nomad é distribuído e possui alta disponibilidade. Permite que todos os nós (que executam sob o papel de servidores) participem das decisões de escalonamento, o que aumenta a taxa de transferência (*throughput*) e diminui a latência.

Apesar de suportar federação de *datacenters* em regiões e a execução de *jobs*, o Nomad não implementa nenhum mecanismo para a execução de *workflows*.

4.5.5 Comparação das Plataformas de Orquestração

A Tabela 4.4 enumera as principais características necessárias para a execução de *workflows* em nuvens federadas e o suporte dado por cada plataforma de orquestração citada nas seções anteriores.

Tabela 4.4: Comparação das Plataformas de Orquestração.

Característica / Plataforma	**Docker Swarm** [20]	**Kubernetes** [34]	**Apache Mesos** [70]	**Nomad** [39]
Execução de *containers*	Sim	Sim	Sim	Sim
Execução de *jobs*	Não	Sim	Sim	Sim
Execução de workflows	Não	Não	Sim	Não
Federação	Não	Sim	Não	Sim

Como pode ser percebido, nenhuma plataforma de orquestração é capaz de prover os quatro requisitos necessários a execução de *workflows* de Bioinformática em nuvens federadas. Assim sendo, este trabalho propõe, no Capítulo 5, uma arquitetura capaz de integrar as plataformas de orquestração existentes em um modelo que acolha todas as características descritas na Tabela 4.4.

4.6 Considerações Finais

Neste capítulo foram apresentadas a história, a evolução e a situação atual da tecnologia de *containers*. Além disso, foram enumeradas as principais diferenças entre *containers* e máquinas virtuais, além das funcionalidades acrescentadas pelo Docker à tecnologia de *containers*. Também foram vistas e comparadas as principais plataformas de orquestração de *containers* existentes na atualidade.

No próximo capítulo será apresentada a arquitetura proposta neste trabalho, bem como os trabalhos relacionados.

Capítulo 5

BioNimbuzBox

No cenário de Bioinformática, *containers* podem auxiliar na distribuição e na reprodução de experimentos científicos [8]. Já existem iniciativas quanto a adoção de *containers* na execução de *workflows* e na criação de repositórios de imagens de aplicações de Bioinformática encapsuladas em *containers* [1] [31] [79]. Essas iniciativas podem se valer da disponibilidade e da capacidade dos provedores de nuvens para atender necessidades de infraestrutura com um baixo comprometimento financeiro [5].

Contudo, como visto no Capítulo 4, por um lado as plataformas de orquestração de *containers* não são capazes de executar *workflows* de Bioinformática em nuvens federadas, e, por outro lado as plataformas de federação de nuvens para execução de *workflows* de Bioinformática carecem de um modelo eficiente para a distribuição, e a instalação de aplicações para a reprodução de *workflows*.

Nesse sentido, foi percebida a necessidade de uma arquitetura baseada em *containers* para *workflows* de Bioinformática, que seja capaz de implementar e gerenciar um ambiente de nuvens federadas. Assim, neste capítulo é detalhada a arquitetura proposta baseada em *containers*, chamada BioNimbuzBox, que é capaz de executar *workflows* de Bioinformática em um ambiente real de nuvens federadas. Os conceitos de serviços apresentados na arquitetura proposta foram baseados nos serviços usados pela plataforma de federação BioNimbuZ [61].

5.1 Visão Geral

A partir do ano de 2015, é possível encontrar na literatura diversos trabalhos que abortam a tecnologia de *containers*, principalmente usando a plataforma Docker, em pesquisas científicas. Dentre os trabalhos pesquisados, Boettiger [8] enumera quatro desafios no tocante a reprodutibilidade de pesquisas científicas:

- "Inferno de dependências"– o autor menciona que, de acordo com pesquisadores da Universidade do Arizona [13], menos de 50% dos *softwares* podem ser compilados ou instalados com sucesso. Isso ocorre pela dificuldade de se reproduzir as dependências do ambiente no qual o software em questão foi concebido;
- Documentação imprecisa – neste ponto, Boettiger [8] indica que a documentação imprecisa é uma das maiores barreiras, principalmente para novatos, na reprodução de pesquisas;
- Deterioração do código – faz menção a como um software se comporta com a atualização de versões de bibliotecas e artefatos utilizados por ele;
- Barreiras para adoção e reutilização de soluções existentes.

Nesse sentido, o mesmo autor [8] sugere o uso da tecnologia Docker para solucionar os três primeiros problemas levantados, e relaciona cinco boas práticas no desenvolvimento de pesquisas: (1) Utilizar o Docker durante o processo de desenvolvimento; (2) Escrever arquivos Dockerfiles em vez do uso de sessões interativas com o *container*; (3) Adicionar testes e verificações no arquivo Dockerfile; (4) Utilizar e prover imagens base apropriadas; e (5) Compartilhar imagens dos *containers* e seus arquivos Dockerfile.

Quanto a performance de utilização de *containers* em *workflows*, Di Tommaso *et al.* [18] avaliaram o impacto, em relação ao tempo de execução final, de três *workflows* com cada uma de suas tarefas encapsuladas em *containers* Docker. Os autores concluíram que, em *workflows* com tarefas longas, a inclusão de *containers* Docker teve impacto mínimo no tempo final de execução do *workflow*. Contudo, para *workflows* com muitas tarefas curtas, o uso do Docker aumentou em cerca de 60% o tempo final de execução do *workflow*.

Nesse sentido, Zheng e Thain [79], com o intuito de reutilizar *containers* já inicializados ou diminuir a quantidade de *containers* criados por um *workflow*, enumeraram algumas estratégias para realizar a integração de *containers* Docker as plataformas de *workflow Makeflow* e *Work Queue*.

Por outro lado, em vez de comparar a execução de *containers* com execuções diretas de aplicações em um sistema operacional (*containers* sempre acrescentarão o tempo de inicialização – de 1 a 2 segundos, ao tempo total de execução), Joy [42] realiza uma comparação de *containers* com máquinas virtuais concluindo que *containers* são 22 vezes mais escaláveis do que máquinas virtuais.

Além disso, Felter *et al.* [23] concluem que, de forma geral, em todos os testes por eles realizados, *containers* excederam ou tiveram resultados de performance iguais aos apresentados pelo hipervisor de máquinas virtuais KVM[1].

[1]KVM – *Kernel-based Virtual Machine* – https://www.linux-kvm.org

5.2 Arquitetura do BioNimbuzBox

Para identificar a nova versão, baseada em *containers*, o BioNimbuZ passará a adotar o nome BioNimbuzBox. O sufixo "Box", tradução da palavra "caixa" para o inglês, faz menção ao encapsulamento das aplicações em *containers* – característica marcante na nova arquitetura.

Além de um novo nome, a arquitetura proposta redesenhou os principais serviços do BioNimbuZ, adequando-os ao conceito de microserviços – que são pequenos serviços autônomos que trabalham juntos [56]. Isso fez-se necessário porque nos últimos anos os sistemas distribuídos passaram de aplicativos monolíticos, com código pesado, para micro-serviços menores e autônomos [56]. Além disso, a comunicação entre os serviços passou a utilizar o padrão REST[2] sob o protocolo HTTP[3], com criptografia baseada em SSL/TLS[4] (maiores detalhes serão descritos nas seções dedicadas ao Serviço de Comunicação e ao Serviço de Segurança deste capítulo).

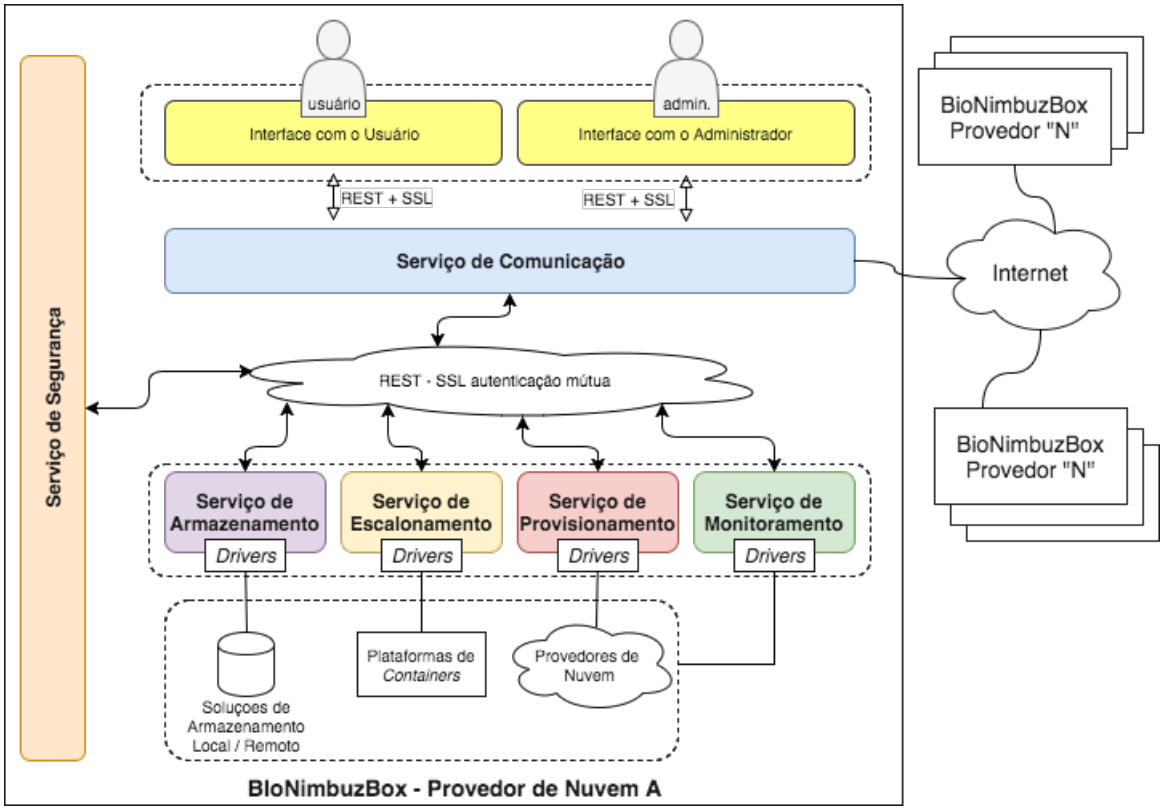

Figura 5.1: Arquitetura do BioNimbuzBox.

Dessa forma, a Figura 5.1 exibe a proposta da arquitetura do BioNimubuzBox, nela se destacam seis serviços fundamentais que serão descritos nas seções subsequentes: o

[2] *REST – Representational state transfer.*
[3] *HTTP – Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto - RFC 2616).
[4] *TLS - Transport Layer Security* (Segurança na Camada de Trasporte - RFC's 2246, 4346 e 5246).

Serviço de Comunicação, o Serviço de Segurança, o Serviço de Provisionamento, o Serviço de Escalonamento, o Serviço de Armazenamento e o Serviço de Monitoramento.

Outra mudança importante realizada no BioNimbuzBox, necessária para suportar os serviços propostos, foi criar um modelo de dados flexível, capaz de incorporar as especificidades de uma ampla quantidade de provedores de nuvem e de plataformas de orquestração de *containers*, sem que para isso seja necessário realizar alterações na arquitetura ou em seu modelo de dados. A Figura 5.2 exibe o modelo de dados proposto. Nela são evidenciadas sete entidades, as quais são:

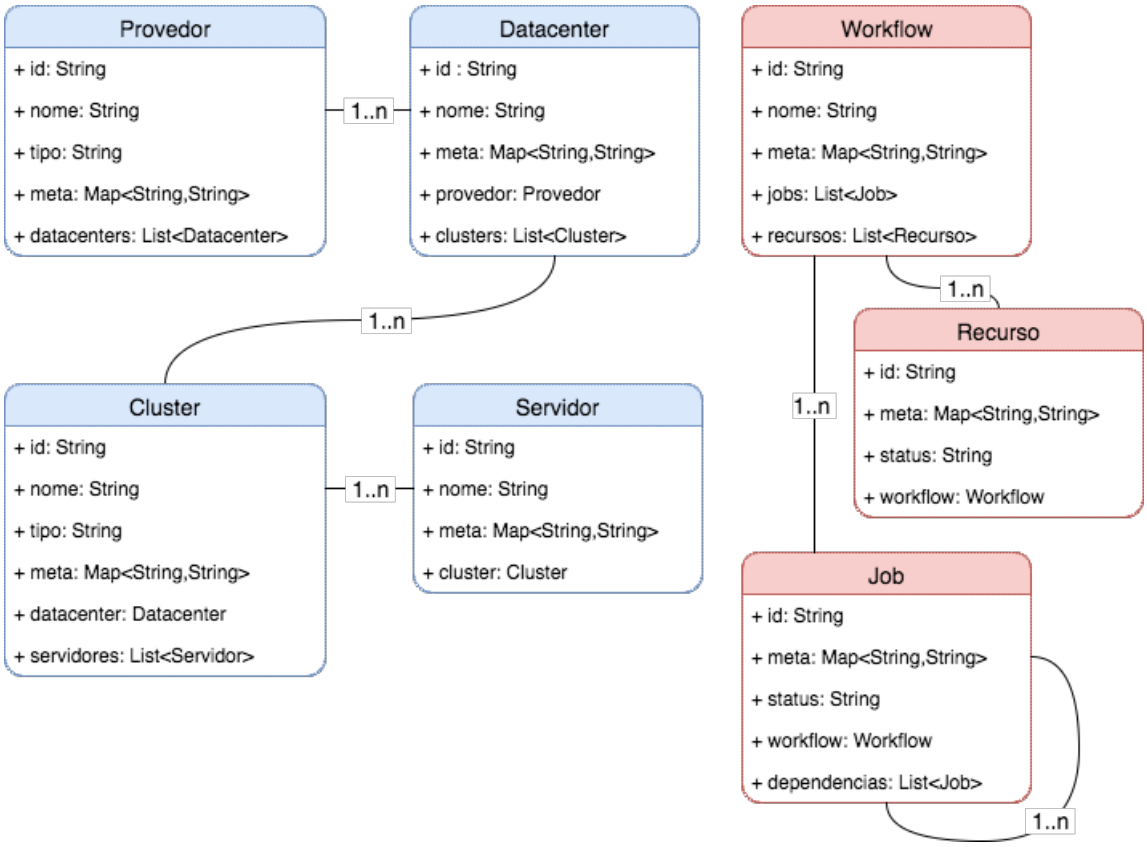

Figura 5.2: Modelo de Dados da Arquitetura BioNimbuzBox.

