

Isabella de Albuquerque Ceravolo

# **O2CMF: Um *Framework* para Experimentação Federada em NFV**

Vitória – ES

2019



Isabella de Albuquerque Ceravolo

## **O2CMF: Um *Framework* para Experimentação Federada em NFV**

Dissertação de mestrado submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do título de Mestre em Informática. Área de concentração: Ciência da Computação.

Universidade Federal do Espírito Santo – UFES

Departamento de Informática

Programa de Pós-Graduação em Informática

Orientador: Magnos Martinello

Vitória – ES

2019





# Agradecimentos

Agradeço a Deus por me sustentar e capacitar para a realização deste mestrado.  
Agradeço aos meus pais por todo carinho e incentivo.  
Agradeço à CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pela concessão da bolsa, viabilizando meus estudos.  
Agradeço ao professor orientador e à equipe do NERDS (Núcleo de Estudos em Redes Definidas por Software) pela colaboração.  
Agradeço à ACM (*Association for Computing Machinery*) por contribuir para minha formação através da concessão do *travel grant* para participação no SIGCOMM.



*“A ti, ó Deus, glorificamos, a ti damos louvor,  
pois o teu nome está perto, as tuas maravilhas o declaram.”  
(Bíblia Sagrada, Salmos 75:1)*



# Resumo

*Testbeds* federados ocupam um papel importante no desenvolvimento de inovações em redes, fornecendo aos pesquisadores um laboratório distribuído para a realização de provas de conceito. Isso é possível através de *frameworks* que transformam recursos físicos em um serviço de experimentação. Contudo, para que um *testbed* continue adequado aos objetivos da comunidade de pesquisa, é necessário evoluir seu serviço de experimentação, incorporando tecnologias emergentes. Uma dessas tecnologias é a virtualização de funções de rede (*Network Functions Virtualization* – NFV), que possibilita que funções de rede tradicionalmente ligadas a dispositivos de hardware sejam executadas na infraestrutura de computação em nuvem. Embora *frameworks* (como o GCF, OCF e OMF) tenham contribuído fortemente para o estabelecimento de federações de *testbeds* de redes, eles não apresentam as características necessárias para suportar NFV. Isso se deve ao emprego de virtualização simples, monitoramento insuficiente e ausência de recursos no catálogo de serviços que possibilitem a construção funções de rede virtuais, além da carência de funcionalidades para sua orquestração. Portanto, esse trabalho propõe um novo *framework* destinado a habilitar a experimentação em NFV. O resultado foi o O2CMF, um *framework* baseado na plataforma de computação em nuvem OpenStack, interoperável com a infraestrutura do Fed4FIRE. Para validar o O2CMF, são apresentadas demonstrações das funcionalidades de gestão do *testbed*, compatibilidade com o Fed4FIRE, isolamento de tráfego, orquestração de NFV e integração com outros domínios (robótica, redes sem fio e OpenFlow). Através dessas provas de conceito, demonstramos que o O2CMF foi capaz de habilitar a experimentação federada em NFV, combinando a interoperabilidade provida por SFA com a flexibilidade e robustez da nuvem, e provendo mecanismos de orquestração de funções de rede virtuais. O O2CMF foi utilizado na implantação de um *testbed* na UFES, através do qual tem apoiado o desenvolvimento de atividades de pesquisa e educação em redes. Além disso, sua documentação de operação e tutoriais de uso motivaram sua adoção na implantação de um *testbed* na Universidade de Bristol.

**Palavras-chave:** Fed4FIRE, OpenStack, NFV.



# Abstract

Federated testbeds support network research by providing a distributed lab. This is possible through frameworks that transform physical resources in an experimentation service. Such service needs to continuously evolve including emerging technologies. Network Function Virtualization (NFV) is an emerging technology which enables to decouple network intelligence from physical hardware. Although frameworks (such as GCF, OCF, and OMF) have strongly contributed to the establishment of network testbeds' federations, they lack features required to support NFV. This is due to the type of virtualization, monitoring, and resources which they offer. Besides that, they lack NFV orchestration functionalities. This work proposes a new framework to enable NFV experimentation. The result was O2CMF, a framework based on the OpenStack cloud computing platform and interoperable with the Fed4FIRE infrastructure. To validate O2CMF, we developed demonstrations showing testbed management, Fed4FIRE compatibility, traffic isolation, NFV orchestration and integration with other domains (robotics, wireless networks, and OpenFlow). These proofs of concept, we demonstrated that O2CMF successfully enabled federated experimentation in NFV, combining the interoperability provided by SFA with the flexibility and robustness of the cloud, and orchestration features. O2CMF have been used to manage a testbed at UFES, supporting research and education activities. In addition, its documentation, covering operation and use, motivated its adoption by the University of Bristol.

**Keywords:** Fed4FIRE, OpenStack, NFV.





# Lista de ilustrações

Figura 1 – Evolução das plataformas experimentais. Fonte: (VANDENBERGHE et al., 2013).	22
Figura 2 – Evolução proposta para 5G. Fonte: (REICHERT, 2018).	25
Figura 3 – Representação do ecossistema FUTEBOL. Fonte: (FUTEBOL, 2016a).	26
Figura 4 – Arquitetura do <i>framework</i> do FUTEBOL. Adaptado de (HAMMAD et al., 2016).	27
Figura 5 – Suíte de ferramentas jFed. Fonte: (FED4FIRE, 2012c).	28
Figura 6 – Relação entre <i>sliver</i> e <i>slice</i> . Fonte: (BECUE et al., 2015).	30
Figura 7 – Representação da arquitetura SFA. Adaptado de: (BECUE et al., 2015).	30
Figura 8 – Representação das funcionalidades da <i>Clearinghouse</i> . Fonte: (BECUE et al., 2015).	31
Figura 9 – Representação das interações do usuário com o AM. Fonte: (BECUE et al., 2015).	31
Figura 10 – Máquina de estados para um <i>sliver</i> . Fonte: (GENI, 2014b)	32
Figura 11 – Representação de um <i>testbed</i> controlado pelo OMF. Fonte: (FIBRE, 2018).	34
Figura 12 – Características da computação em nuvem. Fonte: (SAN ENTHUSIAST, 2016).	38
Figura 13 – Estrutura do OpenStack. Fonte: (AMORIM et al., 2018).	39
Figura 14 – Arquitetura de instalação usual do OpenStack. Fonte: (RED HAT, 2018a).	41
Figura 15 – Modelo usual de implantação do OpenStack. Fonte: (OPENSTACK, 2016).	41
Figura 16 – Interação com o OpenStack via API. Fonte: (FUGA CLOUD, 2018).	42
Figura 17 – Mudança de paradigma proposta por NFV. Fonte: (OSS LINE, 2017).	43
Figura 18 – Representação da arquitetura NFV. Adaptado de: (MIJUMBI et al., 2016b) e (MIJUMBI et al., 2016a).	44
Figura 19 – Arquitetura do Tacker. Adaptado de: (OPENSTACK, 2018m).	46
Figura 20 – Representação da estrutura de um <i>script</i> TOSCA descrevendo uma VNF. Fonte: (AGUIAR; TAVARES, 2017b).	47
Figura 21 – Casos de uso de NFV estabelecidos pelo FUTEBOL.	52
Figura 22 – Arquitetura do O2CMF.	54
Figura 23 – Pacotes que compõem o AM.	58
Figura 24 – Modelo entidade-relacionamento na primeira versão do O2CMF.	61
Figura 25 – Representação da topologia de rede do <i>testbed</i> . Adaptado de: (MIRAN-TIS, 2012).	62
Figura 26 – Orquestração de experimento.	66

Figura 27 – Modelo entidade-relacionamento na segunda versão do O2CMF. . . . .	67
Figura 28 – Modelo entidade-relacionamento na terceira versão do O2CMF. . . . .	69
Figura 29 – Representação do espaço inteligente. Adaptado de: (MARQUES et al., 2017). . . . .	71
Figura 30 – Modelo entidade-relacionamento na última versão do O2CMF. . . . .	76
Figura 31 – Menu de seleção de projeto. . . . .	80
Figura 32 – Painel de gestão de <i>flavors</i> . . . . .	80
Figura 33 – Formulário de criação de <i>flavor</i> . . . . .	81
Figura 34 – Painel exibindo os <i>flavors</i> criados. . . . .	82
Figura 35 – Formulário para definição de metadados do <i>flavor</i> <code>o2cmf.limit</code> . . . . .	83
Figura 36 – Painel de gestão de imagens. . . . .	84
Figura 37 – Painel de topologia de rede do OpenStack. . . . .	84
Figura 38 – Envio de tráfego da <code>vm3</code> para <code>vm1</code> . . . . .	85
Figura 39 – Envio de tráfego da <code>vm3</code> para <code>vm2</code> . . . . .	85
Figura 40 – Envio de tráfego da <code>vm3</code> para <code>vm1</code> e <code>vm2</code> . . . . .	86
Figura 41 – Tela de <i>login</i> do jFed. . . . .	87
Figura 42 – Especificação da topologia do experimento. . . . .	88
Figura 43 – Especificação da imagem a ser utilizada na VM. . . . .	88
Figura 44 – Especificação do <i>flavor</i> da VM. . . . .	89
Figura 45 – Configuração de rede da VM. . . . .	89
Figura 46 – Retorno da chamada ao método <i>ListResources</i> implementado pelo AM. . . . .	89
Figura 47 – Aba <b>RSpec Editor</b> exibindo resultado da definição do experimento. . . . .	90
Figura 48 – Retorno da chamada ao método <i>Create</i> implementado pela <i>Clearinghouse</i> . . . . .	90
Figura 49 – Retorno da chamada ao método <i>Get_Credentials</i> implementado pela <i>Clearinghouse</i> . . . . .	91
Figura 50 – Retorno da chamada ao método <i>Allocate</i> implementado pelo AM. . . . .	91
Figura 51 – Dados persistidos na tabela <code>slice</code> do banco de dados do AM. . . . .	92
Figura 52 – Dados persistidos na tabela <code>vm</code> do banco de dados do AM. . . . .	92
Figura 53 – Dados persistidos na tabela <code>net_iface</code> do banco de dados do AM. . . . .	92
Figura 54 – Retorno da chamada ao método <i>Create</i> implementado pela <i>Clearinghouse</i> . . . . .	93
Figura 55 – Retorno da chamada ao método <i>Provision</i> implementado pelo AM. . . . .	93
Figura 56 – VMs criadas no OpenStack. . . . .	94
Figura 57 – Topologia de rede criada no OpenStack. . . . .	95
Figura 58 – Retorno da chamada ao método <i>PerformOperacionalAction</i> implementado pelo AM. . . . .	95
Figura 59 – Retorno da chamada ao método <i>Status</i> implementado pelo AM. . . . .	96
Figura 60 – Retorno da chamada ao método <i>Describe</i> implementado pelo AM. . . . .	96
Figura 61 – Recursos do experimento prontos para acesso SSH. . . . .	97
Figura 62 – Teste de conectividade através da LAN. . . . .	97

Figura 63 – Teste de conectividade com a Internet. . . . .	97
Figura 64 – Retorno da chamada ao método <i>Delete</i> implementado pelo AM. . . . .	98
Figura 65 – Retorno da chamada ao método <i>Delete</i> implementado pela <i>Clearinghouse</i> . . . . .	98
Figura 66 – Topologia utilizada nos experimentos. . . . .	99
Figura 67 – Parâmetros de execução do teste. . . . .	100
Figura 68 – Resultado da medição da taxa <i>bytes</i> recebidos pela <i>vm2</i> de cada um dos experimentos. . . . .	100
Figura 69 – Repositório de imagens atualizado. . . . .	102
Figura 70 – Resultado da chamada ao método <i>ListResources</i> . . . . .	102
Figura 71 – Estado inicial da topologia de rede do <i>slice Advanced</i> . . . . .	103
Figura 72 – Criação da VNF <i>web_server</i> . . . . .	104
Figura 73 – Painel de gestão de VNFs. . . . .	104
Figura 74 – Painel de gestão de instâncias do <i>slice Advanced</i> . . . . .	105
Figura 75 – Topologia de rede do <i>slice Advanced</i> após criação da VNF. . . . .	105
Figura 76 – Uso de CPU da VDU1. . . . .	106
Figura 77 – Topologia de rede do <i>slice Advanced</i> após <i>scaling</i> da VNF. . . . .	107
Figura 78 – Dados inseridos na tabela <i>template</i> do banco de dados do AM. . . . .	108
Figura 79 – Dados inseridos na tabela <i>parameter</i> do banco de dados do AM. . . . .	108
Figura 80 – Resultado da chamada ao método <i>ListResources</i> . . . . .	109
Figura 81 – Lista de VNFs instanciadas no experimento <i>beginner</i> . . . . .	109
Figura 82 – VM de orquestração e VDU de <i>o2cmf-test-vnf</i> . . . . .	110
Figura 83 – Reserva dos recursos da nuvem e do espaço inteligente através do jFed. . . . .	111
Figura 84 – Experimento provisionado. . . . .	112
Figura 85 – Painel de orquestração dos recursos de robótica. . . . .	113
Figura 86 – Painel de orquestração dos recursos da nuvem. . . . .	113
Figura 87 – Painel de orquestração dos recursos de comunicação sem-fio. . . . .	114
Figura 88 – <i>Log</i> de chamadas realizadas pelo jFed. . . . .	114



# Lista de tabelas

Tabela 1 – Métodos da GENI API. . . . .	33
Tabela 2 – Requisitos do O2CMF. . . . .	54
Tabela 3 – Componentes do O2CMF. . . . .	55
Tabela 4 – Etapas de desenvolvimento do O2CMF. . . . .	56
Tabela 5 – Organização do repositório do O2CMF. . . . .	76
Tabela 6 – Testes de validação por etapa de desenvolvimento. . . . .	79



# Lista de abreviaturas e siglas

AM	Aggregate Manager
API	Application Programming Interface
CLI	Command-Line Interface
CMF	Control and Management Framework
ETSI	European Telecommunications Standard Institute
FUTEBOL	Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory
GENI	Global Environment for Network Innovations
IP	Internet Protocol
LAN	Local Area Network
NFV	Network Functions Virtualization
OCF	OFELIA Control Framework
OMF	Control and Management Framework
O2CMF	OpenStack-based Open Control and Management Framework
SDK	Software Development Kit
SFA	Slice-based Federation Architecture
SO	Sistema Operacional
SSH	Secure Shell
VM	Virtual Machine
VNF	Virtual Network Function





# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>21</b>
<b>1.1</b>	<b>Objetivos</b>	<b>23</b>
<b>1.2</b>	<b>Estrutura do Texto</b>	<b>23</b>
<b>2</b>	<b>TRABALHOS RELACIONADOS</b>	<b>25</b>
<b>2.1</b>	<b>FUTEBOL</b>	<b>25</b>
<b>2.2</b>	<b>Fed4FIRE</b>	<b>28</b>
<b>2.3</b>	<b><i>Frameworks</i> Existentes para Experimentação em Redes</b>	<b>29</b>
2.3.1	GENI Control Framework (GCF)	29
2.3.2	cOntrol and Management Framework (OMF)	33
2.3.3	OFELIA Control Framework (OCF)	34
<b>3</b>	<b>CONCEITOS FUNDAMENTAIS</b>	<b>37</b>
<b>3.1</b>	<b>Computação em Nuvem</b>	<b>37</b>
3.1.1	OpenStack	39
<b>3.2</b>	<b><i>Network Function Virtualization</i> (NFV)</b>	<b>43</b>
3.2.1	Tacker	45
<b>4</b>	<b>PROJETO E IMPLEMENTAÇÃO DO O2CMF</b>	<b>51</b>
<b>4.1</b>	<b>Levantamento de Requisitos</b>	<b>52</b>
<b>4.2</b>	<b>Projeto da Arquitetura</b>	<b>54</b>
<b>4.3</b>	<b>Processo de Implementação do O2CMF</b>	<b>56</b>
4.3.1	Integração da nuvem ao Fed4FIRE	57
4.3.2	Integração de NFV ao Fed4FIRE	63
4.3.3	Suporte à experimentadores iniciantes	67
4.3.4	Experimentação convergente com NFV	70
<b>4.4</b>	<b>Implantação e Testes</b>	<b>76</b>
<b>5</b>	<b>VALIDAÇÃO DO O2CMF</b>	<b>79</b>
<b>5.1</b>	<b>Definição de política para gestão do <i>testbed</i></b>	<b>79</b>
<b>5.2</b>	<b>Reserva e provisionamento</b>	<b>86</b>
<b>5.3</b>	<b>Isolamento entre experimentos</b>	<b>98</b>
<b>5.4</b>	<b>Criação de VNF através de <i>script</i> TOSCA</b>	<b>101</b>
<b>5.5</b>	<b>Criação de VNF através da RSpec</b>	<b>107</b>
<b>5.6</b>	<b>Reserva do espaço inteligente</b>	<b>110</b>
<b>6</b>	<b>CONCLUSÃO</b>	<b>117</b>

REFERÊNCIAS . . . . .	121
-----------------------	-----

APÊNDICES	127
-----------	-----

APÊNDICE A – <i>REQUEST</i> RSPEC CONTENDO DUAS VMS . .	129
---	-----

APÊNDICE B – <i>REQUEST</i> RSPEC REQUISITANDO UMA POR- ÇÃO DOS RECURSOS DA NUVEM E A VM DE ORQUESTRAÇÃO . . . . .	131
--	-----

APÊNDICE C – <i>SCRIPT</i> TOSCA DESCREVENDO SERVIDOR WEB . . . . .	133
--	-----

APÊNDICE D – MODELO DE VNF DISPONIBILIZADO NO CA- TÁLOGO . . . . .	135
---	-----

APÊNDICE E – <i>REQUEST</i> RSPEC PARA CRIAÇÃO DE VNF A PARTIR DE UM MODELO DO CATÁLOGO . . .	137
--	-----

APÊNDICE F – <i>REQUEST</i> RSPEC CONTENDO O ESPAÇO IN- TELIGENTE E SEUS SERVIÇOS NA NUVEM . .	139
---	-----

ANEXOS	141
--------	-----

ANEXO A – MODELO DE VNF CONTENDO POLÍTICAS DE <i>SCALING</i> E MONITORAMENTO . . . . .	143
---	-----

ANEXO B – MODELO DE VNF CONTENDO POLÍTICA DE MO- NITORAMENTO . . . . .	145
---	-----

# 1 Introdução

A construção de provas de conceito constitui uma importante etapa no desenvolvimento de inovações em redes, computação distribuída e computação em nuvem. No entanto, muitos pesquisadores encontravam dificuldades para a realização dessa etapa. Seja por falta de recursos ou pelo risco de interferir na operação da infraestrutura (e afetar outros usuários), muitos se viam limitados ao uso de simuladores. Através do desenvolvimento de federações de *testbeds* de Internet do futuro, pesquisadores encontraram um laboratório distribuído onde podem realizar seus experimentos. Esse ambiente compartilhado possibilita que soluções inovadoras sejam implantadas, testadas e validadas usando recursos reais e em larga escala (BERMAN et al., 2015).

Na construção de um *testbed*, é preciso ir além da implantação da infraestrutura bruta. Para atender às necessidades da comunidade de pesquisa, é necessário prover mecanismos para controle da alocação de recursos; gestão da infraestrutura; identificação de usuários e autorização; além de mecanismos que permitam aos usuários coordenarem seus experimentos (VANDENBERGHE et al., 2013). Esse trabalho é realizado pelo *Control and Management Framework* (CMF): um *framework* de software capaz de gerenciar os recursos físicos, comunicar-se com a federação e ainda prover os mecanismos citados. Dessa forma, o CMF é responsável por transformar recursos físicos em um serviço de experimentação. A partir dessa noção, é possível dizer que o uso bem sucedido de um *testbed* depende das ferramentas utilizadas na composição de seu serviço de experimentação.

Conforme ilustrado na Figura 1, com a evolução das tecnologias de rede e as mudanças no mercado, as federações precisam incluir novos recursos em seus *testbeds* e, conseqüentemente, atualizar ou desenvolver novas ferramentas para suportar a experimentação nesses novos recursos/tecnologias. A nova geração de tecnologias de telecomunicações (5G) tem gerado a demanda por atualização nas plataformas experimentais, uma vez que essa mudança de paradigma exigirá o desenvolvimento de inovações em todas as camadas da rede. Há a necessidade de evoluir para suportar uma quantidade massiva de dispositivos conectados, com exigências maiores de banda e mobilidade. Para habilitar essa transformação, a rede precisa oferecer maior programabilidade, habilitando a gestão dinâmica e o uso eficiente dos recursos (ANDREWS et al., 2014).

Um dos habilitadores da nova geração de redes é a arquitetura de virtualização de funções de rede (*Network Function Virtualization* - NFV). Essa arquitetura possibilita que funções de rede tradicionalmente ligadas a dispositivos de hardware sejam executadas na infraestrutura de computação em nuvem. Isso traz como benefício a capacidade de suportar elasticamente as demandas da rede; agrega agilidade na implantação de serviços

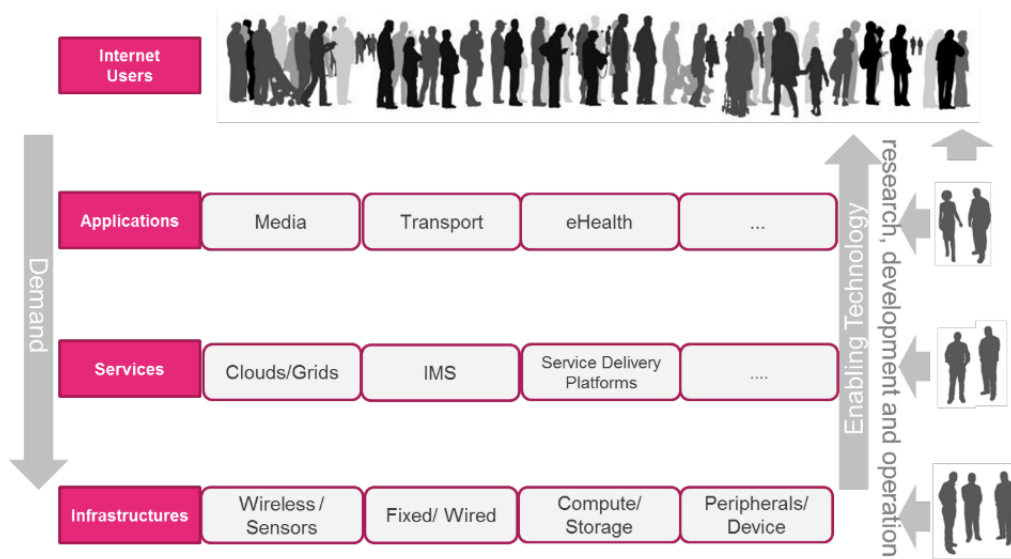


Figura 1 – Evolução das plataformas experimentais. Fonte: (VANDENBERGHE et al., 2013).

de rede; e ainda permite expor a rede como um serviço programável (MIJUMBI et al., 2016b).

Dada a importância de NFV no desenvolvimento do novo paradigma de redes e telecomunicações, é essencial a evolução dos *testbeds* federados para oferecer o suporte necessário à criação de novos serviços e aplicações baseados nessa nova arquitetura. Essa evolução exige a inclusão de novos recursos, de modo a possibilitar a composição de novos serviços de rede, e o desenvolvimento de mecanismos para permitir a orquestração<sup>1</sup> de experimentos com NFV. Ademais, a viabilidade da implementação da arquitetura NFV depende dos mecanismos de virtualização e monitoramento empregados pelo CMF.

Contudo, as federações carecem de ferramentas compatíveis com os requisitos para experimentação em NFV. Em alguns casos, o *testbed* é gerido por um CMF que não oferece monitoramento ou não dispõe de virtualização adequada para prover a abstração da infraestrutura necessária em NFV. Em outros casos, o CMF não dispõe de mecanismos de orquestração que habilitem a gestão de uma função de rede.

Além de requisitos técnicos, é desejável que a experimentação em NFV através da federação possibilite a integração com outros domínios, assim como é esperado nas redes 5G. Desse modo, aproveita-se a heterogeneidade de recursos para promover a inovação e interdisciplinariedade. Também é importante oferecer meios de reduzir a curva de aprendizado, mantendo o equilíbrio entre o controle desejado por usuários experientes e a simplicidade necessária para captar novos usuários.

Tomando como motivação os desafios técnicos apontados e as expectativas da

<sup>1</sup> No contexto deste trabalho, adotamos a definição de orquestração como a capacidade de coordenar ações específicas nos recursos (SOUSA et al., 2018).

comunidade de utilizadores da federação, esse trabalho propõe o desenvolvimento de um novo CMF. Nomeado de O2CMF, esse novo *framework* tem o propósito de fornecer o serviço de experimentação em NFV, sendo interoperável com a estrutura legada de federação.

## 1.1 Objetivos

O objetivo geral desse trabalho é habilitar a criação de experimentos com NFV através de uma plataforma de federação. Como objetivos específicos deste trabalho, é proposto:

- Elicitar os requisitos para integração com a federação, bem como os requisitos para suportar NFV;
- Desenvolver uma arquitetura que habilite a interoperabilidade de NFV com a federação;
- Desenvolver meios para inclusão de usuários não experientes em NFV;
- Habilitar a experimentação em NFV integrada a outro domínio de aplicação como caso de uso.

## 1.2 Estrutura do Texto

Este trabalho se encontra organizado da seguinte maneira:

- O Capítulo 2 apresenta projetos relacionados à federação que influenciaram o desenvolvimento do O2CMF.
- O Capítulo 3 traz uma revisão de conceitos e ferramentas empregados no desenvolvimento do O2CMF.
- O Capítulo 4 detalha o levantamento de requisitos, a arquitetura do O2CMF e o processo de implementação.
- O Capítulo 5 aborda o processo de validação do O2CMF.
- O Capítulo 6 traz a conclusão, apontando as contribuições do O2CMF, os cenários em que esse *framework* tem sido utilizado e reúne sugestões de trabalhos futuros.



## 2 Trabalhos Relacionados

Este capítulo apresenta os conceitos e tecnologias que contextualizam o desenvolvimento deste trabalho. Inicialmente, será apresentado o projeto FUTEBOL. Em seguida, apresentaremos a plataforma experimental Fed4FIRE. Também serão apresentados os principais *frameworks* para controle de *testbeds*.

### 2.1 FUTEBOL

A nova geração de tecnologias de telecomunicações (5G) traz diversos desafios para pesquisadores em redes, criando demanda para o desenvolvimento de novas aplicações, protocolos e tecnologias. Do ponto de vista de redes sem fio, há a necessidade de evoluir para suportar um cenário contendo frequências portadoras muito altas com larguras de banda massivas, densidades extremas em estações rádio-base e um número sem precedentes de antenas. Além disso, há a necessidade de integrar tecnologias sem fio heterogêneas para fornecer cobertura universal de alta taxa. Para apoiar isso, o *core* da rede também terá que alcançar níveis sem precedentes de flexibilidade e inteligência. Essa mudança de paradigma também incluirá novas formas de alocação e realocação de largura de banda, tanto para atender novas aplicações e modelos de negócio quanto para suportar uma “Internet das Coisas” composta por bilhões de dispositivos heterogêneos. Isso exige a consideração conjunta de arquiteturas de redes ópticas e sem fio, além de estratégias de virtualização da rede (ANDREWS et al., 2014). A Figura 2 ilustra essa evolução.



Figura 2 – Evolução proposta para 5G. Fonte: (REICHERT, 2018).

Grandes progressos foram feitos nos últimos anos no desenvolvimento de infraestrutura federada para pesquisa na Europa, através do programa FIRE. Mais recentemente, o projeto FIBRE<sup>1</sup> permitiu a interconexão de fibra óptica de instalações de pesquisa na Europa e no Brasil. No entanto, os problemas de pesquisa de redes sem fio e ópticas são geralmente tratados isoladamente uns dos outros. Essa lacuna motivou a existência do projeto FUTEBOL<sup>2</sup> (acrônimo de *Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory*). Iniciado em 2016, o projeto é conduzido pelos seguintes objetivos(FUTEBOL, 2016b):

1. Implantar *testbeds* na Europa e no Brasil que possam ser acessados externamente para experimentação integrada de tecnologias sem fio e ópticas;
2. Desenvolver e implantar um *framework* para controle convergente de experimentação na fronteira óptica/sem fio, atualmente ausente na infraestrutura de pesquisa FIRE e FIBRE;
3. Realizar pesquisas experimentais utilizando as infraestrutura óptica/sem fio;
4. Criar um ecossistema sustentável de pesquisa colaborativa e parcerias industriais/acadêmicas entre o Brasil e a Europa;
5. Criar materiais de educação e divulgação para um público amplo interessado em questões experimentais em redes sem fio e ópticas.

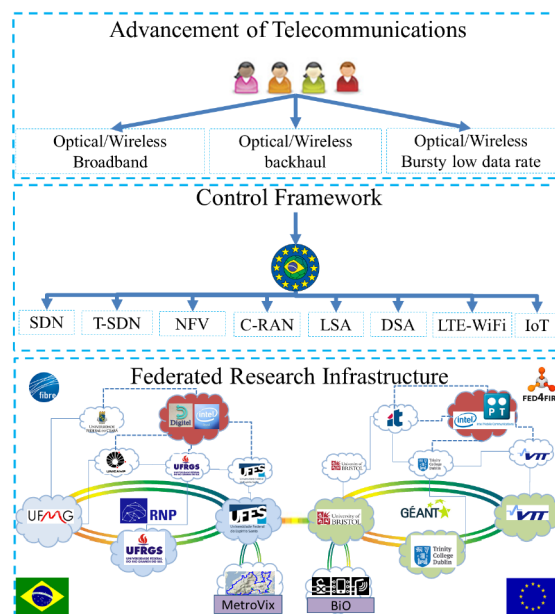


Figura 3 – Representação do ecossistema FUTEBOL. Fonte: (FUTEBOL, 2016a).

<sup>1</sup> <<https://fibre.org.br/>>

<sup>2</sup> <<http://www.ict-futebol.org.br/>>



A Figura 3 ilustra o ecossistema criado pelo FUTEBOL. O avanço das telecomunicações é orientado pelas demandas do usuário final, mas dependente do desenvolvimento do *framework* de controle. Esse *framework* integra diversas tecnologias, dentre as quais está **Network Function Virtualization (NFV)**, a fim de habilitar a convergência da infraestrutura de pesquisa. Por meio dessa abordagem, o FUTEBOL provê uma plataforma experimental conectada à comunidade de telecomunicações (FUTEBOL, 2016a).

O *framework* do FUTEBOL é um aglomerado de ferramentas, do qual este trabalho faz parte. Essas ferramentas se organizam em torno de uma arquitetura comum, apresentada na Figura 4. Cada camada é responsável por um conjunto específico de funcionalidades, distinguindo entre (HAMMAD et al., 2016):

- Camada de serviços: oferece um catálogo de aplicações para facilitar a composição de experimentos.
- Camada de controle de experimento: provê ao usuário programabilidade e automação para gerenciar os recursos no experimento.
- Camada de gestão de *testbed*: responsável pela reserva e instanciação dos recursos, estabelecimento de conectividade e gerenciamento do ciclo de vida do experimento. É representada verticalmente porque se comunica com as demais camadas.
- Camada de virtualização: oferece mecanismos de *slicing* e/ou *scheduling* para habilitar o compartilhamento de recursos entre experimentos. Além disso, a camada de virtualização é responsável por prover isolamento entre experimentos.
- Camada de infraestrutura física convergente: é a coleção de recursos de rede e computação disponíveis para experimentação. Esses recursos são programáveis e incluem rádios, sensores de “Internet das Coisas”, *switches* e enlaces ópticos.

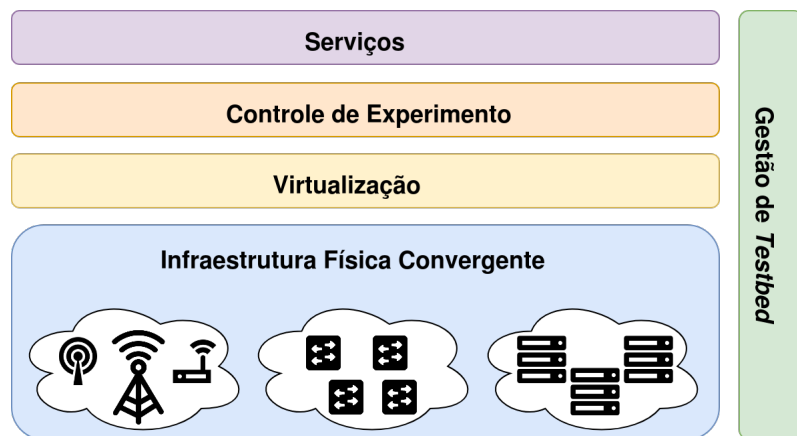


Figura 4 – Arquitetura do *framework* do FUTEBOL. Adaptado de (HAMMAD et al., 2016).

## 2.2 Fed4FIRE

Iniciado em 2012, o Fed4FIRE<sup>3</sup> (acrônimo de *Federation for FIRE*) é um projeto de integração da infraestrutura de pesquisa financiada pelo serviço europeu de pesquisa e experimentação em Internet do futuro (FIRE). O projeto estabeleceu uma estrutura comum de federação, desenvolvendo ferramentas para apoiar usuários no gerenciamento e monitoramento do experimento e para permitir que desenvolvedores se concentrem em suas atividades de teste (FED4FIRE, 2012a).

Dentre as ferramentas desenvolvidas, o jFed<sup>4</sup> alcançou maior popularidade. Trata-se de uma suíte de ferramentas baseadas em Java que atuam como clientes da federação (FED4FIRE, 2012c). Os componentes dessa suíte são ilustrados na Figura 5. JFed UI (também chamado de jFed GUI) permite aos usuários finais provisionar e gerenciar experimentos. JFed Probe ajuda os desenvolvedores de *testbed* a testarem sua implementação da API de federação. JFed Automated Tester realiza testes automatizados, que consistem na execução de um fluxo de experimento completo. Além disso, ele também é empregado no sistema de monitoramento da federação (JFED, 2012).

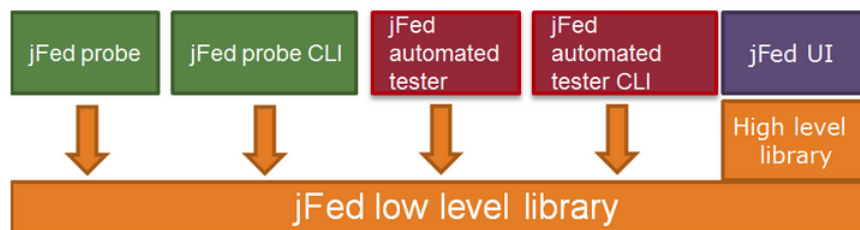


Figura 5 – Suíte de ferramentas jFed. Fonte: (FED4FIRE, 2012c).

Um grande número de *testbeds* existentes na Europa foi adaptado para integrar-se a federação. Esses *testbeds* são voltados para diferentes domínios de redes, como redes ópticas, sem fio, definidas por software, cidades inteligentes, etc (FED4FIRE, 2012a). Dessa forma, o Fed4FIRE alcançou a posição de maior federação mundial de *testbeds* para experimentação em redes, agregando diversas iniciativas de pesquisa e inovação na Europa (FED4FIRE+, 2017).

O Fed4FIRE está continuamente aberto a adesão de *testbeds*, oferecendo diferentes opções de federação (FED4FIRE, 2012b):

- *Associated*: o *testbed* é listado no site do Fed4FIRE, com link para a documentação e informação de contato. Esta opção não requer integração técnica.

<sup>3</sup> <<https://www.fed4fire.eu/>>

<sup>4</sup> <<https://jfed.ilabt.imec.be/>>

- *Light*: o acesso aos recursos do *testbed* é realizado através de uma API exposta via Web. Esta opção não permite controle individual sobre os recursos, mas garante acesso unificado aos experimentadores.
- *Advanced*: o *testbed* implementa a API de federação, provendo ao experimentador controle de todos os estágios do ciclo de vida do experimento (reserva de recursos, instanciação, controle de recursos, monitoramento, etc.).

Os experimentadores podem operar remotamente os *testbeds* oferecidos através do Fed4FIRE. Para ter acesso, basta criar uma conta e um certificado ([FED4FIRE, 2012d](#)).