- **Provedor** – possui as informações de acesso ao provedor de nuvem. Além do identificador único e do nome, fornece uma coleção de metadados do tipo chave-valor para atender a necessidade de informações adicionais do provedor de nuvem. Por exemplo, na nuvem da DigitalOcean é utilizado apenas um "*token*" para acesso, já a Amazon utiliza dois valores: identificador de chave de acesso e a chave de acesso secreta. Nesse caso, ao utilizar metadados, do tipo chave valor, cada objeto poderá ter sua relação de atributos estendidas. Além disso, um Provedor pode ter vários *Datacenters*;

- ***Datacenter*** – unifica os termos Região, Zona, *Datacenter* ou outros similares que estão presentes nos provedores de nuvem. Nesse aspecto, considera-se *Datacenter* o endereço físico de origem dos recursos computacionais disponibilizados pelo provedor de nuvem. Por exemplo, a Amazon disponibiliza recursos computacionais em várias Zonas em uma mesma Região (Leste dos Estados Unidos, América do Sul, Europa). Nesse caso, cada Zona/Região é tido como um *Datacenter*. A segmentação em *datacenters* é importante, uma vez que a latência de rede dentro de um *datacenter* é diferente em relação ao acesso entre *datacenters* [12]. E essa latência pode vir a interferir na formação dos *clusters* que são gerenciados pelas plataformas de orquestração de *containers*. Assim como a entidade Provedor, um *Datacenter* possui um identificador único, um nome e uma coleção de metadados. Além disso, um *Datacenter* define um metadado para identificar sua localização, e ele pode conter vários *Clusters* e está vinculado obrigatoriamente a um Provedor;

- ***Cluster*** – é um conjunto de Servidores de um *Datacenter* gerenciados por uma plataforma de orquestração. Além do identificador único, do nome, e de uma coleção de metadados, um *Cluster* define o atributo tipo, utilizado para identificar a plataforma de orquestração responsável pelo conjunto de recursos entregues em um mesmo *Cluster*;

- **Servidor** – recurso computacional entregue sob o modelo de Infraestrutura como Serviço. Além do identificador único, do nome e dos metadados, define por meio de metadados os atributos papel e tamanho. O atributo papel é utilizado para identificar a função do Servidor no *Cluster*: gerente ou executor. O atributo tamanho dita o dimensionamento do servidor em termos de recursos computacionais (CPU, memória e disco). Um Servidor está vinculado obrigatoriamente a um *Cluster*;

- ***Workflow*** – possui um identificador único, um nome, uma coleção de metadados, uma lista de *Jobs* e uma lista de Recursos. Com auxílio da entidade *Job* e *Recurso*, um *Workflow* é capaz de reproduzir todas as estruturas listadas no Capítulo 2;

- ***Job*** – além dos atributos identificação e metadados, um *Job* está vinculado a um *Workflow* e pode apresentar uma lista de dependências. Adicionalmente, um *Job* contém um atributo *status*, que representa o estado atual do *Job* em questão (novo, aguardando, em execução ou finalizado);

- **Recurso** – a entidade Recurso é semelhante ao *Job*, contudo, não possui dependências e é executada antes de todos os *Jobs* de um *Workflow*. Pode ser utilizada para iniciar recursos de armazenamento necessários a execução do *Workflow*.

Além do modelo de dados, foram definidas duas interfaces de acesso, que ao serem implementadas incorporam novos provedores de nuvem e novas plataformas de orquestração de *containers* ao rol de tecnologias suportadas no arcabouço do BioNimbuzBox. Na prática, e como está demonstrado no Capitulo 6, o BioNimbuzBox é capaz de trabalhar com os diversos provedores de nuvem existentes (neste trabalho serão demonstrados dois provedores), além de diferentes plataformas de orquestração.

No BioNimbuzBox, a implementação de uma dessas Interfaces é conhecida como *Driver*. Assim, como pode ser visto na Figura 5.3, a Interface de Provisionamento norteia os *Drivers* de Provisionamento, enquanto a Interface de Orquestração define os métodos dos *Drivers* de Orquestração. Essas Interfaces e seus *Drivers* podem ser definidas da seguinte maneira:

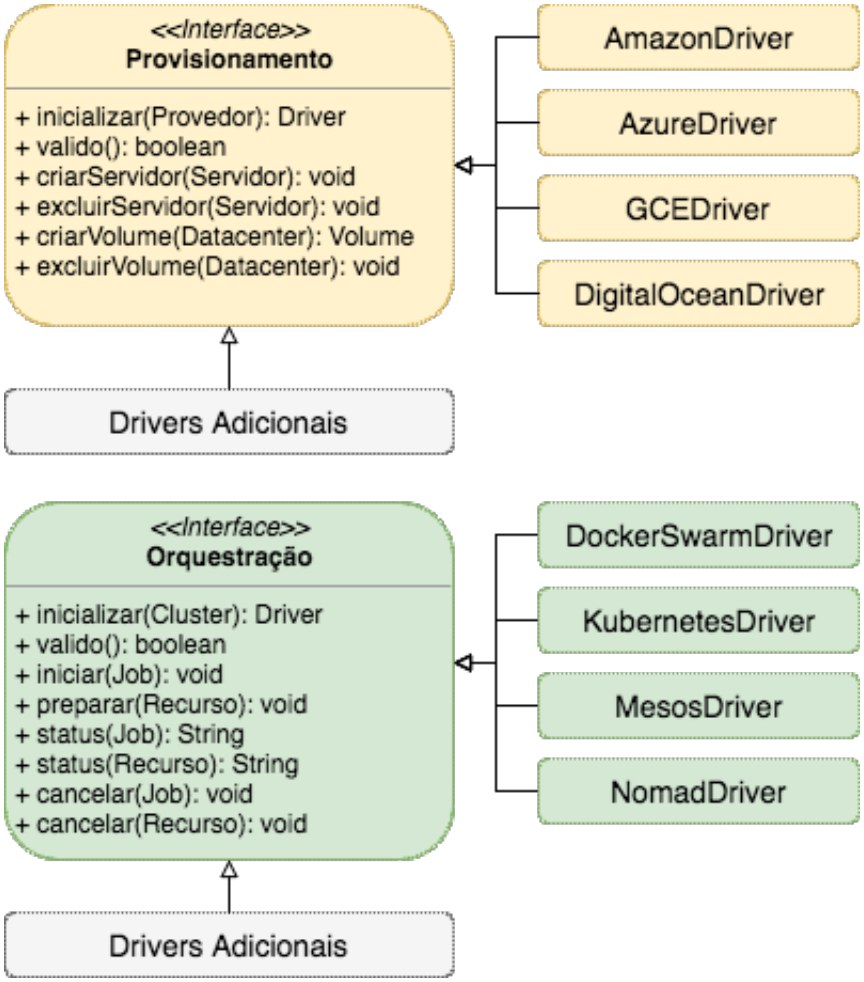

Figura 5.3: Interfaces de Provisionamento e Orquestração.

- **Interface de Provisionamento** – responsável por definir métodos padronizados para o acesso aos diversos provedores de nuvens e seus recursos. A Interface de

Provisionamento, em conjunto com seus *Drivers*, são utilizadas principalmente no Serviço de Provisionamento;

- **Interface de Orquestração** – responsável por definir métodos padronizados para a coordenação e o escalonamento de tarefas nas plataformas de orquestração de *containers* existentes. A Interface de Orquestração, em conjunto com seus *Drivers*, são utilizados nos Serviço de Provisionamento e de Escalonamento. Ela também auxilia na inicialização dos Serviços de Armazenamento e de Monitoramento.

Assim, para um melhor entendimento sobre o BioNimbuzBox, todos os serviços apresentados na Figura 5.1 são detalhados nas próximas seções.

5.2.1 Serviço de Segurança

O Serviço de Segurança, além de realizar a autenticação e a autorização dos usuários e administradores do BioNimbuzBox, é responsável por:

- Manter em segurança a chave privada da Autoridade Certificadora do BioNimbuzBox;
- Assinar com sua chave privada os certificados usados na comunicação entre os serviços;
- Criar um *token* no padrão JWT [41] capaz de ser validado diretamente nos demais Serviços;
- Manter a base de dados de usuários e administradores.

Por ser um serviço vital para o bom funcionamento do BioNimbuzBox, o Serviço de Segurança é o primeiro serviço inicializado na plataforma. Dessa forma, para sua inicialização, o administrador da plataforma deverá providenciar:

a) Um certificado digital em conjunto com sua chave privada – este certificado servirá como Autoridade Certificadora para a assinatura dos demais certificados emitidos para cada um dos serviços do BioNimbuzBox;

b) Um segredo compartilhado – todos os serviços participantes da mesma instalação do BioNimbuzBox devem utilizar o mesmo segredo. Dessa forma, na primeira vez que um serviço solicitar a emissão de um certificado digital ao Serviço de Segurança (por meio do Serviço de Comunicação), esse segredo é verificado para então o certificado ser emitido. Além disso, a utilização de um segredo compartilhado permite ao administrador de uma instalação do BioNimbuzBox forçar uma reinicialização da plataforma ao trocar o segredo compartilhado em tempo de execução;

c) As configurações de acesso a uma solução de banco de dados distribuídos para o armazenamento das informações dos usuários e dos administradores. Na implementação da arquitetura proposta foi utilizada a solução Elasticsearch – um mecanismo de pesquisa e análise distribuído, escalável e em tempo real [35]. Todavia, outras soluções NoSQL distribuídas [38], tais como o Cassandra[5], MongoDB[6] ou CouchDB[7], podem ser facilmente utilizadas, sem prejuízo a execução do BioNimbuzBox, desde que se implementem as interfaces de acesso a dados e se obedeça o modelo de dados apresentado na Figura 5.2.

O Serviço de Segurança pode ser facilmente escalável, inclusive em diferentes nuvens. Para isso o administrador do BioNimbuzBox precisará apenas inicializar novas instâncias do Serviço de Segurança com os mesmos parâmetros de configuração utilizados na primeira inicialização, ou seja, certificado digital, chave privada, segredo compartilhado, e configuração de acesso a base de dados.

Após a inicialização do Serviço de Segurança, os demais serviços podem ser inicializados. Para isso, o administrador deve fornecer o segredo compartilhado utilizado anteriormente e o endereço de acesso de uma ou mais instâncias do Serviço de Segurança. Na primeira mensagem para o Serviço de Segurança, o segredo compartilhado é verificado e um novo certificado digital é emitido e assinado para o serviço em questão, pois essa regra é utilizada para a inicialização de todos os serviços do BioNimbuzBox.

Além de emitir e assinar certificados para a troca segura de mensagens entre todos os serviços, e suas instâncias, o Serviço de Segurança autentica e autoriza usuários a partir da emissão de um *token* JWT [41] que, após criado, pode ser facilmente validado diretamente no serviço acessado, sem uma nova verificação por parte do Serviço de Segurança. Esse processo pode ser visualizado na Figura 5.4.

Dessa forma, antes de realizar qualquer ação no BioNimbuzBox, os usuários precisam ser autenticados e autorizados. Para isso, eles fornecem suas credenciais de acesso ao Serviço de Comunicação (passo 1 da Figura 5.4), que poderá simplesmente repassar as informações para o Serviço de Segurança (uma vez que somente o Serviço de Comunicação é acessível ao usuário) ou realizar algum procedimento prévio. No passo 2 da Figura 5.4, o Serviço de Segurança valida as credenciais de acesso fornecidas pelo usuário e emite um *token* JWT, que é retornado para o solicitante (passo 3 da Figura 5.4).

Um *token* JWT [41] é formado por 3 partes codificadas em Base64[8] e separadas por um ponto final:

[5]Cassandra - http://cassandra.apache.org/
[6]MongoDB - https://www.mongodb.com/
[7]CouchDB - http://couchdb.apache.org/
[8]*Base64 Encoding* - RFC 4648 - https://tools.ietf.org/html/rfc4648

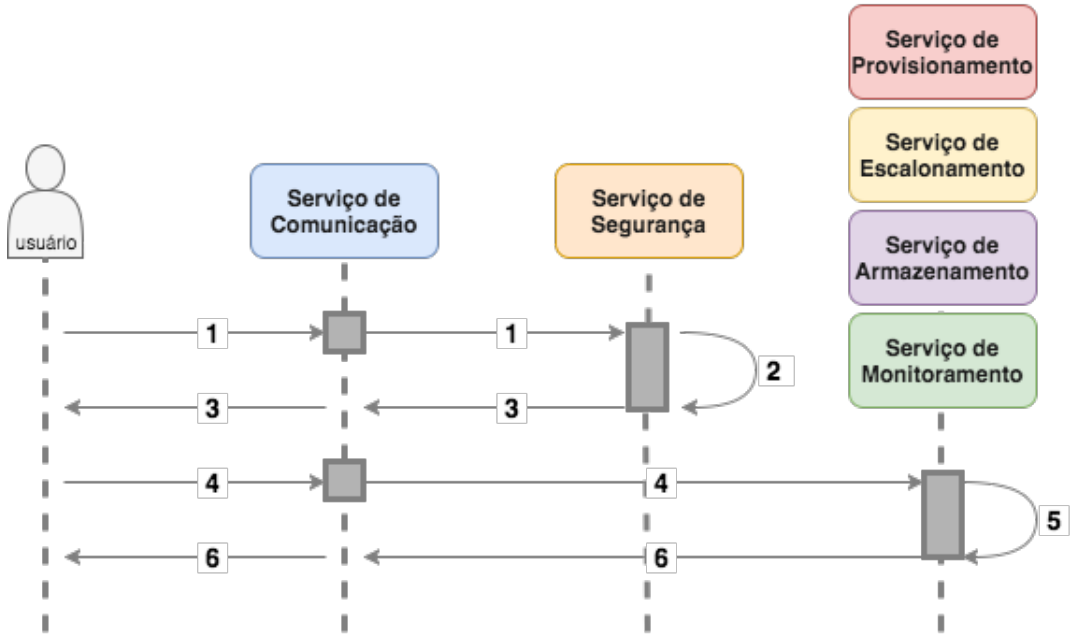

Figura 5.4: Diagrama de Sequência para Emissão e Uso de um *Token* JWT [41].

- Cabeçalho (*Header*) – especifica 2 atributos no formato JSON: o algoritmo *hash* que será utilizado (geralmente, HMAC SHA256[9] ou RSA[10]) e o tipo de *token* (no caso JWT);
- Conteúdo (*Payload*) – JSON contendo informações padronizadas (emissor - *iss*, tempo de expiração - *exp*, assunto - *sub* e audiência - *aud*) além de informações customizadas (por exemplo, perfil do usuário, nome e e-mail);
- Assinatura (*Signature*) – criada a partir de um segredo compartilhado previamente e o algoritmo especificado no Cabeçalho do *token*. Assim, o Cabeçalho (em Base64) e o Conteúdo (em Base64) podem ser assinados garantindo que o utilizador do *token* é quem ele diz ser, e o conteúdo não foi alterado. A seguir um exemplo de como a assinatura é gerada:

```
HMACSHA256(
    base64UrlEncode(header) + "." +  base64UrlEncode(payload),
    secret)
```

De posse do *token* JWT, o usuário passa a realizar chamadas ao Serviço de Comunicação adicionando ao cabeçalho HTTP das mensagens enviadas o *token* JWT emitido previamente (passo 4 da Figura 5.4). No passo 5, antes de processar uma mensagem, o

[9]HMAC SHA256 - RFC 4868 - https://tools.ietf.org/html/rfc4868
[10]RSA - RFC 8017 - https://tools.ietf.org/html/rfc8017

serviço acionado deve verificar a validade do *token*, e somente processar a mensagem caso o *token* seja válido. O passo 6 indica a mensagem processada e devolvida ao solicitante.

5.2.2 Serviço de Comunicação

O Serviço de Comunicação trabalha como um *proxy* entre todos os serviços que integram o BioNimbuzBox. Assim, como pode ser percebido na Figura 5.1, juntamente com o Serviço de Segurança, o Serviço de Comunicação garante que as API's disponibilizadas através dos demais serviços do BioNimbuzBox (Armazenamento, Escalonamento, Provisionamento e Monitoramento) sejam acessadas apenas por usuários, serviços e aplicações autorizadas.