## 2.3 Frameworks Existentes para Experimentação em Redes

Existem diversos *testbeds* de rede em todo o mundo. Cada um deles necessita de um CMF para estar apto a atender os usuários da federação. A seguir são apresentados os CMFs mais populares.

### 2.3.1 GENI Control Framework (GCF)

O projeto estadunidense GENI<sup>5</sup> é uma federação de *testbeds* para experimentação em larga escala em Internet do futuro ([BERMAN et al., 2014](#)). Sendo um dos projetos pioneiros em Internet do futuro, GENI estabeleceu uma arquitetura de federação que se tornou o padrão *de facto* ([HAMMAD et al., 2016](#)). A *Slice-based Federation Architecture* (SFA) define, com base em conceitos de alto nível, o conjunto mínimo de interfaces que permitem a interação em uma federação de infraestrutura. Essas interfaces abrangem os estágios iniciais do ciclo de vida do experimento (como descoberta de recursos, reserva e provisionamento). Dessa forma, a arquitetura SFA habilita a federação de *testbeds* com diferentes tecnologias, pertencentes a diferentes administradores, além de possibilitar a criação de experimentos que combinem recursos disponíveis em diferentes *testbeds* ([PETERSON et al., 2010](#)).

Para habilitar o compartilhamento da infraestrutura, os recursos do *testbed* atribuídos a um usuário não são apenas o substrato físico, mas também versões virtualizadas deste. Na arquitetura SFA, a porção mínima de um recurso que pode ser atribuída a um usuário é chamada de *sliver*. Geralmente, um experimento é composto por um conjunto de *slivers*, o que é denominado *slice* ([PETERSON et al., 2010](#)). A relação entre *sliver* e *slice* é representada na Figura 6.

A partir dessas abstrações de recursos, apresentamos os componentes básicos da arquitetura SFA: o *Aggregate Manager* (AM) e o *Slice Manager*. A interação com esses

---

<sup>5</sup> <http://www.geni.net/>

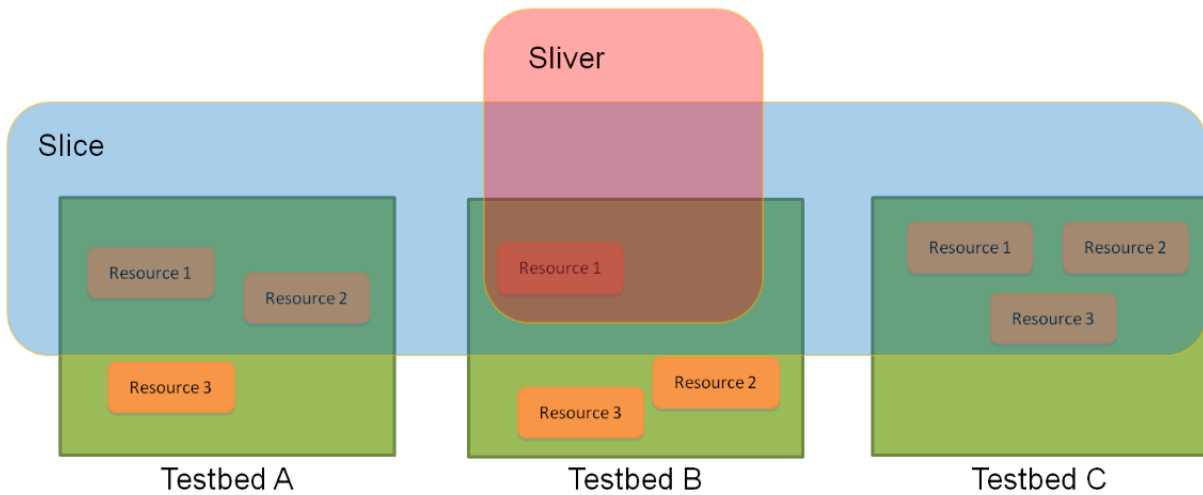


Figura 6 – Relação entre *sliver* e *slice*. Fonte: (BECUE et al., 2015).

componentes se dá através da API SFA definida em (PETERSON et al., 2010). Para facilitar a interação do experimentador com o *testbed*, abstraindo a API, existem clientes SFA (como o jFed). A Figura 7 ilustra a arquitetura SFA.

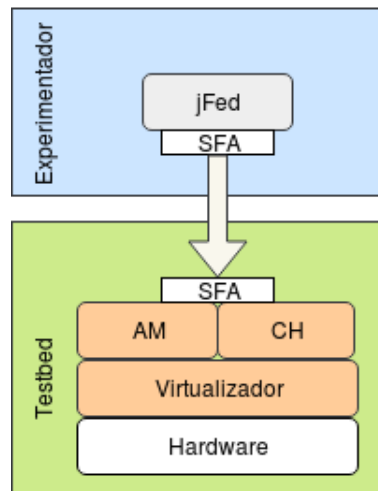


Figura 7 – Representação da arquitetura SFA. Adaptado de: (BECUE et al., 2015).

O *Slice Manager* tem a função de manter um registro dos experimentos (*slices*) e das permissões dos usuários (PETERSON et al., 2010). Na prática, essas funcionalidades são implementadas por um componente chamado *Clearinghouse* (CH). Esse componente acumula funcionalidades de suporte a federação, tais como provedor de identidade, autenticação, além de atribuir credenciais que permitem aos usuários atuar sobre os recursos do experimento (GENI, 2015). A Figura 8 ilustra as funções da *Clearinghouse*.

O *Aggregate Manager* (AM) tem a função de controlar a alocação de *slivers*. Logo, o experimentador precisa interagir com o AM para adicionar recursos ao seu *slice*. A comunicação com o AM envolve um tipo especial de dado: *Resource Specification* (RSpec),

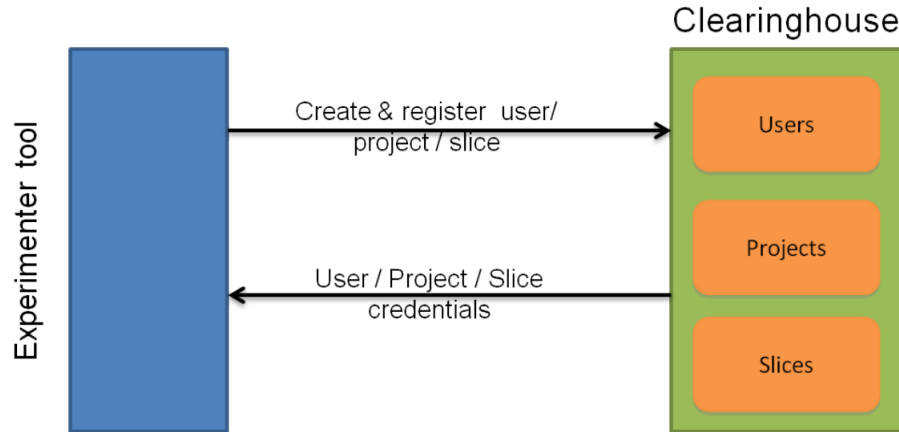


Figura 8 – Representação das funcionalidades da *Clearinghouse*. Fonte: (BECUE et al., 2015).

que é um documento XML que descreve recursos, suas características e dependências. Há três tipos de RSpec: (i) *Advertisement*: usada pelo AM para anunciar ao usuário os recursos sob sua gestão que estão disponíveis; (ii) *Request*: usada pelo usuário para descrever ao AM os recursos desejados para o experimento; e (iii) *Manifest*: usada pelo AM para descrever ao usuário os recursos concedidos a ele (PETERSON et al., 2010). A RSpec é dependente do *testbed*, já que cada um poderá ter tipos de recursos diferentes (representados de maneira própria) (BECUE et al., 2015). As funções do AM estão ilustradas na Figura 9.

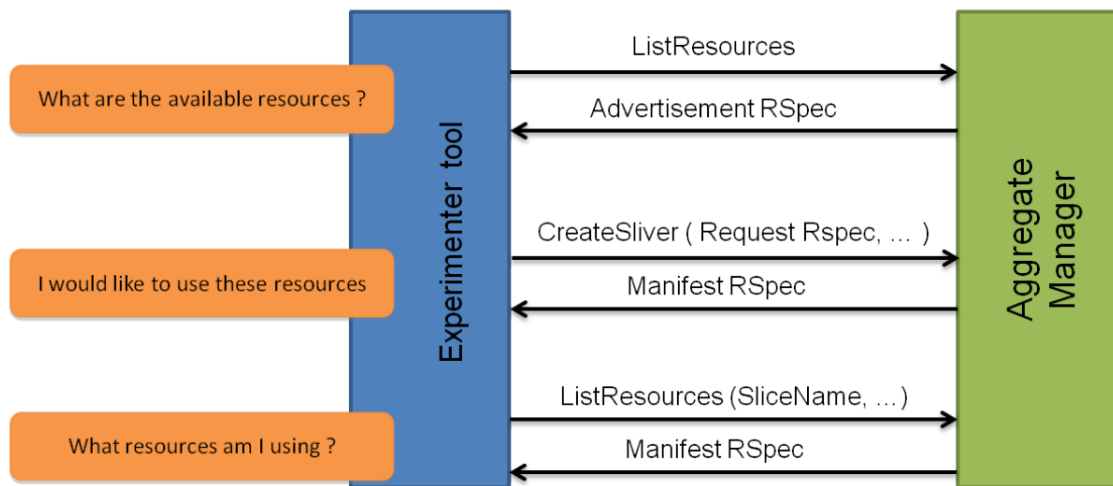


Figura 9 – Representação das interações do usuário com o AM. Fonte: (BECUE et al., 2015).

A federação GENI oferece variados recursos de rede e computação e, por essa razão, faz uso de ferramentas específicas para cada um desses domínios (BERMAN et al., 2014). No entanto, GENI disponibiliza uma implementação de referência da arquitetura SFA, que é o GENI *Control Framework*<sup>6</sup> (GCF). O GCF é composto de um AM; uma

<sup>6</sup> <<https://github.com/GENI-NSF/geni-tools>>

*Clearinghouse*; e ferramentas cliente SFA, que podem ser usadas por desenvolvedores de *testbeds* para testar sua *Clearinghouse* ou AM recém implantado. O AM de exemplo fornece um esqueleto da API SFA (também chamada de GENI API), cabendo ao desenvolvedores a implementação específica para seu *testbed* (BECUE et al., 2015).

A GENI API é a interface através da qual o experimentador se comunica com o a federação para descobrir e reservar recursos. Dessa forma, ao invocar o método *ListResources*, o experimentador é informado dos recursos disponíveis. Uma vez que o experimentador identificou os recursos que deseja, ele pode solicitar sua reserva, usando o método *Allocate*. Caso a reserva seja bem sucedida, ele poderá instanciar os recursos, através do método *Provision*. Em seguida, a ferramenta cliente chama o método *Status*, a fim de verificar se o provisionamento foi concluído. O passo seguinte é iniciar/ligar os recursos através do método *PerformOperationalAction*. A partir daí, o experimentador pode acessar os recursos e executar seu experimento. Caso necessário, ele pode estender sua reserva, usando o método *Renew*. Por fim, o experimentador pode encerrar seu experimento e devolver os recursos usando o método *Delete* (GENI, 2014a). O comportamento de cada método da API se encontra descrito na Tabela 1. Essa tabela, em conjunto com a Figura 10, apresenta as interações entre os componentes na arquitetura SFA.

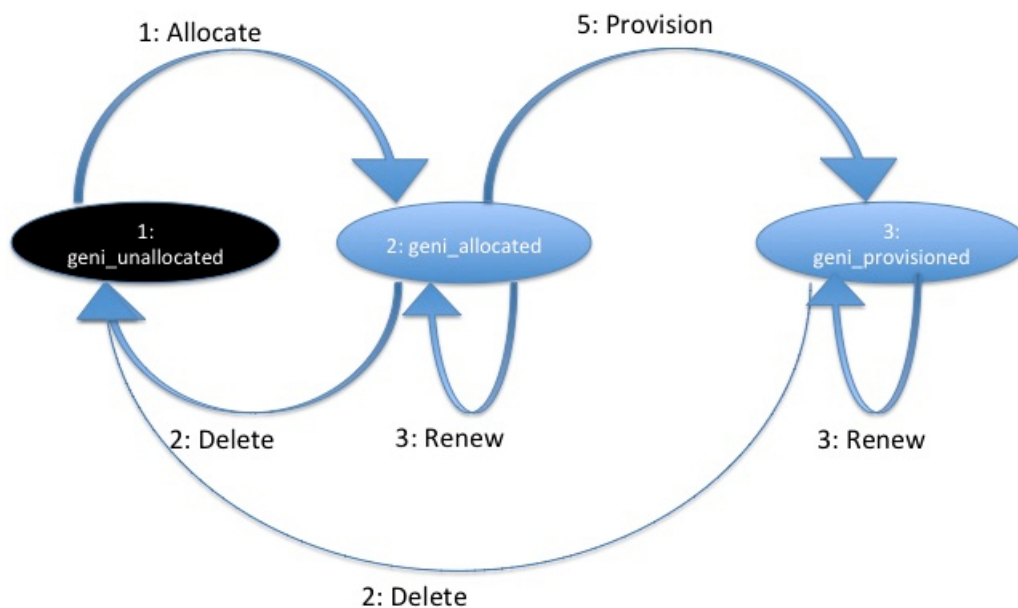


Figura 10 – Máquina de estados para um *sliver*. Fonte: (GENI, 2014b)

A GENI API é fornecida via XML-RPC por meio de uma conexão SSL. Além disso, cada objeto (um usuário, um recurso, um AM ou uma *Clearinghouse*) é identificado com um URN (*Uniform Resource Name*), seguindo o formato `urn:publicid:IDN+<entidade responsável>+<tipo>+<nome>`. São utilizados certificados X.509 v3 (definido na RFC5280) para vincular chaves públicas ao URN. Somente o portador da chave privada correspondente à chave pública contida no certificado pode atuar como o usuário nomeado pelo URN.

Tabela 1 – Métodos da GENI API.

Método	Descrição
<i>GetVersion</i>	Permite consultar informações de configuração do AM, como versão da GENI API e esquemas de RSpec.
<i>ListResources</i>	Permite obter do AM a <i>Advertisement</i> RSpec.
<i>Allocate</i>	Permite reservar recursos. Recebe como parâmetros: (i) descrição dos recursos, através da <i>Request</i> RSpec; (ii) URN do <i>slice</i> (obtida previamente pelo cliente ao solicitar credenciais à <i>Clearinghouse</i> ); e (iii) tempo de duração da reserva (especificado no formato RFC 3339). Esse método retorna uma <i>Manifest</i> RSpec descrevendo os recursos cedidos ao usuário.
<i>Provision</i>	Permite a instanciação dos recursos reservados. Dessa forma, os <i>slivers</i> passam do estado <i>geni_allocated</i> para <i>geni_provisioned</i> . Esse método retorna uma <i>Manifest</i> RSpec, contendo as informações de acesso aos recursos juntamente com sua descrição.
<i>Status</i>	Permite obter atualizações sobre o estado dos <i>slivers</i> após o provisionamento.
<i>PerformOperationalAction</i>	Permite executar uma ação sobre um <i>sliver</i> (ligar, desligar, reiniciar).
<i>Describe</i>	Permite recuperar a <i>Manifest</i> RSpec de um <i>slice</i> . Isso é necessário porque a <i>Manifest</i> RSpec não é necessariamente estática. Ela contém as informações de acesso aos recursos juntamente com sua descrição.
<i>Renew</i>	Permite solicitar ao AM a extensão do prazo de reserva dos recursos. Esse método recebe como argumento o tempo até quando a reserva deve ser mantida (especificado no formato RFC 3339), podendo ser aplicado a <i>slivers</i> que estão no estado <i>geni_allocated</i> ou <i>geni_provisioned</i> .
<i>Delete</i>	Permite excluir um experimento, liberando os recursos associados. Dessa forma, os <i>slivers</i> retornam ao estado <i>geni_unallocated</i> . O AM exclui automaticamente os <i>slivers</i> cuja reserva expirou. Nenhuma outra operação pode ser executada sobre <i>slivers</i> já excluídos.
<i>Shutdown</i>	Permite bloquear um <i>slice</i> cujos <i>slivers</i> apresentam problemas. Dessa forma, nenhuma outra ação poderá ser executada sobre o <i>slice</i> , enquanto o operador do <i>testbed</i> investiga o incidente. Esse método não faz parte da especificação da API, sendo de uso exclusivo do operador.

Dessa forma, é possível identificar e autenticar as entidades que utilizam a API. Por essa razão, cada método da GENI API requer como argumento credenciais (autorização para executar uma ação concedida pela *Clearinghouse*) (GENI, 2016a) (GENI, 2016b) (GENI, 2017).

### 2.3.2 cOntrol and Management Framework (OMF)

O *cOntrol and Management Framework* (OMF) foi originalmente desenvolvido para controlar o *testbed* de rede sem fio da Rutgers University. Inicialmente, o OMF suportava apenas dispositivos ORBIT (hardware para comunicação sem fio IEEE 802.11). Atualmente, o OMF suporta diferentes recursos de rede (com e sem fio) (RAKOTOARIVELO et al., 2010).

O OMF dispõe de uma linguagem específica de domínio (*OMF Experiment Description Language* – OEDL), baseada em Ruby, que permite ao experimentador criar um *script* para controlar seu experimento (*Experiment Description* – ED). O ED detalha os requisitos de recursos, sua configuração inicial e permite especificar ações disparadas por



evento. Além disso, o OMF fornece uma biblioteca para telemetria do experimento (*OMF Monitoring Library* – OML) (RAKOTOARIVELO et al., 2010).

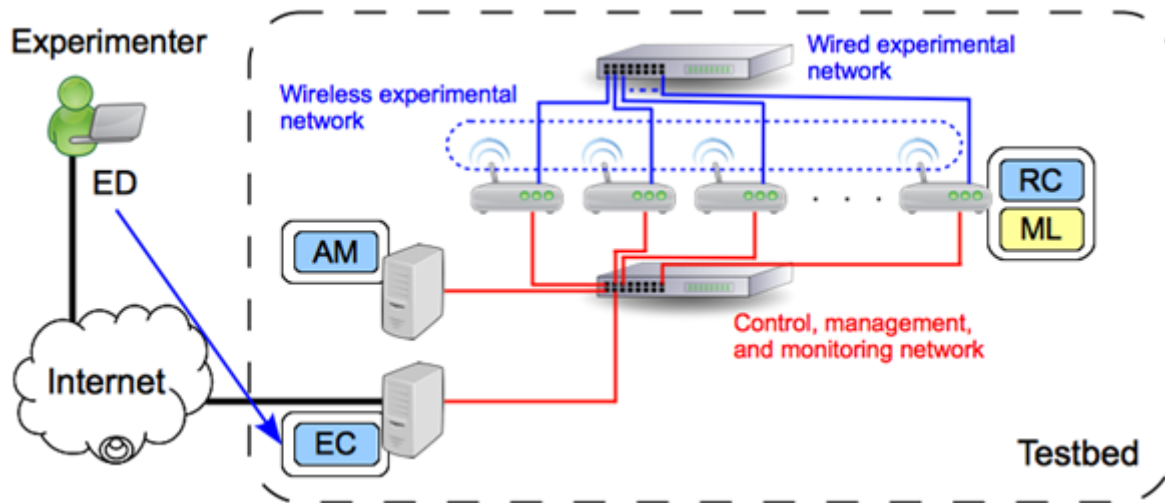


Figura 11 – Representação de um *testbed* controlado pelo OMF. Fonte: (FIBRE, 2018).

A Figura 11 ilustra um *testbed* controlado pelo OMF. Cada recurso físico deve ser gerenciado por um *Resource Controller* (RC). O RC é responsável pela configuração do recurso. Além disso, cada recurso fica associado a uma instância do OML. A rede experimental é dividida em duas redes lógicas: uma dedicada ao tráfego do experimento; e outra dedicada ao tráfego de mensagens de controle e métricas coletadas (FIBRE, 2018).

Além do projeto GENI (BERMAN et al., 2014), outros projetos adotaram o OMF. O FIBRE é uma federação para experimentação em redes, operada instituições acadêmicas brasileiras e pela Rede Nacional de Ensino e Pesquisa (RNP). Inicialmente, o OMF foi utilizado para controlar os recursos sem fio oferecidos pelo FIBRE (SALMITO et al., 2014). Em 2018, o OMF passou a ser o único CMF do FIBRE (CALED et al., 2017).

### 2.3.3 OFELIA Control Framework (OCF)

OFELIA foi um projeto europeu focado em experimentação em OpenFlow. Esse projeto desenvolveu o *OFELIA Control Framework* (OCF) com a finalidade de controlar os *testbeds* de sua federação. O OCF é capaz de controlar recursos de redes (comutadas por pacote e ópticas) e computação (VMs) (SUNE et al., 2014). A arquitetura do OCF é baseada no GCF. Para reservar os recursos, o usuário utiliza um portal Web, chamado Expedient. Esse portal interage com a *Clearinghouse*, para obter autorização para o usuário, e com o *Aggregate Manager*, para instanciar os recursos. Há um AM para cada tipo de recurso: o *Virtualization Aggregate Manager* é responsável pela gestão de máquinas virtuais; e o *OpenFlow Aggregate Manager* é responsável por recursos de rede. A rede virtual alocada para o experimento pode ser controlada usando OpenFlow v1.0 através de um controlador especificado pelo usuário. Além disso, o OpenFlow foi estendido para



integrar-se à inteligência de encaminhamento óptico tradicional. Dessa forma, os usuários têm a liberdade de realizar configurações OpenFlow com base em portas ou comprimentos de onda (HUANG et al., 2017).

Embora o projeto OFELIA tenha finalizado, outros projetos reutilizaram sua infraestrutura e o OCF. O FIBRE fazia uso do OCF para gestão de recursos de computação e OpenFlow (SALMITO et al., 2014) até 2018. OF@TEIN<sup>7</sup> é uma federação asiática para experimentação em OpenFlow que é controlada usando o OCF (HUANG et al., 2017).

---

<sup>7</sup> <<http://oftein.net/projects/OF-TEIN/wiki>>



## 3 Conceitos Fundamentais

Embora os CMFs apresentados no Capítulo 2 ofereçam recursos de computação (VMs), eles empregam simples gerenciadores de virtualização. Isso pode ser explicado pelo fato de que OCF e OMF, por exemplo, foram desenvolvidos com ênfase em redes ópticas e sem fio, respectivamente. No entanto, para implantação de funções de rede virtualizadas, uma virtualização simples impõe limitações para a manipulação da infraestrutura, exigindo maior intervenção do operador e oferecendo pouca flexibilidade aos usuários. A fim de agregar eficiência à gestão dos recursos e prover maior flexibilidade e automação ao usuário, o gerenciador de VMs pode ser substituído por uma plataforma de computação em nuvem. Nesse capítulo abordaremos os conceitos relacionados à computação em nuvem e seu papel na habilitação de NFV (*Network Function Virtualization*).

### 3.1 Computação em Nuvem

Segundo (MELL; GRANCE, 2009), computação em nuvem pode ser definida como um modelo para habilitar acesso à rede de forma ubíqua, conveniente e sob demanda para recursos de computação configuráveis (tais como servidores, armazenamento e aplicativos), possibilitando seu provisionamento (ou liberação) de forma rápida e com intervenção mínima do provedor de serviço. Sendo assim, para que um serviço possa ser caracterizado como computação em nuvem, é essencial que ele seja estruturado tal como apresentado na Figura 12. As características essenciais de um serviço de computação em nuvem são discutidas a seguir:

- ***On-demand self-service***: Um cliente pode provisionar unilateralmente recursos de computação sem exigir interação humana com cada provedor de serviços.
- ***Broad network access***: Os recursos estão disponíveis na rede e podem ser acessados através de plataformas heterogêneas, independentemente de localização.
- ***Resource Pooling***: Os recursos de computação do provedor são agrupados para atender a vários clientes, podendo ser disponibilizados de acordo com a demanda do cliente.
- ***Rapid Elasticity***: Os recursos podem ser provisionados e liberados de forma elástica para escalar de acordo com a demanda.
- ***Measured Service***: Habilitar a medição do consumo de recursos (tais como armazenamento, processamento e largura de banda) de modo a permitir sua otimização automática dos serviços.

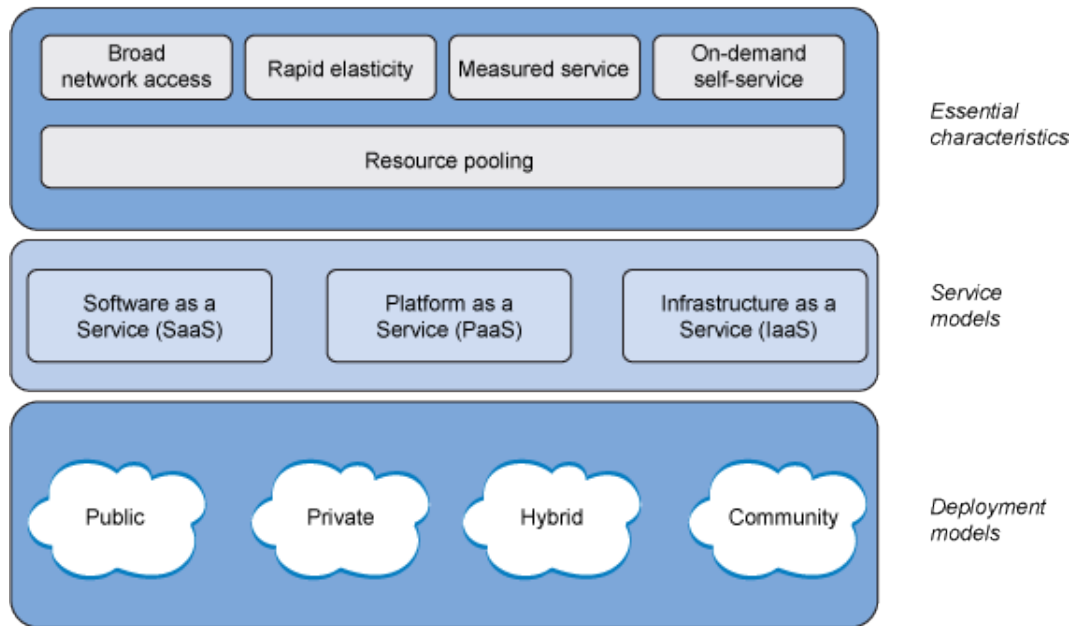


Figura 12 – Características da computação em nuvem. Fonte: (SAN ENTHUSIAST, 2016).

Além disso, a computação em nuvem pode ser provida em três modelos (MELL; GRANCE, 2009):

- **Software as a Service (SaaS):** O usuário utiliza aplicativos em execução em uma infraestrutura de nuvem. Os aplicativos podem ser acessados a partir de vários dispositivos do cliente (através de um navegador Web, por exemplo).
- **Platform as a Service (PaaS):** O usuário pode implantar aplicativos na infraestrutura em nuvem, fazendo uso das linguagens de programação, bibliotecas, serviços e ferramentas oferecidas pelo provedor.
- **Infrastructure as a Service (IaaS):** O usuário é capaz de provisionar recursos de computação (processamento, armazenamento, rede e outros), tendo a liberdade de executar qualquer software a sua escolha.

Ainda de acordo com (MELL; GRANCE, 2009), a infraestrutura de nuvem pode ser implantada nos seguintes modelos:

- **Nuvem Privada:** Implantada para uso exclusivo por uma única organização. Ela pode ser mantida pela organização e/ou por terceiros.
- **Nuvem Comunitária:** É implantada para uso exclusivo por uma comunidade que têm interesses compartilhados (por exemplo, missão, requisitos de segurança, política). Ela pode ser mantida pelas organizações da comunidade e/ou terceiros.

- **Nuvem Pública:** A infraestrutura de nuvem é disponibilizada ao público, sendo mantida por organização comercial, acadêmica ou governamental.
- **Nuvem Híbrida:** Caracterizada por uma composição de infraestruturas de nuvem implantadas em modelos distintos (privada, comunitária ou pública) que permanecem como entidades exclusivas, mas unidas por tecnologia padronizada que permite a portabilidade de dados e aplicativos.

### 3.1.1 OpenStack

Conforme representação na Figura 13, o OpenStack é um sistema operacional de nuvem que permite controlar grandes *pools* de recursos de processamento, armazenamento e rede em um *data center*. Ele é fruto de uma cooperação global de desenvolvedores para produzir uma plataforma aberta de computação em nuvem. O OpenStack é interoperável com tecnologias populares (tanto empresariais como de código aberto), tornando-o adequado para infraestrutura heterogênea. Suas características permitiram conquistar um público constituído por grandes empresas (como Best Buy, Comcast, PayPal) (OPENSTACK, 2018h).

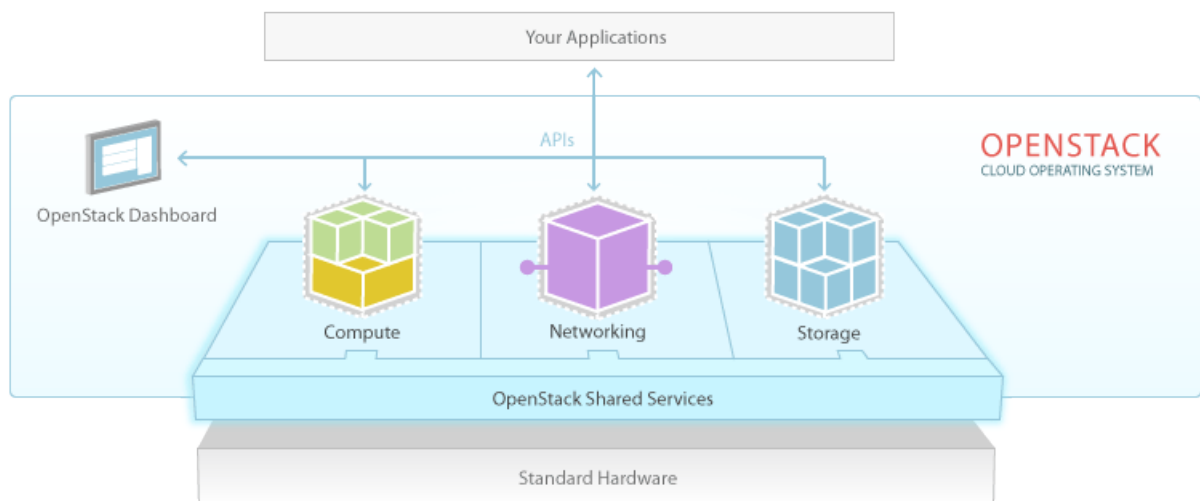


Figura 13 – Estrutura do OpenStack. Fonte: (AMORIM et al., 2018).

O OpenStack adota uma arquitetura modular onde cada módulo é responsável por um tipo de serviço. Há seis módulos básicos e dezenas de módulos opcionais. Dessa forma, é possível criar instalações de nuvem “sob-medida”. Os seis módulos básicos são (RED HAT, 2018b):

- **Nova:** é responsável pela gestão e provisionamento de recursos de processamento.
- **Neutron:** permite gerenciar recursos de rede e prover conectividade a outros serviços do OpenStack.

- **Swift**: é responsável pelo armazenamento de objetos (dados não estruturados).
- **Cinder**: oferece armazenamento tradicional (*block storage*).
- **Keystone**: responsável pela gestão usuários e permissões, além de atuar como um catálogo dos serviços disponíveis na nuvem.
- **Glance**: permite gerenciar imagens de disco para máquinas virtuais.

Além dos módulos essenciais, é comum instalar também os módulos que proveem os serviços de telemetria, orquestração e interface com o usuário. Esses módulos são apresentados a seguir. A Figura 14 ilustra as interações entre os serviços em uma instalação usual do OpenStack.

- **Horizon**: oferece uma *dashboard* acessível via Web que possibilita a administração da nuvem e o provisionamento de recursos de computação (RED HAT, 2018a).
- **Ceilometer**: responsável por prover medições de recursos da nuvem (RED HAT, 2018a).
- **Heat**: implementa um mecanismo de orquestração de infraestrutura que permite a implantação de aplicações na nuvem através de modelos. Os modelos usados pelo Heat consistem em arquivos de texto, pensados para serem lidos/escritos por humanos. Eles contêm a descrição da infraestrutura necessária para a aplicação e os relacionamentos entre os recursos. A partir do modelo, o Heat chama outros serviços do OpenStack para preparar a infraestrutura e implantar a aplicação (OPENSTACK, 2018q).

Considerando os módulos básicos e os opcionais que usualmente são implantados, um cenário de implantação comum é o uso de um nó dedicado aos serviços de gestão da nuvem e um ou mais nós dedicados a prover recursos de computação. Dependendo dos requisitos de desempenho, pode ser implantado um nó dedicado ao serviço de rede. Esse esquema de implantação é exemplificado na Figura 15.

Para a compreensão das funcionalidades providas pelo OpenStack, é preciso conhecer alguns conceitos básicos. **Projetos** são unidades organizacionais na nuvem às quais os usuários são atribuídos. Cada usuário membro do projeto tem um **papel**, ou seja, um conjunto de direitos e privilégios que definem quais ações ele pode executar. Os usuários podem ser membros de um ou mais projetos, possuindo um papel em cada um deles (OPENSTACK, 2018b) (OPENSTACK, 2018f).

Dentro do projeto, os usuários consomem os recursos da nuvem, criando VMs, por exemplo. No OpenStack, cada VM é criada com base em uma configuração pré-definida

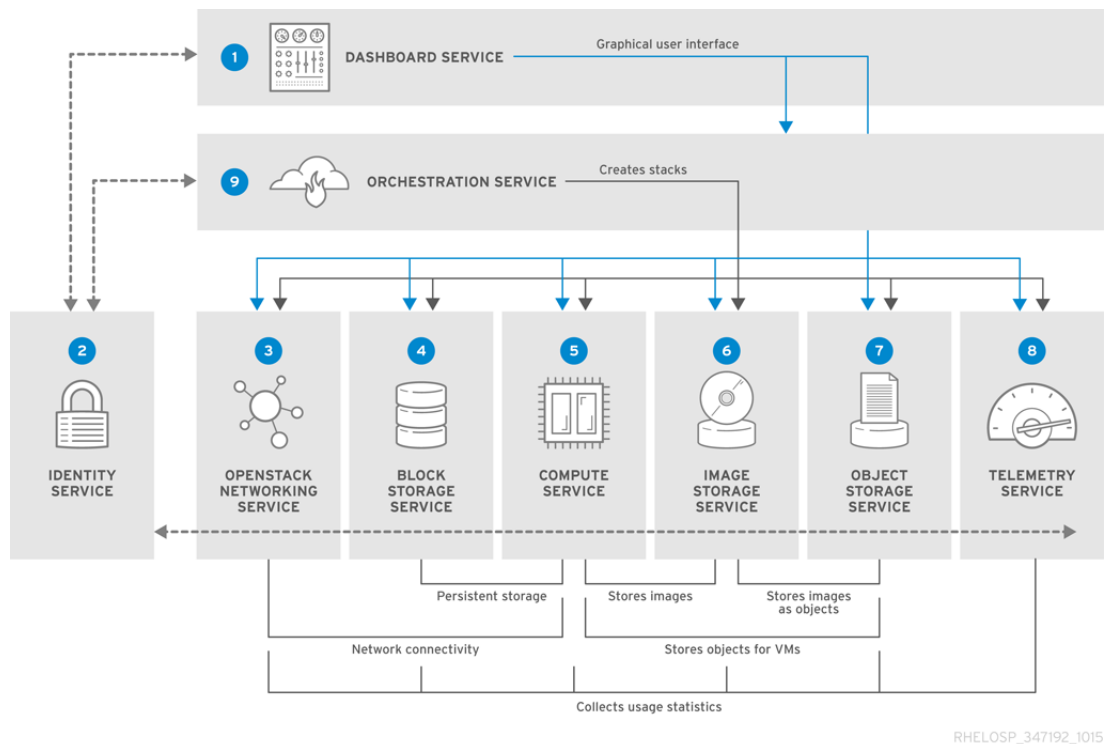


Figura 14 – Arquitetura de instalação usual do OpenStack. Fonte: (RED HAT, 2018a).