Além de funcionar como um *proxy* de mensagens entre usuários, serviços e aplicações; o Serviço de Comunicação informa a ocorrência de eventos relevantes aos demais serviços do BioNimbuzBox. Inicialmente, os serviços devem informar ao Serviço de Comunicação em quais eventos desejam ser alertados. Dessa forma, na ocorrência de eventos relevantes, o Serviço de Comunicação alertará o(s) serviço(s) registrado(s) no evento específico. Cada serviço é responsável por acionar o Serviço de Comunicação na ocorrência de eventos criados por si. Dessa maneira, o Serviço de Comunicação poderá informar a ocorrência de um evento aos demais serviços. Assim, as principais responsabilidades do Serviço de Comunicação são:

- Conhecer e colaborar com os demais serviços dentro de uma federação;
- Receber as mensagens de usuários e de aplicações externas;
- Filtrar o conteúdo das mensagens que por ele trafegam;
- Localizar e entregar as mensagens aos serviços corretos dentro do BioNimbuzBox;
- Controlar o recebimento e a emissão de eventos entre os serviços do BioNimbuzBox.

5.2.3 Serviço de Provisionamento

O Serviço de Provisionamento, conforme visto na Figura 5.5, administra os recursos computacionais fornecidos pelos diferentes provedores de nuvem. Para isso, o Serviço de Provisionamento utiliza uma Interface de Provisionamento para abstrair os detalhes de implementação das API's dos provedores, padronizando o acesso e a utilização desses recursos, no âmbito do BioNimbuzBox.

Como mencionado anteriormente, para que o BioNimbuzBox passe a utilizar determinado provedor de nuvem, será necessário realizar a implementação da Interface de Provisionamento com os detalhes da API do provedor de nuvem desejado.

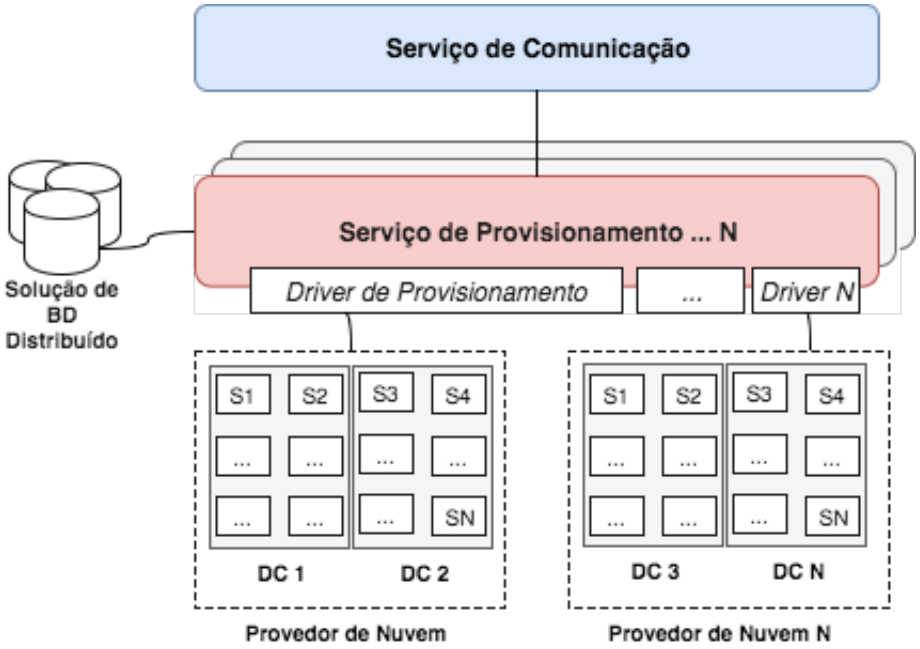

Figura 5.5: Serviço de Provisionamento da Arquitetura BioNimbuzBox.

Os *Drivers* de Provisionamento utilizam as informações definidas nas entidades: Provedor, *Datacenter*, *Cluster* e Servidor para orienta-los durante sua execução. Na Figura 5.3 a Interface de Provisionamento é detalhada em conjunto com seus métodos: *inicializar, valido, criarServidor, excluirServidor, criarVolume e excluirVolume.* O funcionamento do Serviço de Provisionamento, em conjunto com os *Drivers* de Provisionamento, são descritos a seguir.

A partir do acionamento, por parte do usuário administrador, o Serviço de Provisionamento é inicializado com o cadastro dos provedores de nuvem. Para isso, é necessário que o administrador possua as informações de acesso aos provedores desejados. Em seguida, deve-se definir os *Datacenters* que serão utilizados por cada Provedor e quais *Clusters* deverão ser criados. Após a conclusão dessa etapa, inicia-se a criação dos Servidores. Nesse sentido, o Serviço de Provisionamento, ao receber a solicitação de criação de um Servidor, irá seguir o seguinte procedimento:

1 – Identificar o *Clusters*, o *Datacenter* e o Provedor que o Servidor em questão deverá estar vinculado;

2 – Acionar o Serviço de Segurança, para que seja gerado um par de chaves para o novo Servidor;

3 – Inicializar o *Driver* de Provisionamento de acordo com o Provedor identificado. Assim, a inicialização do *Driver* de Provisionamento passa pelas seguintes etapas:

(a) É feita a chamada ao método *inicializar*, passando como parâmetro as informações do Provedor;

(b) O método *inicializa* verifica se o *driver* em questão consta em uma lista de *drivers* inicializados. Essa lista é única por instância do Serviço de Provisionamento, assim:

i. Em caso afirmativo, o *driver* é recuperado da lista e tem seu método *valido* executado;

A. Caso o *driver* continue em estado válido, ele é retornado para o solicitante;

B. Caso o *driver* esteja em um estado inválido, um novo *driver* é inicializado e substituído na lista de *drivers* inicializados.

ii. Em caso negativo, um novo *driver* é inicializado e adicionado na lista de *drivers* inicializados.

4 – Provisionar o novo Servidor utilizando o *Driver* de Provisionamento a partir do método *criarServidor*, e de acordo com as informações do Servidor e o par de chaves gerado. Nesse momento, o Servidor já é inicializado com a plataforma de execução de *Containers* Docker configurada – É utilizada a distribuição CoreOS Container Linux [15].

5 – Além disso, o Serviço de Provisionamento utiliza o *Driver* de Orquestração (de acordo com o tipo de *Cluster*) para:

(a) Caso seja o primeiro Servidor do *Cluster* inicializar um novo *cluster*. Independente do papel escolhido pelo usuário, esse Servidor fará o papel de um gerente. Além disso, o Serviço de Monitoramento é inicializado neste momento (os detalhes serão vistos na Seção 5.2.6);

(b) Caso o *Cluster* já contenha Servidores, adicionar o Servidor ao *Cluster* escolhido, de acordo o papel selecionado (gerente ou executor).

6 – Adicionar as informações do Servidor na solução de banco de dados distribuído.

O método *excluirServidor*, geralmente, é uma tradução direta da API dos provedores de nuvem envolvidos e a Interface de Provisionamento, sendo assim, na maioria das vezes, é executado em um único passo. Os métodos *criarVolume* e *excluirVolume* serão detalhados na Seção 5.2.5.

5.2.4 Serviço de Escalonamento

O Serviço de Escalonamento, mostrado na Figura 5.6, coordena a alocação de recursos para as inúmeras atividades presentes nos variados *workflows* submetidos ao BioNimbuzBox. Para isso, foi definido um arquivo em formato JSON contendo a especificação para a representação de um *workflow*. Nesta especificação constam:

a) Os recursos utilizados pelo *workflow* e seus *jobs*;

b) As configurações de execução de cada *job*;

c) A relação de dependência entre os *jobs*.

Além disso, a definição de um *workflow* deve observar as seguintes restrições:

a) A identificação do *workflow* deve ser única em toda a federação;

b) Cada *job* de um mesmo *workflow* deve ter um identificador único;

c) As dependências entre os *jobs* não podem conter ciclos (grafo acíclico dirigido [76]).

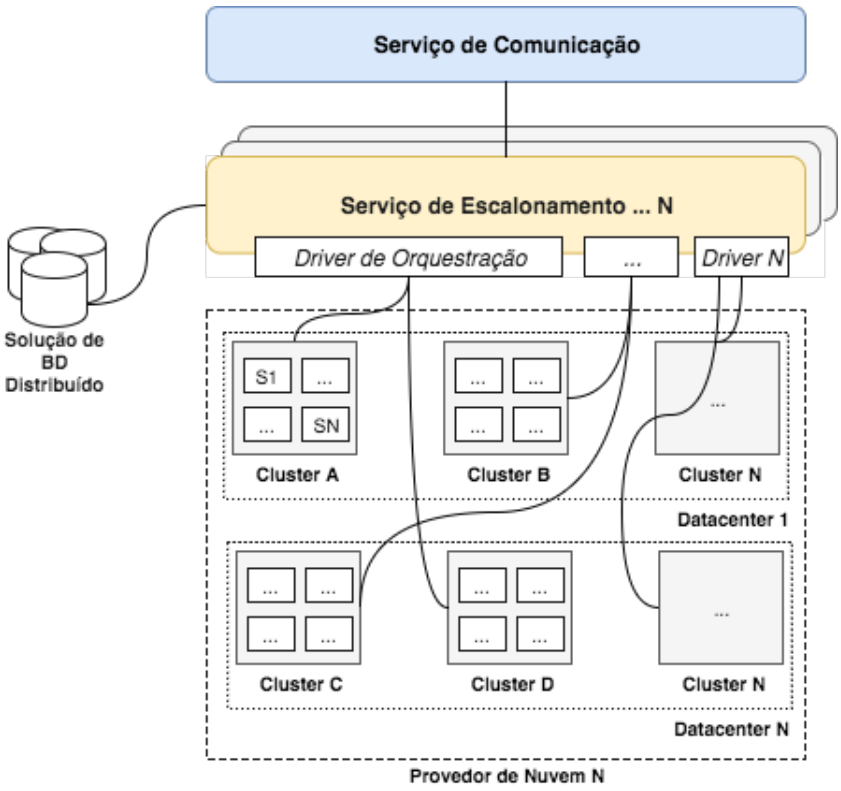

Figura 5.6: Serviço de Escalonamento da Arquitetura BioNimbuzBox.

A Figura 5.7 ilustra um *workflow* de Bioinformática no formato JSON que poderá ser escalonado pelo Serviço de Escalonamento. Na mesma figura é possível perceber a

presença de três *jobs*: *tophat*, *cufflinks* e *cuffmerge*. Além disso, nota-se a dependência (linhas 13, 20 e 27) entre eles e os metadados (imagem e comando) usados em cada *job*. Outra configuração importante é a definição de um recurso que mapeia um volume NFS em um diretório local (linhas 30 a 37).

```
1  {
2    "id":"workflow-1",
3    "nome":"Workflow 1",
4    "meta":{
5      "pesquisador":"Tiago Alves"
6    },
7    "jobs":{
8      "tophat":{
9        "meta":{
10         "imagem":"bionimbuz/tophat",
11         "comando":"tophat -p 4 -G genes.gtf -o /out/"
12       },
13       "dependencies":[]
14     },
15     "cufflinks":{
16       "meta":{
17         "imagem":"bionimbuz/cufflinks",
18         "comando":"cufflinks -p 4 -q -o /out/"
19       },
20       "dependencies":["tophat"]
21     },
22     "cuffmerge":{
23       "meta":{
24         "imagem":"bionimbuz/cufflinks",
25         "comando":"cuffmerge -p 4 -o /out/ -g genes.gtf -s genoma.fa"
26       },
27       "dependencies":["cufflinks"]
28     }
29   },
30   "recursos": {
31     "volume": {
32       "meta":{
33         "servidor-nfs": "123.123.123.123:/volume/workflow1",
34         "mapeamento-local" : "/mnt/workflow1"
35       }
36     }
37   }
38 }
```

Figura 5.7: Exemplo da Representação de um *Workflow* no Formato JSON.

Por utilizar plataformas de orquestração de *containers* disponíveis através de diferentes *Clusters* de Servidores em uma federação, o Serviço de Escalonamento utiliza um algoritmo de escalonamento em duas etapas. Para cada *job* de um *workflow* a primeira etapa de escalonamento indica o *Cluster* responsável pela execução do *job*. Uma vez escolhido o *Cluster* de execução, a plataforma de orquestração responsável por gerenciar o *Cluster* em questão decidirá o Servidor alocado para a execução do *job*. Nesse sentido, atributos definidos no *workflow* e/ou em seus *jobs* podem orientar o algoritmo de escalonamento

quanto a necessidade dos *jobs* serem executados, por exemplo, no mesmo *Cluster* ou até restringir a execução a um conjunto de Servidores, *Datacenters* ou provedores de nuvem.

Por fim, a Figura 5.6 exibe a arquitetura interna do Serviço de Escalonamento. Nela é possível perceber a utilização de *Drivers* de Orquestração na comunicação com diferentes *Clusters*. Além disso, percebe-se a possibilidade de mais de um *Cluster*, orquestrados por diferentes *drivers*, no mesmo *datacenter* e a utilização de uma solução de banco de dados distribuído (assim como o Serviço de Provisionamento), para armazenar as descrições dos *workflows* e os resultados das execuções dos *jobs*.

5.2.5 Serviço de Armazenamento

O Serviço de Armazenamento é responsável por persistir as informações necessárias a execução dos *workflows* de Bioinformática, além dos *logs* de execução e dos arquivos gerados ao término de cada etapa. Nesse sentido, deve conhecer, por meio dos *drivers* de provisionamento, os recursos e os protocolos de acesso oferecidos pelos diversos provedores de nuvem, bem como os mais utilizados em *Datacenters* tradicionais (NFS e iSCSI).

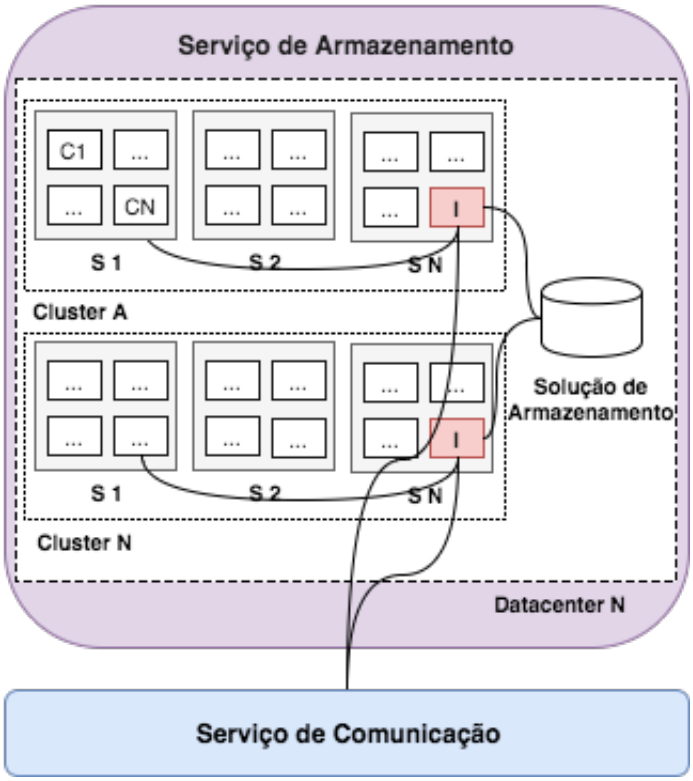

Figura 5.8: Serviço de Armazenamento da Arquitetura BioNimbuzBox.

Como apresentado na Figura 5.8, o Serviço de Armazenamento procura utilizar sempre os recursos mais performáticos disponíveis nos provedores de nuvem utilizados. Para

isso, estabelece como regra que os *containers* apenas terão acesso direto aos dados disponibilizados dentro de um mesmo *Datacenter*, ficando como responsabilidade do Serviço de Armazenamento a sincronização de dados entre *Datacenters*. Essa restrição assegura ao *job* executado em *container* a melhor performance possível, tratando-se de leitura e escrita de dados. Além disso, cada *Cluster* formado pelo BioNimbuzBox possui uma Interface (I) acessível ao usuário, por meio do Serviço de Comunicação, para que sejam enviados e recebidos os arquivos relevantes ao *workflow* que se deseja executar.

5.2.6 Serviço de Monitoramento

O Serviço de Monitoramento é inicializado juntamente com o primeiro Servidor provisionado em um *Cluster*. Dessa forma, cada *Cluster* dentro do BioNimbuzBox é monitorado de forma separada. Além disso, como pode ser percebido na Figura 5.9, o Serviço de Monitoramento é formado por três componentes:

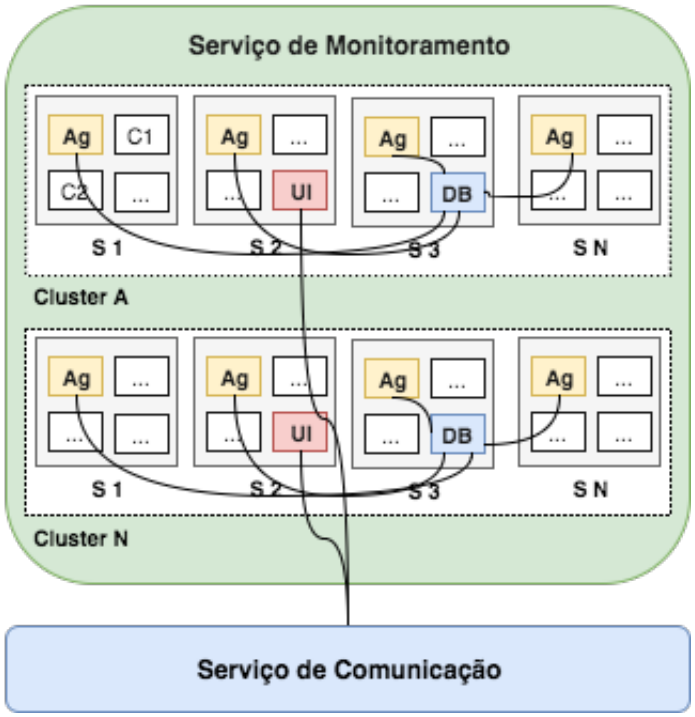

Figura 5.9: Serviço de Monitoramento da Arquitetura BioNimbuzBox.

- Agente de Monitoramento (Ag) – cada servidor provisionado possui um agente de monitoramento encapsulado em *container* que é inicializado pelo *Driver* de Orquestração. Este agente extrai informações de utilização dos recursos computacionais utilizados por cada *container* em execução no servidor (CPU, memória, disco e uso de recursos de entrada e saída *E/S*), enviando-as a um banco de dados disponível no próprio *cluster*;

- Interface de Acesso do Usuário (UI) – também encapsulada em *container*, a UI é acessada pelo usuário através do Serviço de Comunicação. Ela faz a leitura, a partir de um banco de dados disponível no *Cluster*, das informações de utilização de recursos e as exibe em forma de gráfico aos usuários;
- Banco de Dados para Informações de Monitoramento (DB) – cada *Cluster* provisionado recebe uma instância de um banco de dados para que as informações coletadas, pelo agente de monitoramento sejam armazenadas e acessadas pela UI.

Assim, conforme apresentado, o BioNimbuzBox, através da utilização de serviços no modelo de microserviços, apresenta uma arquitetura flexível e escalável, capaz de se adequar a diferentes contextos de execução. Na próxima seção será apresentado um conjunto de trabalhos relacionados ao BioNimbuzBox.

5.3 Trabalhos Relacionados

Além da plataforma de federação BioNimbuZ, que foi detalhada no Capítulo 3, as plataformas de federação, descritas a seguir, estão relacionadas a este trabalho: C-Ports [1], FedUp! [9] e SkyPort [31].

5.3.1 C-Ports

O C-Ports é uma plataforma capaz de executar *containers* Docker através de múltiplos provedores de nuvem e infraestruturas híbridas [1]. Ele foi concebido sob o projeto de código-livre CometCloud [19] [45], que é capaz de construir uma federação definida por software a partir de uma infraestrutura heterogênea. A arquitetura dessa plataforma define dois componentes principais [1]:

- **Solucionador de Restrições (*Constraint Programming Solver*)** – responsável por impor restrições e limites, além de selecionar os recursos que sejam capazes de satisfazerem as restrições e os limites impostos;
- **Escalonador (*Scheduler*)** – responsável por distribuir e executar a carga de trabalho (*workload*) de *containers* em um conjunto de recursos disponíveis, baseado em diferentes objetivos de otimização.

Abdelbaky *et al.* [1] citam dois cenários no qual o C-Ports foi testado:

a) Distribuição de *containers* de acordo com a carga (*Cloud-bursting*) – neste cenário um usuário executa *containers* em um *cluster* local que possui seus recursos monitorados. Novos *containers* são escalonados no *cluster* local enquanto a capacidade

(parametrizável) deste *cluster* não for atingida. A partir do momento em que o limite de recursos é atingido, o C-Ports passa a escalonar novos *containers* em outros recursos disponíveis na nuvem;

b) Execução através de nuvens – neste cenário, o C-Ports baseia-se no custo por hora e capacidade dos recursos fornecidos pelos provedores de nuvem para distribuir *containers* entre esses provedores.

Além disso, na mesma obra [1], os autores concluem que o C-Ports foi capaz de executar *containers* em uma federação composta por cinco provedores de nuvem (IBM Bluemix[11], Amazon AWS[12], Google Cloud Engine[13], Chamaleon Cloud[14] e Future Systems[15]) e dois *clusters*.

Do ponto de vista funcional, o C-Ports apresenta similaridades com o BioNimbuzBox. Contudo, além de possuírem arquiteturas diferentes, o C-Ports não coordena a execução de *workflows* de Bioinformática em uma federação de nuvens, pois ele se preocupa essencialmente com aplicações que executam como serviço (por exemplo, servidores de aplicação e bancos de dado).

5.3.2 FedUp!

O FedUp! é uma plataforma projetada para criar federações através de um modelo de serviço similar ao entregue por uma Plataforma como Serviço – *PaaS* [9]. Neste sentido, a arquitetura do FedUp! é capaz de:

a) Gerenciar todo o ciclo de vida de uma federação (*Federation as a Service*);

b) Permitir que uma nuvem: (1) participe como membro de diversas federações; (2) candidate-se como membro de outras federações de acordo com uma descrição de recursos que ela pode fornecer e/ou que ela necessite; (3) migre facilmente de uma federação para outra;

c) Manter um repositório central que armazena os registros relacionados as federações geradas.

No que tange ao processo de criação e de manutenção de federações, o FedUp! enumera uma lista taxativa de requisitos. Dessa forma, para os autores, uma federação [9]:

[11]IBM Bluemix – https://www.ibm.com/cloud-computing/bluemix/
[12]Amazon AWS – https://aws.amazon.com
[13]Google Cloud Engine – https://cloud.google.com/
[14]Chamaleon Cloud – https://www.chameleoncloud.org/
[15]Future Systems – https://portal.futuresystems.org/

a) Deve suportar nuvens heterogêneas;

b) Deve possuir um baixo impacto na infraestrutura existente das nuvens participantes;

c) Deve permitir a geração e a destituição de federações de forma transparente respeitando cada nuvem participante;

d) Deve suportar a participação de novas nuvens membro, assim como sua destituição;

e) Deve suportar a geração automática de federações;

f) Deve gerar federações orquestradas por um sistema automático;

g) Deve fornecer mecanismos de controle de acesso aos recursos federados;

h) Deve permitir que o proprietário de uma federação especifique os valores utilizados em *tags*[16] pré-definidas.

Além disso, como sugestão para a implementação da arquitetura do FedUp!, os autores sugerem o uso de três tecnologias: *containers* Docker para encapsular aplicações; a ferramenta de configurações SaltStack[17] para coordenar a instalação dos serviços responsáveis pela federação e a plataforma de orquestração de *containers* Kubernetes.

Em relação ao BioNimbuzBox, a plataforma de federação FedUp! [9] aborda um modelo diferente para federações. Nesse modelo, as nuvens participantes tem maior autonomia, podendo inclusive escolher em quais federações desejam participar. Contudo, o FedUp! não possui resultados quanto a sua implementação e assim como o C-Ports, a arquitetura do FedUp! não possui um modelo para a coordenação de *workflows* de Bioinformática.

5.3.3 SkyPort

O SkyPort [31] é uma solução em *containers* Docker construída como uma extensão da plataforma de análise de dados AWE/Shock [66]. Nesse sentido, a plataforma AWE/Shock fornece ambientes escaláveis de execução de *workflows* para dados científicos na nuvem e a extensão SkyPort distribui e executa aplicações encapsuladas em *containers* nessa plataforma.

Para melhor compreender o SkyPort é necessário explorar os três principais componentes da plataforma AWE/Shock, que são [31]:

- **Shock** – é um sistema de gerenciamento de dados desenhado como um sistema de armazenamento de objetos (similar ao Amazon S3 [3]). Os dados armazenados

[16] tags – marcações no formato de texto utilizadas para identificar objetos e/ou recursos.
[17] SaltStack – https://saltstack.com

no Shock são representados a partir de um identificador único e de metadados que descrevem informações de proveniência científica e computacional. O Shock fornece uma API para armazenar, consultar e recuperar dados e metadados. Além disso, permite que os dados sejam indexados de diferentes maneiras, viabilizando paralelismos e otimizações para formatos de arquivos relevantes para Bioinformática (FASTA, FASTQ e BAM) [31];

- **Servidor AWE (AWE *server*)** – o servidor AWE é um gerenciador de recursos e um escalonador de *jobs*. Ele recebe como parâmetro de entrada um documento contendo a descrição de um *workflow* e cria unidades de trabalho menores que serão processadas – *checked out* – a partir dos executores AWE;
- **Executor AWE (AWE *worker*)** – os executores AWE extraem – *checked out* – e executam as unidades de trabalho criadas pelo servidor AWE. As configurações de um executor determinam quais unidades de trabalho ele pode extrair do servidor AWE, o grupo o qual faz parte, e uma lista de comandos que podem ser executadas por ele. Após extrair uma unidade de trabalho do servidor, o executor realiza o *download*, a partir do Shock, de todos os arquivos de entrada e base de dados. Após o processamento da unidade de trabalho pelo executor, as bases de dados são mantidas em cache, os arquivos de entrada são excluídos e o resultado do processamento é enviado para Shock.

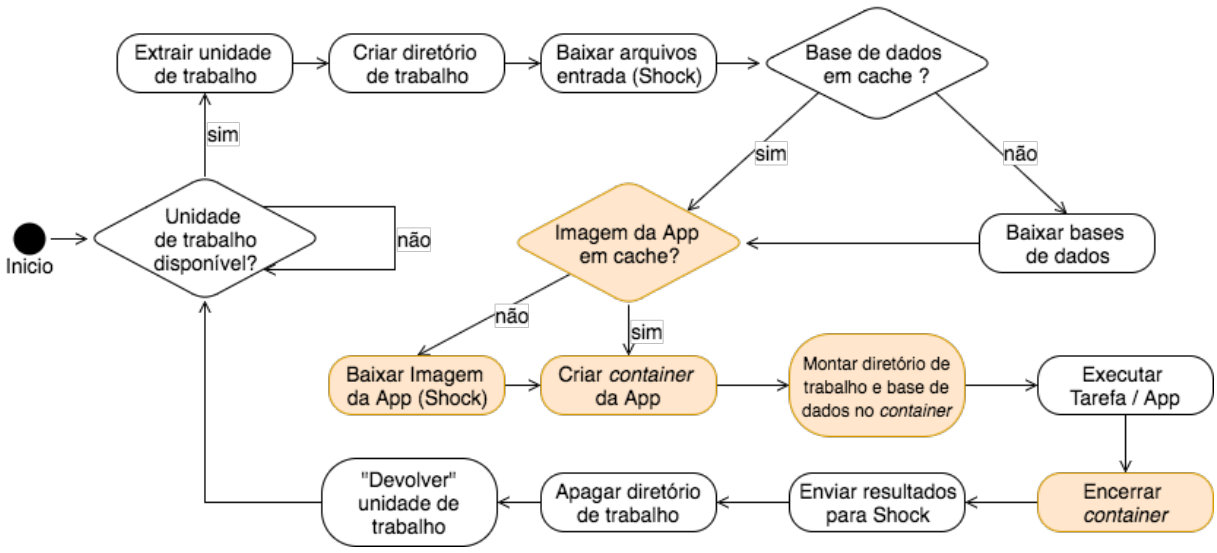

Figura 5.10: Diagrama de Atividades da Plataforma SkyPort AWE/Shock [31].

A Figura 5.10 exibe um diagrama de atividades das ações realizadas pelo SkyPort para executar uma unidade de trabalho de um *workflow*. Além das ações já listadas

anteriormente na descrição do Executor AWE, o SkyPort inclui as seguintes ações ao fluxo tradicional do executor:

- Verificar se a imagem da aplicação já foi disponibilizada pelo executor e encontra-se em cache (Condição "Imagem da App em cache?" conforme apresentado na Figura 5.10);
- Baixar do Shock a imagem da aplicação (Atividade "Baixar Imagem da App (Shock)" da Figura 5.10);
- Criar um *container* Docker baseada na imagem da aplicação (Atividade "Criar *container* da App" da Figura 5.10);
- Montar diretórios de trabalho e base de dados no *container* (Atividade de mesmo nome da Figura 5.10);
- Encerrar o *container* criado (Atividade "Encerrar *container*" da Figura 5.10).

O SkyPort é a plataforma que mais se assemelha com a proposta do BioNimbuzBox. Contudo, como pode ser percebido na Tabela 5.1, o SkyPort utiliza apenas o modelo de Infraestrutura como Serviço para adquirir recursos dos provedores de nuvem, enquanto o BioNimbuzBox, além de utilizar a Infraestrutura como Serviço é capaz de trabalhar com Plataformas de Orquestração de *Containers* entregues no modelo de Plataforma como Serviço.

Tabela 5.1: Comparação entre as Características dos Trabalhos Relacionados.

Características	**BioNimbuZ** [49] [61] [62]	**C-Port** [1]	**FedUp!** [9]	**SkyPort** [31]	**BioNimbuzBox**
Coordenação e execução de *workflows* de Bioinformática	Sim	Não	Não	Sim	Sim
Solução para distribuição e instalação de aplicações	Não	Sim	Sim	Sim	Sim
Capacidade de utilizar diferentes plataformas de orquestração de *containers* sob o modelo PaaS	Não	Sim	Não	Não	Sim
Utilizado sob o modelo de Software como Serviço SaaS	Sim	Não	Não	Não	Sim

Além dessas plataformas, o BioNimbuzBox compartilha grande parte das suas características com o BioNimbuZ. Contudo, o BioNimbuZ não possui uma solução para a

distribuição e a instalação de aplicações de Bioinformática (suas versões e dependências), devendo essas atividades serem realizadas manualmente. Além disso, o BioNimbuzBox utilizou o conceito de serviços inseridos pelo BioNimbuZ para propor uma atualização de arquitetura, no qual cada serviço passa a funcionar de maneira independente (expondo, quando necessário, uma API de acesso).