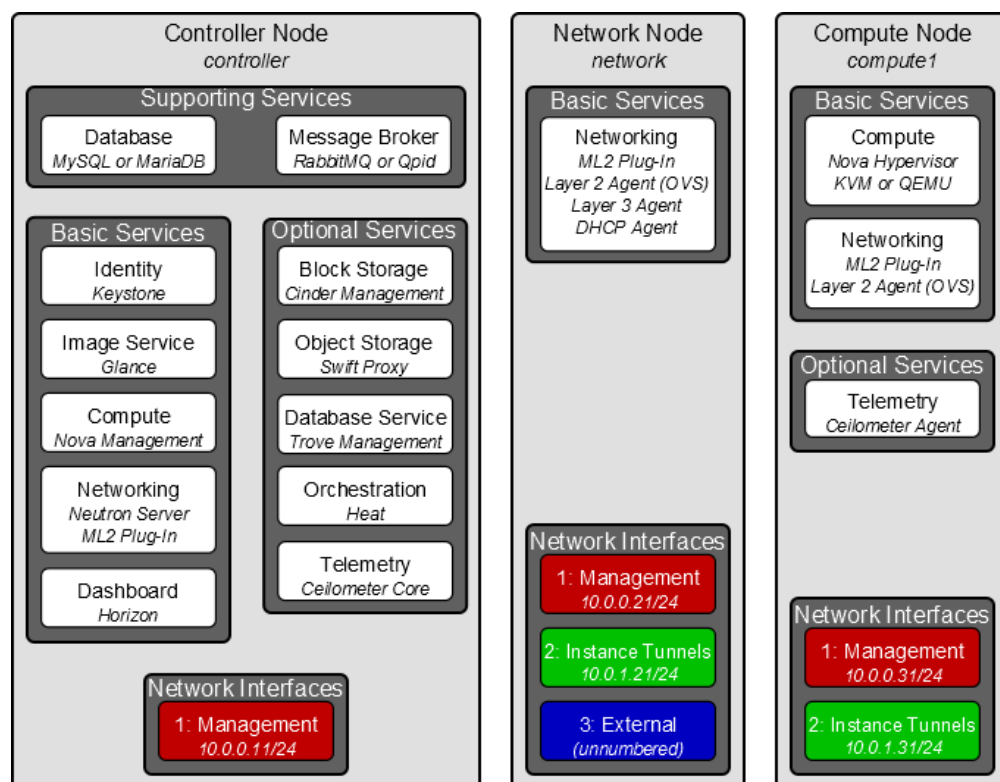


Figura 15 – Modelo usual de implantação do OpenStack. Fonte: (OPENSTACK, 2016).

de hardware, chamada de *flavor*, que define uma quantidade de CPUs virtuais, memória e disco (OPENSTACK, 2018c). Além disso, sempre que uma VM é criada, ela recebe

automaticamente um endereço IP privado (*fixed IP*) que será usado para comunicação com as demais VMs no projeto. Para habilitar a comunicação com redes externas (provendo conectividade com a Internet ou viabilizando o acesso SSH) é preciso associar um *floating IP* à VM. Esse tipo de endereço é obtido de um *pool* previamente configurado pelo administrador da nuvem (OPENSTACK, 2018e), utilizando uma faixa de IPs públicos ou de IPs privados acessíveis a partir de outras redes no *data center* (MIRANTIS, 2012).

Os serviços providos pelos módulos podem ser geridos a partir de uma API *RESTful*. Essa API é utilizada tanto na interação entre os módulos (como o Horizon), como por administradores e aplicações externas que utilizam a nuvem. Nesse último caso, para simplificar o uso da API do OpenStack, são disponibilizados clientes CLI e SDK. A Figura 16 exemplifica essa arquitetura. Oficialmente, o OpenStack mantém três clientes para sua API, todos desenvolvidos para a linguagem Python: **Openstackclient**<sup>1</sup>, uma interface CLI para todos os serviços do OpenStack; **Openstacksdk**<sup>2</sup>, SDK oficial do OpenStack; **Shade**<sup>3</sup>, uma biblioteca cliente para o OpenStack.(OPENSTACK, 2018l).

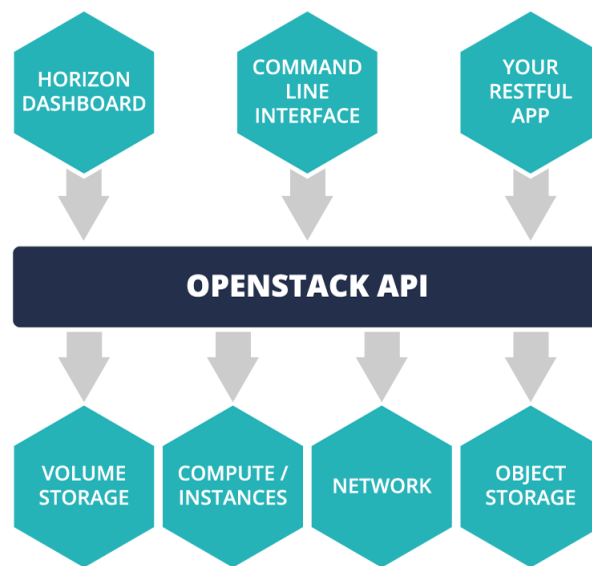


Figura 16 – Interação com o OpenStack via API. Fonte: (FUGA CLOUD, 2018).

Qualquer que seja o cliente/SDK, é preciso fornecer previamente dados de identificação do usuário e do projeto em nome do qual ele executará as ações. Os clientes CLI, particularmente, esperam obter esses dados em variáveis de ambiente. Para simplificar esse processo de configuração, o OpenStack disponibiliza um arquivo *shell script* com as credenciais do usuário no projeto. Esse *script* é chamado de *RC file*. O usuário pode obtê-lo acessando o projeto através do Horizon (OPENSTACK, 2018k).

<sup>1</sup> <<https://pypi.org/project/python-openstackclient/>>

<sup>2</sup> <<https://pypi.org/project/openstacksdk/>>

<sup>3</sup> <<https://pypi.org/project/shade/>>



## 3.2 Network Function Virtualization (NFV)

Tradicionalmente, os provedores de serviços de telecomunicações estabelecem suas operações sobre uma infraestrutura que emprega dispositivos físicos proprietários para cada função que faz parte de um determinado serviço. Nesse contexto, a infraestrutura evolui lentamente, apresentando desvantagens como: programabilidade reduzida, baixa interoperabilidade, falta de elasticidade. Consequentemente, há maior resistência à mudanças e demora na implantação de novos serviços. *Network Function Virtualization* (NFV) foi proposta como uma forma de abordar essas dificuldades, aproveitando a tecnologia de virtualização para oferecer uma nova maneira de projetar, implantar e gerenciar serviços de rede. A ideia principal de NFV é o desacoplamento entre o dispositivo físico de rede e as funções que são executadas neles. Isso significa que uma função de rede (como *switch*, *firewall*) pode ser executada como um software qualquer. Dessa forma, é possível “mover” a inteligência da rede para o *data center*. Um determinado serviço pode ser decomposto em um conjunto de funções de rede virtuais, possibilitando ser instanciado em diferentes locais e sem necessariamente exigir a compra e a instalação de novo hardware. Sendo assim, NFV tem o potencial de reduzir despesas e agilizar a implantação de novos serviços, cuja aplicação não é restrita ao campo das telecomunicações (MIJUMBI et al., 2016b). A Figura 17 ilustra a proposta de NFV.

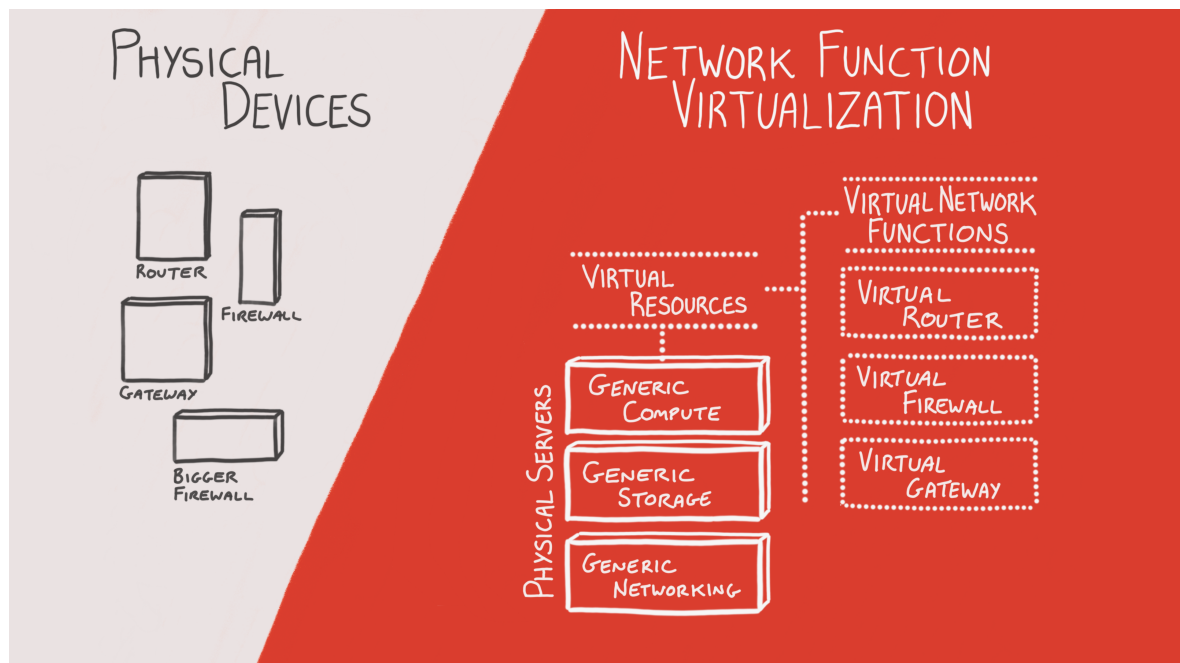


Figura 17 – Mudança de paradigma proposta por NFV. Fonte: (OSS LINE, 2017).

NFV despertou tamanho interesse que algumas operadoras de telecomunicações buscaram o *European Telecommunications Standards Institute* (ETSI) a fim de criar uma comunidade para discutir e padronizar a arquitetura NFV (*Industry Specification Group for NFV* – ETSI ISG NFV). Essa comunidade publicou algumas especificações, que incluem

uma visão geral da infraestrutura, arquitetura, gerenciamento, segurança, resiliência e qualidade de serviço (MIJUMBI et al., 2016b). A arquitetura proposta por (ETSI NFV ISG, 2014) é representada na Figura 18 e é composta por três macro-elementos, descritos a seguir.

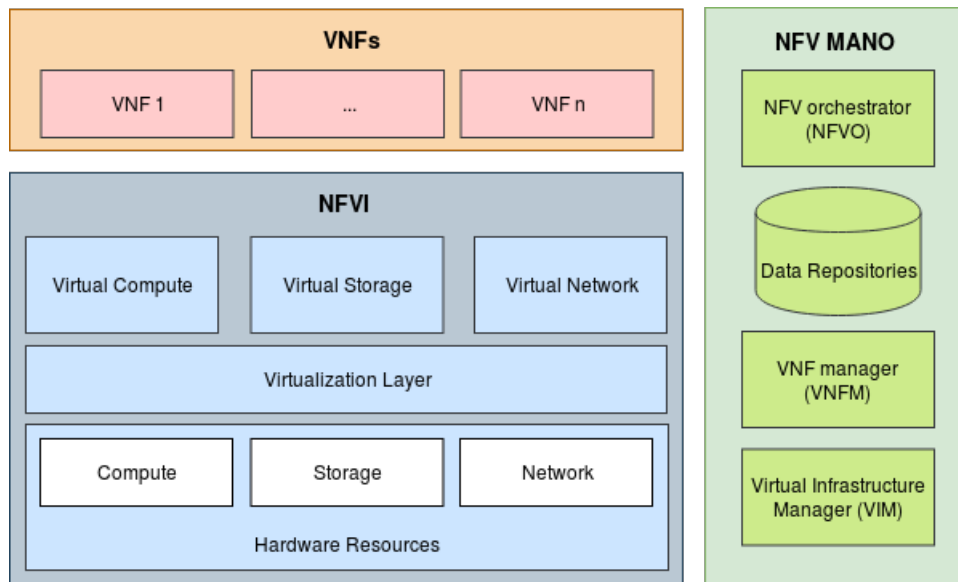


Figura 18 – Representação da arquitetura NFV. Adaptado de: (MIJUMBI et al., 2016b) e (MIJUMBI et al., 2016a).

- **Virtual Network Function (VNF):** Uma função de rede é um bloco funcional dentro de uma infraestrutura de rede que possui interfaces externas e comportamento bem definidos (por exemplo: *firewall*, servidor DHCP, roteador). Uma VNF é uma implementação de uma função de rede implantada em recursos virtuais, como uma VM (MIJUMBI et al., 2016b).
- **Network Function Virtualization Infrastructure (NFVI):** é a combinação de recursos de hardware (geralmente, hardware de “prateleira”) e software (para virtualização) que compõem o ambiente no qual as VNFs são implantadas (MIJUMBI et al., 2016b).
- **NFV Management and Orchestration (NFV MANO):** habilita o provisionamento de VNFs e as operações relacionadas, como a configuração das VNFs e da infraestrutura em que essas funções são executadas. O NFV MANO é composto pelos seguintes componentes (MIJUMBI et al., 2016a):
  - **Virtual Infrastructure Manager (VIM):** responsável pela gestão dos recursos do NFVI. Um VIM pode ser genérico ou especializado em lidar com um determinado tipo de recurso ou provedor de infraestrutura.

- **VNF Manager (VNFM)**: responsável pelo gerenciamento do ciclo de vida das VNFs. Um VNFM pode administrar uma ou mais instâncias de uma mesma VNF ou de VNFs distintas.
- **NFV Orchestrator (NFVO)**: suas funções se dividem entre orquestração de recursos e orquestração de serviços. A primeira fornece suporte a instanciação de VNFs, abstraindo o acesso aos recursos do NFVI e habilitando a governança das instâncias de VNF. A orquestração de serviços permite compor serviços usando VNFs e gerenciar a topologia das instâncias integrantes desse serviço.
- **Data Repositories**: são bancos de dados que mantêm o catálogo de VNFs (um conjunto de modelos que descreve as características de implementação e operação das VNFs); o catálogo serviços de rede (um conjunto de modelos, que definem como os serviços podem ser implementados, indicando as VNFs necessárias e o encadeamento destas); registro das instâncias de VNFs; e registro da alocação dos recursos do NFVI.

Além dos componentes apresentados, a arquitetura NFV define interfaces que podem ser usadas para comunicação interna aos componentes e para a comunicação entre estes. É importante destacar que se trata de uma arquitetura projetada para suportar múltiplos domínios (instalações). Essa característica favorece a escalabilidade e promove a flexibilização do local de implantação dos serviços (MIJUMBI et al., 2016a).

### 3.2.1 Tacker

O Tacker é um módulo do OpenStack dedicado à gestão e orquestração de VNFs. Sua arquitetura, ilustrada na Figura 19, é baseada em três componentes centrais: Catálogo NFV, NFVO e VNFM. Esses componentes colaboram entre si para implementar a NFV *Orchestration* API<sup>4</sup>. Além disso, o VNFM e o NFVO utilizam o OpenStack como VIM fazendo uso dos serviços de orquestração de infraestrutura providos pelo Heat e de autenticação e autorização providos pelo Keystone (OPENSTACK, 2018m).

O componente VNFM é responsável pelas funcionalidades de gestão do ciclo de vida das VNFs (criação, atualização, remoção); monitoramento; gerência de falhas; desempenho e segurança; escalar a VNF com base em políticas; facilitar a configuração inicial da VNF. O componente NFVO possui uma visão global dos serviços na infraestrutura e a usa para prover funcionalidades de orquestração dos recursos e dos serviços de rede. O catálogo funciona como um repositório de modelos de VNFs (OPENSTACK, 2018m) (AGUIAR; TAVARES, 2017a).

<sup>4</sup> <<https://developer.openstack.org/api-ref/nfv-orchestration/v1/index.html>>

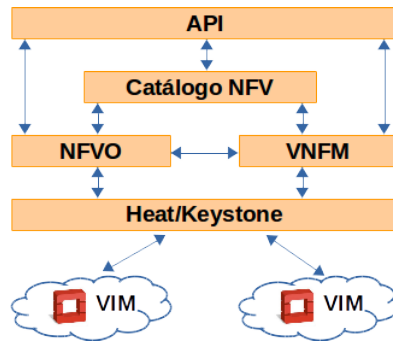


Figura 19 – Arquitetura do Tacker. Adaptado de: (OPENSTACK, 2018m).

Os modelos de VNF são descritores que indicam requisitos de implantação da VNF e seu comportamento. Eles devem ser escritos utilizando a linguagem TOSCA<sup>5</sup> e o formato YAML<sup>6</sup>. Curiosamente, o Heat, cujos serviços são utilizados pelo Tacker, adota outra linguagem. Por esse motivo, os *scripts* em TOSCA são internamente traduzidos para *scripts* HOT (*Heat Orchestration Template*) (AGUIAR; TAVARES, 2017b).

O *script* TOSCA descrevendo a VNF deve apresentar três tipos de componentes (nós): *Virtual Deployment Unit* (VDU), *Virtual Link* (VL) e *Connection Point* (CP). Uma VNF pode ser hospedada em uma ou mais VDUs. Cada VDU é implementada como uma VM. A descrição da VDU no *script* TOSCA indica a imagem de SO e o *flavor* da VM (também é possível descrever diretamente as propriedades de memória, CPU e disco). VL representa a ligação a uma rede lógica. CP atua como uma interface de rede para VDU, conectando-a com um VL (OPENSTACK, 2018p). A estrutura geral de um *script* TOSCA descrevendo uma VNF é apresentada na Figura 20.

Além de descrever a infraestrutura para acomodar a VNF, o modelo também pode conter políticas que especificam o que deve ser feito sempre que a ação de escalar (*scaling*, na nomenclatura do Tacker) for invocada. Há diferentes formas de escalar, sendo possível diminuir recursos (*scale in*) ou aumentá-los (*scale out*). Além disso, há dois tipos de *scaling*: (i) *scaling* horizontal, em que são adicionadas/removidas VDUs; e (ii) *scaling* vertical, em que são adicionados/removidos recursos de uma VDU existente (AGUIAR, 2017). A Listagem 3.1 apresenta a definição de uma política de *scaling* horizontal. Note que a política especifica qual VDU deve ser replicada no ato do *scaling*, através do atributo **targets**. Observe também que está definido como o *scaling* deve ser realizado (adicionando/removendo uma VDU, desde que haja ao menos uma instância e, no máximo, três instâncias). O *script* TOSCA completo se encontra no Anexo A.

```

1 policies:
2   - SP1:
3     type: toska.policies.tacker.Scaling
4     targets: [VDU1]
```

<sup>5</sup> <<http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html>>

<sup>6</sup> <<http://yaml.org/>>

```

tosca_definitions_version:
    Define a versão do TOSCA que o template se baseia.
    A versão atual é tosca_simple_profile_for_nfv_1_0_0.

tosca_default_namespace:
    Campo opcional. Um namespace para incluir schema, types
    ↪ version etc.

description:
    Uma descrição curta sobre o template.

metadata:
    template_name: Nome do template.

topology_template:
    Descreve a topologia da VNF no campo node_template
    node_template:
        Descreve tipos de nós da VNF.
        VDU:
            Descreve as propriedades e capacidades das VDUs.
        CP:
            Descreve as propriedades e capacidades dos pontos
            ↪ de conexão (CP).
        VL:
            Descreve as propriedades e capacidades dos links
            ↪ virtuais (VL).

```

Figura 20 – Representação da estrutura de um *script* TOSCA descrevendo uma VNF.  
Fonte: (AGUIAR; TAVARES, 2017b).

```

5     properties:
6         increment: 1
7         cooldown: 120
8         min_instances: 1
9         max_instances: 3
10        default_instances: 1

```

Listagem 3.1 – Política de *scale*.

Caso seja definida também uma política de monitoramento, é possível escalar de forma automática (*auto-scaling*). Nesse caso, a condição monitorada serve como gatilho para a ação de escalar (OPENSTACK, 2018a). A Listagem 3.2 exemplifica esse cenário, onde há uma política de monitoramento (*vdu\_cpu\_usage\_monitoring\_policy*) que define dois gatilhos, um para *scaling out* (*vdu\_hcpu\_usage\_scaling\_out*) e outro para *scale in* (*vdu\_lcpu\_usage\_scaling\_in*), ambos baseados no uso de CPU. Sempre que a condição de algum dos gatilhos for satisfeita, a política de *scaling* SP1 deve ser executada. Detalhes do *script* TOSCA podem ser consultados no Anexo A.

```

1  policies:
2    - SP1:
3        type: tosca.policies.tacker.Scaling
4        targets: [VDU1]
5        properties:

```

```

6      increment: 1
7      cooldown: 120
8      min_instances: 1
9      max_instances: 3
10     default_instances: 1
11
12     - vdu_cpu_usage_monitoring_policy:
13       type: tosca.policies.tacker.Alarming
14       triggers:
15         vdu_hcpu_usage_scaling_out:
16           event_type:
17             type: tosca.events.resource.utilization
18             implementation: ceilometer
19           metric: cpu_util
20           condition:
21             threshold: 80
22             constraint: utilization greater_than 80%
23             granularity: 60
24             evaluations: 1
25             aggregation_method: mean
26             resource_type: instance
27             comparison_operator: gt
28           metadata: SG1
29           action: [SP1]
30
31     vdu_lcpu_usage_scaling_in:
32       event_type:
33         type: tosca.events.resource.utilization
34         implementation: ceilometer
35       metric: cpu_util
36       condition:
37         threshold: 10
38         constraint: utilization less_than 10%
39         granularity: 60
40         evaluations: 1
41         aggregation_method: mean
42         resource_type: instance
43         comparison_operator: lt
44       metadata: SG1
45       action: [SP1]

```

Listagem 3.2 – Políticas para *scale* automático.

Também é possível especificar uma política de monitoramento, para executar outras ações sobre as VDUs (como reiniciar, desligar) ([AGUIAR; TAVARES, 2017c](#)). A Listagem 3.3 contém um exemplo de política de monitoramento para re-instanciar a VDU1 caso a CPU se mantenha com utilização maior que 50% por 300 segundos. O *script* TOSCA completo se encontra no Anexo B.

```

1 policies:
2   - vdu1_cpu_usage_monitoring_policy:
3     type: tosca.policies.tacker.Alarming
4     triggers:
5       vdu_hcpu_usage_respanning:
6         event_type:

```

```
7         type: toska.events.resource.utilization
8         implementation: ceilometer
9     metric: cpu_util
10    condition:
11        threshold: 50
12        constraint: utilization greater_than 50%
13        granularity: 300
14        evaluations: 1
15        aggregation_method: mean
16        resource_type: instance
17        comparison_operator: gt
18    metadata: VDU1
19    action: [respawn]
```

Listagem 3.3 – Política de monitoramento.





## 4 Projeto e Implementação do O2CMF

Conforme discutido no Capítulo 1, os *testbeds* precisam evoluir continuamente, acompanhando o desenvolvimento científico. Apesar de todo o interesse em NFV, as federações já estabelecidas ainda caminham de forma tímida para a inclusão desse novo paradigma. Impulsionado pelo objetivo de habilitar a pesquisa convergente entre redes ópticas e sem fio, o projeto FUTEBOL construiu uma nova plataforma experimental federada, na qual NFV é uma peça chave.

Para habilitar a implantação de funções de rede virtuais, é preciso prover infraestrutura e mecanismos de gestão adequados, conforme discutido na Seção 3.2. Segundo (MIJUMBI et al., 2016b), a nuvem é a opção mais barata para implementação de NFV. A rapidez e flexibilidade na implantação de novos serviços, além da elasticidade, fazem da computação em nuvem uma excelente opção, oferecendo a chance de alcançar eficiência e redução de despesas. Considerando os modelos de computação em nuvem discutidos na Seção 3.1, pode-se observar que o modelo *IaaS* corresponde aos recursos físicos e virtuais em NFVI, enquanto que VNFs são semelhantes ao modelo de serviço *SaaS*. Apesar disso, NFV requer mais esforços do que simplesmente transferir funções de rede para a nuvem. Há a necessidade de adaptar tanto a forma como serviços e aplicativos são desenvolvidos e entregues, como os ambientes de nuvem (aprimorando os mecanismos de orquestração) (MIJUMBI et al., 2016b).

Os CMFs apresentados na Seção 2.3, embora consagrados no ambiente federado, não estão preparados para experimentação em NFV. A inclusão da nuvem na federação, por si só, representa um avanço. Com base na definição apresentada em (MELL; GRANCE, 2009) e no conhecimento adquirido sobre a infraestrutura legada do Fed4FIRE, é possível afirmar que os CMFs empregam simples gerenciadores de virtualização (como Xen e KVM) para oferecer máquinas virtuais com pouco grau de escolha dos requisitos (opções de imagem de disco e configuração de CPU, memória, etc). Além disso, não é possível aumentar ou diminuir a quantidade de recursos no *slice* dinamicamente, sendo necessário finalizar o experimento e criá-lo novamente. Outra carência é a falta de suporte à medição. Dessa forma, os CMFs falham em apresentar algumas das características essenciais de um serviço de computação em nuvem, tais como *Rapid Elasticity* e *Measured Service*. Sendo assim, o serviço oferecido por esses CMFs não se enquadra plenamente no modelo *IaaS*. Ademais, a inexistência de um catálogo de serviços e a falta de mecanismos de orquestração de experimentos demonstram que os modelos *PaaS* e *SaaS* não são oferecidos. Por fim, a nuvem também se destaca por compartilhar a infraestrutura empregando mecanismos de isolamento entre os usuários.

Os problemas apontados motivaram o desenvolvimento do O2CMF (acrônimo de *OpenStack-based open control and management framework*), um CMF para experimentação em NFV. Neste capítulo, abordaremos o processo de entendimento dos requisitos, o projeto da arquitetura e a implementação.

## 4.1 Levantamento de Requisitos

A fim de elicitar os requisitos do O2CMF, foi preciso considerar: (i) os casos de uso de NFV na plataforma experimental FUTEBOL; (ii) as diretrizes para o desenvolvimento de *testbeds* FUTEBOL; e (iii) o processo de federação ao Fed4FIRE.

Segundo (HAMMAD et al., 2016), a plataforma experimental FUTEBOL deve expor aos experimentadores um catálogo de serviços. Esse catálogo deve prover todos os recursos necessários para a composição de uma VNF, assim como modelos de VNFs. Cada modelo de VNF deve especificar uma imagem de sistema operacional e um perfil de recursos (quantidade de memória, armazenamento, CPUs virtuais e interfaces de rede). A reserva e instanciação de recursos pode ser feita tomando como base um modelo ou uma composição especificada pelo usuário. Espera-se também que o experimentador disponha de meios para controlar o ciclo de vida das VNFs, podendo utilizar políticas para isso. A Figura 21 ilustra os casos de uso.

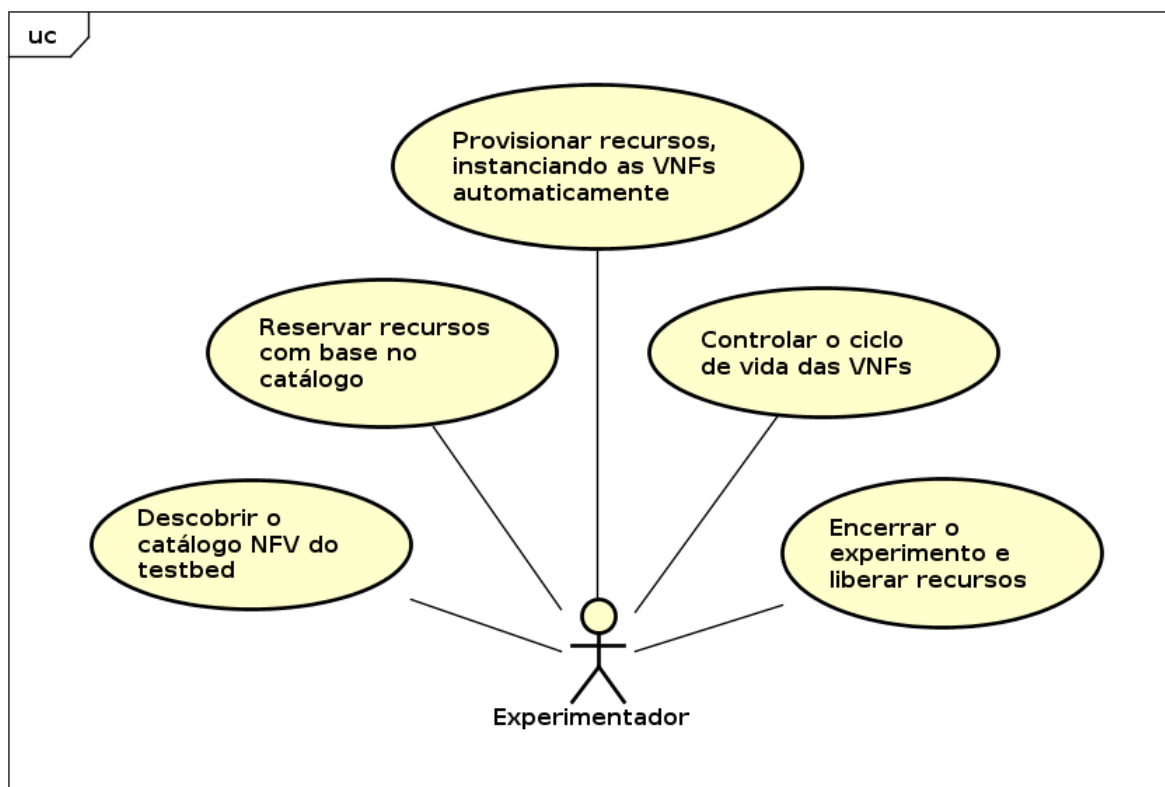


Figura 21 – Casos de uso de NFV estabelecidos pelo FUTEBOL.

No projeto FUTEBOL, cada *testbed* é administrado por uma instituição parceira, que tem autonomia para selecionar os tipos de recursos oferecidos, o CMF e as políticas de uso. Como forma de manter a interoperabilidade, estabeleceu-se algumas especificações a serem seguidas pelos *testbeds*. Em (HAMMAD et al., 2016), foi determinado que a infraestrutura de experimentação do FUTEBOL seria desenvolvida para ser federada a fim de oferecer suporte a experimentadores no acesso e provisionamento de recursos heterogêneos distribuídos entre os *testbeds*. Por essa razão, foi definido que o FUTEBOL aproveitaria a estrutura do Fed4FIRE, para permitir que os experimentadores se concentrem em suas tarefas de experimentação, em vez de particularidades técnicas, como aprender a trabalhar com diferentes ferramentas para cada federação/*testbed* e solicitar contas em cada um separadamente.

Conforme discutido na Seção 2.2, o Fed4FIRE oferece diferentes modos de federação. Os *testbeds* FUTEBOL devem federar-se no modo *Advanced*, a fim de oferecer uma experiência de usuário completa. Além disso, é definido em (HAMMAD et al., 2016) que os CMFs utilizados devem dispor de ferramenta(s) para permitir ao experimentador controlar e configurar os recursos, executar o cenário do experimento e coletar os resultados. Por fim, em (MARTINELLO et al., 2018) estabeleceu-se que cada *testbed* deve prover suporte a usuários iniciantes, oferecendo maior automação, e a usuários especialistas, oferecendo programabilidade profunda dos recursos. O suporte a usuários iniciantes pode ser feito com o uso de configurações *default* e modelos de experimento, além da documentação. O suporte aos usuários especialistas pode ser habilitado ao expor mais parâmetros de configuração da infraestrutura, considerando outras camadas além da aplicação.

O Fed4FIRE adota a arquitetura SFA que, conforme discutido na Seção 2.3.1, tem como componentes básicos o AM e a *Clearinghouse*. No Fed4FIRE, a *Clearinghouse* é centralizada e há uma autoridade certificadora própria. Dessa forma, cada *testbed* federado no modo *Advanced* fica encarregado apenas de implementar seu AM e confiar na autoridade certificadora da federação (FED4FIRE, 2012b).

Embora não esteja especificado pelo FUTEBOL ou Fed4FIRE, foi considerado desejável oferecer ao operador do *testbed* o controle da granularidade dos recursos oferecidos através de políticas. Os objetivos são evitar a degradação de performance em função do compartilhamento; e favorecer a segurança, prevenindo que um *slice* ocupe os recursos de forma a tornar o *testbed* indisponível. Além disso, para oferecer segurança e privacidade aos usuários, foi considerado necessário prover isolamento entre os experimentos, que é uma carência dos CMFs apresentados anteriormente (tais como OCF e OMF). Um resumo dos requisitos levantados encontra-se na Tabela 2.

Tabela 2 – Requisitos do O2CMF.

ID	Descrição
R01	Oferecer ao experimentador um catálogo de serviços contendo modelos de VNF, diferentes perfis de recursos e imagens de SO.
R02	Reservar recursos com base no catálogo, possibilitando a instânciação automática de VNFs.
R03	Possibilitar ao experimentador especificar políticas para a gestão da VNF.
R04	Permitir execução automatizada do cenário do experimento.
R05	Prover suporte a dois perfis de usuários: Iniciante e Especialista.
R06	Implementar um AM compatível com a API SFA.
R07	Suportar as credenciais geradas pelo Fed4FIRE.
R08	Permitir que o operador do <i>testbed</i> estabeleça políticas de compartilhamento de recursos.
R09	Prover isolamento entre os experimentos.

## 4.2 Projeto da Arquitetura

Considerando os requisitos elicitados, identificamos os principais componentes arquiteturais do O2CMF. Esses componentes são: (i) o **AM**, que será responsável pelas funcionalidades referentes à interação com a federação/experimentador; (ii) o **Orquestrador**, que será responsável por prover ao experimentador mecanismos de controle do experimento; e (iii) a **Plataforma de Computação em Nuvem**, que atuará como gerenciadora da infraestrutura virtual. Esses componentes são representados na Figura 22, que esquematiza a arquitetura do O2CMF.

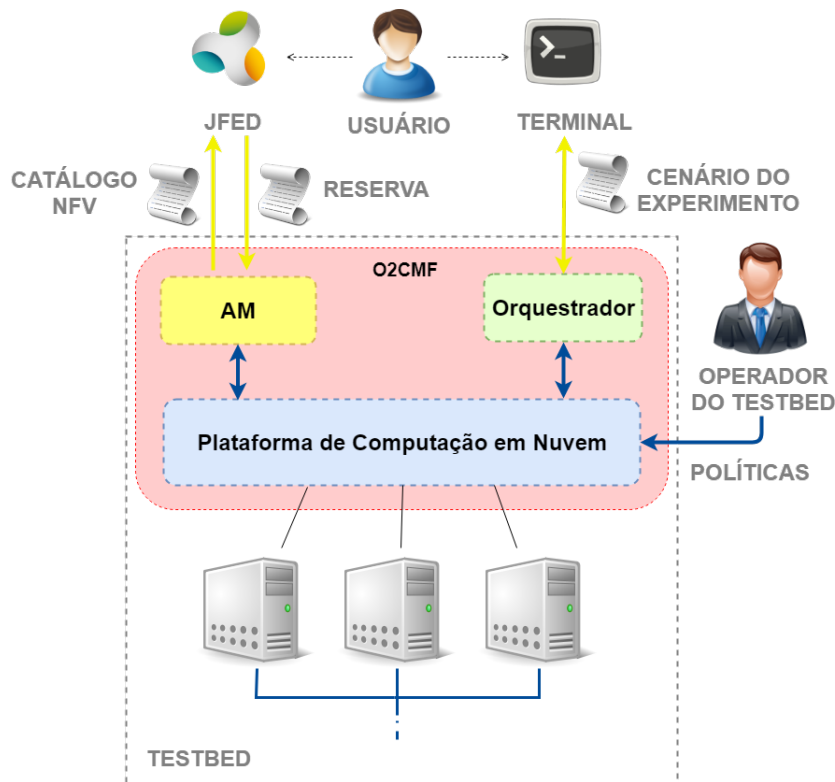


Figura 22 – Arquitetura do O2CMF.