Os trabalhos relacionados apresentados nesta seção podem ter suas características comparadas na Tabela 5.1. Dessa forma, como pode set notado na Tabela 5.1, o BioNimbuzBox foi a única arquitetura que contemplou todas as características enumeradas.

5.4 Considerações Finais

Neste capítulo foram abordadas as principais características propostas pelo BioNimbuzBox, uma arquitetura baseada em *containers* que utiliza os conceitos de separação em serviços trazidos pelo BioNimbuZ para execução de *workflows* de Bioinformática. Além disso, foram discutidos os trabalhos relacionados, apresentando as semelhanças e as diferenças em relação ao trabalho proposto.

No próximo capítulo serão apresentados os resultados obtidos na utilização do BioNimbuzBox na execução de um *workflow* de Bioinformática.

Capítulo 6

Resultados Obtidos

Neste capítulo são apresentados os resultados obtidos na execução de um *workflow* real de Bioinformática na arquitetura do BioNimbuzBox. Para isso, foram propostos quatro cenários distintos de execução, possibilitando, ao final deste capítulo, identificar claramente, pelo uso de uma arquitetura baseada em *containers*, as vantagens e as dificuldades encontradas no processamento de *workflows* de Bioinformática em nuvens federadas. Assim, será possível, especialmente, avaliar o *overhead* provocado na utilização do BioNimbuzBox.

6.1 Metodologia

Para os testes realizados com o BioNimbuzBox, optou-se por dividir os experimentos em quatro cenários diferentes. Em cada um deles são descritos os objetivos do teste, o ambiente de execução, os resultados obtidos e a análise dos resultados. Assim, os testes foram realizados utilizando-se um *workflow* real de Bioinformática que, a partir de um genoma de referência, executa seis *jobs* distribuídos em quatro etapas: preparação (*job* Preparação), construção do índice (*job* bowtie2 build [48]), mapeamento (*jobs* tophat2 "a" e "b" [44]) e montagem (*jobs* cufflinks "a" e "b" [72]). A Figura 6.1 ilustra este *workflow*, no qual, além dos *jobs* mencionados, nota-se ainda:

- As dependências entre os *jobs*;
- Os arquivos gerados e os arquivos utilizados por cada um dos *jobs*.

Além disso, a Figura 6.1 apresenta o *job* Preparação com uma cor diferente dos demais *jobs*. Isso se dá uma vez que a execução do *job* Preparação utiliza uma imagem de *container* diferente daquela empregada nos demais *jobs*. Nesse sentido, é importante mencionar a existência de uma imagem auxiliar de *container* que é utilizada para a sincronização dos

dados processados e gerados por cada *job*. Assim sendo, para o *workflow* testado foram utilizadas três imagens de *containers* diferentes:

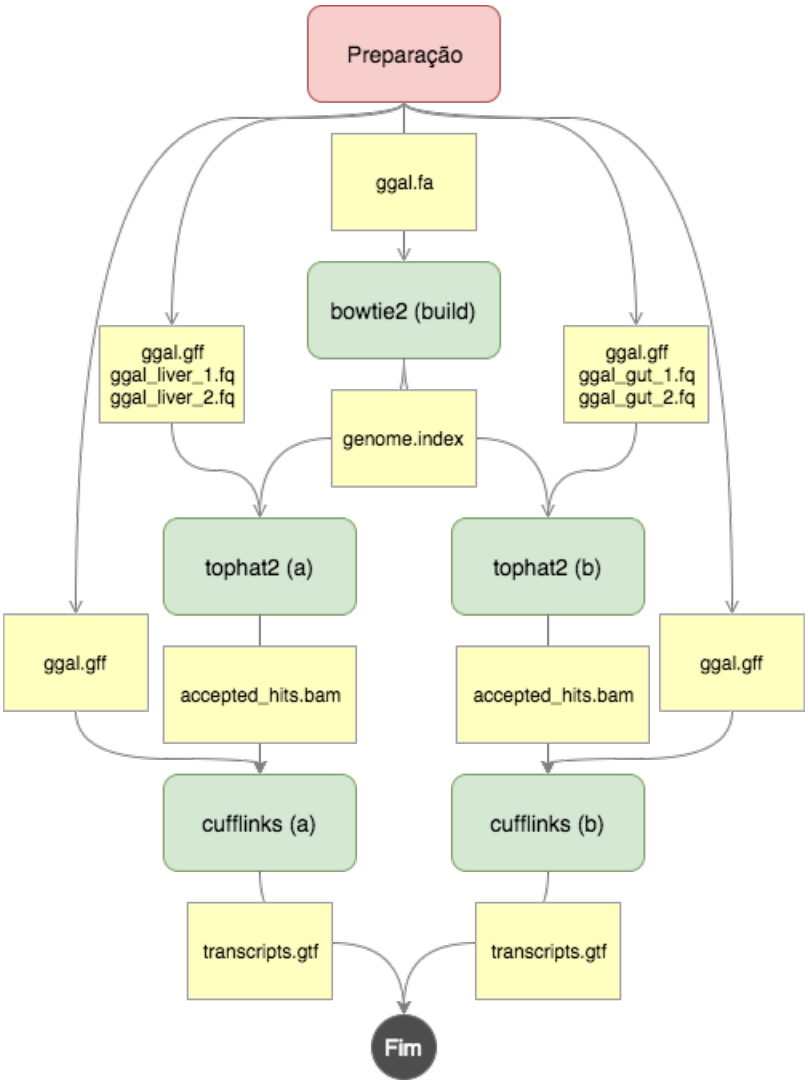

Figura 6.1: *Workflow* Utilizado nos Testes deste Trabalho.

- Preparação – responsável por efetuar o *donwload* dos arquivos inicias que serão utilizados pelos demais *jobs*, além de preparar a estrutura de diretórios que irão compor a execução do *workflow*;
- Sincronização – garante que os dados de entrada estão atualizados e prontos para a execução do *job* (*download* antes de executar o *job*) e realiza a sincronização dos dados gerados pelo *job* (*upload* após a execução do *job*). As duas ações de sincronização (*download* e *upload*) são executadas no mesmo *cluster* no qual o *job* foi escalonado;

- Aplicação – no *workflow* apresentado na Figura 6.1 será utilizada uma única imagem de *container* com todas as aplicações, e suas dependências, instaladas (bowtie2 [48], tophat2 [44] e cufflinks [72]). É possível utilizar imagens de *container* separadas por aplicação, contudo, nesse caso, como a aplicação tophat2 possui dependência com a aplicação bowtie2, há ganho na união das aplicações em uma única imagem.

Outro ponto que cabe destaque é em relação ao tamanho dos arquivos de entrada. Nesse sentido, foram utilizados para todos os testes seis arquivos de tamanhos reduzidos, listados na Tabela 6.1.

Tabela 6.1: Arquivos de Entrada Utilizados na Execução do *Workflow*

Nome do Arquivo	**Tamanho**
ggal.fa	170 Kb
ggal.gff	26 Kb
ggal_gut_1.fq	676 Kb
ggal_gut_2.fq	676 Kb
ggal_liver_1.fq	676 Kb
ggal_liver_2.fq	676 Kb

Além do exposto, conforme serão vistas nas ilustrações presentes em cada cenário de execução a seguir, a infraestrutura utilizada pelo BioNimbuzBox foi executada inteiramente a partir do provedor de nuvem DigitalOcean[1], em um *datacenter* denominado "NYC1", localizado na cidade de Nova Iorque – EUA.

A informação da localização da infraestrutura do BioNimbuzBox será necessária para compreender os resultados obtidos em se tratando da execução do *workflow* exposto em diferentes *datacenters* e na federação de nuvens. Para os cenários propostos, o BioNimbuzBox executou integralmente quatro dos seus seis serviços (Serviço de Comunicação, Serviço de Armazenamento, Serviço de Provisionamento e Serviço de Escalonamento) e, parcialmente, o Serviço de Segurança (não foi utilizada autenticação do usuário e o *token* JWT, uma vez que foi considerado que o usuário já estaria logado no BioNimbuzBox). E o Serviço de Monitoramento foi desligado uma vez que nos testes as informações de utilização de processamento, memória e E/S não serão analisadas. Para coleta de informações do tempo de duração dos *jobs* e do *workflow*, foram utilizas as entradas disponível nos *logs* do Serviço de Escalonamento. Também é importante mencionar que cada serviço foi executado de maneira autônoma em um *container* próprio, executando em um servidor com dois processadores, 2Gb de memória e 40Gb de armazenamento em disco

[1]DigitalOcean – https://digitalocean.com

SSD[2]. Além disso, foi utilizado o Elasticseach[3] – solução de banco de dados NoSQL distribuída – para armazenamento das informações relativas a infraestrutura provisionada pelo BioNimbuzBox e o registro do *workflow* testado.

Por fim, para cada cenário a ser apresentado, foram realizadas 10 execuções consecutivas do *workflow* testado (com intervalo de 10 segundos entre elas). Após cada execução, todos os arquivos (iniciais e gerados) eram excluídos e os *logs* de execução eram coletados.

6.2 Cenário A

Neste cenário, como pode ser visto na Figura 6.2, foi comparado a execução do *workflow* de referência em um servidor (*nyc1-a*) executando no mesmo provedor de nuvem – DigitalOcean – (e *datacenter* - NYC1) da infraestrutura do BioNimbuzBox, com a execução em servidor (*sfo1-a*) no mesmo provedor, mas em um *datacenter* (SFO1 – São Francisco - EUA) a uma distância considerável (aproximadamente 5 mil quilômetros) da infraestrutura do BioNimbuzBox.

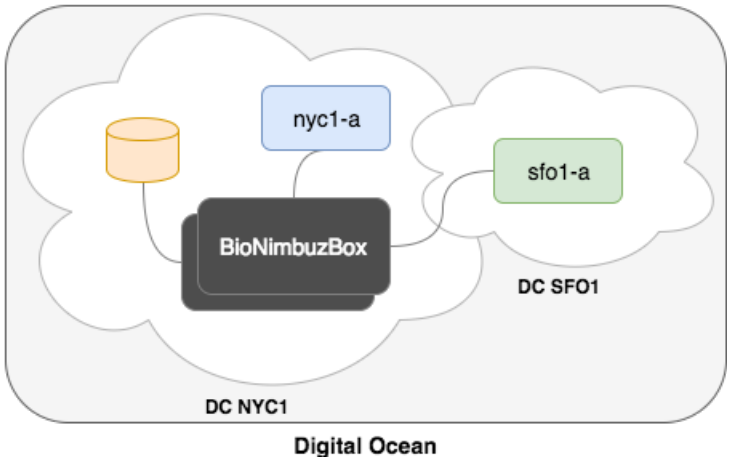

Figura 6.2: Infraestrutura do Cenário A de Teste.

O objetivo deste teste é compreender a relevância da proximidade dos dados em relação a localização do recurso físico e a influência do mecanismo de *cache* de imagens de *containers* na execução de *workflows*.

Além da infraestrutura do BioNimbuzBox, foram provisionados dois servidores (*nyc1-a* e *sfo1-a*) idênticos em termos de configuração (1 processador, 512Mb de memória e 20Gb de armazenamento SSD).

[2] *SSD – Solid State Drive* (Disco de Estado Sólido)
[3] Elasticsearch – https://www.elastisearch.co

6.2.1 Resultados do Cenário A

Primeiramente, o *workflow* testado foi executado 10 vezes no servidor *nyc1-a* (conforme os resultados apresentados na Tabela 6.2). Em seguida, a mesma quantidade de execuções foi realizada no servidor *sfo1-a* (resultados apresentados na Tabela 6.3).

Tabela 6.2: Cenário A - Execução no Servidor *nyc1-a*.

Job	**Download** (segundos)			**Execução** (segundos)			**Upload** (segundos)			**Total** (segundos)		
	Min.	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**
1-prepare	0	0	0	6	7	6,1	6	26	8,3	12	33	14,4
2-bowtie-build	6	6	6	6	**56**	11	6	6	6	18	69	23,8
3-tophat2-a	6	6	6	22	22	22	6	6	6	34	35	34,9
3-tophat2-b	6	6	6	21	22	21,9	6	6	6	34	35	34,9
4-cufflinks-a	6	6	6	6	6	6	6	6	6	19	19	19
4-cufflinks-b	6	6	6	6	6	6	6	6	6	19	19	19
Workflow Total										**91**	**163**	**98,5**

Neste cenário existia uma única opção para o Serviço de Escalonamento, que no primeiro conjunto de execuções foi apresentado como escolha apenas o servidor *nyc1-a* e para o segundo conjunto de execuções somente o servidor *sfo1-a*.

Tabela 6.3: Cenário A - Execução no Servidor *sfo1-a*.

Job	**Download** (segundos)			**Execução** (segundos)			**Upload** (segundos)			**Total** (segundos)		
	Min.	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**
1-prepare	0	0	0	9	14	9,5	25	32	26,5	34	46	36,6
2-bowtie-build	9	9	9	9	**69**	15,4	9	13	9,8	27	88	34,7
3-tophat2-a	9	9	9	21	29	25,6	9	17	9,8	40	53	45,5
3-tophat2-b	9	9	9	21	29	25,5	9	17	9,8	40	53	45,4
4-cufflinks-a	9	9	9	9	13	9,4	9	9	9	27	32	28,2
4-cufflinks-b	9	13	9,4	9	13	9,4	9	9	9	27	32	28,7
Workflow Total										**136**	**220**	**151,2**

6.2.2 Análise do Cenário A

Tomando como referência as Tabelas 6.2 e 6.3 é perceptível a diferença entre os tempos de *download*, *upload* e o total dos *jobs* executados no *datacenter* NYC1 e SFO1. Além disso, as execuções realizadas no servidor *sfo1-a* tiveram uma variação maior entre os tempos

mínimos e máximos. Esse comportamento é explicável, uma vez que o servidor *nyc1-a* está localizado no mesmo *datacenter* da infraestrutura do BioNimbuzBox (portanto, o fluxo de dados de sincronização de arquivos com o Serviço de Armazenamento ocorre no mesmo *datacenter*). Por outro lado, o servidor *sfo1-a* encontra-se a uma distância considerável em relação à origem dos dados (Serviço de Armazenamento em NYC1).

Outro fato interessante ocorre na primeira execução do *job* 2-bowtie-build. Neste momento é apresentado o tempo máximo de execução do *job*, o qual foi 56 segundos no servidor *nyc1-a* e 69 segundos no servidor *sfo1-a*. Isso ocorre porque a plataforma de execução de *containers* identifica que a imagem do *container* ("Aplicação"), responsável pela execução do *job*, não se encontra disponível no servidor e precisa ser transferida.

Dessa forma, o tempo de execução exibido nas tabelas também inclui o tempo levado para realizar o *download* da imagem em questão. Como a imagem utilizada para a execução da aplicação *bowtie-build* mantém-se no servidor, as demais execuções (*tophat2* e *cufflinks*) não precisam realizar *download* da imagem, não tendo seus tempos de execução afetados, ou seja, apresentam tempo de execução mínimo e máximo muito semelhantes. Cabe ainda observar que nas execuções subsequentes do *workflow* a imagem do *container* permaneceu em *cache* tornando as execuções do bowtie-build (a partir da segunda execução) mais rápidas.