Para as etapas pré-experimento (descoberta de recursos, reserva e provisionamento),

o experimentador poderá utilizar o jFed para interagir com o AM. Será através do AM que o experimentador irá descobrir o catálogo, com base no qual ele preparará a reserva para seu experimento. O experimentador voltará a se comunicar com o AM a fim de reservar e provisionar os recursos. O AM, por sua vez, interagirá com a nuvem para instanciar os recursos pedidos pelo usuário. Após o provisionamento, o experimentador estabelecerá uma conexão SSH com o orquestrador para controlar os recursos na nuvem alocados para o experimento, podendo utilizar uma descrição de um cenário para ser executado automaticamente. Detalhes das interações entre os componentes serão abordados durante o processo de implementação.

Através dessa arquitetura, o O2CMF visa oferecer a interoperabilidade provida pela GENI API, com orquestração de experimento e monitoramento semelhantes ao OMF, aliados à flexibilidade e robustez providos pela nuvem. Para viabilizar a implementação, buscamos tecnologias que pudessem apoiar o desenvolvimento de cada um dos componentes do O2CMF. Em 2016, quando esse trabalho se iniciava, o **OpenStack** foi identificado como um dos principais componentes de uma estrutura arquitetural NFV baseada em nuvem (MIJUMBI et al., 2016b). Diferente de uma simples plataforma de gerenciamento de virtualização, ele é compatível com a definição estabelecida em (MELL; GRANCE, 2009) e disponibiliza seus serviços através de um conjunto consistente de APIs (RED HAT, 2018b). Por esses motivos, o OpenStack foi a plataforma de computação em nuvem selecionada para compor o O2CMF. O **Tacker** é um módulo do OpenStack que habilita a implantação e orquestração serviços de rede usando VNFs (OPENSTACK, 2018m). Por ser compatível com a especificação de NFV proposta por (ETSI NFV ISG, 2014) e integrado ao OpenStack, o Tacker foi selecionado para compor o orquestrador do O2CMF. Como base para o desenvolvimento do AM, o Fed4FIRE recomenda o **GCF** (FED4FIRE, 2012b).

A Tabela 3 resume as características da arquitetura do O2CMF, apresentando a função de cada componente, os requisitos que ele deve implementar e a tecnologia que servirá de base para seu desenvolvimento.

Tabela 3 – Componentes do O2CMF.

Componente	Propósito	Requisitos	Tecnologia
AM	Expor recursos de NFV ao Fed4FIRE.	R01, R02, R05, R06 e R07	GCF
Orquestrador	Habilitar o controle de experimento em NFV.	R03, R04	Tacker
Plataforma de Computação em Nuvem	Gerenciar a infraestrutura virtual.	R08, R09	OpenStack

### 4.3 Processo de Implementação do O2CMF

Habilitar a experimentação federada em NFV é um objetivo complexo. Há a necessidade de manter-se aderente a duas especificações distintas (SFA e NFV), que, embora tenham sido projetadas para fins distintos, têm influência sobre itens similares (representação dos recursos e serviços, tipos de dados e interações entre os componentes). Dessa forma, optamos pelo desenvolvimento incremental do O2CMF, que possibilitou controlar a complexidade do desenvolvimento e apresentar entregas periódicas para acompanhamento por parte do consórcio FUTEBOL. O processo de desenvolvimento foi dividido em quatro etapas:

1. **Integração da nuvem ao Fed4FIRE:** Essa etapa tem o objetivo de implantar a infraestrutura de nuvem e desenvolver uma versão inicial do AM que oferece VMs. Sua contribuição é a adaptação do modelo de dados da federação para a nuvem.
2. **Integração de NFV ao Fed4FIRE:** O objetivo dessa etapa é habilitar a criação de VNFs, inserindo funcionalidades de orquestração de NFV. Sua contribuição é a integração entre o bloco MANO da arquitetura NFV e a federação.
3. **Suporte à experimentadores iniciantes:** O objetivo dessa etapa é habilitar a criação de experimentos a partir de modelos de VNFs. Sua contribuição é reduzir a complexidade e aumentar a automação na criação de experimentos em NFV.
4. **Experimentação convergente com NFV:** Essa etapa tem o objetivo de integrar NFV com outras tecnologias a fim de habilitar a experimentação entre recursos heterogêneos, formando uma prova de conceito da interoperabilidade esperada das redes 5G.

A Tabela 4 apresenta a relação entre as etapas de desenvolvimento e os requisitos trabalhados. Alguns requisitos aparecem em mais de uma etapa porque serão completados de forma incremental. Esse é o caso dos requisitos R01 e R02, que são intrinsecamente relacionados, pois cada etapa adicionará novos recursos ao catálogo. No caso do requisito R05, que envolve dois perfis de experimentadores, o perfil Especialista será trabalhado na etapa 2 e o perfil Iniciante na etapa 3.

Tabela 4 – Etapas de desenvolvimento do O2CMF.

<b>Etapas de Desenvolvimento</b>	<b>Requisitos Trabalhados</b>
Integração da nuvem ao Fed4FIRE	R01, R02, R06, R07, R08, R09
Integração de NFV ao Fed4FIRE	R01, R02, R03, R04, R05
Suporte à experimentadores iniciantes	R01, R02, R05
Experimentação convergente com NFV	–

Nas seções a seguir, serão descritas as etapas de codificação do O2CMF, apresentando o processo de integração dos requisitos e decisões de projeto.

### 4.3.1 Integração da nuvem ao Fed4FIRE

Nessa etapa, foi implantada a nuvem e desenvolvida uma versão básica do AM, que oferece ao experimentador VMs com conectividade LAN. Os requisitos trabalhados nessa etapa foram:

- **R01** – Oferecer ao experimentador um catálogo de serviços contendo modelos de VNF, diferentes perfis de recursos e imagens de SO.
- **R02** – Reservar recursos com base no catálogo, possibilitando a instanciação automática de VNFs.
- **R06** – Implementar um AM compatível com a API SFA.
- **R07** – Suportar as credenciais geradas pelo Fed4FIRE.
- **R08** – Permitir que o operador do *testbed* estabeleça políticas de compartilhamento de recursos.
- **R09** – Prover isolamento entre os experimentos.

O AM exemplo distribuído com GCF, foi a base para a criação do AM do O2CMF e possibilitou a implementação do requisito **R06**. Como se trata apenas de um “esqueleto” desenvolvido em Python, foi necessário desenvolver outros pacotes a fim de prover funcionalidades essenciais para a gestão de recursos reais, como persistência, controle de acesso e comandos de interação com a nuvem. Os pacotes que compõem o AM são representados na Figura 23. Além disso, o GCF permite configurar de quais *Clearinghouses* o AM pode aceitar credenciais. Dessa forma, bastou adicionar o certificado do Fed4FIRE fornecido pela equipe de suporte para implementar o requisito **R07**.

Enquanto o pacote **GCF** provê a estrutura necessária para envio/recebimento das mensagens SFA, o pacote **Dispatchers** implementa cada um dos métodos da API SFA. Os demais pacotes proveem serviços para apoiar a implementação desses métodos. O pacote **TestbedResources** contém as classes que modelam os recursos do *testbed*. O pacote **Database** é responsável pela persistência das reservas. O pacote **Drivers** é responsável pela interação com a nuvem. O pacote **ProxySSH** habilita o acesso do experimentador aos recursos. Ao longo dessa seção serão apresentados detalhes de cada pacote, assim como as interações entre os pacotes para a instanciação de um experimento.

O pacote **Dispatchers** é a parte central do AM. Nele está contida uma classe que herda do AM exemplo (do pacote GCF), sobrescrevendo os métodos da API SFA para adaptá-los à proposta do O2CMF. Dessa forma, esse pacote tem a função de mediar as interações com a federação, recebendo as requisições SFA e as transformando em comandos menos abstratos direcionados aos demais pacotes.



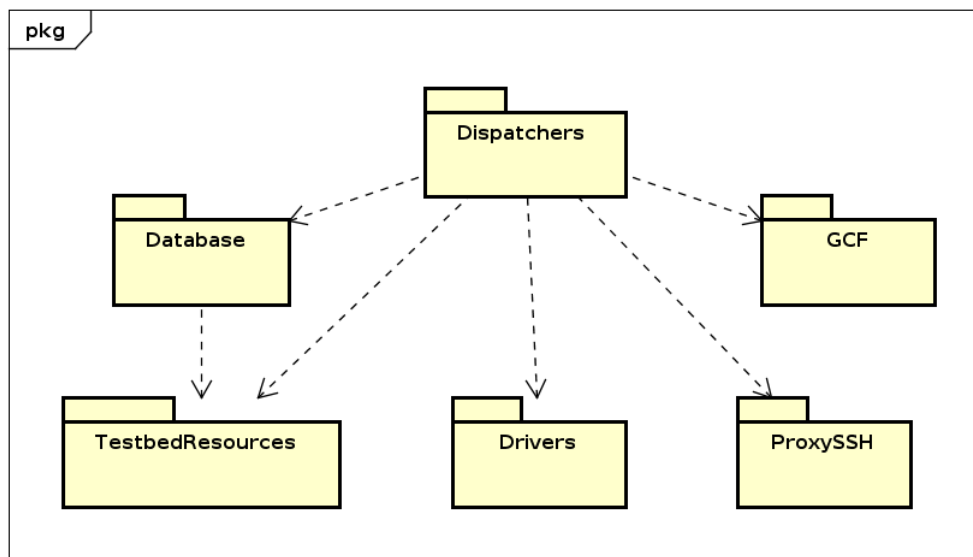


Figura 23 – Pacotes que compõem o AM.

No contexto do O2CMF, os recursos são a infraestrutura virtual (computação, armazenamento e rede), que é manipulada através dos serviços do OpenStack. O pacote **Dispatchers** contém uma interface que define as ações que podem ser solicitadas pelo **Dispatchers** ao OpenStack e uma classe que implementa essa interface. Inicialmente, para interagir com as APIs do OpenStack, foi utilizado o `Openstackclient`. Por ter um escopo amplo e prover uma interação mais “direta” com a API *RESTful*, ele recebe mais atualizações e oferece uma gama maior de funcionalidades. No entanto, ele não é recomendado para o desenvolvimento de aplicações, por retornar dicionários ao invés de objetos que representam os recursos. Por essa razão, ele foi substituído pelo `Openstacksdk`. Porém, a documentação pobre, métodos que não apresentavam o comportamento esperado e as quebras a cada nova versão do OpenStack, levaram ao seu abandono. A partir de então, vem sendo utilizado o `Shade` que, apesar do escopo menor, se mostrou mais estável.

O pacote **TestbedResources** contém um conjunto de classes que representam recursos gerenciados pelo *testbed*. Nessa versão, esses recursos são **ComputeNode**, **VirtualMachine**, **Image** e **User**. Essas classes são utilizadas internamente pelo AM para manipular os dados referentes ao recursos para experimentação.

Devido ao requisito de ser compatível com a arquitetura SFA (R06), a especificação de recursos para experimentação deve ser feita através de RSpec. Sendo assim, a RSpec é utilizada na interação com a federação e usuários, enquanto as classes no pacote **TestbedResources** são utilizadas na interação entre pacotes do AM. A representação dos recursos na RSpec difere da adotada pelas classes do modelo orientado a objetos. No que diz respeito à representação adotada na RSpec, o caso dos recursos VM e rede LAN subjacente, o Fed4FIRE estabelece uma representação padronizada (FED4FIRE, 2014). A responsabilidade por traduzir a representação da RSpec para a representação em objeto (e



vice-versa) é do pacote `Dispatchers`.

Quando o *ListResources* é invocado, o `Dispatchers` cria um objeto `ComputeNode`, que representa o nó de computação do OpenStack, e requisita ao `Drivers` todos os *flavors* e imagens de SO disponíveis na nuvem, a fim de montar uma lista de recursos. Com base nessa lista, o AM prepara a *Advertisement* RSpec, traduzindo da representação em objeto dos recursos para elementos XML da RSpec. Essa *Advertisement* RSpec é apresentada na Listagem 4.1.

```

1 <node component_id="urn:publicid:IDN+futebol.inf.ufes.br+node+compute_node"
2   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+am"
3   component_name="compute_node"
4   exclusive="false">
5   <sliver_type name="vm-m1.large">
6     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu17"/>
7     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+centos7"/>
8     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+fedora27"/>
9     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
10    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+cirros"/>
11  </sliver_type>
12  <sliver_type name="vm-m1.medium">
13    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu17"/>
14    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
15    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+cirros"/>
16  </sliver_type>
17  <sliver_type name="vm-m1.small">
18    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
19    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+cirros"/>
20  </sliver_type>
21  <available now="true"/>
22 </node>

```

Listagem 4.1 – Excerto da *Advertisement* RSpec.

A *Advertisement* RSpec constitui o catálogo do *testbed* (requisito **R01**) e é com base nela que o experimentador prepara sua *Request* RSpec (requisito **R02**). Quando o *Allocate* é invocado, `Dispatchers` faz o *parser* da RSpec transformando elementos XML (definidos pela federação) em recursos (internos ao *testbed*). A Listagem 4.2 apresenta a especificação de uma VM nomeada como “instance1” com *flavor* `m1.small`, imagem do Ubuntu 16 e uma interface de rede (com o *fixed* IP 192.168.0.3).

```

1 <node client_id="instance1" exclusive="false"
2   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
3   <sliver_type name="vm-m1.small">
4     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
5   </sliver_type>
6   <interface client_id="node0:eth0">
7     <ip address="192.168.0.3" netmask="255.255.255.0" type="ipv4"/>
8   </interface>
9 </node>

```

Listagem 4.2 – Excerto da *Request* RSpec, exibindo a definição de uma VM.

A RSpec também pode conter a especificação de uma *bridge*, apresentada na Listagem 4.3, que representa uma LAN. As VMs são conectadas à *bridge* através de *links*, conforme apresentado na Listagem 4.4. Note que as pontas do *link* são especificadas através da *tag interface\_ref*. A informação sobre a VM e sua configuração de rede (*bridge* e *links*) são aglutinadas pelo *Dispatchers* em um objeto da classe *VirtualMachine*. Por fim, para que o AM consiga identificar posteriormente a quem cedeu os recursos e habilitar o acesso a estes, o usuário é representado através da classe *User*.

```

1 <node client_id="bridge0" exclusive="false"
2   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
3   <sliver_type name="bridge"/>
4   <interface client_id="bridge0:eth0"/>
5   <interface client_id="bridge0:eth1"/>
6   <interface client_id="bridge0:eth2"/>
7 </node>

```

Listagem 4.3 – Excerto da *Request* RSpec, exibindo a definição de uma *bridge*.

```

1 <link client_id="link1">
2   <component_manager name="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm"/>
3   <interface_ref client_id="instance1:eth0"/>
4   <interface_ref client_id="bridge0:eth1"/>
5   <link_type name="lan"/>
6 </link>

```

Listagem 4.4 – Excerto da *Request* RSpec, exibindo a definição de um *link*.

O pacote *Database* contém classes responsáveis pela gestão de dados persistidos e mapeamento objeto-relacional. Dessa forma, sempre que um método SFA é invocado, *Dispatchers* recorre a *Database* para recuperar ou armazenar os dados das reservas. A escolha pelo modelo relacional foi motivada pela natureza estruturada dos dados a serem persistidos. Contudo, a representação das entidades no modelo entidade-relacionamento foi simplificada em relação ao modelo de classes do GCF. Essa simplificação teve o objetivo de facilitar a recuperação dos dados e manter sua consistência, pois muitas classes apresentam atributos redundantes e dependência cíclica. Além disso, a divisão de responsabilidades entre as classes nem sempre é bem delimitada e o conceito de herança é massivamente aplicado. O resultado é apresentado na Figura 24, em que é possível observar a presença de entidades oriundas do pacote *TestbedResources* (*user*, *vm*) relacionadas com entidades oriundas do pacote GCF (*slice*, *sliver*, *resource*), além de entidades exclusivas do modelo relacional (*cloud\_request*, *net\_iface*) criadas para facilitar a recuperação dos dados. A gestão da persistência foi feita utilizando o padrão de projeto DAO (*Data Access Object*) em conjunto com a biblioteca *SQLAlchemy*<sup>1</sup> para mapeamento objeto-relacional. O sistema gerenciador de banco de dados escolhido foi o MySQL, em função de suas ferramentas para *design* e gestão de bases de dados.

<sup>1</sup> <<https://www.sqlalchemy.org/>>

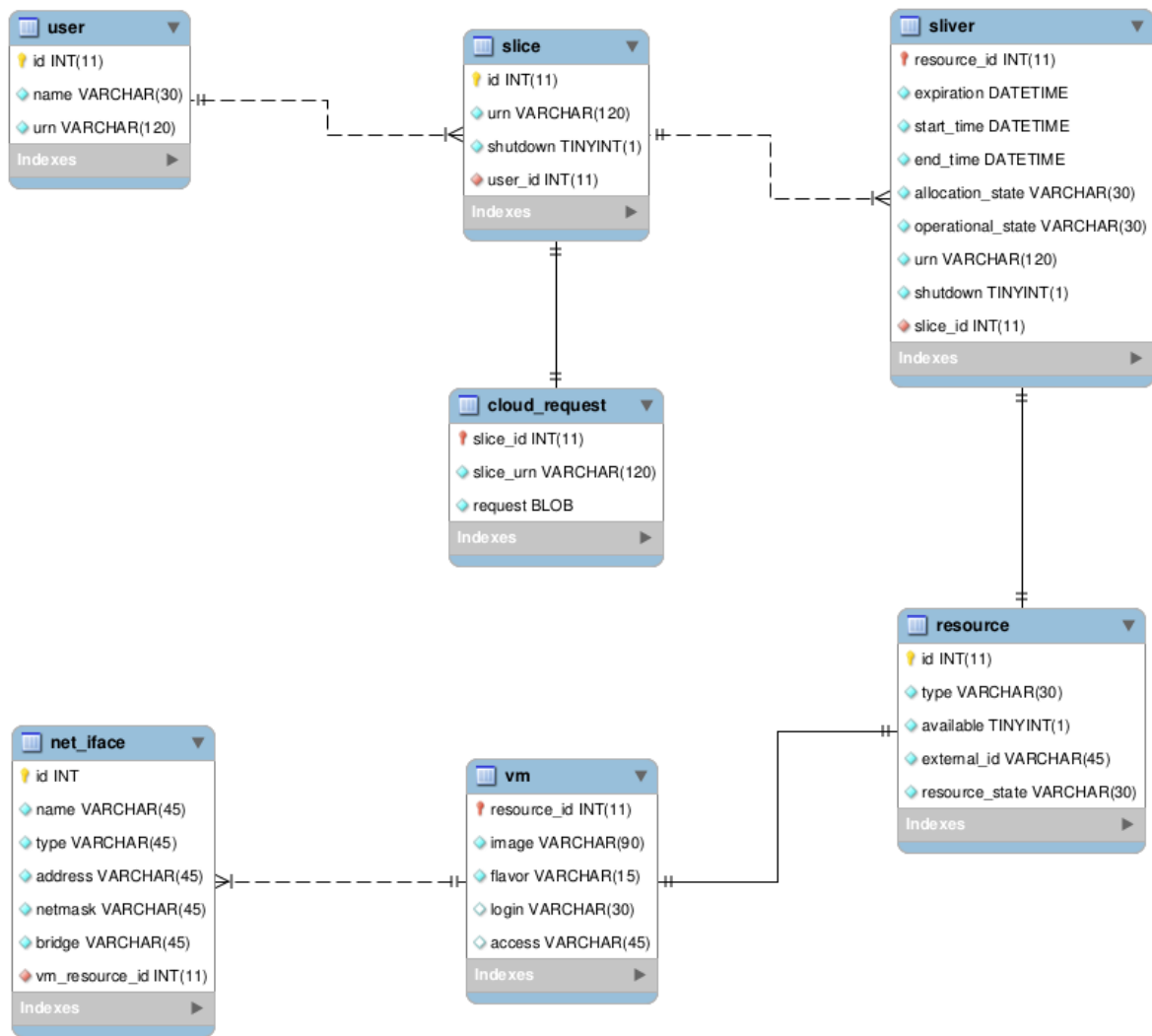


Figura 24 – Modelo entidade-relacionamento na primeira versão do O2CMF.

Conforme explicado na Seção 3.1.1, para dar acesso às VMs criadas no *testbed*, é preciso configurar um *pool* de *floating* IPs no OpenStack. Devido à inviabilidade de prover IPs públicos para as VMs dos experimentadores, optamos por utilizar uma faixa de IPs privados. Esses IPs privados são acessíveis a partir de outras redes internas ao *data center*, como aquela que provê conectividade ao AM. No entanto, essa configuração não é suficiente para habilitar o acesso para o público externo (experimentadores). Por essa razão, foi implantada uma VM no *data center* com a função de atuar como *proxy* SSH. Essa VM *proxy* tem conectividade com os *floating* IPs, conforme ilustrado na Figura 25, e também está associada a um IP público através de NAT (*Network Address Translation*). Dessa forma, ela serve de ponte para o acesso do experimentador às suas VMs. Além disso, foi desenvolvido o pacote ProxySSH, que fica instalado na VM *proxy*.

O pacote ProxySSH auxilia no estabelecimento de acesso externo aos recursos de computação no *testbed*. Esse pacote expõe uma API *RESTful* que permite adicionar/remover o par (*username*, *chave pública*) no sistema de contas de usuário da VM *proxy*.

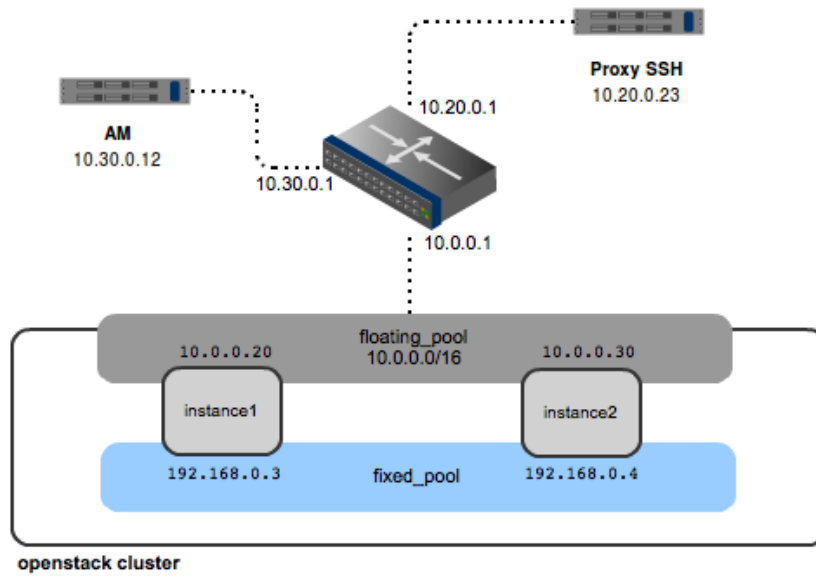


Figura 25 – Representação da topologia de rede do *testbed*. Adaptado de: (MIRANTIS, 2012).

Dessa forma, sempre que um experimento passa pelo método *Provision*, o Dispatchers obtém de Drivers o *floating* IP de cada VM criada e, em seguida, requisita ao ProxySSH a inclusão dos dados de *login* do experimentador. Como resultado, o Dispatchers inclui os dados de acesso na *Manifest* RSpec, apresentada na Listagem 4.5 (através da *tag services*). Quando o método *Delete* for invocado, o Dispatchers irá requisitar ao Drivers a liberação dos recursos, solicitará a Database a remoção da reserva do banco de dados e ao ProxySSH solicitará a remoção da conta do experimentador.

```

1 <node client_id="node0" exclusive="false"
2   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
3   <sliver_type name="vm-m1.small">
4     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
5   </sliver_type>
6   <interface client_id="node0:eth0">
7     <ip address="192.168.0.3" netmask="255.255.255.0" type="ipv4"/>
8   </interface>
9   <services xmlns:proxy="http://jfed.iminds.be/proxy/1.0">
10     <proxy:proxy proxy="isabella@futebol.inf.ufes.br:22"
11       for="isabella@10.0.0.20:22"/>
12     <login authentication="ssh-keys" hostname="futebol.inf.ufes.br"
13       port="22" username="isabella"/>
14     <login authentication="ssh-keys" hostname="10.0.0.20"
15       port="22" username="isabella"/>
16   </services>
17 </node>

```

Listagem 4.5 – Excerto da *Manifest* RSpec, exibindo uma VM com seus dados de acesso.

Além das funcionalidades providas através da GENI API, o O2CMF oferece isolamento entre os experimentos (requisito **R09**) a fim de dar privacidade e segurança ao experimentador. Cada experimento provisionado se transforma em um projeto no OpenStack. Sendo assim, o experimentador tem visibilidade apenas dos recursos atribuídos ao seu projeto. Além disso, cada *bridge* da RSpec se transforma em uma rede LAN virtual. Dessa forma, as VMs do experimentador possuem IPs privados acessíveis somente na rede local atribuída ao seu projeto, assegurando o isolamento de tráfego. Para reforçar, as imagens de disco utilizadas nas VMs foram preparadas com o pacote cloud-init<sup>2</sup>. Assim, a autenticação com senha para conexão SSH foi desabilitada e o cloud-init permitiu injetar a chave do usuário na VM, de modo que ele seja o único com acesso.

Para evitar que os recursos da nuvem se esgotem subitamente, o OpenStack permite a configuração de *quotas* e a customização de *flavors* e imagens. *Quotas* são limites operacionais para um projeto definidos pelo administrador da nuvem (OPENSTACK, 2018j). Dessa forma, o operador do *testbed* pode especificar que cada projeto criado poderá conter até 5 VMs ou utilizar até 30 GB de disco, por exemplo. Os *flavors*, assim imagens e outros recursos no OpenStack, podem ser associados a propriedades (chamadas de metadados), a fim de definir seu comportamento (OPENSTACK, 2018g). Assim, o administrador da nuvem pode alterar os metadados do *flavor* de forma a limitar o máximo de largura de banda consumida, por exemplo. Dessa maneira, o operador do *testbed* obtém um controle fino da granularidade dos recursos oferecidos (requisito **R08**).

O trabalho realizado nessa etapa inicial de desenvolvimento se encontra publicado em (CERAVOLO et al., 2017). A versão inicial do AM foi desenvolvida com base na GENI API (requisito R06) e configurada para utilizar a *Clearinghouse* do Fed4FIRE (requisito R07). Nessa etapa, o catálogo exposto pelo AM consistiu de perfis de recursos (*flavors*) e imagens de SO (requisito R01), permitindo a reserva e provisionamento de VMs com conectividade LAN (requisito R02). A implantação do OpenStack, além viabilizar a criação das VMs, possibilitou ao operador do *testbed* definir políticas de compartilhamento de recursos (requisitos R08) e proveu isolamento entre os experimentos (requisito R09). Dessa forma, atingimos o objetivo dessa etapa, habilitando a interação com a nuvem através da federação. As próximas etapas de desenvolvimento que tomaram como base a estrutura de pacotes desenvolvida nessa etapa para incluir funcionalidades de NFV.

### 4.3.2 Integração de NFV ao Fed4FIRE

Nessa etapa, introduzimos NFV no ambiente da federação. Tendo em vista as necessidades de um usuário de perfil especialista, foi incluído o Orquestrador e oferecida a possibilidade reservar infraestrutura para criar VNFs customizadas. Os requisitos trabalhados nessa etapa foram:

---

<sup>2</sup> <<https://cloud-init.io/>>

- **R01** – Oferecer ao experimentador um catálogo de serviços contendo modelos de VNF, diferentes perfis de recursos e imagens de SO.
- **R02** – Reservar recursos com base no catálogo, possibilitando a instanciação automática de VNFs.
- **R03** – Possibilitar ao experimentador especificar políticas para a gestão da VNF.
- **R04** – Permitir execução automatizada do cenário do experimento.
- **R05** – Prover suporte a dois perfis de usuários: Iniciante e Especialista.

Apesar da padronização da representação de VM e rede LAN na RSpec, o Fed4FIRE dá a liberdade aos *testbeds* de definirem a representação dos demais tipos de recursos oferecidos. Isso foi essencial para habilitar a inclusão de novos recursos na federação. Além dos *flavors* e imagens, o catálogo do O2CMF passou a exibir também a quantidade de recursos virtuais “brutos” disponíveis na infraestrutura virtual. Através da *tag vim*, exibida no final da Listagem 4.6, o experimentador fica informado sobre o quanto de memória, CPUs virtuais e armazenamento está disponível na nuvem.

```

1 <node component_id="urn:publicid:IDN+futebol.inf.ufes.br+node+compute_node"
2   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+am"
3   component_name="compute_node" exclusive="false">
4   <sliver_type name="vm-m1.large">
5     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+nfv-orchestrator"/>
6     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
7     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+cirros"/>
8   </sliver_type>
9   <sliver_type name="vm-m1.medium">
10    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+nfv-orchestrator"/>
11    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
12    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+cirros"/>
13  </sliver_type>
14  <sliver_type name="vm-m1.small">
15    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+nfv-orchestrator"/>
16    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
17    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+cirros"/>
18  </sliver_type>
19  <available now="true"/>
20 </node>
21 <vim component_id="urn:publicid:IDN+futebol.inf.ufes.br+tenant+openstack"
22   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+am"
23   component_name="openstack" exclusive="false">
24   <sliver_type name="tenant">
25     <properties disk="1143" ram="147" vcpu="28"/>
26   </sliver_type>
27   <available now="true"/>
28 </vim>

```

Listagem 4.6 – Excerto da *Advertisement* RSpec.

Conforme apresentado na Listagem 4.6, o tipo de *sliver* provido pelo recurso *vim* é *tenant*, uma alusão ao projeto no OpenStack. O objetivo em oferecer o projeto como recurso é permitir que o experimentador reserve parte da infraestrutura “bruta” para implantar VNFs totalmente definidas por ele. Sendo assim, para preparar a *Request* RSpec, o usuário deve estimar quanta memória, CPU e disco será preciso reservar. Para auxiliá-lo, o site do *testbed* UFES<sup>3</sup> traz a descrição de cada um dos *flavors* e imagens, além de tutoriais. De posse desses dados, o experimentador insere um elemento *vim* na RSpec, conforme exemplificado na Listagem 4.7.

```

1 <vim client_id="vim0" exclusive="false"
2   component_manager_id="urn:publicid:IDN+futebolufes+authority+am">
3   <sliver_type name="tenant">
4     <properties disk="120" ram="5" vcpu="3"/>
5   </sliver_type>
6 </vim>

```

Listagem 4.7 – Excerto da *Request* RSpec, exibindo a definição de um *Tenant*.

Considerando a arquitetura NFV, a etapa de desenvolvimento anterior preparou a infraestrutura virtual (NFVI) e estabeleceu parte do bloco MANO, através do OpenStack (atuando como VIM). Para que fosse possível implantar e orquestrar VNFs, era preciso completar o bloco MANO, estabelecendo o VNFM e o NFVO. A solução proposta foi incluir o Tacker, que provê ambos. A definição de VNFs através da linguagem TOSCA habilita tanto a definição de políticas, como o monitoramento e a execução automatizada de ações no experimento. Isso permitiu atender ao requisito **R03** – que diz respeito a possibilitar ao usuário definir políticas para a gestão de sua VNF – e ao requisito **R04** – que diz respeito a execução automatizada do experimento.

É possível interagir com o Tacker através de um cliente CLI, o Tackerclient<sup>4</sup>. Porém, para que ele pudesse ser visível ao usuário, através do catálogo do O2CMF, preparamos uma imagem que contém o Tackerclient instalado (apresentada no catálogo como *nfv-orchestrator*). Para que o experimentador consiga criar suas VNFs, ele precisa incluir em sua *Request* RSpec a especificação da VM de orquestração, exibida na Listagem 4.8.

```

1 <node client_id="orchestrator" exclusive="false"
2   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+am">
3   <sliver_type name="vm-m1.small">
4     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+nfv-orchestrator"/>
5   </sliver_type>
6 </node>

```

Listagem 4.8 – Excerto da *Request* RSpec, exibindo a definição da VM de orquestração.

Quando o experimento for provisionado, será criado um projeto no OpenStack, cujos recursos (memória, CPU e disco) serão ajustados para as quantidades indicadas na

<sup>3</sup> <<http://futebol.inf.ufes.br/>>

<sup>4</sup> <<https://pypi.org/project/python-tackerclient/>>



reserva, através do uso de *Quotas*. Também é criada uma LAN no projeto, com a finalidade de prover conectividade para as futuras VNFs. A VM de orquestração recebe um *fixed* IP dessa LAN e também é configurada com o *RC File* do usuário, de modo a preparar o Tackerclient para o experimentador.

Após a instanciação e configuração inicial dos recursos, o usuário poderá fazer SSH na VM de orquestração. A partir desse momento, o experimentador poderá criar suas VNFs, descrevendo-as em TOSCA e usando o Tackerclient para instanciá-las<sup>5</sup>. Após a instanciação, o experimentador poderá consultar os endereços de IP, através do Tackerclient<sup>6</sup>, e realizar o acesso SSH a partir da VM de orquestração. A Figura 26 ilustra o processo de orquestração do experimento.

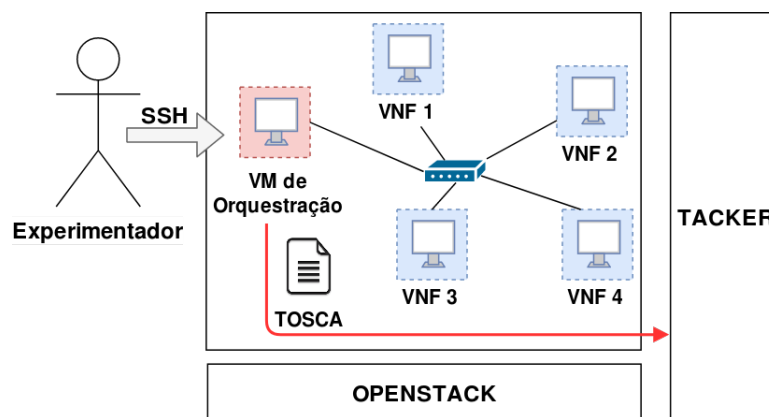


Figura 26 – Orquestração de experimento.

A estrutura de pacotes proposta na versão anterior foi mantida. Apenas um novo recurso foi incluído, através da classe *Tenant*, e feitas atualizações em *Dispatchers*, *Drivers* e *Database* para suportar esse novo recurso. O modelo entidade-relacionamento, exibido na Figura 27, também foi atualizado com a entidade *tenant*.

O trabalho realizado nessa etapa se encontra publicado em (CERAVOLO et al., 2018). As funcionalidades de NFV desenvolvidas nessa fase tiveram como foco atender usuários com perfil especialista (requisito **R05**). As VNFs podem ser definidas através de *script* TOSCA. Entretanto, não é viável inserir TOSCA na RSpec, pois TOSCA é uma linguagem cuja indentação é baseada em espaços ou tabulações. Como a RSpec é enviada de forma comprimida, a indentação do *script* seria quebrada. Imitar a estrutura da linguagem TOSCA na RSpec também não é viável pela complexidade do *parsing* e validação que teriam de ser realizados pelo AM. Ainda que o TOSCA fosse integrado à RSpec, ocorreria uma violação do protocolo estabelecido por SFA, uma vez que o Tacker acopla o provisionamento da VNF à definição de suas políticas (que é uma funcionalidade de orquestração). Em função dessas limitações, optamos por oferecer a infraestrutura “bruta” no catálogo (requisito

<sup>5</sup> O tutorial de uso do O2CMF se encontra em <<http://futebol.inf.ufes.br/projeto/index.php/tutorials/>>.

<sup>6</sup> A descrição detalhada dos comandos suportados pelo Tackerclient está disponível em <<http://futebol.inf.ufes.br/projeto/index.php/vnf-templates/>>.