6.3 Cenário B

No Cenário B, como visto na Figura 6.3, o *workflow* de referência é executado em um *cluster* com 3 servidores em um *datacenter* remoto (SFO1). Mais uma vez, cada servidor (*sfo1-a*, *sfo1-b* e *sfo1-c*) apresenta a mesma configuração do cenário anterior (1 processador, 512Mb e 20Gb de disco SSD). Para esse *cluster*, todas as máquinas foram recriadas através do Serviço de Provisionamento e, portanto, não possuíam nenhuma informação em *cache*.

Na Figura 6.3 é possível perceber um repositório de dados compartilhado entre os três servidores. Como descrito no Capítulo 5 – na seção dedicada ao Serviço de Armazenamento, esse repositório de dados funciona como um *cache* no nível de *datacenter*. A implementação desse *cache* utilizou um serviço NFS[4] executado a partir do servidor *sfo1-a* e disponibilizado para os demais servidores do *cluster* (*sfo1-b* e *sfo1-c*).

O objetivo deste teste é esclarecer o funcionamento do mecanismo de *cache* de imagens de *containers* em um *cluster*.

[4] *NFS – Network File System* (Sistema de Arquivos de Rede)

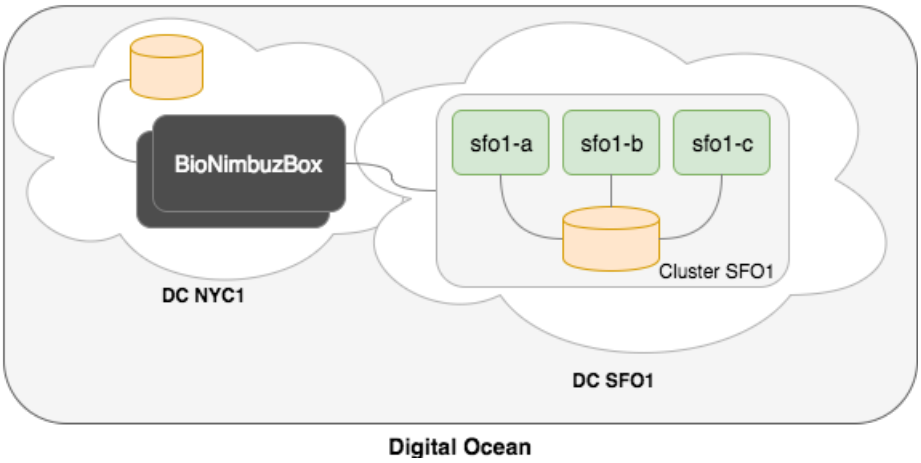

Figura 6.3: Infraestrutura do Cenário B de Teste.

6.3.1 Resultados do Cenário B

Para este teste, além da Tabela 6.4, que consolida todas as 10 execuções do *workflow* de referência realizadas no *cluster* formado pelos servidores (*sfo1-a*, *sfo1-b* e *sfo1-c*) no *data-center* SFO1, será apresentado a Tabela 6.5, que exibe o servidor alocado para executar cada *job* da primeira execução do *workflow* testado; e a Tabela 6.6, que é semelhante a Tabela 6.5, mas considera a segunda execução do mesmo *workflow*.

Tabela 6.4: Cenário B - Execução no *Cluster* SFO1.

Job	**Download** (segundos)			**Execução** (segundos)			**Upload** (segundos)			**Total** (segundos)		
	Min.	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**
1-prepare	0	0	0	9	14	9,5	25	29	26,3	34	40	36,2
2-bowtie-build	9	34	11,5	9	**54**	13,5	9	13	9,4	27	102	34,5
3-tophat2-a	9	9	9	17	**104**	27,7	9	13	9,4	35	128	47
3-tophat2-b	9	13	9,4	17	21	18,6	9	17	9,8	35	44	38,7
4-cufflinks-a	9	9	9	9	9	9	9	9	9	27	28	27,4
4-cufflinks-b	9	9	9	9	**68**	14,9	9	38	11,9	27	115	36
Workflow Total										**135**	**307**	**153,9**

6.3.2 Análise do Cenário B

Na Tabela 6.4 é possível perceber que além do *job* 2-bowtie-build, os *jobs* 3-tophat2-a e 4-cufflinks-b apresentaram, respectivamente, tempo de execução máximo bem superior aos *jobs* 3-tophat2-b e 4-cufflinks-a. Esse comportamento pode ser explicado a partir

da Tabela 6.5, que exibe em qual servidor cada *job* foi alocado na primeira execução do *workflow*.

Nesse sentido, nota-se que o *job* 2-bowtie-build é executado no servidor *sfo1-c* e, como na primeira execução do *workflow* a imagem de *container* para aplicações ainda não havia sido utilizada nesse servidor, a plataforma de orquestração de *containers* precisou transferir essa imagem. O mesmo comportamento repetiu-se no *job* seguinte (3-tophat2-a), que executou no servidor *sfo1-b*. Contudo, para o *job* subsequente (3-tophat2-b), foi utilizada a imagem em *cache* que já havia sido transferida para o servidor *sfo1-c* (o mesmo ocorreu para o *job* 4-cufflinks-a executando no servidor *sfo1-c*). Por fim, para o *job* 4-cufflinks-b foi necessário que o servidor *sfo1-a* também transferisse a imagem de *container* para aplicações.

Tabela 6.5: Alocação de *Jobs* na 1ª Execução do *Workflow* Testado no *Cluster* SFO1

Job	**Alocação**	**Download** (segundos)	**Execução** (segundos)	**Upload** (segundos)	**Total** (segundos)
1-prepare	sfo1-a	0	14	26	40
2-bowtie-build	**sfo1-c**	34	**54**	13	102
3-tophat2-a	**sfo1-b**	9	**104**	13	128
3-tophat2-b	sfo1-c	9	17	17	44
4-cufflinks-a	sfo1-c	9	9	9	27
4-cufflinks-b	**sfo1-a**	9	**68**	38	115
Workflow Total					307

A Tabela 6.6, que exibe a segunda execução do *workflow* testado, apresenta um comportamento diferente daquele encontrado na primeira execução. Nessa tabela, todos os *jobs* possuem tempo de execução próximos ao mínimo que foi consolidado na Tabela 6.4 (inclusive para o tempo total de execução do *workflow*). Esse comportamento justifica-se uma vez que todos os servidores já possuíam em *cache* uma cópia da imagem de *container* para aplicações.

Os testes realizados nos Cenários C e D terão seus resultados avaliados de forma integrada na Seção 6.6. Dessa forma, a Seção 6.4 – Cenário C e a Seção 6.5 – Cenário D contarão apenas com a descrição do ambiente de execução, os objetivos do teste e os resultados obtidos.

6.4 Cenário C

No Cenário C o *workflow* testado é executado em dois *clusters* com três servidores cada. Um dos *clusters* está localizado no *datacenter* SFO1 e o outro em NYC3. Os *clusters*

Tabela 6.6: Alocação de *Jobs* na 2ª Execução do *Workflow* Testado no *Cluster* SFO1.

Job	**Alocação**	**Download** (segundos)	**Execução** (segundos)	**Upload** (segundos)	**Total** (segundos)
1-prepare	sfo1-c	0	9	25	35
2-bowtie-build	sfo1-a	9	9	9	27
3-tophat2-a	sfo1-b	9	21	9	39
3-tophat2-b	sfo1-a	9	17	9	36
4-cufflinks-a	sfo1-c	9	9	9	28
4-cufflinks-b	sfo1-a	9	9	9	27
Workflow Total					**136**

foram provisionados com o Serviço de Provisionamento em um mesmo provedor de nuvem, DigitalOcean, e as configurações de cada servidor mantiveram-se as mesmas dos cenários anteriores (1 processador, 512Mb e 20Gb de disco SSD). A mesma implementação de *cache* no nível de *datacenter* adotada no Cenário B foi utilizada neste cenário. A Figura 6.4 ilustra o Cenário C.

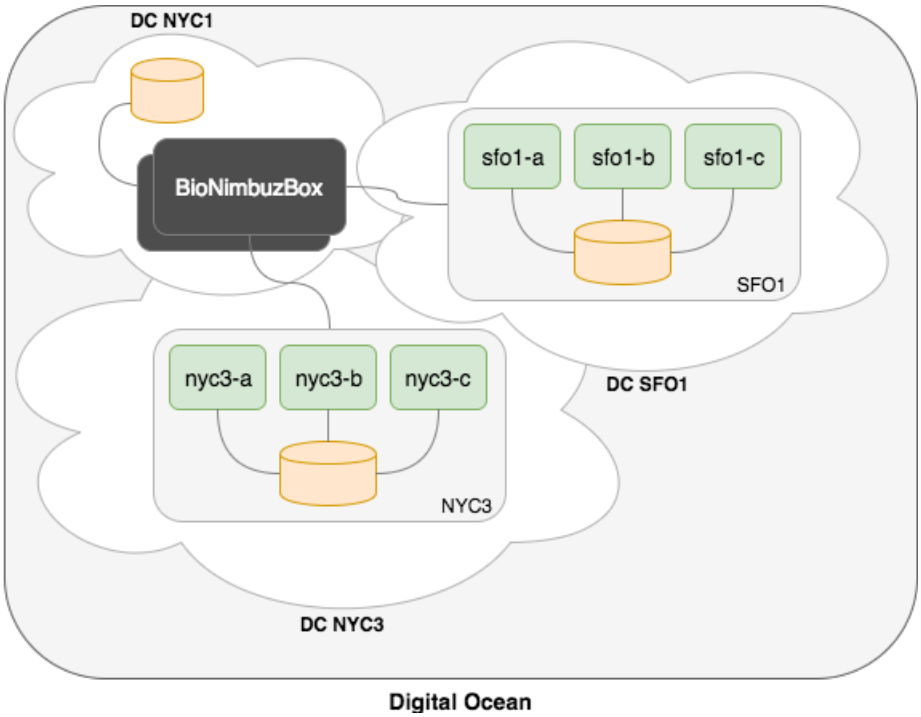

Figura 6.4: Infraestrutura do Cenário C de Teste.

Para o escalonamento dos *jobs* neste teste, o Serviço de Escalonamento selecionou, aleatoriamente, um *cluster* entre os dois disponíveis. Uma vez escolhido o *cluster* de exe-

cução, a plataforma de orquestração de *containers* procurou eleger, de forma igualitária, a seleção do servidor [20].

O objetivo deste teste é demonstrar como a utilização de um *cluster* de servidores (NYC3) próximo a origem dos dados (Serviço de Armazenamento) pode compensar a utilização de um *cluster* mais distante (SFO1).

6.4.1 Resultados do Cenário C

Para o Cenário C, o resultado consolidado das execuções do *workflow* testado pode ser visualizado na Tabela 6.7.

Tabela 6.7: Execução nos *clusters* sfo1 e nyc3.

Job	**Download** (segundos)			**Execução** (segundos)			**Upload** (segundos)			**Total** (segundos)		
	Min.	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**
1-prepare	0	0	0	6	9	8,4	9	29	21,7	15	39	30,3
2-bowtie-build	6	29	8,9	6	62	12,5	6	15	7,5	18	107	29,6
3-tophat2-a	6	9	7,8	15	77	28,9	6	9	7,8	28	90	45,1
3-tophat2-b	6	33	10,2	15	110	30,3	6	17	8,6	28	162	49,9
4-cufflinks-a	6	29	12,5	6	75	14,1	6	37	10,6	18	131	37,9
4-cufflinks-b	6	9	6,6	6	9	6,6	6	9	6,6	18	28	20,6
Workflow Total										**97**	**368**	**152,7**

Além da Tabela 6.7, a Figura 6.5 é importante para a análise deste teste porque exibe um gráfico com o tempo total de cada execução do *workflow* testado em cada um dos cenários propostos neste capítulo. A análise dos resultados obtidos neste teste serão realizadas em conjunto com os testes do Cenário D, na Seção 6.6.

6.5 Cenário D

A infraestrutura construída pelo BioNimbuzBox para o Cenário D inclui uma federação entre os provedores de nuvem DigitalOcean e Amazon AWS. Neste cenário, foram utilizados três *datacenters* diferentes, sendo dois na DigitalOcean (SFO1 e NYC3) e um na Amazon AWS (US-EAST-1). Em cada *datacenter* foi provisionado um *cluster* diferente (SFO1, NYC3 e US-EAST-1) formado por três servidores cada. A Figura 6.6 ilustra o Cenário D.

A configuração dos servidores provisionados na nuvem da Amazon AWS era composta por 1 processador, 613Mb de memória e 8Gb para armazenamento em disco. A configuração dos servidores na nuvem da DigitalOcean permaneceu igual a utilizada nos cenários

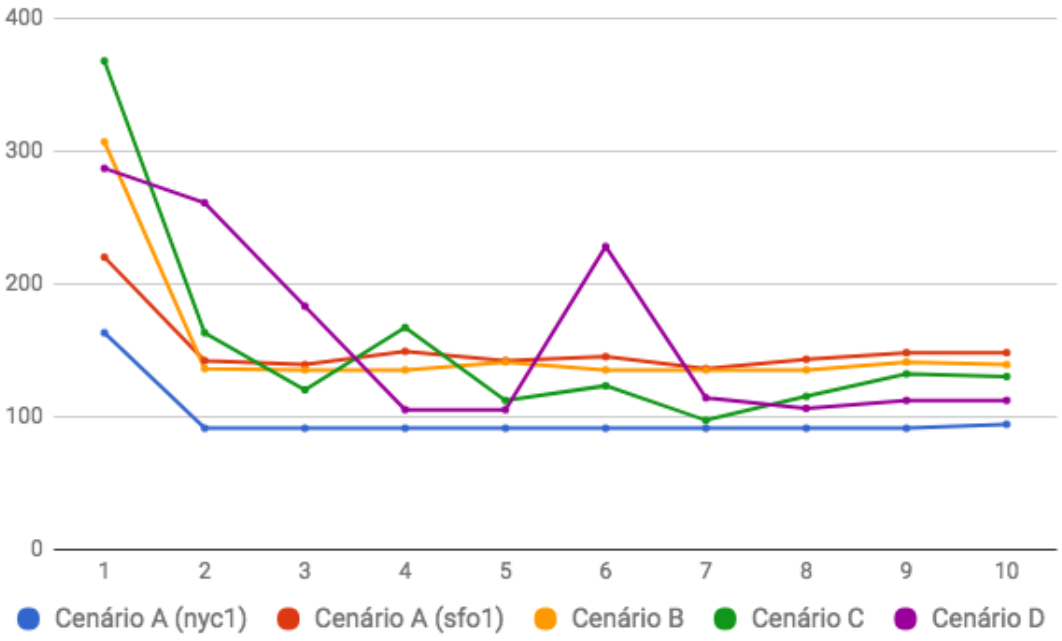

Figura 6.5: Comparação do Tempo Total de Execução do *Workflow* Testado.

anteriores (1 processador, 512Mb de memória e 20Gb de disco SSD). Além disso, a mesma implementação de *cache* no nível de *datacenter* adotadas nos Cenário B e C foi utilizada neste cenário. Além disso os *jobs* foram escalonados seguindo a mesma estratégia adotada nos testes do Cenário C. O objetivo deste teste é estudar a execução do *workflow* escalonado em uma federação de nuvens.