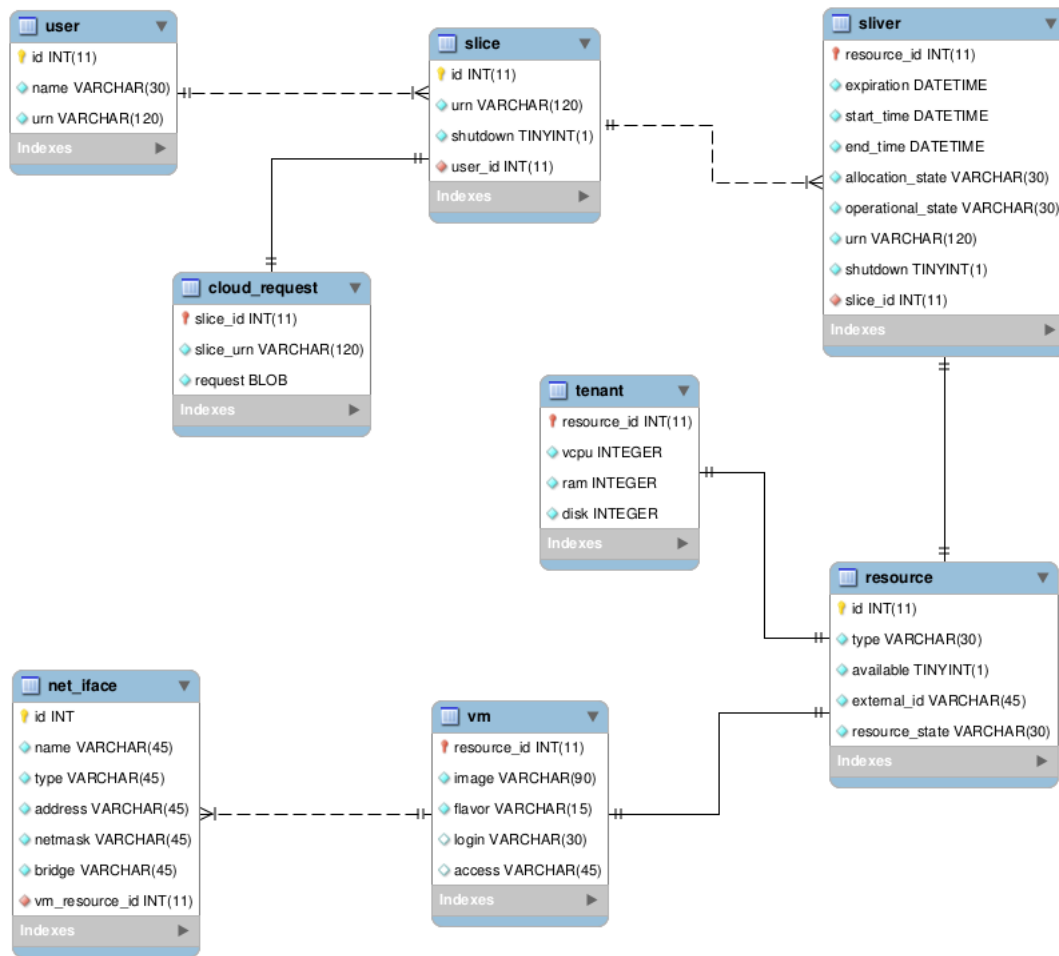


Figura 27 – Modelo entidade-relacionamento na segunda versão do O2CMF.

R01) para que o experimentador possa utilizá-la para a implementação de suas VNFs (requisito R02). A inclusão do Orquestrador possibilitou utilizar um *script* TOSCA para criar VNFs customizadas, contendo políticas de monitoramento e *scaling* (requisitos R03 e R04). Dessa forma, o O2CMF oferece programabilidade e flexibilidade para a realização de experimentos complexos em NFV.

#### 4.3.3 Suporte à experimentadores iniciantes

Nessa etapa, ampliamos as funcionalidades de NFV, com ênfase na usabilidade para os usuários de perfil iniciante. Os requisitos trabalhados nessa etapa foram:

- **R01** – Oferecer ao experimentador um catálogo de serviços contendo modelos de VNF, diferentes perfis de recursos e imagens de SO.
- **R02** – Reservar recursos com base no catálogo, possibilitando a instanciação automática de VNFs.
- **R05** – Prover suporte a dois perfis de usuários: Iniciante e Especialista.

Para completar o requisito **R01**, faltava a inclusão de modelos de VNF no catálogo. Conforme visto na Seção 3.2.1, a linguagem TOSCA pode ser aplicada em NFV para descrever VNFs. Porém, a escrita de um descritor de VNF (modelo) é uma tarefa que exige conhecimento prévio da linguagem TOSCA, do OpenStack e de NFV. Dessa forma, as funcionalidades do O2CMF atingiam apenas um público muito especializado. Tornar NFV mais “acessível”, atraindo um público mais amplo (e possivelmente interdisciplinar), foi a motivação para a inclusão de modelos de VNF no catálogo do O2CMF. A *Advertisement* RSpec, que desde a primeira versão exibe os *flavors* e imagens disponíveis, passou a exibir também um conjunto de modelos de VNF (através da *tag nfv*), conforme observado na Listagem 4.9.

```

1 <node component_id="urn:publicid:IDN+futebol.inf.ufes.br+node+compute_node"
2   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+am"
3   component_name="compute_node" exclusive="false">
4   <sliver_type name="vm-m1.large">
5     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+o2cmf-orchestrator"/>
6     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
7     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+cirros"/>
8   </sliver_type>
9   <sliver_type name="vm-m1.medium">
10    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+o2cmf-orchestrator"/>
11    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
12    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+cirros"/>
13  </sliver_type>
14  <sliver_type name="vm-m1.small">
15    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+o2cmf-orchestrator"/>
16    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
17    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+cirros"/>
18  </sliver_type>
19  <available now="true"/>
20 </node>
21 <nfv component_id="urn:publicid:IDN+futebol.inf.ufes.br+nfv+nfv_catalog"
22   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+am"
23   component_name="nfv_catalog" exclusive="false">
24   <sliver_type name="vnf">
25     <template name="web-server"/>
26     <template name="firewall"/>
27     <template name="load-balancer"/>
28     <template name="test"/>
29   </sliver_type>
30   <available now="true"/>
31 </nfv>

```

Listagem 4.9 – Excerto da *Advertisement* RSpec.

No entanto, dispor apenas de *scripts* TOSCA predefinidos é inflexível e reduz a aplicabilidade das VNFs nos cenários pretendidos por cada experimentador. A solução proposta foi permitir a modificação de parâmetros do *script* TOSCA, cujos novos valores podem ser especificados na *Request* RSpec (conforme exemplificado Listagem 4.10 na *tag parameter*). Para habilitar essa modificação, foi necessário utilizar a base de dados do AM para armazenar os *scripts* TOSCA (modelos de VNF), pois o catálogo de NFV interno ao

Tacker não permite que o TOSCA seja alterado após sua inclusão. As mudanças feitas no esquema do banco para acomodar os modelos de VNF são ilustradas na Figura 28. A entidade `template` representa o modelo de VNF e a entidade `parameter` representa os parâmetros que podem ser customizados no modelo. Ambas precisam ser previamente populadas pelo operador do *testbed*.

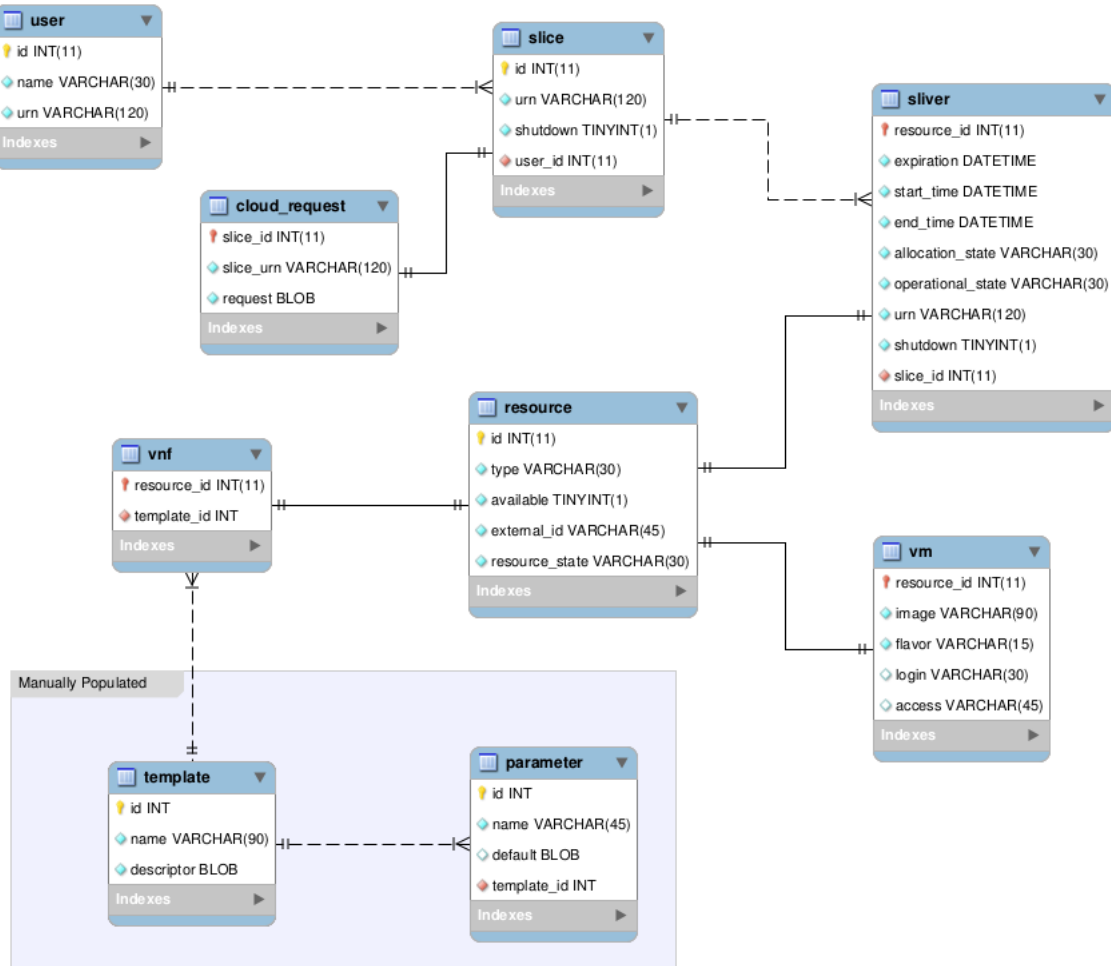


Figura 28 – Modelo entidade-relacionamento na terceira versão do O2CMF.

Apesar da VNF passar a ser especificada no ato da reserva, a VM orquestradora ainda é necessária (conforme apresentado na Listagem 4.10), permitindo descobrir os *fixed* IPs das VDUs e fazer acesso SSH. Isso ocorre porque não há como determinar previamente um *fixed* IP para a VDU (lembre que o modelo é genérico e pode ser instanciado *n* vezes, até dentro de um mesmo projeto) e por conta da dificuldade em controlar o endereçamento das VDUs surgidas no processo de *scaling*.

```

1 <node client_id="node0" exclusive="false"
2   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+am">
3   <sliver_type name="vm-m1.small">
4     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+o2cmf-orchestrator"/>
5   </sliver_type>
6 </node>

```

```
7 <nfv client_id="my_vnf"  
8   component_manager_id="urn:publicid:IDN+futebolufes+authority+am"  
9   exclusive="false">  
10  <sliver_type name="vnf">  
11    <template name="web-server">  
12      <parameter name="max_instances" value="7"/>  
13    </template>  
14  </sliver_type>  
15 </nfv>
```

Listagem 4.10 – Excerto da *Request* RSpec.

Durante o provisionamento de um experimento contendo modelos de VNFs (requisito **R02**), o pacote **Drivers** precisa interagir também com o Tacker, visto que as funcionalidades de NFV estão fora do escopo do Shade. Usar o Tackerclient não era possível, pois seria preciso configurar no AM os dados do *RC file* do usuário. Por essa razão, foram implementadas chamadas *REST* para a API do Tacker e do Keystone. Esses novos métodos do pacote **Drivers** são invocados sempre que é preciso provisionar um modelo, obtendo autorização com o Keystone para então solicitar a criação/remoção da VNF com Tacker. Desse modo, sempre que um modelo de VNF precisar ser instanciado, o AM recuperará o TOSCA do banco de dados (alterará os parâmetros, se necessário) e interagirá com o orquestrador (Tacker) para criar a VNF. Após o provisionamento, o experimentador deve acessar a VM orquestradora e entrar com o comando `tacker vnf-list` para obter uma listagem das VNFs instanciadas e o endereço IP de cada VDU.

A inserção do modelos no catálogo do O2CMF (requisito R01) possibilitou a implantação automática de VNFs (requisito R02), além de contornar as restrições mencionadas anteriormente para inclusão de TOSCA na RSpec. O uso de modelos beneficia, principalmente, os usuários iniciantes em NFV (requisito R05), pois abstrai a complexidade da escrita do *script* TOSCA enquanto agrega automação ao processo de construção do experimento. Além disso, a possibilidade de configurar parâmetros do modelo através da RSpec, permite oferecer um pouco de flexibilidade e programabilidade sem abrir mão da simplicidade.

#### 4.3.4 Experimentação convergente com NFV

Essa última etapa surgiu da necessidade de integrar NFV com recursos provenientes de outros domínios tecnológicos, a fim de habilitar a experimentação convergente. A integração visa prover um ambiente onde recursos heterogêneos são interoperáveis, favorecendo o desenvolvimento das redes 5G. Apesar de todos os requisitos do O2CMF estarem completos na etapa anterior, sua contribuição ainda estava isolada. Para mudar esse cenário, o O2CMF foi inserido em um *showcase* do FUTEBOL que demonstra o controle remoto de robô através de rede sem fio.

O cenário da demonstração é composto pelo *data center* (abrigando a nuvem

federada pelo O2CMF) e o espaço inteligente<sup>7</sup>. O espaço inteligente, ilustrado na Figura 29, é um ambiente com 4 câmeras, 4 *access points*, 1 *switch* OpenFlow e 1 robô móvel sem inteligência local. O objetivo da demonstração é fazer o robô seguir uma trajetória especificada pelo experimentador. O robô contém apenas os componentes necessários para comunicação sem fio e execução de comandos de controle. As câmeras coletam imagens do ambiente e as enviam para a nuvem, onde estarão implantados serviços para processar as imagens, calcular a localização do robô e gerar comandos de controle. Os comandos são enviados de volta ao robô, que os recebe através de uma rede sem fio controlada via OpenFlow (MARQUES et al., 2017).

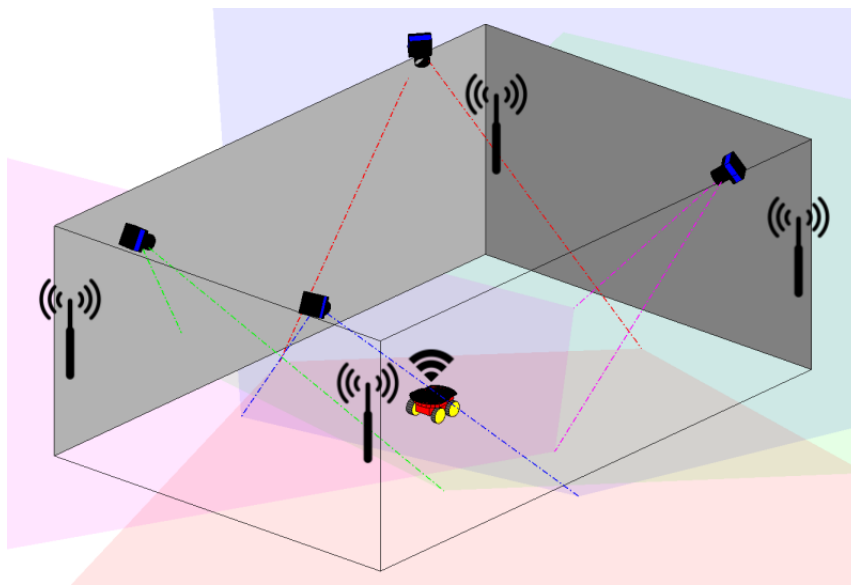


Figura 29 – Representação do espaço inteligente. Adaptado de: (MARQUES et al., 2017).

Integrar o espaço inteligente à federação exigiu ir além exibição de um novo recurso no catálogo. Para que essas aplicações de controle do espaço inteligente pudessem ser implantadas na nuvem e usufruírem dos benefícios de NFV (escalabilidade, políticas de monitoramento, etc), era necessário que elas fossem transformadas em imagens de disco. O grupo de pesquisa Viros<sup>8</sup>, desenvolvedor do espaço inteligente, encarregou-se dessa adaptação e forneceu a imagem de cada serviço. Além dos serviços que habilitam o deslocamento do robô, outras aplicações se fazem necessárias a fim de permitir ao experimentador programar a infraestrutura do espaço inteligente de modo a adaptá-lo ao cenário de seu experimento. Essas aplicações foram desenvolvidas por outros membros do grupo de pesquisa NERDS para atuar na orquestração da conectividade sem fio (baseado em (MARTINEZ et al., 2018)) e nuvem. Além dos orquestradores, foi desenvolvida uma aplicação que permite a telemetria do experimento (baseada em (SANTOS et al., 2018)).

<sup>7</sup> <<http://viros.ufes.br/Intelligent-Space/>>

<sup>8</sup> <<http://viros.ufes.br/>>

Essas aplicações foram disponibilizadas como imagem de disco. Desse modo, o catálogo de serviços foi atualizado (conforme Listagem 4.11) para incluir as seguintes imagens:

- **is\_camera\_gateway**: contém o serviço responsável pela configuração das câmeras (taxa de *frames* por segundo, padrão de cor, resolução) e aquisição de imagens.
- **is\_image\_processing**: oferece um serviço de tratamento das imagens das câmeras, detectando o robô nas imagens e fornecendo as coordenadas de sua localização.
- **is\_robot\_controller**: oferece um serviço que recebe como entrada a trajetória desejada para o robô e a localização deste. O serviço usa esses dados para gerar comandos de direção e aceleração, a fim de orientar o robô na realização do trajeto pretendido pelo experimentador.
- **is\_sdn\_controller**: contém o serviço de controle da conectividade sem fio. Esse serviço baseado em OpenFlow é responsável pela configuração das zonas cobertas pelos *access points*, canais e frequência. Além disso, esse serviço também implementa uma solução de *handover*.
- **is\_mpeg\_server**: compõe o serviço de *handover*.
- **experiment\_orchestrator**: reúne clientes dos orquestradores, permitindo a configuração dos recursos do experimento e a coleta de métricas.
- **is\_rabbit\_mq**: oferece um serviço de fila de mensagens. Esse serviço possibilita a comunicação entre os demais serviços.

```

1 <node component_id="urn:publicid:IDN+futebol.inf.ufes.br+node+compute_node"
2   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+am"
3   component_name="compute_node"
4   exclusive="false">
5   <sliver_type name="vm-m1.medium">
6     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+experiment_orchestrator"/>
7     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_image_processing"/>
8     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_camera_gateway"/>
9     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_rabbit_mq"/>
10    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_robot_controller"/>
11    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_sdn_controller"/>
12    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_mpeg_server"/>
13    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
14    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+cirros"/>
15  </sliver_type>
16  <sliver_type name="vm-m1.small">
17    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+experiment_orchestrator"/>
18    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_image_processing"/>
19    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_camera_gateway"/>
20    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_rabbit_mq"/>
21    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_robot_controller"/>
22    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_sdn_controller"/>
23    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_mpeg_server"/>

```

```

24 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
25 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+cirros"/>
26 </sliver_type>
27 <sliver_type name="vnf-m1.medium">
28 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+experiment_orchestrator"/>
29 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_image_processing"/>
30 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_camera_gateway"/>
31 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_rabbit_mq"/>
32 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_robot_controller"/>
33 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_sdn_controller"/>
34 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_mpeg_server"/>
35 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
36 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+cirros"/>
37 </sliver_type>
38 <sliver_type name="vnf-m1.small">
39 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+experiment_orchestrator"/>
40 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_image_processing"/>
41 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_camera_gateway"/>
42 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_rabbit_mq"/>
43 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_robot_controller"/>
44 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_sdn_controller"/>
45 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_mpeg_server"/>
46 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
47 <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+cirros"/>
48 </sliver_type>
49 <available now="true"/>
50 </node>
51 <node component_name="intelligent_space" exclusive="false"
52     component_id="urn:publicid:IDN+futebol.inf.ufes.br+intelligent_space+intelligent_space"
53     component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+am">
54 <sliver_type name="intelligent_space"/>
55 <available now="true"/>
56 </node>

```

Listagem 4.11 – Excerto da *Advertisement* RSpec.

A *Request* RSpec atualizada é apresentada na Listagem 4.12. Os serviços que atuam no controle da trajetória do robô e no controle da conectividade sem fio são aplicações de tempo real, em que a habilidade de escalar de forma sensível à demanda possibilitaria manter a qualidade de seu serviço. Por essa razão, elas foram especificadas como VNFs. Diferente da versão anterior, é possível especificar a imagem e o *flavor* da VNF diretamente na RSpec, de modo similar ao que é feito com VMs. Isso é reflexo da implementação de VNFs, que mudou para se integrar ao novo mecanismo de orquestração da nuvem. O espaço inteligente também está representado na Listagem 4.12. Embora o AM não execute nenhuma ação sobre os recursos físicos dele, sua presença é necessária para registrar sua alocação. Os parâmetros que constam em sua descrição na RSpec (taxa de aquisição das câmeras, trajetória, número de voltas, tempo de duração do experimento e área de cobertura dos *access points*) são utilizados para automatizar a configuração do cenário do experimento.

```

1 <node client_id="orchestrator" exclusive="false"
2     component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">

```



```

3  <sliver_type name="vm-m1.small">
4    <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+experiment_orchestrator"/>
5  </sliver_type>
6 </node>
7 <node client_id="image-processing" exclusive="false"
8   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
9   <sliver_type name="vnf-m1.small">
10     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_image_processing"/>
11   </sliver_type>
12 </node>
13 <node client_id="camera-gateway" exclusive="false"
14   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
15   <sliver_type name="vnf-m1.small">
16     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_camera_gateway"/>
17   </sliver_type>
18 </node>
19 <node client_id="rabbit-mq" exclusive="false"
20   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
21   <sliver_type name="vnf-m1.small">
22     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_rabbit_mq"/>
23   </sliver_type>
24 </node>
25 <node client_id="robot-controller" exclusive="false"
26   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
27   <sliver_type name="vnf-m1.small">
28     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_robot_controller"/>
29   </sliver_type>
30 </node>
31 <node client_id="sdn-controller" exclusive="false"
32   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
33   <sliver_type name="vnf-m1.small">
34     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_sdn_controller"/>
35   </sliver_type>
36 </node>
37 <node client_id="mpeg-server" exclusive="false"
38   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
39   <sliver_type name="vnf-m1.small">
40     <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_mpeg_server"/>
41   </sliver_type>
42 </node>
43 <node client_id="ispace" exclusive="false"
44   component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
45   <sliver_type name="intelligent_space">
46     <camera fps="15"/>
47     <robot duration="30" laps="3" trajectory="circle"/>
48     <wireless_layout x="0" y="0"/>
49   </sliver_type>
50 </node>

```

Listagem 4.12 – Excerto da *Request* RSpec.

Ao processar a *Request* RSpec, o AM gera um arquivo com os parâmetros do espaço inteligente e os nomes das VNFs e o deixa na VM de `experiment_orchestrator`, a fim de que os serviços e orquestradores possam ler suas configurações iniciais. Desse modo, o experimento fica pré-configurado e, quando o experimentador acessar a VM de



`experiment_orchestrator`, ele precisará apenas iniciar a execução do experimento. Caso necessite, ele poderá utilizar os clientes dos orquestradores para alterar essas configurações, como a área coberta por cada *access point* e a taxa de aquisição das câmeras, por exemplo.

Para o contexto do *showcase*, o serviço de orquestração da nuvem oferecido pelo Tacker não se mostrou adequado. As exigências apertadas em relação ao tempo de resposta das aplicações hospedadas como VNFs motivaram a mudança de orquestrador da nuvem, pois o *scaling* horizontal realizado pelo Tacker leva um tempo considerável para criar uma nova instância e distribuir a carga. Além disso, nenhum dos serviços utilizados nesse *showcase* pode ser paralelizado. Dessa forma, a opção mais apropriada no contexto do *showcase* é o *scaling* vertical, que consiste na adição de recursos na mesma instância e não é implementado pelo Tacker. Sendo assim, exclusivamente para essa versão do O2CMF, o Tacker foi substituído pelo *Scaler*, um orquestrador de nuvem que implementa *scaling* vertical. Em função dessa mudança, as VNFs passaram a ser implementadas através de VMs.

O *data center* e o espaço inteligente se localizam em prédios vizinhos e estão conectados diretamente através de um enlace de fibra óptica. Para facilitar a interação com a nuvem, os dispositivos no espaço inteligente receberam endereços do *pool* de *floating* IPs. Dessa forma, todas as VMs e VNFs recebem um *floating* IP, que é usado tanto na comunicação com os dispositivos como para acesso SSH. Também por essa razão, qualquer tentativa de especificar um *fixed* IP é ignorada.

Como efeito das mudanças no catálogo anunciado pelo O2CMF, a representação interna dos recursos precisou ser adaptada. Anteriormente, VM e VNF eram entidades distintas. Isso ocorria porque as VNFs eram gerenciadas pelo Tacker, o que limitava o controle que o AM tinha sobre esse tipo de recurso. Com a substituição do Tacker, a responsabilidade pelo ciclo de vida das VNFs foi transferida para o AM. Dado que VM e VNF agora são implementadas na nuvem da mesma forma, elas passaram a ser representadas pela mesma classe (`ComputeInstance`), sendo diferenciadas apenas pelo valor do atributo *type* (herdado de `Resource`). Além disso, foi incluída a classe `IntelligentSpace` a fim de representar o espaço inteligente. Essas mudanças feitas nos pacotes `TestbedResources` e `Database` podem ser observadas através do modelo entidade-relacionamento, apresentado na Figura 30.

Através da integração com o espaço inteligente, o O2CMF pode entregar ao experimentador uma plataforma para experimentação convergente. Essa plataforma permite ao experimentador descobrir, reservar e provisionar recursos de domínios diferentes (nuvem, robótica, redes de pacote e sem fio) de forma integrada.

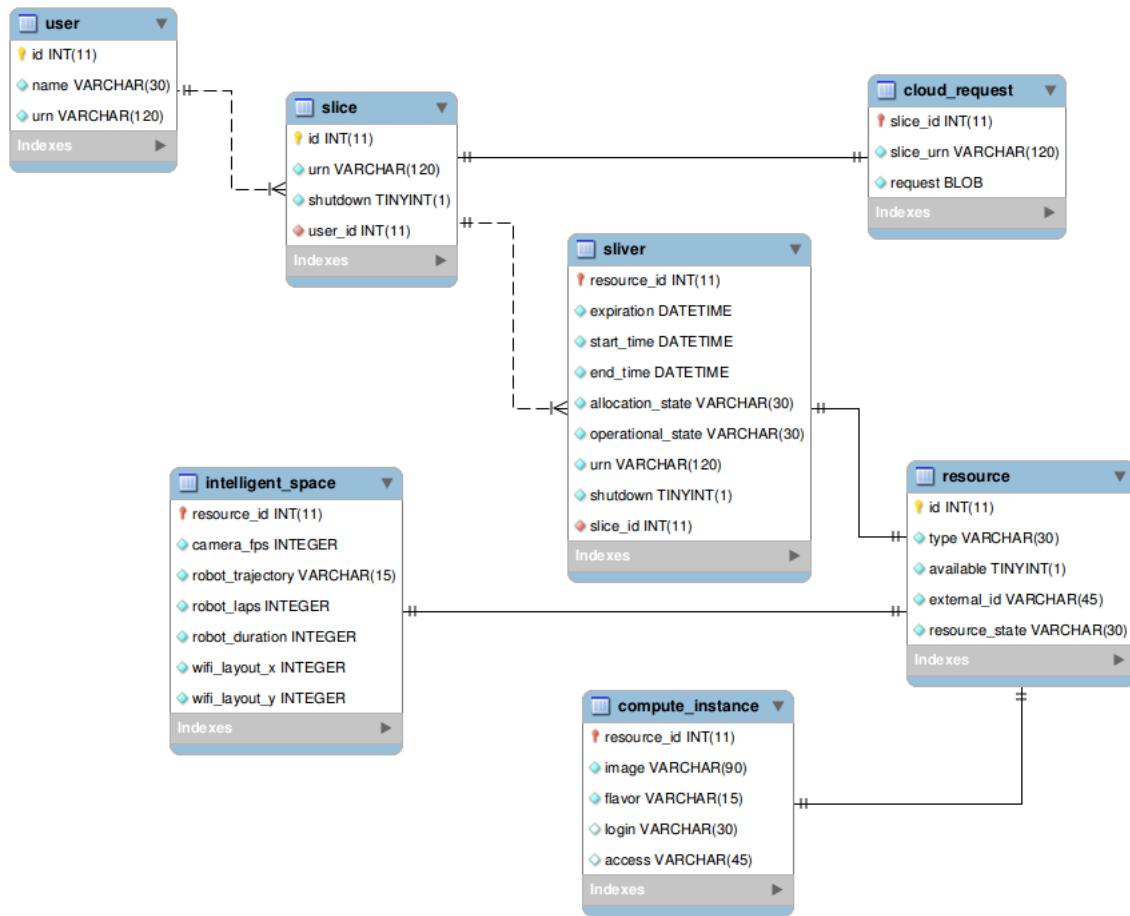


Figura 30 – Modelo entidade-relacionamento na última versão do O2CMF.

## 4.4 Implantação e Testes

O código fonte gerado nas três primeiras etapas de desenvolvimento está disponível em <https://gitlab.com/futebol/O2CMF>. O código da última etapa ainda não está disponível publicamente, visto que o *showcase* com o espaço inteligente ainda não foi oficialmente apresentado. O repositório apresenta uma *branch* para cada etapa, conforme descrito na Tabela 5. O processo de instalação encontra-se descrito na *wiki* do repositório. Na página <https://gitlab.com/futebol/O2CMF/wikis/installation>, há informação sobre as tecnologias utilizadas, requisitos da instalação, topologia e configuração.

Tabela 5 – Organização do repositório do O2CMF.

<b>Etapas</b>	<b>Branch</b>
Integração da nuvem ao Fed4FIRE	<i>master</i>
Integração de NFV ao Fed4FIRE	<i>nfv_ofc</i>
Suporte à experimentadores iniciantes	<i>nfv_prototype</i>
Experimentação convergente com NFV	—

Durante cada etapa do processo de desenvolvimento do O2CMF, foram realizadas verificações do comportamento do AM. Inicialmente, enquanto o O2CMF não estava

federado, foi utilizado o cliente SFA de linha de comando distribuído com o GCF (Omni). Em função das limitações do Omni, passou-se a utilizar o jFed *Probe*.

No jFed *Probe* é possível chamar cada método SFA individualmente ou utilizar um teste automatizado. Esse teste consiste em criar um experimento contendo uma única VM, sendo o mesmo teste realizado pelo sistema de monitoramento do Fed4FIRE para checar o status do *testbed*. O processo de configuração do jFed *Probe* para interagir com um AM ainda não federado é também descrito na *wiki* do O2CMF, na página <<https://gitlab.com/futebol/O2CMF/wikis/testing>>. Nessa página também se encontram instruções sobre a utilização dessa ferramenta.

Após a inclusão no Fed4FIRE, o jFed *Probe* foi substituído pelo jFed GUI, pois os testes automáticos do *Probe* se aplicam somente a VMs. O jFed GUI é a interface utilizada pelos experimentadores, que possibilitou avaliar o comportamento do O2CMF de forma fiel ao percebido pelo usuário final. Além disso, a substituição agregou flexibilidade e permitiu realizar testes mais elaborados.



## 5 Validação do O2CMF

A fim de atestar a conformidade do O2CMF com sua especificação foram realizados testes. Esses testes foram organizados de acordo com os objetivos de cada etapa de desenvolvimento e de modo a cobrir todos os requisitos. A Tabela 6 apresenta a relação entre as etapas, os testes relacionados e os respectivos requisitos a validar.

Tabela 6 – Testes de validação por etapa de desenvolvimento.

Etapa de Desenvolvimento	Teste	Requisitos
Integração da nuvem ao Fed4FIRE	Definição de política para gestão do <i>testbed</i>	R08
	Reserva e provisionamento	R01, R02, R06, R07
	Isolamento entre experimentos	R09
Integração de NFV ao Fed4FIRE	Criação de VNF através de <i>script</i> TOSCA	R03, R04, R05
Suporte à experimentadores iniciantes	Criação de VNF através da RSpec	R05
Experimentação convergente com NFV	Reserva do espaço inteligente	–

### 5.1 Definição de política para gestão do *testbed*

Essa demonstração tem o objetivo de validar o requisito R08, referente à definição de políticas para o compartilhamento da infraestrutura. O cenário da demonstração consistirá na criação de uma política para limitação da largura de banda. Serão analisados dois *flavors*, ambos com a mesma configuração (quantidade de memória, disco e CPU virtual). Enquanto um deles utiliza uma política para limitar a largura de banda, o outro tem como política não restringir a largura de banda. A comprovação da efetividade da política de limitação de largura de banda será feita por meio de medições de vazão na interface de rede das VMs. A funcionalidade a ser validada neste teste é voltada ao operador do *testbed*.

A preparação para o teste consistiu na instalação do OpenStack, versão Rocky, em um único servidor (*deploy all-in-one*). Após a implantação da nuvem, o primeiro passo foi acessar a *dashboard* de administração (Horizon) utilizando as credenciais de administrador. Foi necessário estar no projeto **admin**, conforme Figura 31, para realizar as alterações pretendidas.

Em seguida, foi acessado o menu **Admin > Compute > Flavors**, conforme exibido na Figura 32. Através desse painel é possível gerenciar os *flavors* disponíveis na nuvem. Os *flavors* listados na imagem são nativos do OpenStack, criados no processo de instalação. Note que ao lado de cada *flavor* há a opção **Update Metadata**. Essa opção permite criar/modificar metadados, que servem para definir propriedades do recurso. Nesse teste,



Figura 31 – Menu de seleção de projeto.

os metadados serão utilizadas para criar a política de gestão. Os *flavors* nativos não foram modificados. Ao invés disso, utilizou-se a opção **Create Flavor** para criar os *flavors* utilizados nesse teste.

Admin / Compute / Flavors

## Flavors

Filter

Displaying 5 items

<input type="checkbox"/>	Flavor Name	VCPUs	RAM	Root Disk	Ephemeral Disk	Swap Disk	RX/TX factor	ID	Public	Metadata	Actions
<input type="checkbox"/>	m1.large	4	8GB	80GB	0GB	0MB	1.0	4	Yes	No	Update Metadata ▼
<input type="checkbox"/>	m1.medium	2	4GB	40GB	0GB	0MB	1.0	3	Yes	No	Update Metadata ▼
<input type="checkbox"/>	m1.small	1	2GB	20GB	0GB	0MB	1.0	2	Yes	No	Update Metadata ▼
<input type="checkbox"/>	m1.tiny	1	512MB	1GB	0GB	0MB	1.0	1	Yes	No	Update Metadata ▼
<input type="checkbox"/>	m1.xlarge	8	16GB	160GB	0GB	0MB	1.0	5	Yes	No	Update Metadata ▼

Displaying 5 items

Figura 32 – Pannel de gestão de *flavors*.

Na Figura 33, é apresentado o formulário de criação de um *flavor*. Nele são especificadas as quantidades de memória, CPUs virtuais e disco. É possível especificar tanto disco raiz (para acomodar uma imagem proveniente do repositório do Glance) como disco efêmero (armazenamento existente somente durante a execução da VM). Embora haja a opção de especificar a taxa de recepção e transmissão (Rx/Tx), determinando assim a largura de banda, essa opção é implementada apenas para sistemas baseados em Xen ou NSX (OPENSTACK, 2018c). Por essa razão, a política de limitação de largura de banda foi estabelecida utilizando metadados.

Conforme mostrado na Figura 34, foram criados dois *flavors*: `o2cmf.base` e `o2cmf.limit`. Ambos contam com 1024MB de memória, 1 CPU virtual e 20GB de disco raiz. O passo seguinte foi alterar os metadados do *flavor* `o2cmf.limit`.

**Create Flavor** ✕

**Flavor Information** \* **Flavor Access**

**Name** \*

**ID** ⓘ

**VCPUs** \*

**RAM (MB)** \*

**Root Disk (GB)** \*

**Ephemeral Disk (GB)**

**Swap Disk (MB)**

**RX/TX Factor**

Flavors define the sizes for RAM, disk, number of cores, and other resources and can be selected when users deploy instances.