6.5.1 Resultados do Cenário D

O resultado consolidado das execuções do *workflow* testado neste teste podem ser vistas na Tabela 6.8. Além disso, a Figura 6.5 exibe um gráfico com o tempo total de cada execução do *workflow* em cada um dos cenários propostos neste capítulo. A comparação e análise deste cenário e do Cenário C são realizadas na próxima seção.

6.6 Análise dos Cenários C e D

Por meio dos resultados apresentados nas Tabelas 6.7 e 6.8 e, em conjunto com a Figura 6.5, nota-se que, ao se comparar os testes nos Cenários C e D, o tempo mínimo de execução do *workflow* testado foi menor no Cenário C. Contudo, esse comportamento só foi verificado na sétima execução do *workflow*, quanto (1) a imagem de *container* de aplicações já estava em *cache* em todos os servidores de todos os *clusters* e, como pode ser visto na Tabela 6.9, (2) porque cinco dos seis *jobs* do *workflow* testado foram escalonados em

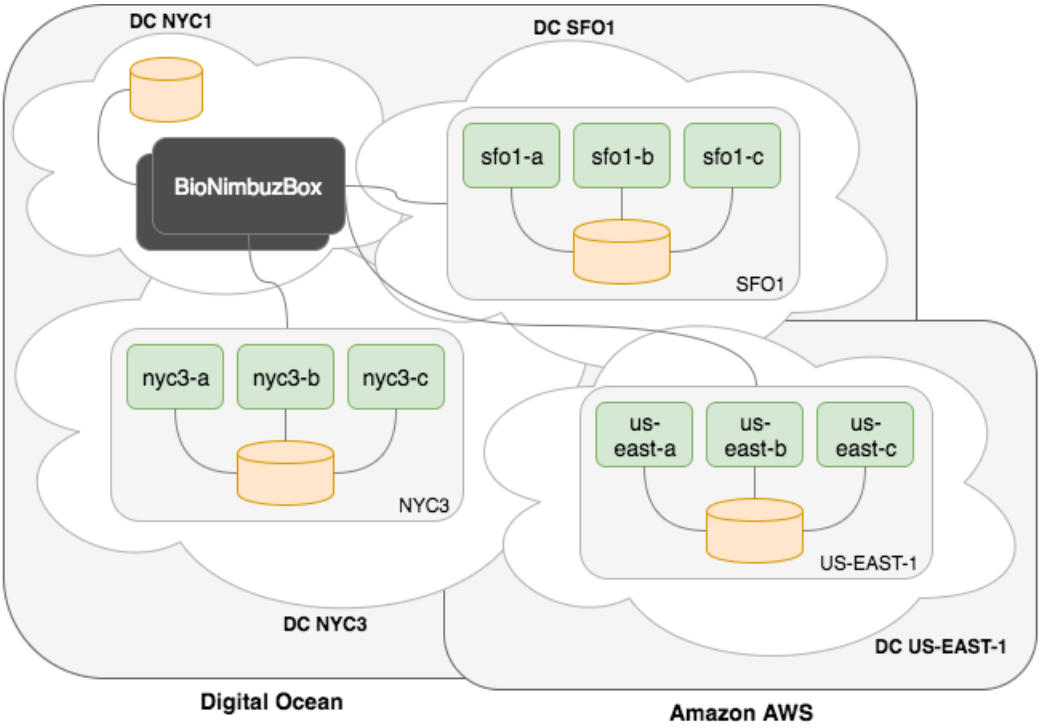

Figura 6.6: Infraestrutura do Cenário D de Teste.

servidores do *cluster* NYC3 (que é o *datacenter* mais próximo ao *datacenter* de execução do Serviço de Armazenamento).

Além disso, um ponto que chama a atenção na Figura 6.5 é em relação à sexta execução do *workflow* no Cenário D. Nesta execução específica, ao se comparar a tendência das execuções anteriores, ocorre um aumento considerável ao tempo transcorrido para o *workflow* testado chegar ao fim. O entendimento desse comportamento passa pela análise das alocações dos *jobs* nas execuções anteriores e do tempo de execução de cada *job*. A Tabela 6.10 exibe todas as alocações e os tempos de execução de todos os *jobs* para todas as execuções do *workflow* testado no Cenário D. A partir dessa tabela é possível concluir que, na sexta execução, o *job* 3-tophat2-b foi escalonado no servidor *us-east-b* e que, antes disso, o mesmo servidor não havia recebido nenhum outro *job*. Desse modo, não havia em *cache* a imagem do *container* para aplicações. Além disso, o tempo de execução (incluindo o tempo para transferir a imagem do *container*) do *job* 3-tophat2-b na sexta execução do *workflow* testado foi o maior dentre todas as execuções da aplicação tophat no Cenário D.

Ainda sob a análise da Figura 6.5, é possível perceber que de acordo com que as imagens de *containers* vão sendo armazenadas em *cache* (quarta execução em diante), os testes nos cenários que possuem uma quantidade maior de servidores próximos ao *datacenter* responsável pelo Serviço de Armazenamento do BioNimbuzBox começam a apresentar tempos de execução menores. Assim, a partir da oitava execução, a infraestrutura no Ce-

Tabela 6.8: Execução em uma Federação composta pelos *Clusters* sfo1, nyc3 e us-east-1.

Job	**Download** (segundos)			**Execução** (segundos)			**Upload** (segundos)			**Total** (segundos)		
	Min.	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**	**Min.**	**Max.**	**Med.**
1-prepare	0	0	0	6	12	8,8	6	21	11,7	12	30	21
2-bowtie-build	6	9	7,5	6	88	20,7	6	9	7,2	19	104	36,2
3-tophat2-a	6	9	7,2	17	85	27,5	6	9	6,9	31	102	42,4
3-tophat2-b	6	13	7,7	15	137	42,8	6	13	7,3	28	150	58,6
4-cufflinks-a	6	17	9,1	6	54	16,9	6	9	7,8	19	80	34,2
4-cufflinks-b	6	9	7,2	6	80	14,6	6	9	7,2	19	93	29,6
Workflow Total										**105**	**287**	**161,3**

Tabela 6.9: Alocação de *Jobs* na 7ª Execução do *Workflow* Testado no Cenário C

Job	**Alocação**	**Download** (segundos)	**Execução** (segundos)	**Upload** (segundos)	**Total** (segundos)
1-prepare	**nyc3-1**	0	6	9	15
2-bowtie-build	**nyc3-3**	6	6	6	18
3-tophat2-a	sfo1-1	9	17	9	35
3-tophat2-b	**nyc3-1**	6	15	6	28
4-cufflinks-a	**nyc3-2**	6	6	9	22
4-cufflinks-b	**nyc3-1**	6	6	6	19
Workflow Total					**97**

nário D, passou a apresentar melhor resultado do que a dos Cenários C e B, e até mesmo do Cenário A (quando executado no *datacenter* SFO1).

Por fim, a Tabela 6.10 exibe, em termos absolutos e em percentuais, o *overhead* na execução do *workflow* testado no BioNimbuzBox. Considerando os tempos de sincronização de dados (entrada e saída), de *download* da imagem do *container* de aplicações e de execução do *container*, o BioNimbuzBox, em termos absolutos, acrescentou aproximadamente 6 segundos ao tempo total de execução do *workflow* testado (sendo o acréscimo mínimo de 3 segundos, e máximo de 8 segundos). Esse mesmo resultado também foi obtido nos Cenário A, B e C.

6.7 Considerações Finais

Desde o Cenário A foi perceptível a atuação do *cache* de imagens de *containers* na otimização do tempo de execução total do *workflow* de referência. Além disso, o local de

Tabela 6.10: Alocação de *Jobs* do *Workflow* Testado no Cenário D.

#	Job	Aloc.	Down.	Exec.	Up.	Total
1	1-prepare	us-east-c	0	10	6	17
1	2-bowtie-build	sfo1-c	9	62	9	81
1	3-tophat2-a	nyc3-c	9	85	6	102
1	3-tophat2-b	sfo1-a	13	62	13	89
1	4-cufflinks-a	sfo1-b	17	54	9	80
1	4-cufflinks-b	sfo1-c	9	9	9	27
1	**Workflow Total**					**280**
1	**Workflow Total (BioNimbuzBox)**					**287** | **7** | **2,50%**
2	1-prepare	sfo1-b	0	9	17	26
2	2-bowtie-build	us-east-a	9	88	6	104
2	3-tophat2-a	sfo1-b	9	17	9	35
2	3-tophat2-b	sfo1-c	9	17	9	35
2	4-cufflinks-a	sfo1-a	9	9	9	27
2	4-cufflinks-b	us-east-c	6	80	6	93
2	**Workflow Total**					**258**
2	**Workflow Total (BioNimbuzBox)**					**261** | **3** | **1,16%**
3	1-prepare	nyc3-c	0	12	6	19
3	2-bowtie-build	us-east-a	6	6	6	19
3	3-tophat2-a	us-east-c	6	25	6	38
3	3-tophat2-b	nyc3-a	10	96	6	113
3	4-cufflinks-a	nyc3-b	11	52	6	70
3	4-cufflinks-b	sfo1-b	9	9	9	27
3	**Workflow Total**					**178**
3	**Workflow Total (BioNimbuzBox)**					**183** | **5** | **2,81%**
4	1-prepare	nyc3-c	0	6	6	12
4	2-bowtie-build	sfo1-a	9	9	9	28
4	3-tophat2-a	nyc3-a	6	19	6	31
4	3-tophat2-b	sfo1-b	9	17	9	35
4	4-cufflinks-a	sfo1-a	9	9	9	27
4	4-cufflinks-b	us-east-a	6	6	6	19
4	**Workflow Total**					**98**
4	**Workflow Total (BioNimbuzBox)**					**105** | **7** | **7,14%**
5	1-prepare	nyc3-a	0	9	9	19
5	2-bowtie-build	us-east-c	6	6	6	19
5	3-tophat2-a	sfo1-b	9	17	9	35
5	3-tophat2-b	us-east-a	6	22	6	35
5	4-cufflinks-a	sfo1-c	9	9	9	27
5	4-cufflinks-b	sfo1-b	9	9	9	27
5	**Workflow Total**					**100**
5	**Workflow Total (BioNimbuzBox)**					**105** | **5** | **5,00%**
Todos os valores expressos em segundos						

#	Job	Aloc.	Down.	Exec.	Up.	Total
6	1-prepare	sfo1-b	0	9	17	26
6	2-bowtie-build	us-east-c	6	6	6	19
6	3-tophat2-a	us-east-c	6	22	6	35
6	3-tophat2-b	us-east-b	6	137	6	150
6	4-cufflinks-a	sfo1-a	9	9	9	27
6	4-cufflinks-b	sfo1-b	9	9	9	27
6	**Workflow Total**					**222**
6	**Workflow Total (BioNimbuzBox)**					**228** | **6** | **2,70%**
7	1-prepare	us-east-b	0	9	9	19
7	2-bowtie-build	sfo1-a	9	9	9	27
7	3-tophat2-a	us-east-c	6	22	6	35
7	3-tophat2-b	nyc3-c	6	22	6	35
7	4-cufflinks-a	sfo1-b	9	9	9	27
7	4-cufflinks-b	nyc3-a	6	6	6	19
7	**Workflow Total**					**108**
7	**Workflow Total (BioNimbuzBox)**					**114** | **6** | **5,56%**
8	1-prepare	us-east-c	0	6	9	16
8	2-bowtie-build	sfo1-a	9	9	9	27
8	3-tophat2-a	us-east-b	6	22	6	36
8	3-tophat2-b	nyc3-c	6	15	6	28
8	4-cufflinks-a	us-east-c	6	6	6	19
8	4-cufflinks-b	nyc3-a	6	6	6	19
8	**Workflow Total**					**98**
8	**Workflow Total (BioNimbuzBox)**					**106** | **8** | **8,16%**
9	1-prepare	sfo1-a	0	9	21	30
9	2-bowtie-build	nyc3-c	6	6	6	19
9	3-tophat2-a	sfo1-b	9	17	9	35
9	3-tophat2-b	us-east-b	6	25	6	38
9	4-cufflinks-a	nyc3-c	6	6	6	19
9	4-cufflinks-b	nyc3-a	6	6	6	19
9	**Workflow Total**					**106**
9	**Workflow Total (BioNimbuzBox)**					**112** | **6** | **5,66%**
10	1-prepare	sfo1-b	0	9	17	26
10	2-bowtie-build	us-east-c	6	6	6	19
10	3-tophat2-a	us-east-b	6	29	6	42
10	3-tophat2-b	nyc3-a	6	15	6	28
10	4-cufflinks-a	nyc3-c	6	6	6	19
10	4-cufflinks-b	us-east-c	6	6	6	19
10	**Workflow Total**					**106**
10	**Workflow Total (BioNimbuzBox)**					**112** | **6** | **5,66%**
Todos os valores expressos em segundos						

origem das informações que serão processadas pelos *jobs*, bem como o local utilizado para a sincronização dos dados, contribuíram para aumentar ou diminuir o tempo de execução total do *workflow* testado em todos os cenários.

Outro ponto observado nos testes deste capítulo foi como o escalonamento aleatório de *jobs*, em um cenário de nuvens federadas, pode interferir no tempo total de execução

de um *workflow*.

Neste capítulo também foi observado que o *overhead* acrescentado pelo BioNimbuzBox foi inferior a 8 segundos, mesmo considerando as variações máximas e mínimas de execução do *workflow* W1.

No próximo capítulo serão apresentadas a conclusão deste trabalho e os trabalhos futuros.

Capítulo 7

Conclusão e Trabalhos Futuros

A execução de *workflows* de Bioinformática traz uma série de desafios. Dentre eles, a preparação do ambiente computacional adequado é uma atividade que demanda bastante tempo e conhecimento. Além disso, a ausência de recursos computacionais apropriados torna-se uma barreira adicional no processo de reprodução de experimentos científicos. Dessa forma, este trabalho propôs uma arquitetura baseada em *containers* e entregue sob o modelo de *Software* como Serviço – *SaaS*, capaz de abstrair as dificuldades impostas no processo de preparação do ambiente computacional adequado e, para a execução de *workflows* de Bioinformática utilizando esse ambiente.

Nesse contexto, o uso da tecnologia de *containers* auxiliou no processo de instalação e distribuição das aplicações necessárias a execução de *workflows* de Bioinformática. Ademais, o uso de *containers* para a execução de aplicações tem se mostrado uma alternativa flexível [8], e com ganho de performance [23] em comparação com o modelo de máquinas virtuais.

No que tange a disponibilidade de recursos computacionais, o uso de diferentes provedores de nuvem, coordenados sob um modelo de federação, apresentou-se como uma escolha acertada no provisionamento do ambiente computacional necessário às execuções de *workflows*. Nesse sentido, a arquitetura do BioNimbuzBox, ao empregar interfaces de provisionamento e de orquestração, possibilitou tanto a utilização de recursos entregues sob o modelo de Infraestrutura como Serviço – *IaaS*, quanto sob o modelo de Plataforma como Serviço – *PaaS*.

Diante do exposto, a arquitetura do BioNimbuzBox proposta neste trabalho, demonstrou, através de execuções reais em diferentes cenários, ser capaz de gerenciar recursos computacionais entregues por diferentes provedores de nuvem (sob modelos distintos), e coordenar a distribuição, a instalação e a execução de aplicações necessárias ao processamento de *workflows* de Bioinformática.

Como trabalho futuro, propõe-se a inclusão de um serviço de elasticidade que, em

conjunto com o serviço de provisionamento e de escalonamento, seja capaz de empregar um modelo de predição que considere (1) a disponibilidade dos recursos computacionais e seu custo; e (2) o tempo de execução dos diferentes *jobs* pendentes de escalonamento; na decisão de provisionamento automático de recursos.

Referências

[1] Abdelbaky, M., Diaz-Montes, J., Parashar, M., Unuvar, M., and Steinder, M. Docker containers across multiple clouds and data centers. In *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)* (Dec 2015), pp. 368–371. 36, 52, 53, 56

[2] Amazon Web Services LLC. Amazon Elastic Compute Cloud (EC2). Disponível em `http://aws.amazon.com/pt/ec2/`, 2012. Acessado em 05 de junho de 2015. 33

[3] Amazon Web Services LLC. Amazon Simple Storage Service (Amazon S3). `http://aws.amazon.com/s3/`, 2015. Acessado em 29 de maio de 2015. 54

[4] Amazon Web Services LLC. Amazon EC2 service limits. Disponível em `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-resource-limits.html`, 2016. Acessado em 30 de julho de 2016. 12

[5] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. A view of cloud computing. *Communications of the ACM 53*, 4 (2010), 50–58. 6, 8, 10, 11, 12, 15, 36

[6] Arun Chandrasekaran, Andrew Lerner, D. S. Containers will change your data center infrastructure and operations strategy. Disponível em `https://www.gartner.com/doc/3267118`, 2016. Acessado em 30 de julho de 2016. 30

[7] Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M. H., and Vahi, K. Characterization of scientific workflows. In *2008 Third Workshop on Workflows in Support of Large-Scale Science* (Nov 2008), pp. 1–10. xi, 13, 15

[8] Boettiger, C. An introduction to docker for reproducible research. *SIGOPS Oper. Syst. Rev. 49*, 1 (Jan. 2015), 71–79. 2, 36, 37, 73

[9] Bottoni, P., Gabrielli, E., Gualandi, G., Mancini, L. V., and Stolfi, F. *FedUp! Cloud Federation as a Service*. Springer International Publishing, Cham, 2016, pp. 168–182. 52, 53, 54, 56

[10] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems 25*, 6 (2009), 599–616. 6

[11] Canonical Ltd. Linux containers. Disponível em `https://linuxcontainers.org/`, 2016. Acessado em 30 de julho de 2016. 25, 28

[12] Choy, S., Wong, B., Simon, G., and Rosenberg, C. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games* (Piscataway, NJ, USA, 2012), NetGames '12, IEEE Press, pp. 2:1–2:6. 40

[13] Collberg, C., Proebsting, T., Moraila, G., Shankaran, A., Shi, Z., and Warren, A. M. Measuring reproducibility in computer systems research. *Department of Computer Science, University of Arizona, Tech. Rep* (2014). 37

[14] Corbet, J. Seccomp and sandboxing. *LWN. net, May* (2009). 25

[15] CoreOS, Inc. CoreOS. Disponível em `https://coreos.com/`, 2016. Acessado em 07 de setembro de 2016. 30, 47

[16] Costa, H. H. d. P. M. Controle de acesso na plataforma de nuvem federada BioNimbuZ, 2015. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 16

[17] Crockford, D. JSON. Disponível em `http://json.org/`, 2012. Acessado em 05 de junho de 2015. 17

[18] Di Tommaso, P., Palumbo, E., Chatzou, M., Prieto, P., Heuer, M. L., and Notredame, C. The impact of docker containers on the performance of genomic pipelines. *PeerJ 3* (2015), e1273. 1, 2, 24, 37

[19] Diaz-Montes, J., AbdelBaky, M., Zou, M., and Parashar, M. Cometcloud: Enabling software-defined federations for end-to-end application workflows. *IEEE Internet Computing 19*, 1 (Jan 2015), 69–73. 52

[20] Docker, Inc. Docker Swarm. Disponível em `https://docs.docker.com/swarm/overview/`, 2016. Acessado em 20 de agosto de 2016. 30, 35, 67

[21] Docker, Inc. What is docker. Disponível em `https://www.docker.com/what-docker`, 2016. Acessado em 30 de julho de 2016. xi, 2, 24, 25, 26, 29

[22] Docker, Inc. Docker compose. Disponível em `https://www.docker.com/products/docker-machine`, 2017. Acessado em 20 de maio de 2017. 31

[23] Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (March 2015), pp. 171–172. 24, 37, 73

[24] Ferry, N., Rossini, A., Chauvel, F., Morin, B., and Solberg, A. Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on* (2013), IEEE, pp. 887–894. 1, 11

[25] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext Transfer Protocol–http/1.1, Request For Coments (RFC 2616). Disponível em `http://tools.ietf.org/pdf/rfc2616.pdf`, 1999. Acessado em 05 de junho de 2015. 17

[26] Gartner, Inc. Gartner says cloud computing will become the bulk of new IT spend by 2016. Disponível em http://www.gartner.com/newsroom/id/2613015, 2013. Acessado em 30 de julho de 2016. 1

[27] Gartner, Inc. Gartner says worldwide cloud infrastructure-as-a-service spending to grow 32.8 percent in 2015. Disponível em http://www.gartner.com/newsroom/id/3055225, 2015. Acessado em 30 de julho de 2016. 1

[28] Gartner, Inc. Gartner says by 2020, a corporate "no-cloud"policy will be as rare as a "no-internet"policy is today. Disponível em http://www.gartner.com/newsroom/id/3354117, 2016. Acessado em 20 de maio de 2017. 1

[29] Gartner, Inc. Gartner says by 2020 "cloud shift"will affect more than $1 trillion in IT spending. Disponível em http://www.gartner.com/newsroom/id/3384720, 2016. Acessado em 20 de maio de 2017. 1

[30] Gartner, Inc. IT glossary – hybrid cloud computing. Disponível em http://www.gartner.com/it-glossary/hybrid-cloud-computing/, 2017. Acessado em 25 de maio de 2017. 1

[31] Gerlach, W., Tang, W., Keegan, K., Harrison, T., Wilke, A., Bischof, J., D'Souza, M., Devoid, S., Murphy-Olson, D., Desai, N., et al. Skyport: container-based execution environment management for multi-cloud scientific workflows. In *Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds* (2014), IEEE Press, pp. 25–32. xi, 36, 52, 54, 55, 56

[32] Global Inter-Cloud Technology Forum. Use cases and functional requirements for inter-cloud computing. Tech. rep., GICTF, 2010. 11

[33] Google CloudPlataform. Quotas - app engine — google cloud platform. Disponível em https://cloud.google.com/appengine/docs/quotas, 2016. Acessado em 30 de julho de 2016. 12

[34] Google Inc. Google Kubernetes. Disponível em http://kubernetes.io/, 2016. Acessado em 20 de agosto de 2016. 30, 32, 35

[35] Gormley, C., and Tong, Z. *Elasticsearch: The Definitive Guide*. "O'Reilly Media, Inc.", 2015. 43

[36] Govil, S. B., Thyagarajan, K., Srinivasan, K., Chaurasiya, V. K., and Das, S. An approach to identify the optimal cloud in cloud federation. *International Journal of Cloud Computing and Services Science 1*, 1 (2012), 35. 12

[37] Grozev, N., and Buyya, R. Inter-cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience 44*, 3 (2014), 369–390. 11

[38] Han, J., E, H., Le, G., and Du, J. Survey on nosql database. In *2011 6th International Conference on Pervasive Computing and Applications* (Oct 2011), pp. 363–366. 43

[39] Hashicorp . Nomad by Hashicorp. Disponível em `https://www.nomadproject.io/`, 2016. Acessado em 30 de julho de 2016. 30, 34, 35

[40] Hildred, T. The history of containers. Disponível em `http://rhelblog.redhat.com/2015/08/28/the-history-of-containers/`, 2016. Acessado em 20 de agosto de 2016. 25

[41] Jones, M., Bradley, J., and Sakimura, N. Json Web Token (JWT). Tech. rep., RFC 7519, 2015. xi, 42, 43, 44

[42] Joy, A. M. Performance comparison between linux containers and virtual machines. In *2015 International Conference on Advances in Computer Engineering and Applications* (March 2015), pp. 342–346. 24, 37

[43] Júnior, B., and de Oliveira, W. Escalonador de tarefas para o plataforma de nuvens federadas BioNimbuZ usando beam search iterativo multiobjetivo, 2016. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 16

[44] Kim, D., Pertea, G., Trapnell, C., Pimentel, H., Kelley, R., and Salzberg, S. L. Tophat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biology 14*, 4 (2013), R36. 58, 60

[45] Kim, H., and Parashar, M. Cometcloud: An autonomic cloud engine. *Cloud Computing: Principles and Paradigms* (2011), 275–297. 52

[46] Kurze, T., Klems, M., Bermbach, D., Lenk, A., Tai, S., and Kunze, M. Cloud federation. *Cloud Computing 2011* (2011), 32–38. 11

[47] Kwei-Jay, L., and Gannon, J. Atomic Remote Procedure Call. *Software Engineering, IEEE Transactions on SE-11*, 10 (1985), 1126–1135. 17

[48] Langmead, B., and Salzberg, S. L. Fast gapped-read alignment with bowtie 2. *Nat Meth 9*, 4 (04 2012), 357–359. 58, 60

[49] Lima, D., Moura, B., Ribeiro, E. ; Aráujo, A. P. F., Walter, M. E., Holanda, M. T., and Oliveira, G. A Storage Policy for a Hybrid Federated Cloud Platform executing Bioinformatics Applications. *C4BIE 2014: Cloud for Business, Industry and Enterprises* (2014). Proceedings of the 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2014). 15, 16, 17, 18, 56

[50] Linux Foundation. Open container initiative. Disponível em `https://www.opencontainers.org/`, 2016. Acessado em 30 de julho de 2016. 28, 29

[51] Luscombe, N. M., Greenbaum, D., Gerstein, M., et al. What is bioinformatics? a proposed definition and overview of the field. *Methods of information in medicine 40*, 4 (2001), 346–358. 12, 13

[52] Mell, P., and Grance, T. The NIST definition of cloud computing. *Communications of the ACM 53*, 6 (2010), 50. 1, 6, 7, 8, 9

[53] Menage, P. B. Adding generic process containers to the linux kernel. In *Proceedings of the Linux Symposium* (2007), vol. 2, Citeseer, pp. 45–57. 25

[54] Microsoft. Microsoft azure subscription and service limits, quotas, and constraints. Disponível em `https://azure.microsoft.com/en-us/documentation/articles/azure-subscription-service-limits/`, 2016. Acessado em 30 de julho de 2016. 12

[55] National Library of Medicine (US). Genetics home reference. help me understand genetics. `https://ghr.nlm.nih.gov/primer`, 2015. Acessado em 04 de junho de 2017. 14

[56] Newman, S. *Building microservices*. "O'Reilly Media, Inc.", 2015. 38

[57] Paula, R. d. Proveniência de dados em workflows de bioinformática. *Universidade de Brasília-UnB* (2013). xi, 13, 14

[58] Raicu, I., Foster, I. T., and Zhao, Y. Many-task computing for grids and supercomputers. In *2008 Workshop on Many-Task Computing on Grids and Supercomputers* (Nov 2008), pp. 1–11. 13

[59] Ramos, V. d. A. Um sistema gerenciador de workflows científicos para a plataforma de nuvens federadas BioNimbuZ, 2016. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 16

[60] Red Hat, Inc. What are containers? Disponível em `https://www.redhat.com/en/insights/containers`, 2016. Acessado em 20 de agosto de 2016. 24, 25

[61] Rosa, M., Moura, B., Vergara, G., Santos, L., Ribeiro, E., Holanda, M., Walter, M. E., and Araujo, A. BioNimbuZ: A federated cloud platform for bioinformatics applications. *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) 00* (2016), 548–555. 2, 16, 36, 56

[62] Saldanha, H., Araújo, A., Borges, C., Ribeiro, E., Setubal, J. C., Walter, M. E., Holanda, M., Gallon, R., and Togawa, R. *Towards a hybrid federated cloud platform to efficiently execute bioinformatics workflows*. INTECH Open Access Publisher, 2012. 16, 17, 56

[63] Siriwardena, P. Mutual authentication with TLS. In *Advanced API Security*. Springer, 2014, pp. 47–58. 31

[64] Steen, M. V., and Tanenbaum, A. *Distributed Systems: Principles and Paradigms*. Prentice Hall; 2 edition, 2006. 17

[65] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, F., Dabek, F., and Balakrishnan, H. Chord: a scalable peer-to-peer lookup protocol for internet applications. *Networking, IEEE/ACM Transactions on 11*, 1 (2003), 17–32. 16

[66] Tang, W., Wilkening, J., Desai, N., Gerlach, W., Wilke, A., and Meyer, F. A scalable data analysis platform for metagenomics. In *2013 IEEE International Conference on Big Data* (Oct 2013), pp. 21–26. 54

[67] The Apache Software Foundation. Apache Zookeeper. Disponível em `http://zookeeper.apache.org/`, 2010. Acessado em 05 de junho de 2015. xi, 17, 18, 34

[68] The Apache Software Foundation. Apache Avro. Disponível em `http://avro.apache.org/`, 2012. Acessado em 05 de junho de 2015. 17

[69] The Apache Software Foundation. Apache Foundation. Disponível em `http://apache.org/`, 2015. Acessado em 30 de março de 2015. 17

[70] The Apache Software Foundation. Apache Mesos. Disponível em `http://mesos.apache.org`, 2016. Acessado em 30 de julho de 2016. 30, 33, 35

[71] Tost, A., and De Jesús, J. Escalabilidade e elasticidade para padrões de aplicativo virtual no IBM PureApplication System. Disponível em `http://www.ibm.com/developerworks/br/websphere/techjournal/1309_tost/`, 2013. Acessado em 20 de agosto de 2016. 7

[72] Trapnell, C., Roberts, A., Goff, L., Pertea, G., Kim, D., Kelley, D. R., Pimentel, H., Salzberg, S. L., Rinn, J. L., and Pachter, L. Differential gene and transcript expression analysis of rna-seq experiments with tophat and cufflinks. *Nat. Protocols 7*, 3 (03 2012), 562–578. 58, 60

[73] Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review 39*, 1 (2008), 50–55. 6, 7

[74] Vergara, G. F., Moura, B., and Araújo, A. Elasticidade em uma infraestrutura de nuvens federadas BioNimbuZ, 2014. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 16

[75] Villegas, D., Bobroff, N., Rodero, I., Delgado, J., Liu, Y., Devarakonda, A., Fong, L., Sadjadi, S. M., and Parashar, M. Cloud federation in a layered service model. *Journal of Computer and System Sciences 78*, 5 (2012), 1330–1344. 11

[76] Wang, L. *Directed Acyclic Graph*. Springer New York, New York, NY, 2013, pp. 574–574. 48

[77] Watson, R. Orchestrating docker container-based cloud and microservices applications. Disponível em `https://www.gartner.com/doc/3287319`, 2016. Acessado em 20 de agosto de 2016. 30

[78] Zhang, Q., Cheng, L., and Boutaba, R. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications 1*, 1 (2010), 7–18. 10

[79] Zheng, C., and Thain, D. Integrating containers into workflows: A case study using makeflow, work queue, and docker. In *Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing* (New York, NY, USA, 2015), VTDC '15, ACM, pp. 31–38. 2, 36, 37