Cancel Create Flavor

Figura 33 – Formulário de criação de *flavor*.

O formulário para definir as propriedades do *flavor* `o2cmf.limit` é apresentado na Figura 35. Há diversos grupos de propriedades, organizados por finalidade. Conforme informações apresentadas no formulário, no grupo **Flavor Quota** estão agrupadas propriedades relativas à ajuste de CPU, disco, controle tráfego e largura de banda. As propriedades relacionadas à largura de banda formam o subgrupo **Virtual Interface QoS**. Essas propriedades permitem a configuração das taxas média e máxima de envio/recepção, além da quantidade máxima de *bytes* que podem ser enviados/recebidos a cada vez. A configuração é feita considerando valores em *kilobytes* por segundo (OPENSTACK, 2018c). Conforme exibido na Figura 35, essas propriedades foram configuradas para limitar a banda em 500 MB/s. Quando essas propriedades não são especificadas, a interface utiliza a mesma largura de banda da rede a qual está ligada.

Após a configuração dos *flavors*, foi a vez de preparar a imagem utilizada no teste. Durante a instalação do OpenStack, é adicionada ao repositório do Glance a imagem `cirros`, que é uma distribuição mínima do Linux (OPENSTACK, 2018d). Porém, essa imagem possui limitações que dificultam a instalação de pacotes essenciais para os próximos

Admin / Compute / Flavors

## Flavors

Displaying 7 items

<input type="checkbox"/>	Flavor Name	VCPUs	RAM	Root Disk	Ephemeral Disk	Swap Disk	RX/TX factor
<input type="checkbox"/>	m1.large	4	8GB	80GB	0GB	0MB	1.0
<input type="checkbox"/>	m1.medium	2	4GB	40GB	0GB	0MB	1.0
<input type="checkbox"/>	m1.small	1	2GB	20GB	0GB	0MB	1.0
<input type="checkbox"/>	m1.tiny	1	512MB	1GB	0GB	0MB	1.0
<input type="checkbox"/>	m1.xlarge	8	16GB	160GB	0GB	0MB	1.0
<input type="checkbox"/>	o2cmf.base	1	1GB	20GB	0GB	0MB	1.0
<input type="checkbox"/>	o2cmf.limit	1	1GB	20GB	0GB	0MB	1.0

Displaying 7 items

Figura 34 – Painel exibindo os *flavors* criados.

passos desse teste. Por essa razão, foi preciso inserir uma nova imagem no repositório do Glance. Para criar uma nova imagem, o OpenStack disponibiliza o guia (OPENSTACK, 2018i), que descreve como obter, criar e modificar imagens para VMs compatíveis com o OpenStack. No contexto do teste, uma imagem do Ubuntu 16, obtida em (OPENSTACK, 2018d), era suficiente. Após a inclusão na nuvem, o repositório de imagens ficou conforme apresentado na Figura 36.

Com os *flavors* e imagens preparados, foram criadas três instâncias: *vm1* baseada no *flavor* *o2cmf.limit* (com política), *vm2* e *vm3* baseadas no *flavor* *o2cmf.base* (sem política). Essas VMs foram configuradas com *fixed* IPs da rede *selfservice* (192.0.2.0/24), sendo as únicas instâncias existentes na nuvem, conforme exibido no painel de topologia de rede do OpenStack (menu **Project** > **Network** > **Network Topology**), apresentado na Figura 37.

Após a instanciação das VMs, foi instalado em cada uma delas o *iperf*<sup>1</sup>, que é um software para medição da largura de banda. O *iperf* foi utilizado nesse teste com a finalidade de gerar tráfego de modo a ocupar toda a banda disponível, evidenciando as diferenças de largura de banda entre os *flavors* *o2cmf.base* e *o2cmf.limit*. Como

<sup>1</sup> <<https://iperf.fr/>>



### Update Flavor Metadata ✕

You can specify resource metadata by moving items from the left column to the right column. In the left column there are metadata definitions from the Glance Metadata Catalog. Use the "Custom" option to add metadata with the key of your choice.

#### Available Metadata

- > CPU Pinning +
- ▼ Flavor Quota +
- > CPU Limits +
- > Disk QoS +
- > Guest Memory Backing +
- > libvirt Driver Options +
- > Random Number Generator +
- > Trusted Compute Pools (Intel® TXT) +
- > Virtual CPU Topology +

#### Existing Metadata

quota:vif_inbound_av...	5000000	-
quota:vif_inbound_burst	5000000	-
quota:vif_inbound_peak	5000000	-
quota:vif_outbound_a...	5000000	-
quota:vif_outbound_b...	5000000	-
quota:vif_outbound_p...	5000000	-

**Quota: VIF Outbound Peak** (*quota:vif\_outbound\_peak*)

Network Virtual Interface (VIF) outbound peak in kilobytes per second. Specifies maximum rate at which an interface can send data.

✕ Cancel
Save

Figura 35 – Formulário para definição de metadados do *flavor* `o2cmf.limit`.

a nuvem foi implantada em um único servidor, hospedando somente `vm1`, `vm2` e `vm3`, o tráfego gerado pelo `iperf` circulará apenas pelo barramento do servidor. Como forma de acompanhar os resultados do teste, foi utilizado o Ceilometer para medir as taxas de envio e recepção nas interfaces de rede de cada uma das VMs. Com o apoio da plataforma de monitoramento Grafana<sup>2</sup> foi possível visualizar as métricas coletadas pelo Ceilometer.

A primeira parte deste teste consistiu em verificar se a largura de banda da `vm1` se manteve menor ou igual a 500 MB/s, conforme estabelecido pela política. Utilizando `vm1` como servidor `iperf` e `vm3` como cliente, foram enviados pacotes utilizando o protocolo TCP, durante 60 segundos. O gráfico apresentado na Figura 38 foi obtido através do Grafana e exibe a taxa de *bytes* recebidos pela `vm1` ao longo do teste. Note que em nenhum momento essa taxa foi maior que 500 MB/s, comprovando que a política foi respeitada. O mecanismo de *traffic shaping* que assegura o cumprimento dessa política é de responsabilidade do

<sup>2</sup> <<https://grafana.com/>>

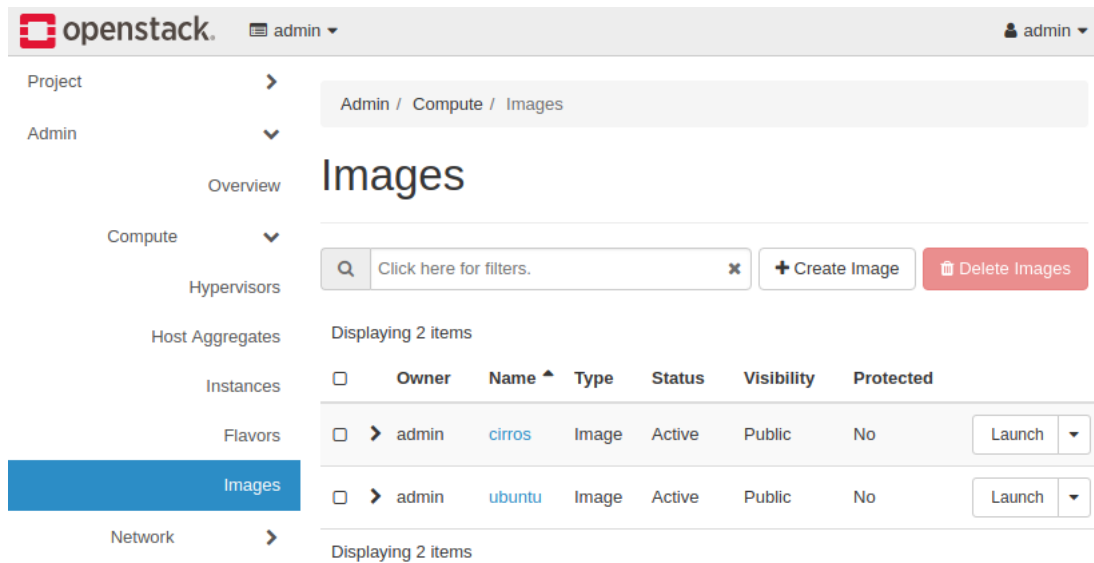


Figura 36 – Painel de gestão de imagens.

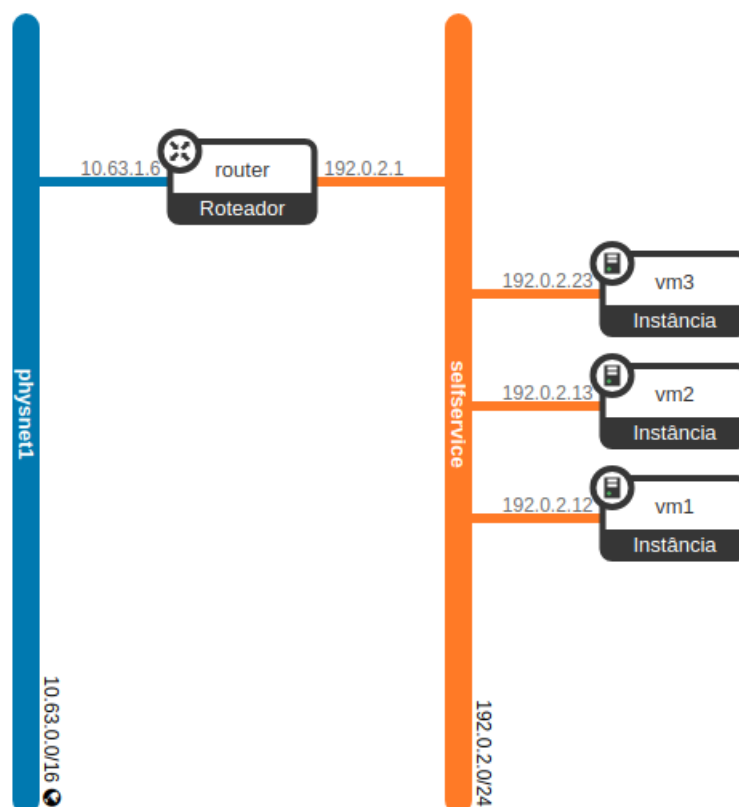


Figura 37 – Painel de topologia de rede do OpenStack.

Nova, sendo implementado através do utilitário TC<sup>3</sup> para controle de tráfego no Linux (OPENSTACK, 2019a).

A segunda parte do teste consistiu em verificar o comportamento do *flavor* sem

<sup>3</sup> <<https://lartc.org/manpages/tc.txt>>

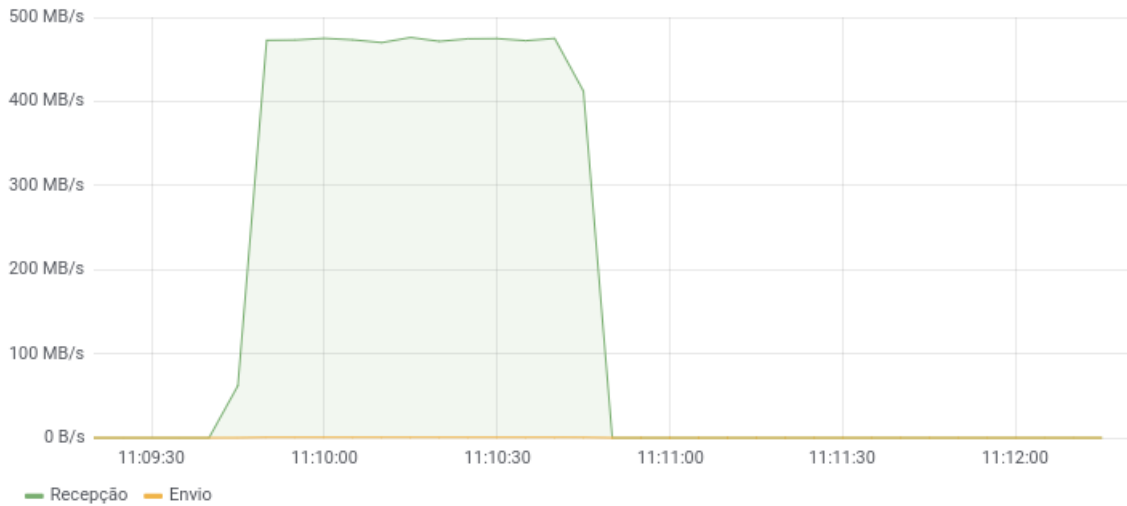


Figura 38 – Envio de tráfego da vm3 para vm1.

política. Utilizando **vm2** como servidor iperf e **vm3** como cliente, foram enviados pacotes utilizando o protocolo TCP, durante 60 segundos. O gráfico apresentado na Figura 39 exibe a taxa de *bytes* recebidos pela **vm2** ao longo do teste. Conforme esperado, no *flavor* sem política, o iperf tentou ocupar a capacidade máxima da rede entre **vm2** e **vm3** (barramento do servidor). Note que a taxa recebida pela **vm2** (acima de 2 GB/s) foi bastante superior à taxa recebida pela **vm1** nas mesmas condições de teste.

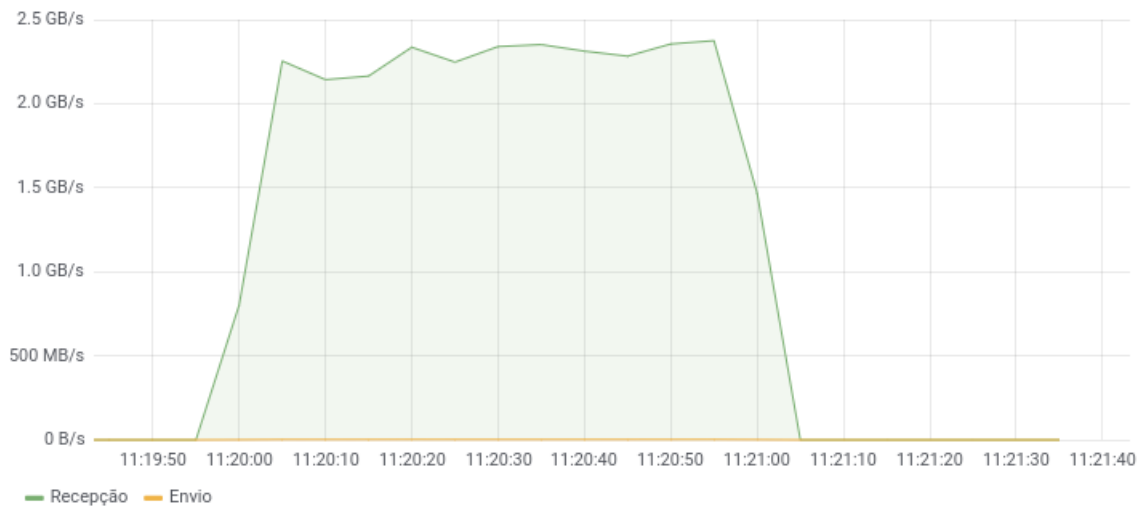


Figura 39 – Envio de tráfego da vm3 para vm2.

A terceira parte do teste consistiu em avaliar o comportamento de **vm3** (cliente iperf) interagindo com **vm1** e **vm2** (ambas como servidor iperf). Para isso, **vm3** enviou pacotes para **vm1** e **vm2** simultaneamente, utilizando o protocolo TCP, durante 60 segundos. O gráfico apresentado na Figura 40 exibe a taxa de *bytes* enviados por **vm3** e as taxas de *bytes* recebidos por **vm1** e **vm2** ao longo do teste. Embora **vm3** tenha sido criada através do *flavor* sem política (`o2cmf.base`), a comunicação com **vm1** (criada com o *flavor* `o2cmf.limit`)

respeitou a política estabelecida, enquanto que a comunicação com `vm2` (criada com o `flavor o2cmf.base`, sem política associada) utilizou a maior parte da banda.

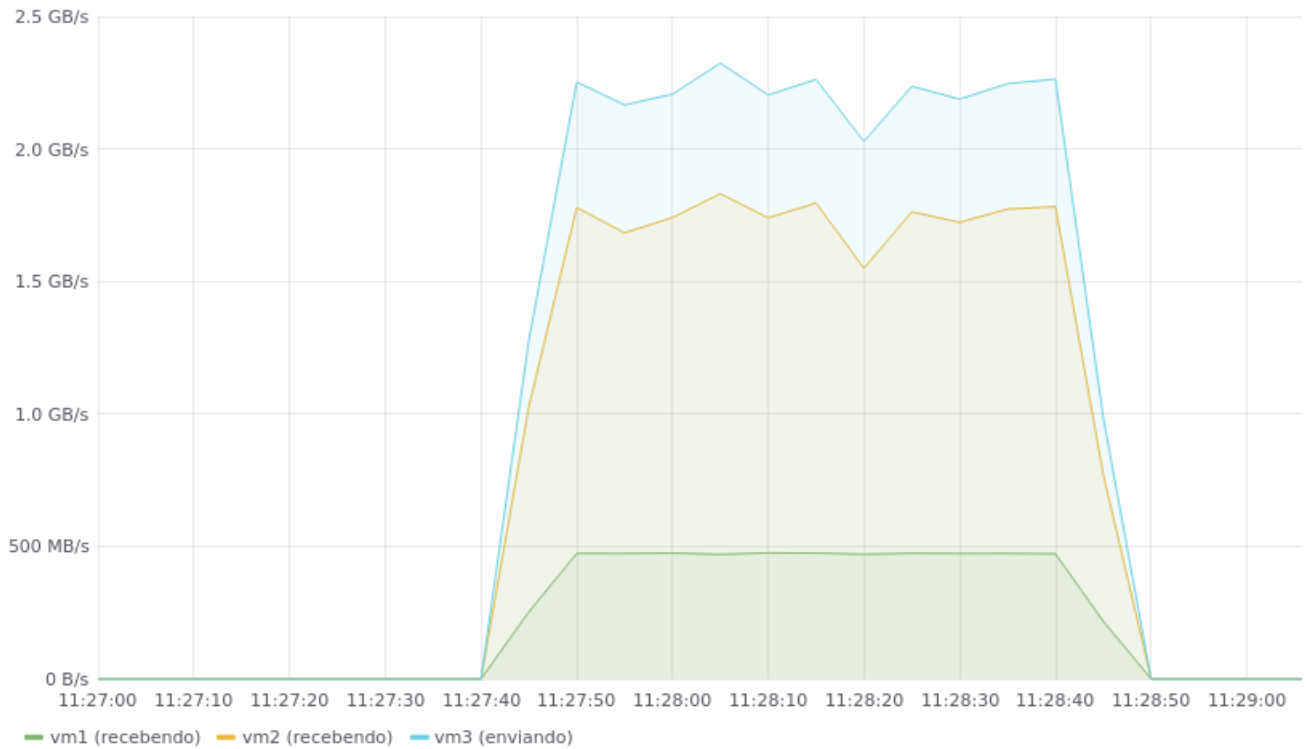


Figura 40 – Envio de tráfego da `vm3` para `vm1` e `vm2`.

Através dos resultados apresentados nos gráficos, foi possível confirmar a efetividade da política de limitação de largura de banda. Dessa forma, validamos a conformidade do O2CMF com o requisito R08.

## 5.2 Reserva e provisionamento

Essa demonstração tem o objetivo de validar a compatibilidade do AM com o Fed4FIRE. Portanto, serão avaliadas a conformidade com a API SFA (requisito R06) e a integração com a *Clearinghouse* do Fed4FIRE (requisito R07). Essa avaliação se dará através da criação de um experimento utilizando o jFed. Esse experimento será composto de duas VMs conectadas a uma LAN. Consequentemente, também serão validadas as funcionalidades de descoberta do catálogo (requisito R01) e de reserva e provisionamento de recursos (requisito R02).

A preparação para o teste consistiu na implantação do AM e do *Proxy* SSH no *data center*. A instalação de nuvem utilizada permaneceu a mesma do teste anterior. Além disso, o jFed foi instalado em uma máquina cliente (externa ao *data center*). O primeiro passo desse teste foi realizar *login* no jFed. Conforme exibido na Figura 41, o jFed aceita credenciais do Fed4FIRE ou GENI. Foram utilizadas credenciais do Fed4FIRE.

Figura 41 – Tela de *login* do jFed.

Após o *login*, foi acessado o menu **Experiment Definition > New** para criar a reserva de um novo experimento. Nessa tela, o usuário pode escrever uma RSpec (na aba **RSpec Editor**) ou usar recursos representados graficamente para desenhar seu experimento (na aba **Topology Editor**). Nesse segundo caso, o usuário também pode especificar as características dos recursos de forma gráfica. Dessa forma, foi projetado um experimento contendo duas VMs (**vm1** e **vm2**) conectadas por uma *bridge* (**bridge0**), conforme exibido na Figura 42.

Após desenhar a topologia do experimento, partimos para a especificação dos recursos. Ao clicar com o botão direito do mouse no ícone da VM, o jFed apresenta a opção **Configure Node**, que apresenta uma janela onde é possível configurar a imagem, o *flavor* e as interfaces de rede da VM. Desse modo, a **vm1** foi configurada para ser implantada no *testbed* FUTEBOL na UFES, usando a imagem do Ubuntu (Figura 43), o *flavor* *m1.small* (Figura 44) e a interface de rede *eth0* foi configurada com o endereço IP 10.255.255.111 (Figura 45). Esse processo de configuração também foi executado para **vm2**, diferenciando apenas o endereço IP utilizado (10.255.255.112).

O jFed abstrai as chamadas à *Clearinghouse* e aos AMs federados, para que o usuário tenha uma interação simples e intuitiva com a federação. Embora não seja diretamente visível ao usuário, a GENI API e as RSpecs, apresentadas na Seção 2.3.1, continuam sendo utilizadas. As opções de imagens e *flavors*, por exemplo, são obtidas pelo jFed através da *Advertisement* RSpec. Por meio do painel de *logs* do jFed, foi possível ver a *Advertisement* RSpec resultado da chamada ao método *ListResources*, conforme exibido na Figura 46.

A especificação do experimento feita através da interface gráfica é traduzida pelo jFed em uma *Request* RSpec. Ela fica disposta na aba **RSpec Editor**, como pode ser visto na Figura 47. O conteúdo integral dessa *Request* RSpec se encontra no Apêndice A.

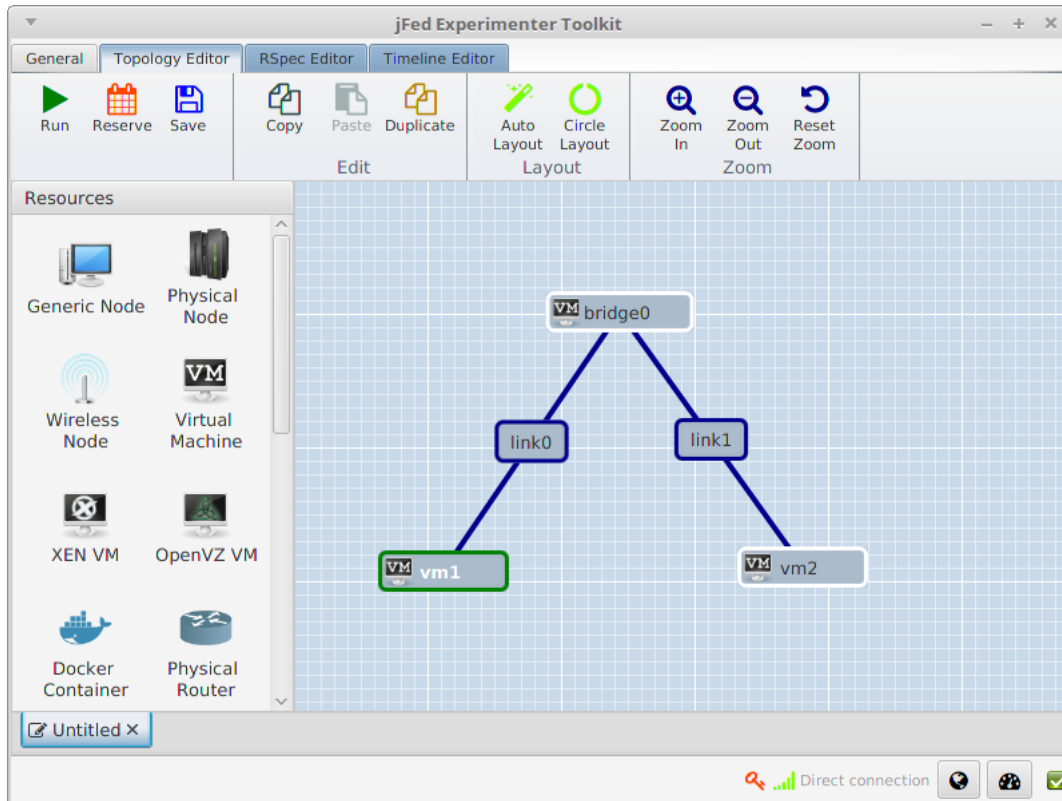


Figura 42 – Especificação da topologia do experimento.

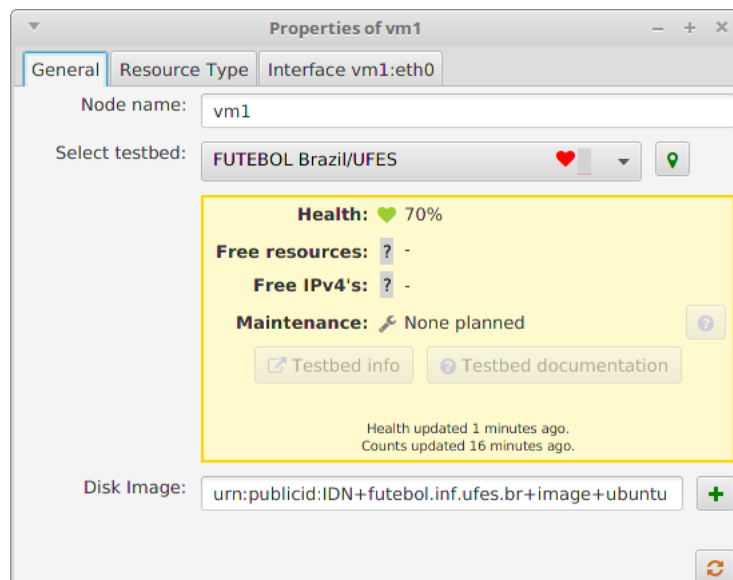


Figura 43 – Especificação da imagem a ser utilizada na VM.

Após a especificação do experimento, prosseguimos para a reserva e instanciação. Para iniciar o experimento, é preciso informar um nome para o experimento e a duração da reserva. O nome dado ao experimento foi  $t_2$  e sua duração foi especificada em 1 hora.

No jFed, todo o processo criação do experimento, que envolve diversas chamadas ao AM e à *Clearinghouse*, é abstraído. Porém, com o apoio do painel de *logs* do jFed, é

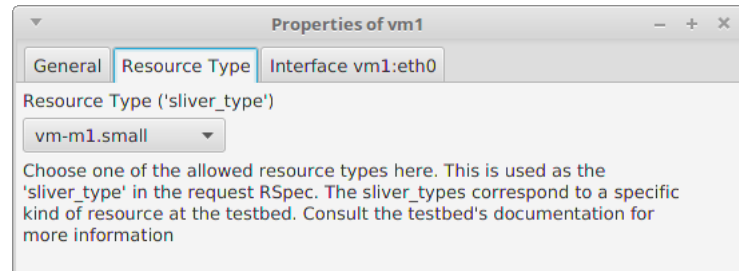
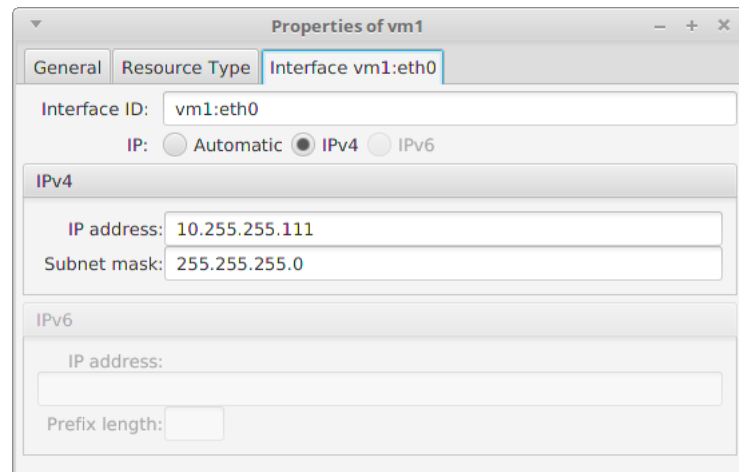
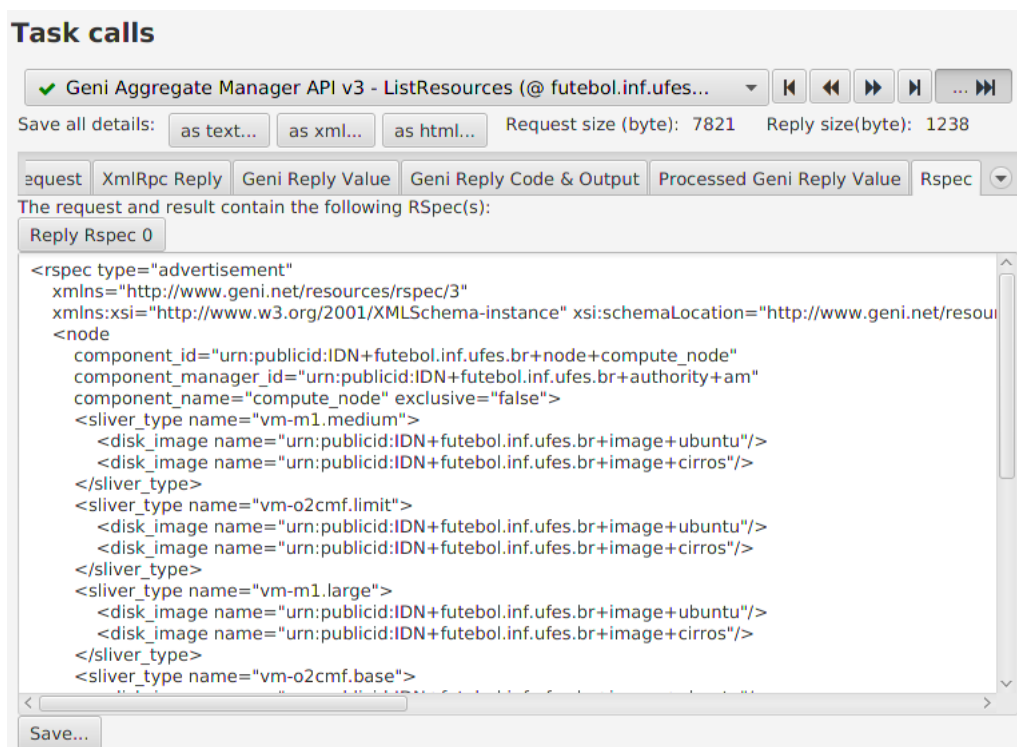
Figura 44 – Especificação do *flavor* da VM.

Figura 45 – Configuração de rede da VM.

Figura 46 – Retorno da chamada ao método *ListResources* implementado pelo AM.

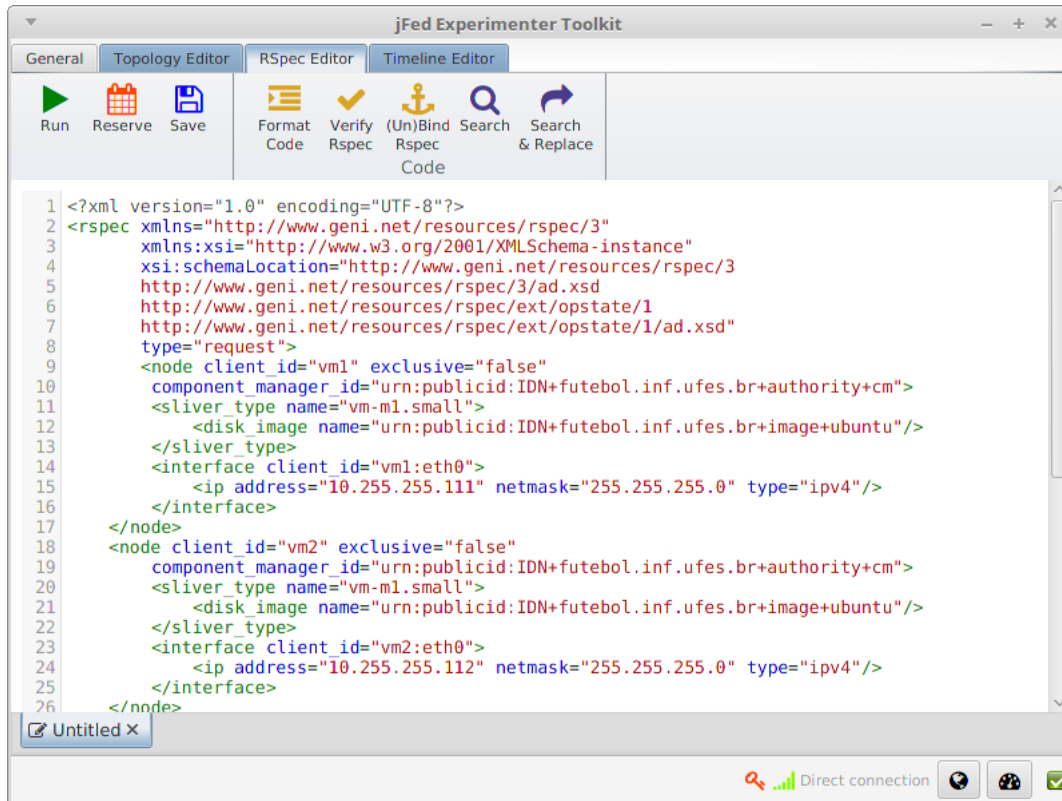


Figura 47 – Aba RSpec Editor exibindo resultado da definição do experimento.

possível acompanhar detalhadamente as interações com a federação. O primeiro passo para a criação do experimento foi registrar um *slice* com o nome *t2*. Isso foi feito através da invocação do método *Create* implementado pela *Clearinghouse*, cujo retorno é apresentado na Figura 48.

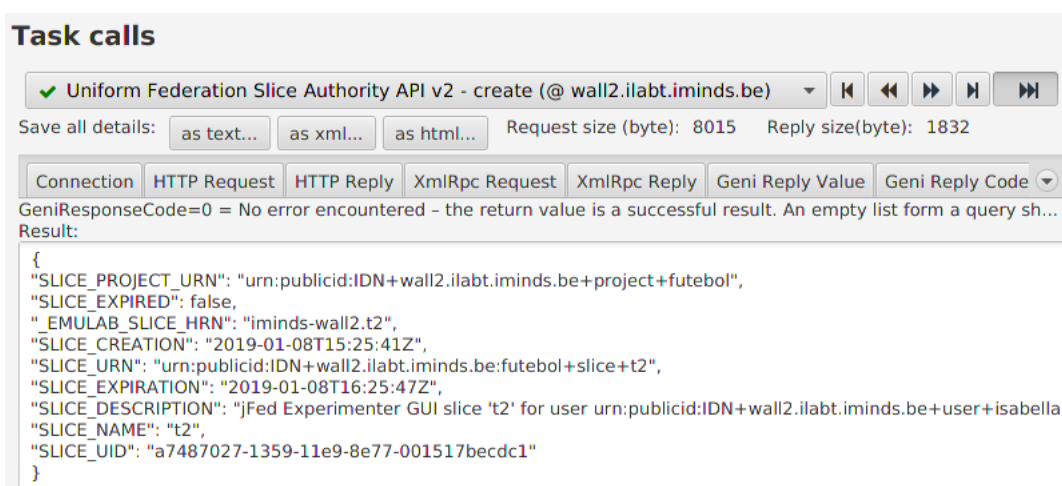


Figura 48 – Retorno da chamada ao método *Create* implementado pela *Clearinghouse*.

Após o registro do *slice*, foi preciso obter a permissão em nome do usuário para atuar nele inserindo *slivers* (recursos). Isso foi feito através do método *Get\_Credentials* implementado pela *Clearinghouse*. O retorno dessa chamada é apresentado na Figura 49.



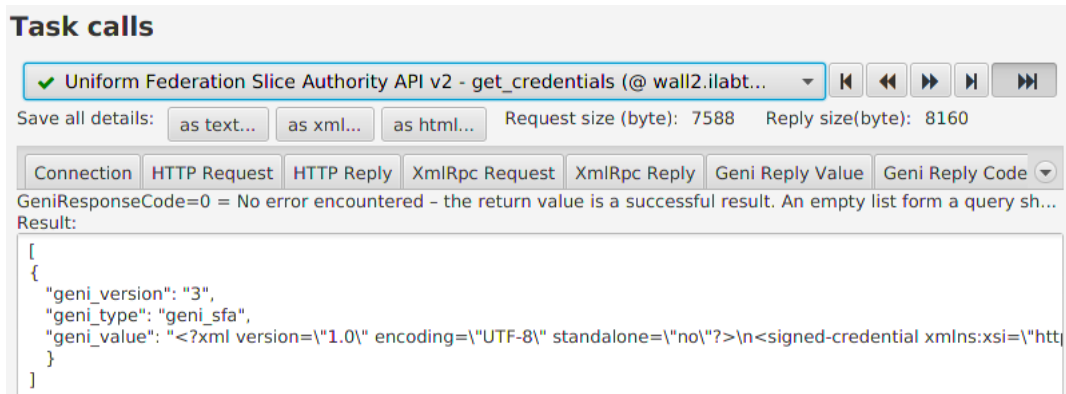


Figura 49 – Retorno da chamada ao método *Get\_Credentials* implementado pela *Clearinghouse*.

O passo seguinte foi reservar os recursos no *testbed*. Utilizando a *Request* RSpec, a URN do *slice t2* e a permissão (obtida na chamada anterior), o *jFed* invocou o método *Allocate*, implementado pelo AM. O retorno, contendo a *Manifest* RSpec, é exibido na Figura 50.

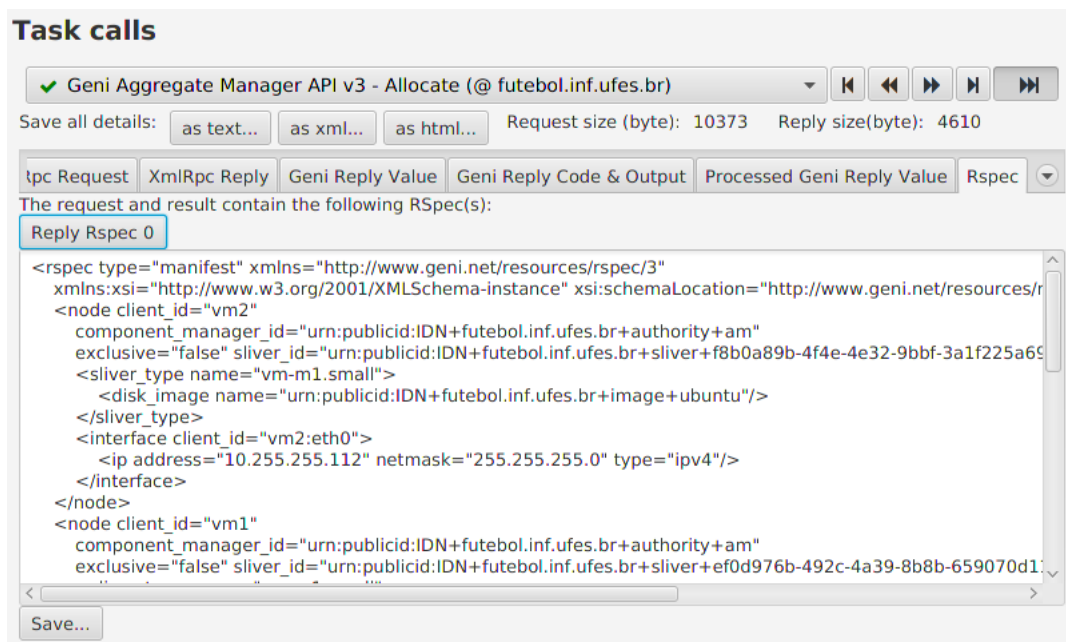


Figura 50 – Retorno da chamada ao método *Allocate* implementado pelo AM.

Após a execução do método *Allocate*, foi utilizado o MySQL Workbench para inspecionar o banco de dados do AM. Foram realizadas *queries* nas tabelas, que permitiram confirmar que a reserva estava devidamente registrada. A figura 51 exibe o resultado da *query* realizada na tabela *slice*, cujo resultado mostra o experimento *t2*. Na tabela *vm* há o registro de *vm1* e *vm2*, exibido na Figura 52, contendo a especificação de imagem (*ubuntu*) e *flavor* (*m1.small*). As configurações de rede das VMS estão na tabela *net\_iface*, contendo os IPs especificados na *Request* RSpec, conforme Figura 53.

resource x sliver x slice x

Limit to 1000 rows

1 • SELECT \* FROM amcloud.slice;

Result Grid Filter Rows: Edit: Export/Import:

#	id	urn	shutdown	user_id
1	2	urn:publicid:IDN+wall2.ilabt.iminds.be:futebol+slice+t2	0	9
*	NULL	NULL	NULL	NULL

Figura 51 – Dados persistidos na tabela `slice` do banco de dados do AM.

Query 9 x user x sliver x vm x

Limit to 1000 rows

1 • SELECT \* FROM amcloud.vm;

Result Grid Filter Rows: Edit: Export:

#	resource_id	image	flavor	login	access
1	3	ubuntu	m1.small	NULL	NULL
2	4	ubuntu	m1.small	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL

Figura 52 – Dados persistidos na tabela `vm` do banco de dados do AM.

net\_iface x

Limit to 1000 rows

1 • SELECT \* FROM amcloud.net\_iface;

Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:

#	id	name	type	address	netmask	bridge	vm_resource_id
1	3	eth0	ipv4	10.255.255.112	255.255.255.0	bridge0:eth1->link1	3
2	4	eth0	ipv4	10.255.255.111	255.255.255.0	bridge0:eth0->link0	4
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 53 – Dados persistidos na tabela `net_iface` do banco de dados do AM.

Até esse momento, os recursos estavam reservados para o experimento `t2`, mas ainda não estavam instanciados no OpenStack. Antes de requisitar o provisionamento, o `jFed` precisou se comunicar com a *Clearinghouse*, invocando novamente o método `get_credentials`. Com a permissão retornada por esse método, o `jFed` chamou o método `Create` para que a *Clearinghouse* faça o registro de dois *slivers* (`vm1` e `vm2`) associados ao *slice* `t2`. O retorno dessa chamada é apresentado na Figura 54.

Em seguida, o `jFed` invocou o método `Provision`, passando como parâmetros a URN do *slice* e a permissão (obtida com `get_credentials`). Para atender a essa requisição, o AM recuperou a reserva do experimento `t2` e transformou a especificação dos recursos em requisições para o OpenStack. Desse modo, o *slice* é implementado como um projeto no

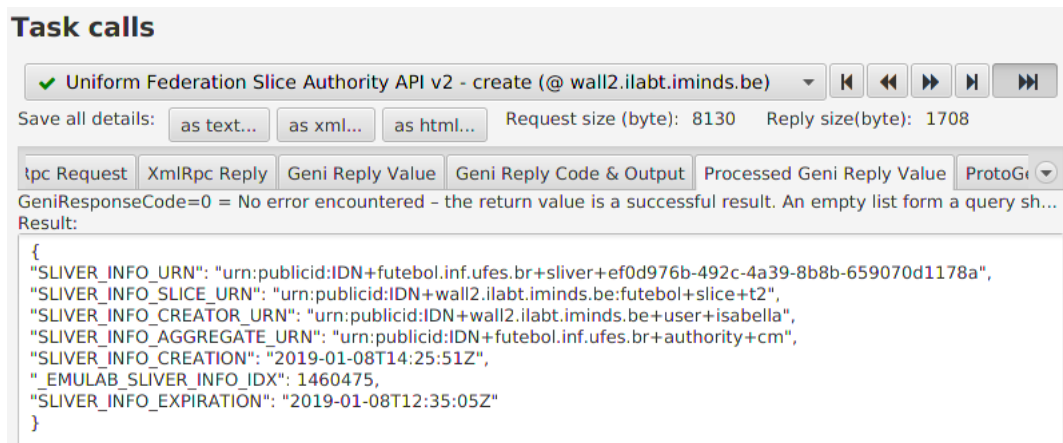


Figura 54 – Retorno da chamada ao método *Create* implementado pela *Clearinghouse*.

OpenStack, os *slivers* são as VMs instanciadas nele e a *bridge* é uma rede privada. Como resultado, o AM envia a *Manifest* RSpec atualizada, conforme Figura 55, contendo os dados para acesso SSH nas VMs.

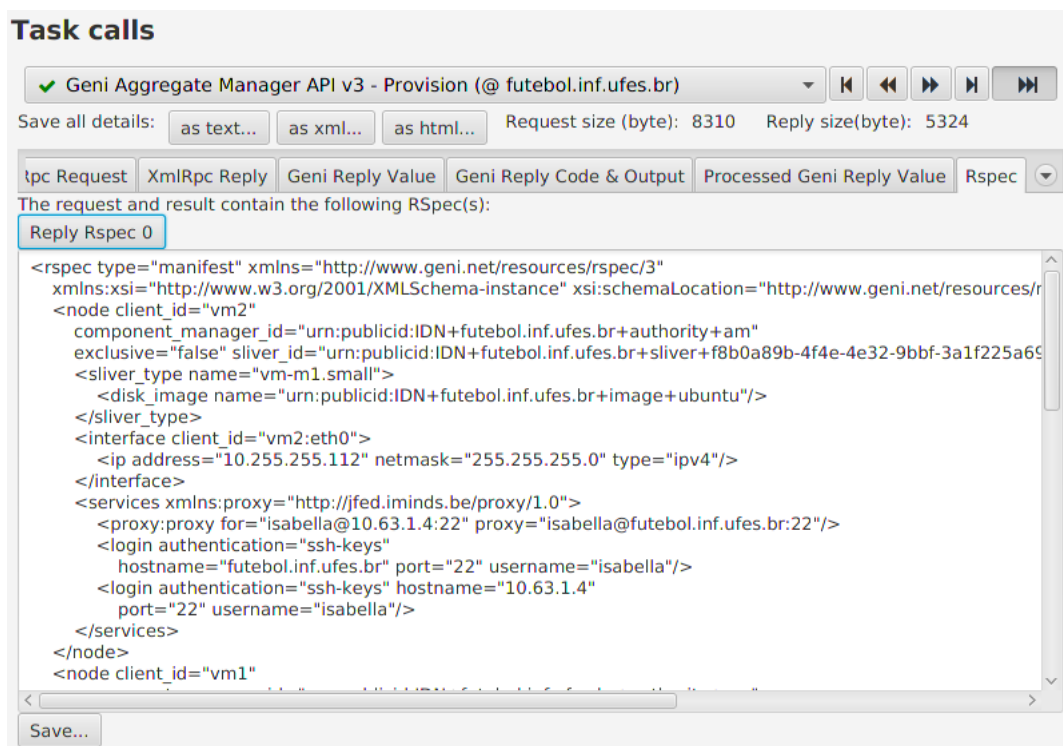


Figura 55 – Retorno da chamada ao método *Provision* implementado pelo AM.

Após a execução do método *Provision*, foi acessado o Horizon para verificar os recursos criados. Através da Figura 56, é possível observar que foram criadas *vm1* e *vm2* com os *fixed* IPs especificados na reserva (10.255.255.111 e 10.255.255.112) e que há *floating* IPs associados a elas (10.63.1.9 e 10.63.1.4). Essas instâncias estão no projeto FUTEBOL+slice+t2, conforme exibido no canto superior esquerdo da figura. No painel de topologia de rede, exibido na Figura 57, podemos observar que as VMs foram conectadas à

rede privada de nome `bridge0`, conforme especificado na RSpec. Essa rede foi conectada, por meio da `router`, à rede `physnet1` que provê os *floating* IPs.

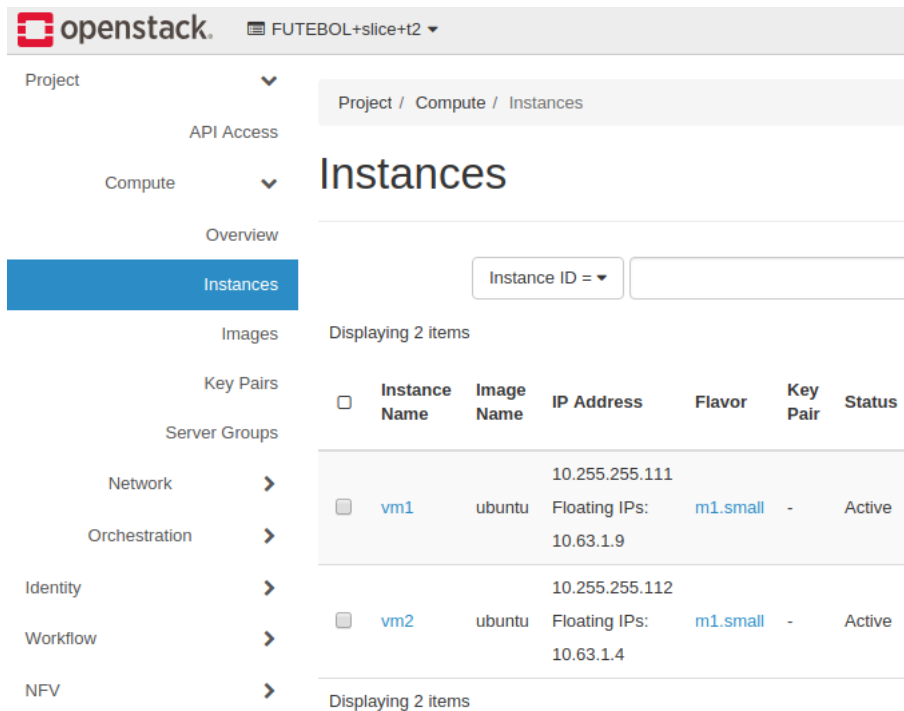


Figura 56 – VMs criadas no OpenStack.

Após o método *Provision*, o jFed continua interagindo com o AM na intenção de certificar ao usuário que todos os recursos estão prontos para uso. Isso ocorre porque, dependendo do tipo de recurso, o processo de instanciação e o preparo para acesso podem ser demorados. Esse não é o caso do O2CMF, em que os recursos estão totalmente prontos após o *Provision*. Contudo, o O2CMF precisa implementar essas “formalidades” para ser plenamente compatível com a arquitetura SFA.

Por essa razão, o jFed invocou o método *PerformOperacionalAction*, conforme Figura 58. Esse método apenas altera o *status* dos recursos para provisionados. Em seguida, foi invocado o método *Status*, conforme Figura 59. Esse método retorna o estado de cada um dos *slivers*, indicando se eles já estão prontos para uso. Por fim, o jFed invocou o método *Describe* (Figura 60) para obter a *Manifest* RSpec atualizada com dados de acesso.

Após a execução de todas as etapas pré-experimento, os recursos ficaram disponíveis para acesso SSH, conforme Figura 61. Foram acessadas às VMs com o objetivo de validar funcionamento do *proxy* SSH e testar a conectividade. Na `vm1` foi realizado um ping para `vm2`. O resultado, apresentado na Figura 62, valida o funcionamento da rede `bridge0`. Na `vm2` foi realizado um ping para o endereço IP 8.8.8.8 (DNS público do Google). O resultado, exibido na Figura 63, demonstra funcionamento da conectividade com a Internet.

Após acessar as VMs e testar a conectividade, foi solicitado o encerramento do

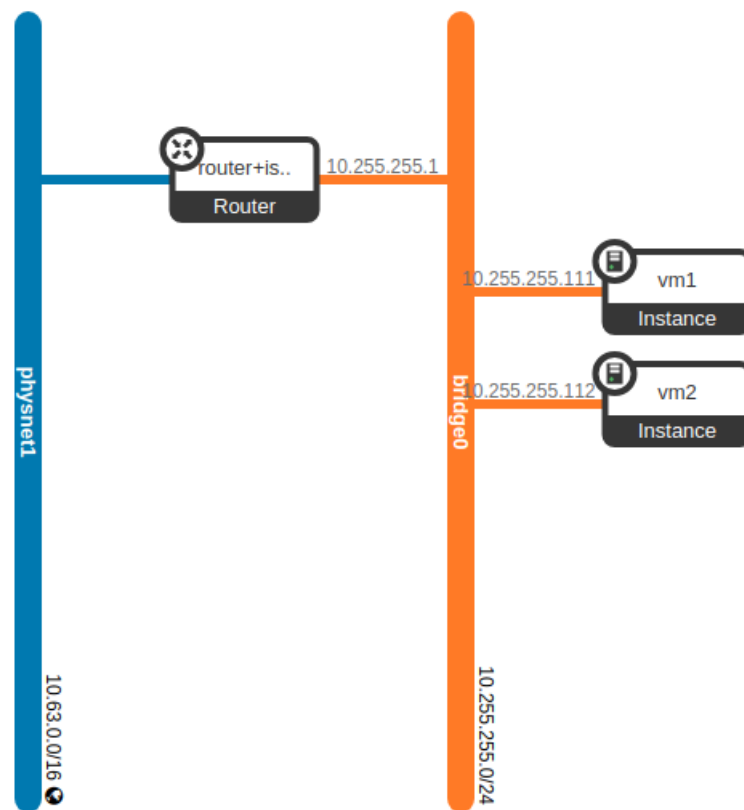


Figura 57 – Topologia de rede criada no OpenStack.

**Task calls**

✓ Geni Aggregate Manager API v3 - PerformOperationalAction (@ futebol.in... ⏮ ⏪ ⏩ ⏭

Save all details: as text... as xml... as html... Request size (byte): 7574 Reply size(byte): 2251

y XmlRpc Request XmlRpc Reply Geni Reply Value Geni Reply Code & Output Processed Geni Reply Value

GeniResponseCode=0 = Success

Result:

```
[
  {
    "geni_expires": "2019-01-08T16:25:47+00:00",
    "geni_allocation_status": "geni_provisioned",
    "geni_end_time": "2019-01-08T16:25:47+00:00",
    "geni_start_time": "2019-01-08T15:25:06+00:00",
    "geni_operational_status": "geni_ready",
    "geni_error": "",
    "geni_sliver_urn": "urn:publicid:IDN+futebol.inf.ufes.br+sliver+f8b0a89b-4f4e-4e32-9bbf-3a1f225a6944"
  },
  {
    "geni_expires": "2019-01-08T16:25:47+00:00",
    "geni_allocation_status": "geni_provisioned",
    "geni_end_time": "2019-01-08T16:25:47+00:00",
    "geni_start_time": "2019-01-08T15:25:06+00:00",
    "geni_operational_status": "geni_ready",
    "geni_error": "",
    "geni_sliver_urn": "urn:publicid:IDN+futebol.inf.ufes.br+sliver+ef0d976b-492c-4a39-8b8b-659070d1178a"
  }
]
```

Figura 58 – Retorno da chamada ao método *PerformOperationalAction* implementado pelo AM.

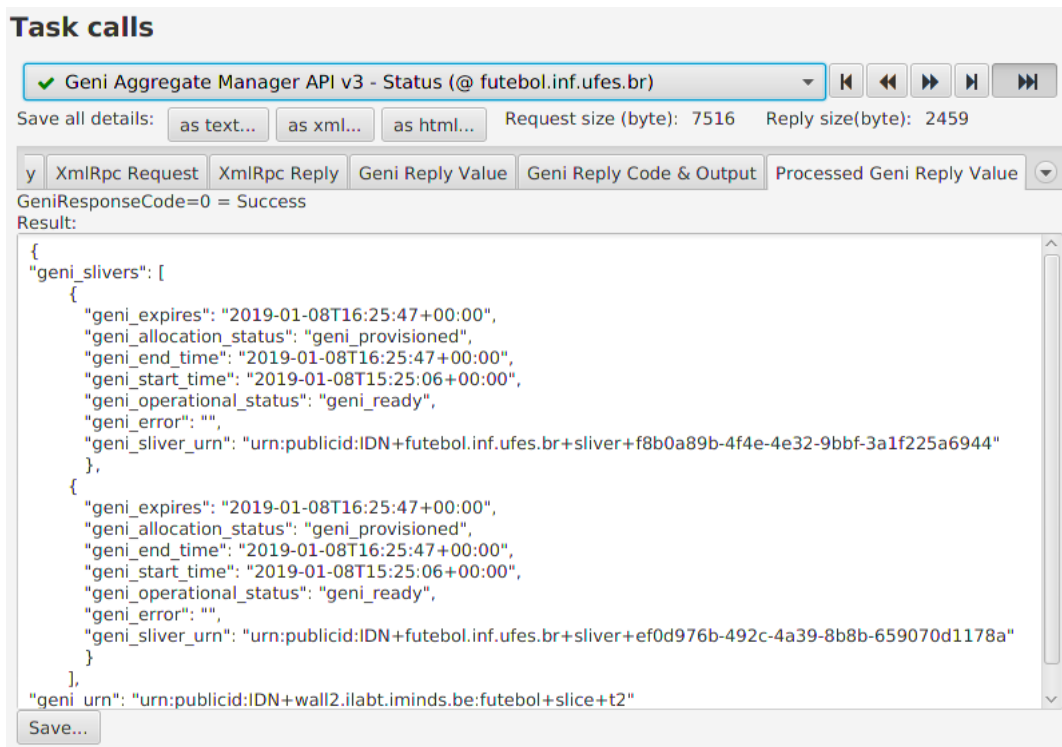


Figura 59 – Retorno da chamada ao método *Status* implementado pelo AM.

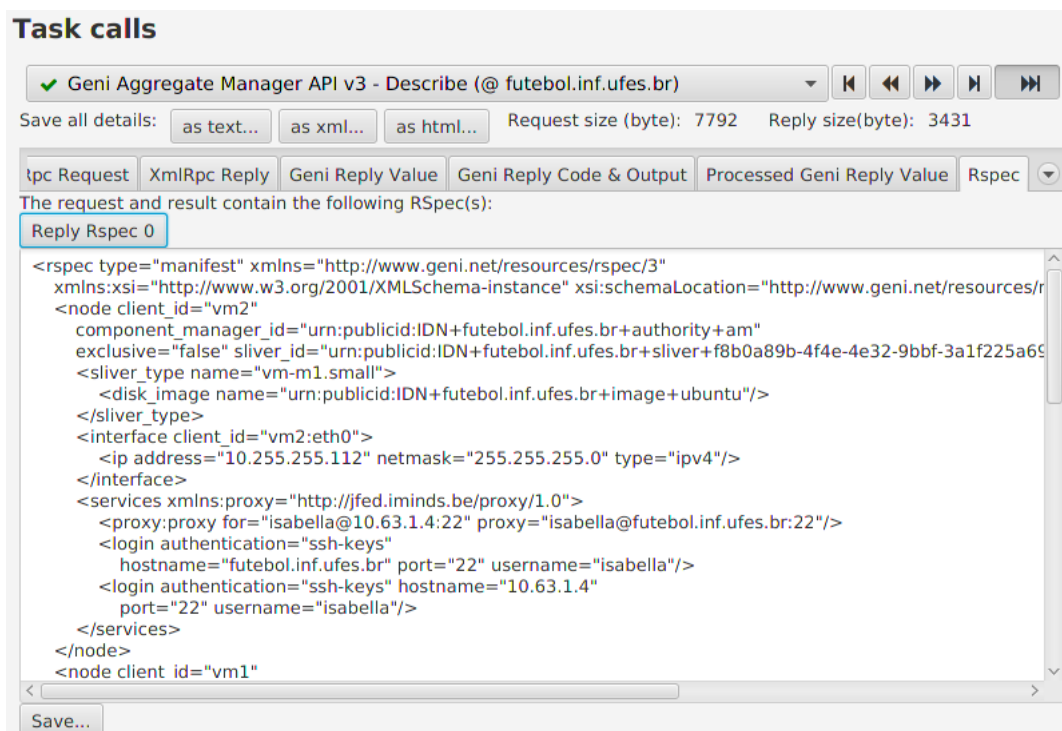


Figura 60 – Retorno da chamada ao método *Describe* implementado pelo AM.

experimento. Para isso, o jFed invocou o método *Delete*, em que o AM libera os recursos no OpenStack e remove a reserva do banco de dados. O resultado dessa chamada é apresentado na Figura 64. Em seguida, o jFed solicitou à *Clearinghouse* a remoção dos registros dos



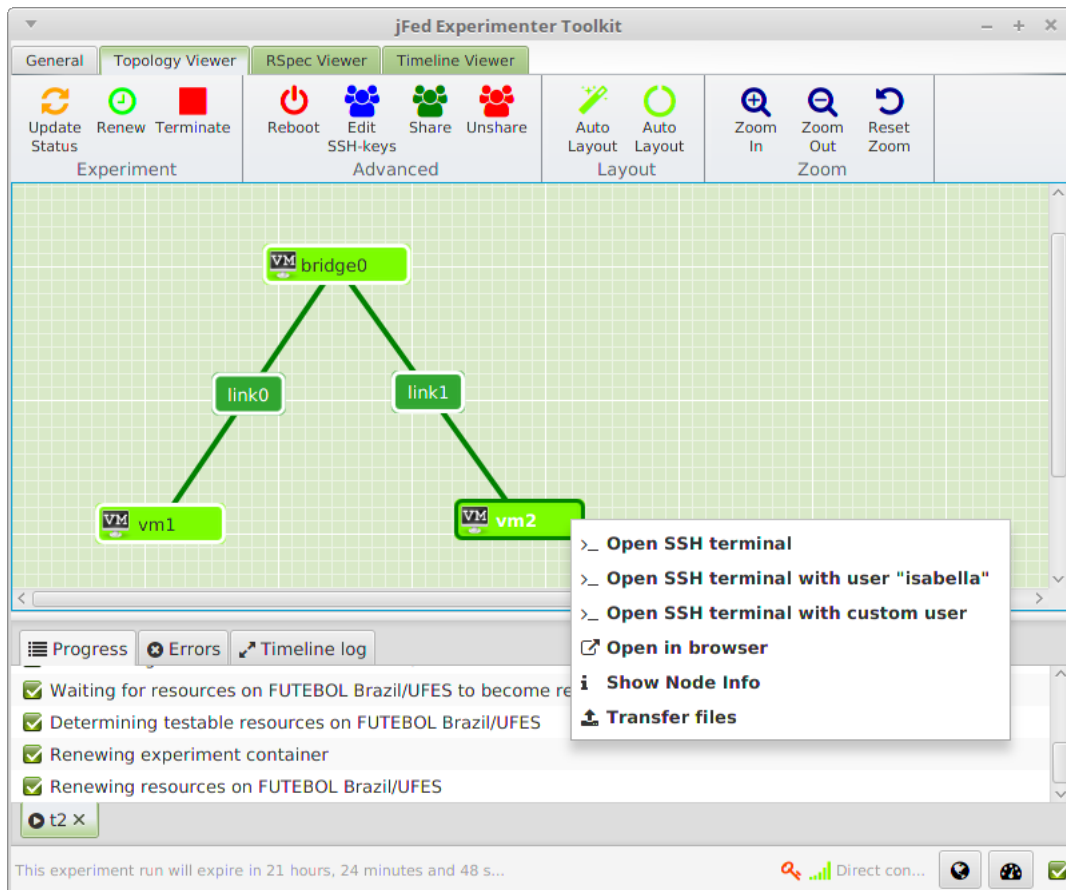


Figura 61 – Recursos do experimento prontos para acesso SSH.

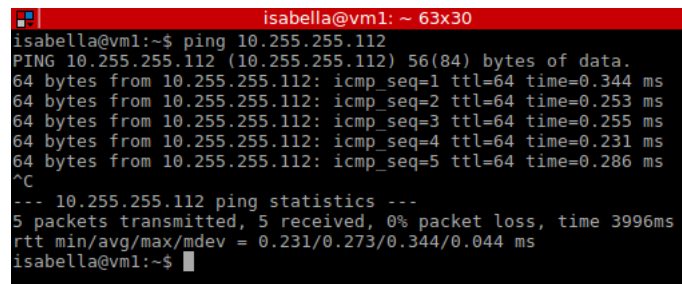


Figura 62 – Teste de conectividade através da LAN.

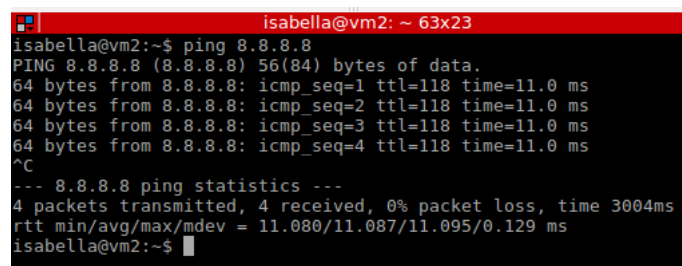


Figura 63 – Teste de conectividade com a Internet.

*slices* e *slivers* do experimento *t2*. Isso foi feito através do método *Delete*, cujo resultado é

apresentado na Figura 65. Com ambas as chamadas executadas com sucesso, foi finalizado com êxito o ciclo de vida do experimento.

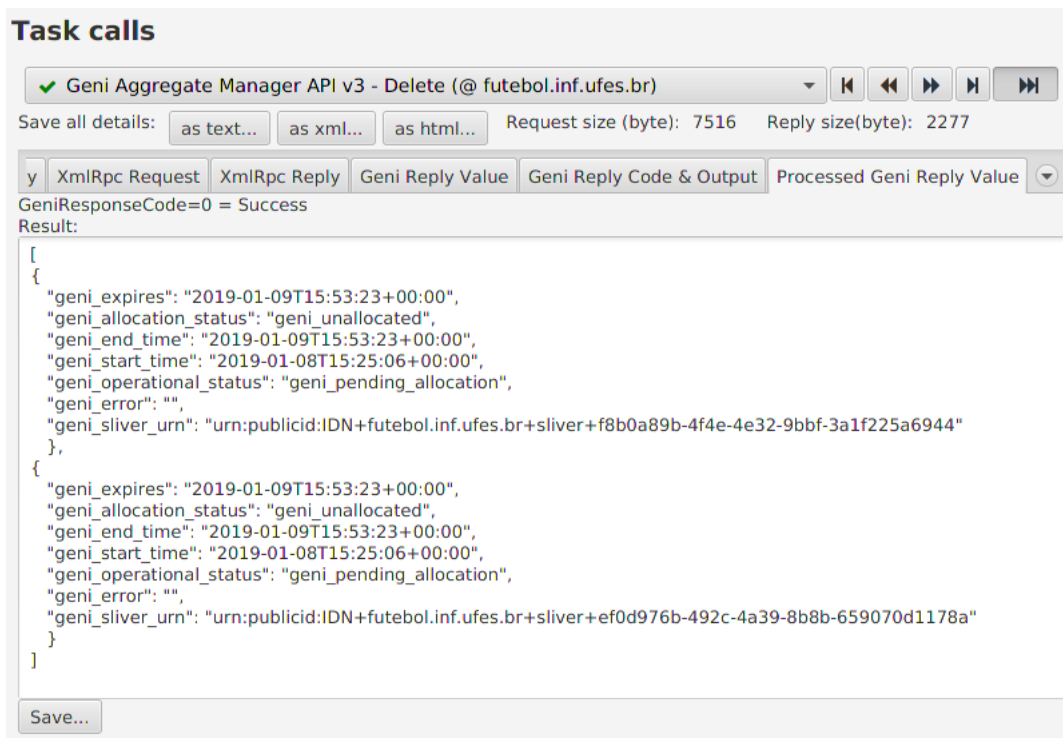


Figura 64 – Retorno da chamada ao método *Delete* implementado pelo AM.

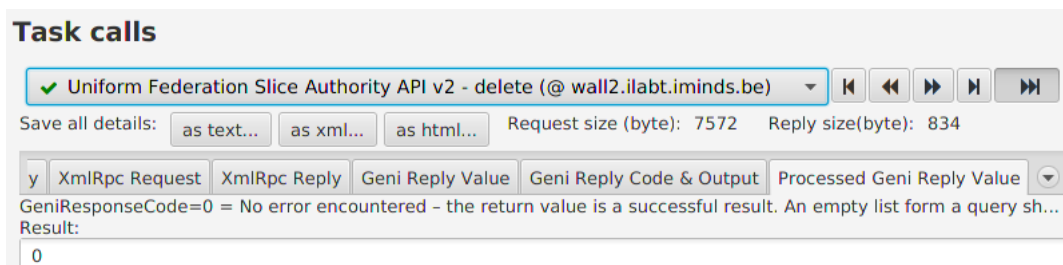


Figura 65 – Retorno da chamada ao método *Delete* implementado pela *Clearinghouse*.

Com a execução completa do ciclo de vida de um experimento foi possível confirmar que o AM implementa corretamente cada um dos métodos da API SFA (requisito R06). Consequentemente, também foram validadas as funcionalidades de descoberta do catálogo (requisito R01) e de reserva e provisionamento de recursos (requisito R02). Além disso, o uso do jFed também permitiu validar a integração com o Fed4FIRE (requisito R07).

### 5.3 Isolamento entre experimentos

Essa demonstração tem o objetivo de validar o isolamento entre experimentos (requisito R09). A demonstração consistirá em segregar o tráfego de rede entre experimentos



que serão criados a partir da mesma RSpec (provocando sobreposição de endereços IP). É esperado que o tráfego fique restrito ao experimento onde foi gerado, embora o mesmo endereço IP de destino seja utilizado no outro experimento.

Para preparar o cenário do teste, foi utilizada a *Request* RSpec gerada no teste anterior (Seção 5.2), disponível no Apêndice A. Através dessa RSpec, foram criados dois experimentos (*SliceA* e *SliceB*) que concorreram pelos recursos do *testbed*. Note que, por terem sido criados a partir da mesma RSpec, ambos os experimentos continham VMs e *bridge* com os mesmos nomes, além de utilizarem os mesmos *fixed* IPs. Desse modo, os dois *slices* possuem a mesma topologia, representada na Figura 66, com *vm1* utilizando o endereço 10.255.255.111 e *vm2* utilizando o endereço 10.255.255.112.

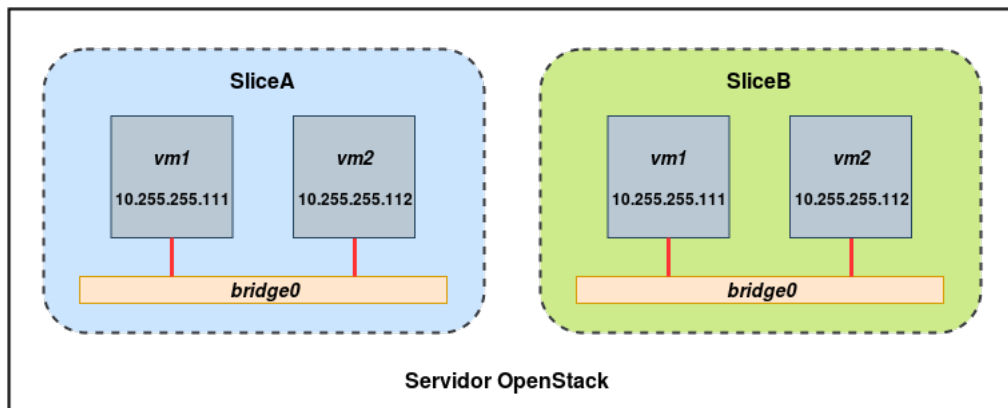


Figura 66 – Topologia utilizada nos experimentos.

A avaliação do mecanismo de isolamento de tráfego de rede foi realizada através do envio de tráfego da *vm1* do *SliceA* para o endereço IP 10.255.255.112, usado tanto na *vm2* do *SliceA* como na *vm2* do *SliceB*. Conforme já explicado no teste anterior (Seção 5.2), cada experimento é provisionado como um projeto no OpenStack. Portanto, é o OpenStack que deve assegurar a segregação de tráfego entre os projetos.

O primeiro passo foi abrir uma conexão SSH com cada uma das VMs dos *SliceA* e *SliceB*. Em seguida foi iniciado o servidor *iperf* na *vm2* de cada experimento. Para testar o isolamento, a *vm1* do *SliceA* atuou como cliente *iperf*, gerando tráfego com destino ao endereço 10.255.255.112. Foram enviados pacotes usando o protocolo TCP, durante 60 segundos, conforme Figura 67.

A taxa de *bytes* recebidos na interface de rede das *vm2* do *SliceA* e do *SliceB* foram monitoradas utilizando o Grafana. O resultado é apresentado na Figura 68, onde o gráfico superior é referente à *vm2* do *SliceA* e o gráfico inferior é referente à *vm2* do *SliceB*. Note que a *vm2* do *SliceA* registrou taxas superiores a 2 GB/s, enquanto que a *vm2* do *SliceB* não recebeu nenhum *byte*, conforme esperado.

Através dos dados apresentados nos gráficos, podemos comprovar a efetividade do mecanismo de isolamento de tráfego entre projetos no OpenStack, validando o requisito R09.

```

isabella@vm1:~$ sudo iperf -c 10.255.255.112 -f M -t 60 -i 2
sudo: unable to resolve host vm1
-----
Client connecting to 10.255.255.112, TCP port 5001
TCP window size: 0.04 MByte (default)
-----
[ 3] local 10.255.255.111 port 52236 connected with 10.255.255.112 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 3]  0.0- 2.0 sec  4510 MBytes 2255 MBytes/sec
[ 3]  2.0- 4.0 sec  3959 MBytes 1979 MBytes/sec
[ 3]  4.0- 6.0 sec  4246 MBytes 2123 MBytes/sec
[ 3]  6.0- 8.0 sec  4055 MBytes 2028 MBytes/sec
[ 3]  8.0-10.0 sec  3917 MBytes 1959 MBytes/sec
[ 3] 10.0-12.0 sec  4349 MBytes 2175 MBytes/sec
[ 3] 12.0-14.0 sec  4010 MBytes 2005 MBytes/sec
[ 3] 14.0-16.0 sec  4144 MBytes 2072 MBytes/sec
[ 3] 16.0-18.0 sec  4149 MBytes 2074 MBytes/sec

```

Figura 67 – Parâmetros de execução do teste.

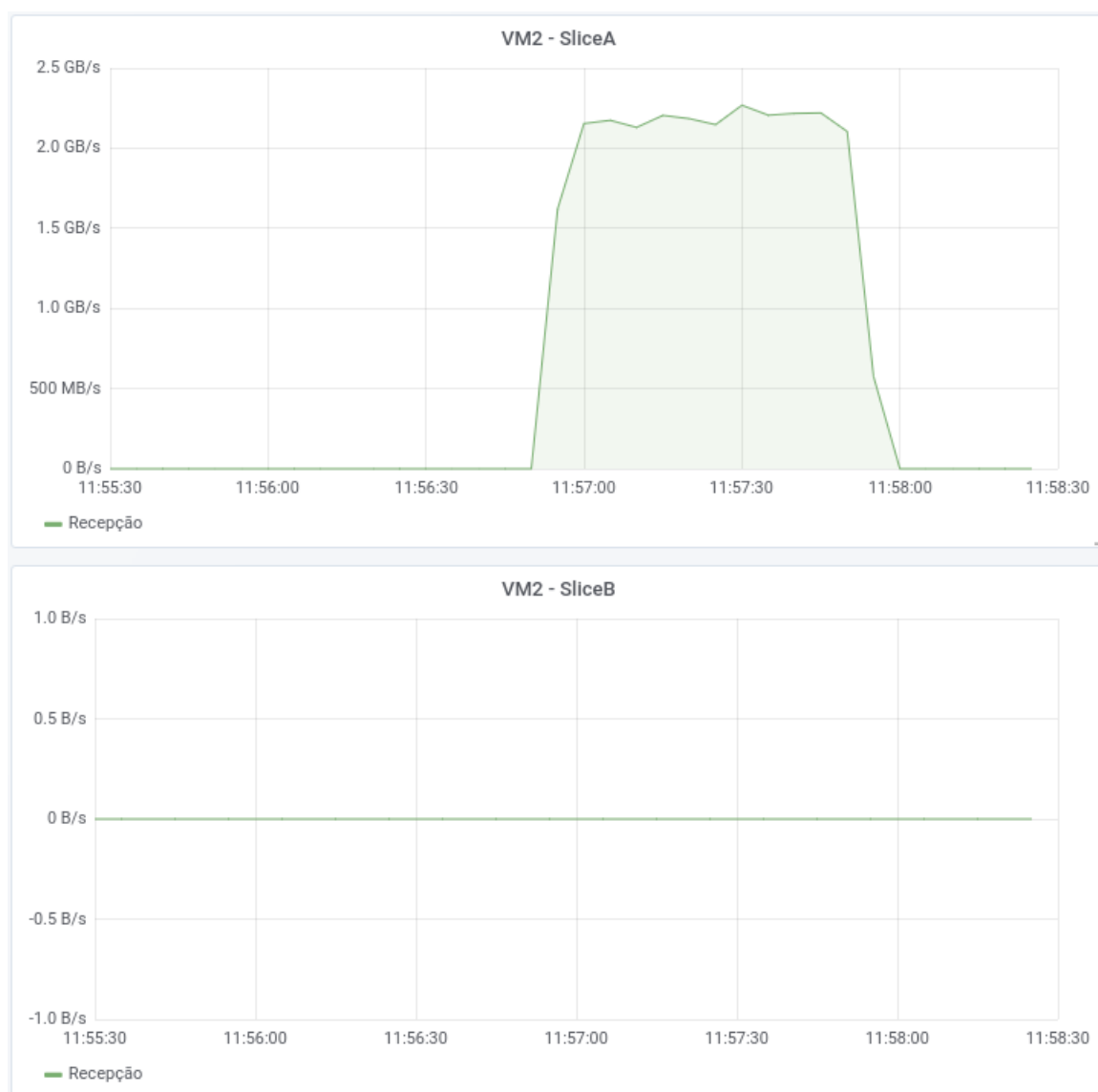


Figura 68 – Resultado da medição da taxa *bytes* recebidos pela *vm2* de cada um dos experimentos.

Esse mecanismo é oferecido pelo Neutron, que pode implementar a segregação de tráfego de duas formas diferentes: (i) através de VLANs, usando cabeçalho IEEE 802.1Q nos pacotes;

ou (ii) tunelamento através da camada de enlace (*L2 tunnels*), usando encapsulamento GRE. A instalação do OpenStack empregada nos testes foi configurada para a utilização de VLANs. Dessa forma, a *bridge* presente na *Request* RSpec é provisionada como uma rede privada no OpenStack, implementada através de uma VLAN. O OpenStack atribui um identificador (ID) de VLAN e executa automaticamente operações de *tag/untag* sempre que um pacote sai ou chega às interfaces de rede das VMs a ela conectadas. As redes de VLAN que compartilham a mesma rede física são isoladas umas das outras em camada 2. Isso possibilitou isolar o tráfego gerado no experimento, mesmo as redes dos *SliceA* e *SliceB* possuindo o mesmo nome (*bridge0*) e contendo espaços de endereços IP sobrepostos (OPENSTACK, 2019b). Uma descrição detalhada da implementação dos mecanismos de isolamento de rede do OpenStack pode ser encontrada em (HAMMAD et al., 2017).

## 5.4 Criação de VNF através de script TOSCA

Essa demonstração tem o objetivo de validar as funcionalidades de orquestração de NFV. Portanto, serão avaliados mecanismos de gestão de VNF através de políticas (requisito R03) e de execução automatizada de experimento (requisito R04). Essa avaliação se dará através da criação, utilizando *script* TOSCA, de VNF com política de *auto-scaling*. Consequentemente, será validado o suporte ao experimentador de perfil **especialista** (requisito R05).

Conforme discutido na Seção 3.2.1, a implantação de uma VNF é feita com base na descrição feita em um *script* TOSCA. Desse modo, com uma VNF é possível utilizar tanto a funcionalidade de execução automatizada de experimento como a funcionalidade de gestão através de políticas. Isso se deve ao fato de que o TOSCA contém a especificação dos atributos (como as VDUs e a topologia) e comportamento (como políticas de monitoramento e *scaling*) da VNF.

Para avaliar as funcionalidades de orquestração foi considerado um cenário onde uma página Web precisa se manter disponível aos usuários. Para isso, ela é hospedada através de uma VNF de servidor Web, associada a uma política para escalar automaticamente. Desse modo, se a quantidade de requisições aumenta, a política assegura a instanciação novas VDUs hospedando a página. O servidor Web foi implementado através de uma imagem do Ubuntu 16 com o Apache<sup>4</sup> instalado.

A preparação para o teste exigiu incluir novas imagens do repositório do Glance. Conforme Figura 69, foram incluídas as imagens do orquestrador (*nfv\_orchestrator*) e do servidor web (*vnf\_web\_server*). A primeira tem o objetivo de habilitar o acesso às funcionalidades de orquestração, e a outra tem a finalidade de compor a VNF utilizada no teste. Além disso, foi preciso atualizar o AM e o banco de dados para a versão *nfv\_ofc*.

<sup>4</sup> <<https://httpd.apache.org/>>

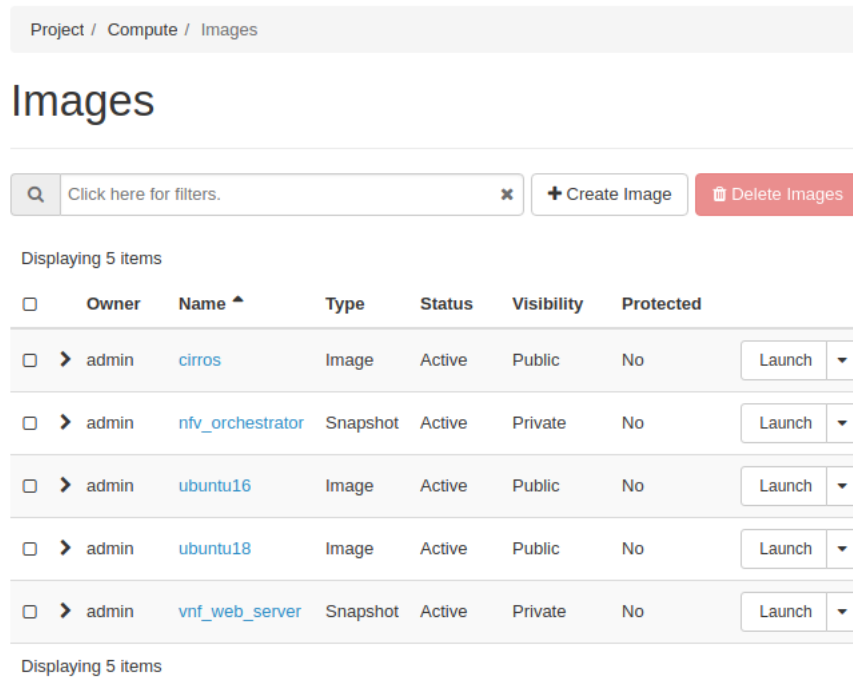


Figura 69 – Repositório de imagens atualizado.

Antes de preparar a *Request* RSpec, verificamos se a *Advertisement* RSpec já estava atualizada com o recurso **tenant**. Através do painel de *logs* do jFed, foi chamado o método *ListResources*. O resultado, exibido na Figura 70, permite identificar que **tenant** já está disponível na *Advertisement* RSpec. Conforme discutido na Seção 4.3.2, o recurso do tipo **tenant** tem a função de possibilitar a reserva de recursos “brutos” da nuvem. Esses recursos (memória, CPU e disco) serão utilizados na implantação da VNF de servidor Web.

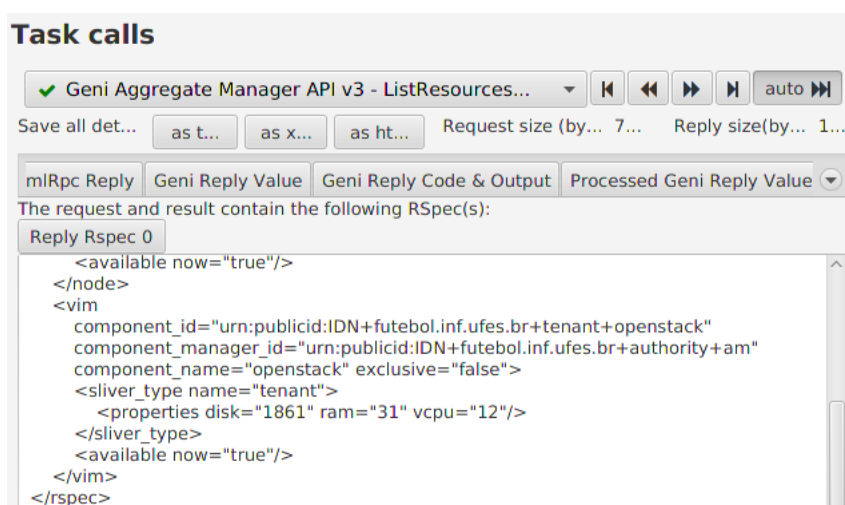


Figura 70 – Resultado da chamada ao método *ListResources*.

Neste teste não foi possível especificar o experimento no modo gráfico (Topology Editor), pois não há ícone para o recurso do tipo **tenant**. Desse modo, o experimento pre-

cisou ser especificado utilizando o **RSpec Editor**. A *Request* RSpec criada está disponível no Apêndice B. Nela está especificada a VM de orquestração e um recurso do tipo **tenant**. Assim como na versão inicial do O2CMF, cada experimento (*slice*) é implementado como um projeto no OpenStack. Com *tenant*, o que muda é que se torna possível deixar uma porção extra de recursos para serem utilizados com a implantação de VNFs. A partir dessa *Request* RSpec, foi criado um experimento com o nome de **Advanced** e duração de 1 dia.

Após a reserva e provisionamento dos recursos, foi acessado o Horizon para verificar os recursos instanciados. Conforme Figura 71, que mostra o painel de topologia de rede, o projeto contém apenas a VM **orchestrator**. Ela está ligada à rede privada **selfservice** e, por ter um *floating* IP, também está ligada à rede **physnet1**. Contudo, apenas a rede privada (**selfservice**) pode ser utilizada para dar conectividade às VNFs.

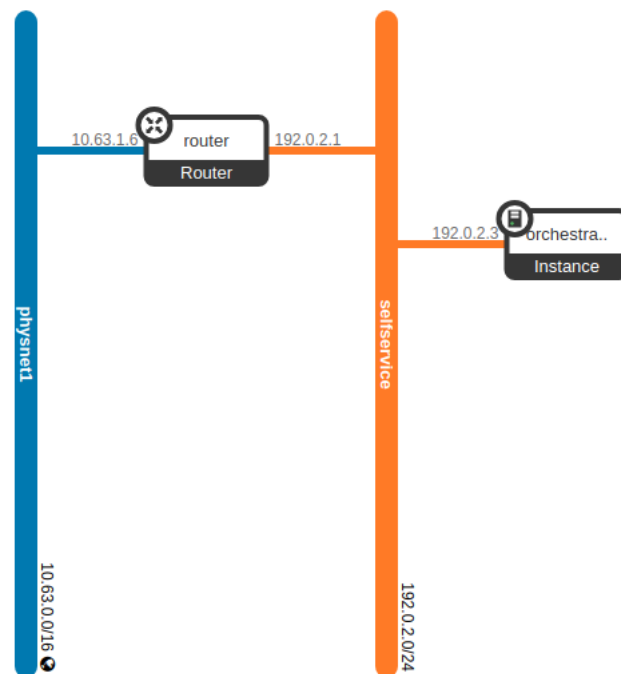


Figura 71 – Estado inicial da topologia de rede do *slice* **Advanced**.

O passo seguinte foi acessar a VM de orquestração (**orchestrator**) para criar a VNF. Primeiro, foi criado o *script* TOSCA descrevendo a VNF de servidor Web. Essa VNF foi descrita contendo 1 VDU utilizando a imagem *vnf\_web\_server*, 1 GB de memória, 1 CPU e 20 GB de disco. Além disso, esse *script* contém uma política para a realização de *auto-scaling*, implantando uma nova VDU sempre que for ultrapassado 70% de uso de CPU. O *script* TOSCA dessa VNF está disponível no Apêndice C. Em seguida, a VNF foi instanciada utilizando o Tackerclient. Conforme Figura 72, foi preciso informar um nome para a VNF (**web\_server**), além de seu descritor em TOSCA (*webserver.yaml*).

Em seguida, voltamos ao Horizon para verificar as VNFs criadas. No menu **NFV > VNF Manager** temos o painel de gestão de VNFs. Na Figura 73 é possível ver a VNF **web\_server**

```

isabella@orchestrator: ~
isabella@orchestrator:~$ tacker vnf-create --vnfd-template webserver.yaml web_server
Created a new vnf:
+-----+-----+
| Field | Value |
+-----+-----+
| created_at | 2019-02-04 17:47:00.229464 |
| description | O2CMF Test - Web Server |
| error_reason | |
| id | cc9fb8e1-3aec-4d1d-bafd-4ca57400c4c3 |
| instance_id | ee4aacd8-05c9-49ff-8040-376ec782c666 |
| mgmt_url | |
| name | web_server |
| placement_attr | {"vim_name": "VIM0"} |
| status | PENDING_CREATE |
| tenant_id | 52e35a0583574a18acd8bbf1d53a326cc |
| updated_at | |
| vim_id | f2d28c18-473a-4b39-a6ac-695bfdb0a32e |
| vnfd_id | a8b9a3ce-c813-40d9-b96a-d1871f5e1ffb |
+-----+-----+
isabella@orchestrator:~$

```

Figura 72 – Criação da VNF `web_server`.

ativa. Como cada VDU é implementada como uma VM, verificamos também o painel de gestão de instâncias (Figura 74), onde podemos identificar a VM correspondente à VDU1. A topologia de rede atualizada é apresentada na Figura 75. Desse modo, confirmamos a implantação bem sucedida do cenário do experimento.

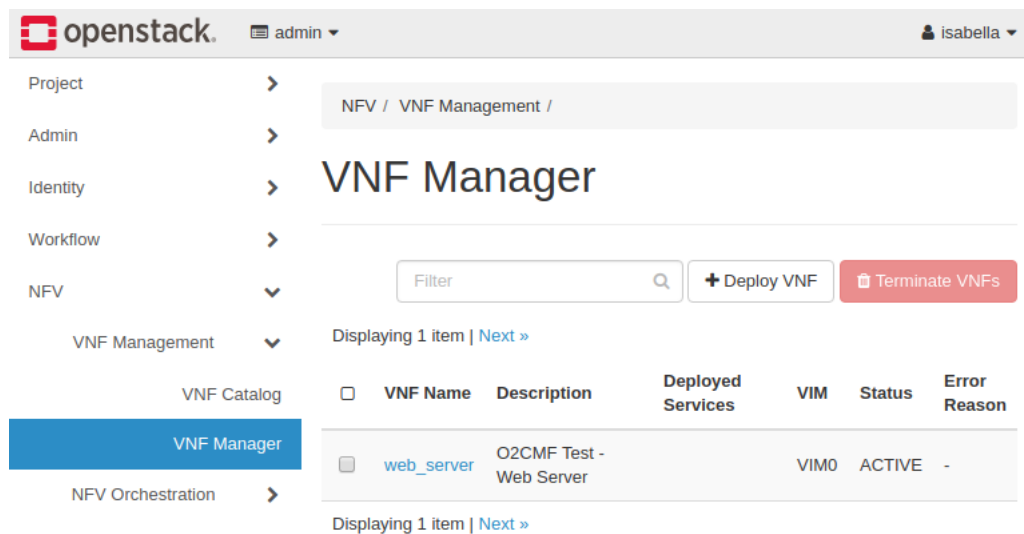
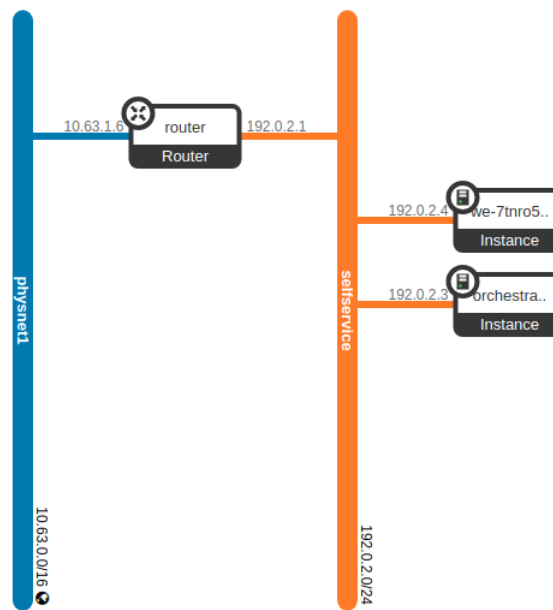


Figura 73 – Painel de gestão de VNFs.

A etapa seguinte do teste consistiu na validação da política de *auto-scaling* da VNF. Conforme o *script* TOSCA disponível no Apêndice C, a política é composta de um alarme e uma ação. O alarme, exibido na Listagem 5.1, estabelece um “gatilho” para acionar a ação de *scaling out*. Esse gatilho é uma condição relacionada a uma métrica: sempre que a média da utilização de CPU das VDUs ultrapassar 70%, deve ser disparada a ação SP1. Desse modo, o mecanismo que automatiza o *scaling* é implementado através de atuação conjunta dos módulos Ceilometer (telemetria), Aodh (sistema de alarme) e

<input type="checkbox"/> Instance Name	Image Name	IP Address	Flavor
<input type="checkbox"/> <a href="#">we-7tnro5-4ubjtn3ueho-wvehpxhhsffy-VDU1-isx7a2kaisnq</a>	<a href="#">vnf_web_server</a>	192.0.2.4	<a href="#">web_server_cc9fb8e1-3aec-4d1d-bafd-4ca57400c4c3-SP1_group-wpxjcx7tnro5-4ubjtn3ueho-wvehpxhhsffy-VDU1_flavor-5pop56prry4a</a>
<input type="checkbox"/> <a href="#">orchestrator</a>	<a href="#">nfv_orchestrator</a>	192.0.2.3 Floating IPs: 10.63.1.25	<a href="#">m1.small</a>

Displaying 2 items

Figura 74 – Painel de gestão de instâncias do *slice* Advanced.Figura 75 – Topologia de rede do *slice* Advanced após criação da VNF.

Gnocchi (armazenamento de séries de dados) ([AGUIAR; CORREA; TAVARES, 2018](#)).

```

1  - vdu_cpu_usage_monitoring_policy:
2    type: toska.policies.tacker.Alarming
3    triggers:
4      vdu_hcpu_usage_scaling_out:
5        event_type:
6          type: toska.events.resource.utilization
7          implementation: ceilometer
8        metric: cpu_util
9        condition:
10         threshold: 70
11         constraint: utilization greater_than 70%
12         granularity: 5
13         evaluations: 1
14         aggregation_method: mean
15         resource_type: instance
16         comparison_operator: gt
17     metadata: SG1
18     action: [SP1]

```

Listagem 5.1 – Alarme compoendo a política de *auto-scaling*.

A ação SP1, exibida na Listagem 5.2, estabelece que a VNF é inicialmente implantada com 1 VDU (`default_instances`). Quando a ação de *scaling* for acionada, será implantada 1 nova VDU (`increment`), podendo chegar até 3 VDUs implantadas (`max_instances`). Caso o alarme dispare quando há 3 VDUs ou dentro de um intervalo de 120 segundos desde o último disparo do alarme (`cooldown`), a ação de *scaling* não será executada (AGUIAR; CORREA; TAVARES, 2018).

```

1  - SP1:
2      type: tosca.policies.tacker.Scaling
3      targets: [VDU1]
4      properties:
5          increment: 1
6          cooldown: 120
7          min_instances: 1
8          max_instances: 3
9          default_instances: 1

```

Listagem 5.2 – Ação compondo a política de *auto-scaling*.

A configuração do cenário para provocar o *auto-scaling* consistiu na instalação do Siege<sup>5</sup> na VM de orquestração para gerar requisições. Essas requisições foram feitas com o comando `siege -c 4000 -d 3 -t 1M 192.0.2.4`. Ou seja, foram emulados 4000 clientes concorrentes que enviaram requisições durante 1 minuto, com um atraso randômico de 0 a 3 segundos entre elas. O destino das requisições foi o endereço IP da VDU1 (192.0.2.4). Como o Siege estava configurado para utilizar o protocolo HTTP, as requisições eram direcionadas à página *default* do Apache hospedado em VDU1. O impacto dessas requisições na utilização de CPU da VDU1 é apresentado na Figura 76. Nesse gráfico podemos notar que, durante alguns segundos, o uso de CPU superou os 70%, tornando verdadeira a condição de acionamento do alarme para escalar.

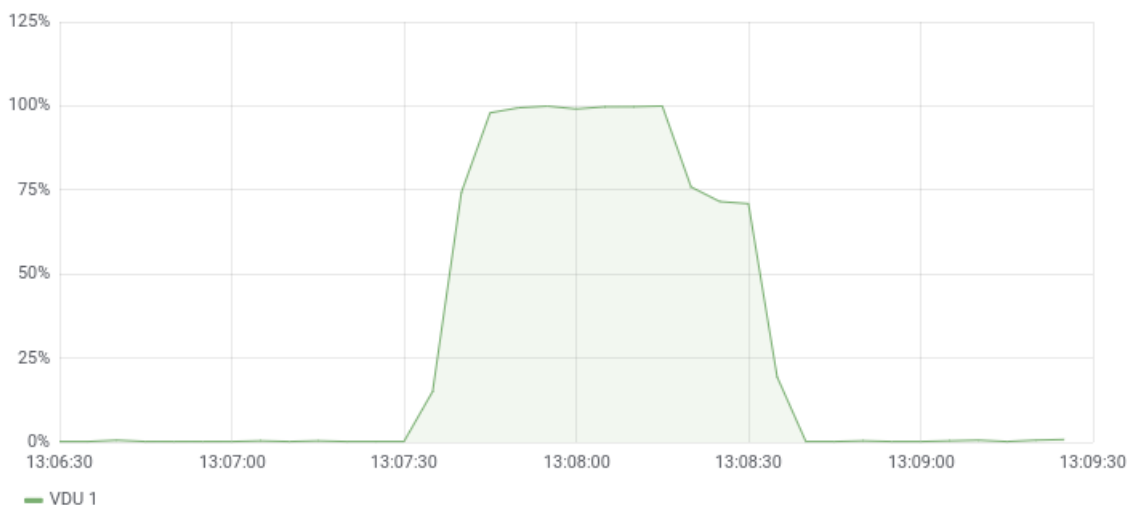


Figura 76 – Uso de CPU da VDU1.

<sup>5</sup> <<https://www.joedog.org/siege-home/>>



Para que a política de auto-scaling seja considerada efetiva, é preciso que uma nova VDU tenha sido implantada. Isso foi confirmado através da verificação da topologia de rede no Horizon. Conforme apresentado na Figura 77, há uma nova instância com endereço IP 192.0.2.6, correspondendo à nova VDU de `web_server`.

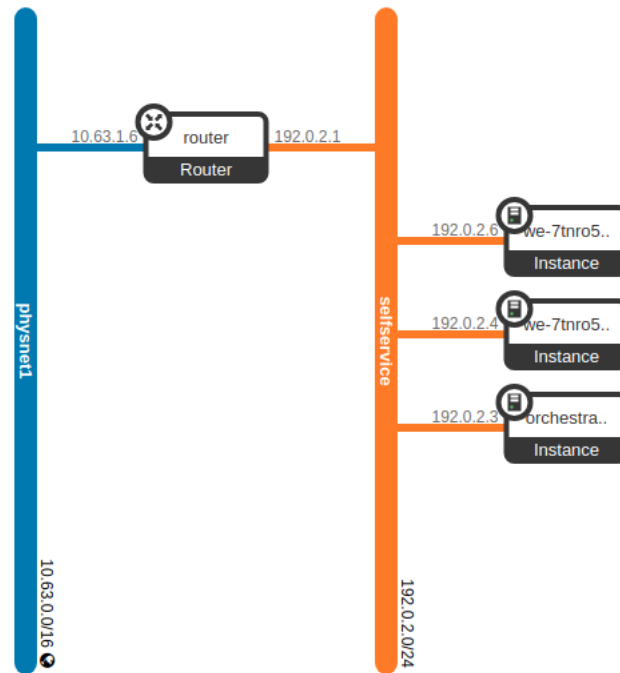


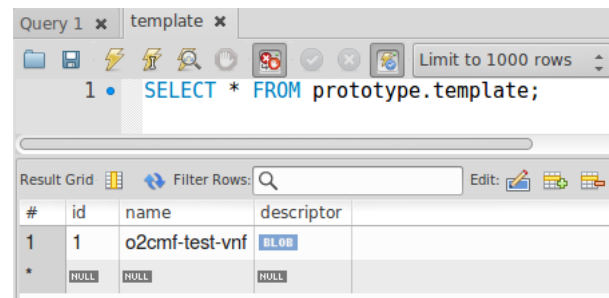
Figura 77 – Topologia de rede do *slice Advanced* após *scaling* da VNF.

Através das figuras obtidas do OpenStack, comprovamos a efetividade do *script* TOSCA na implantação automática do cenário do experimento (requisito R04). Além de prover automação na construção de experimentos com NFV, TOSCA também dá suporte à definição de políticas para gestão da VNF. Nesse teste, utilizamos uma política para escalar a VNF de modo automático. Com os resultados obtidos, validamos a implementação da política tal como especificado (requisito R03). Desse modo, foi validado o suporte ao experimentador de perfil **especialista** (requisito R05).

## 5.5 Criação de VNF através da RSpec

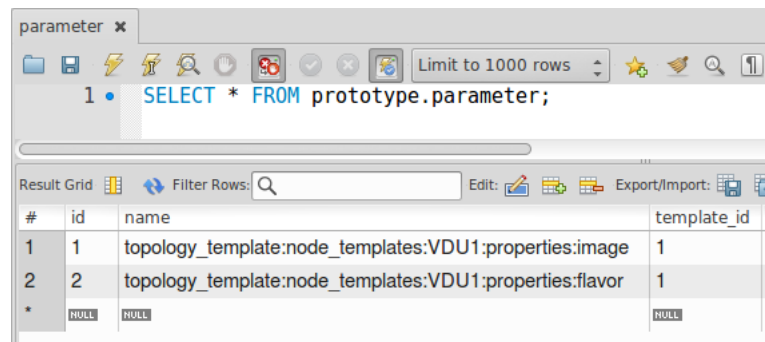
Essa demonstração tem o objetivo de validar o requisito R05, referente ao suporte a diferentes perfis de usuário. Ela é voltada ao experimentador em busca de uma alternativa ao TOSCA para especificação de VNFs. O cenário da demonstração consistirá na instanciação de uma VNF especificada a partir da RSpec. Será utilizado um modelo de VNF disponível no catálogo do O2CMF. Além disso, parâmetros desse modelo serão customizados na RSpec. A funcionalidade será validada através da comparação do modelo do catálogo com as características da VNF implantada.

A preparação para o teste consistiu em atualizar o AM e o banco de dados para a versão *nfv\_prototype*. Além disso, foi necessário popular o banco de dados com um modelo de VNF e seus parâmetros. O modelo de VNF é o *script* TOSCA que a descreve. A Figura 78 mostra que foi inserido na tabela `template` um *script* TOSCA, em formato binário (*BLOB*), e apelidado de `o2cmf-test-vnf`. Esse *script* está disponível no Apêndice D. A imagem e o *flavor* da VNF foram os parâmetros escolhidos para serem passíveis de customização pelo experimentador. A Figura 79 mostra esses parâmetros inseridos na tabela `parameter`.



#	id	name	descriptor
1	1	o2cmf-test-vnf	BLOB

Figura 78 – Dados inseridos na tabela `template` do banco de dados do AM.



#	id	name	template_id
1	1	topology_template:node_templates:VDU1:properties:image	1
2	2	topology_template:node_templates:VDU1:properties:flavor	1

Figura 79 – Dados inseridos na tabela `parameter` do banco de dados do AM.

Antes de preparar a *Request* RSpec, verificamos o resultado do método *ListResources* usando o painel de *logs* do jFed. A Figura 80, que mostra um trecho da *Advertisement* RSpec, permite identificar que o modelo `o2cmf-test-vnf` já está disponível.

Assim como no teste anterior (Seção 5.4), não foi possível especificar o experimento no modo gráfico (*Topology Editor*), pois não há ícone para o recurso modelo de VNF. Sendo assim, o experimento foi especificado utilizando o *RSPEC Editor*. A *Request* RSpec criada está disponível no Apêndice E. Nela está especificado um modelo de VNF e a VM de orquestração. O modelo de VNF está especificado para utilizar a imagem *nfv\_web\_server* e o *flavor* *o2cmf.base*. A partir dessa RSpec, foi criado um experimento com o nome de *beginner* e duração de 1 hora.

Após a reserva e provisionamento dos recursos, foi acessada a VM de orquestração (*orchestrator*). Com o comando `tacker vnf-list`, que lista as VNFs instanciadas, foi

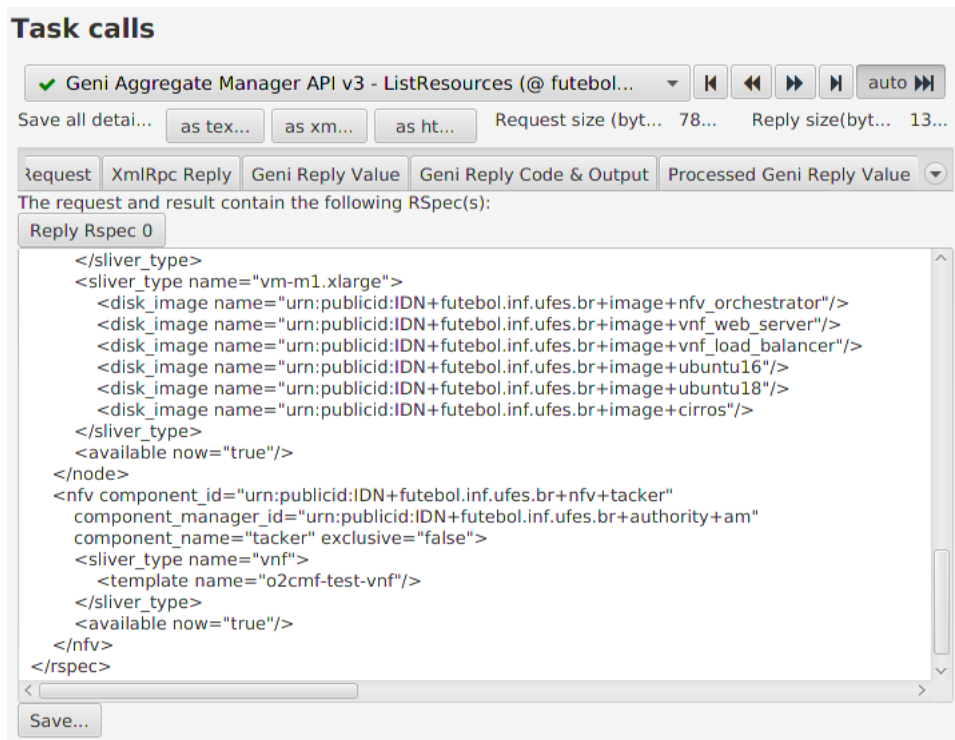


Figura 80 – Resultado da chamada ao método *ListResources*.

verificado que a VNF `o2cmf-test-vnf` havia sido realmente instanciada, conforme Figura 81. O resultado também mostra o endereço IP de sua VDU (192.0.2.24).

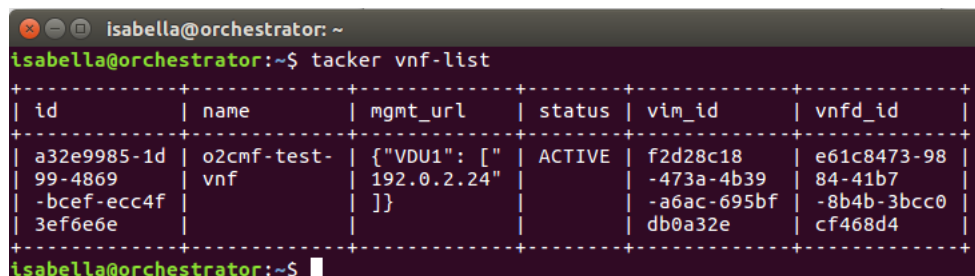


Figura 81 – Lista de VNFs instanciadas no experimento *beginner*.

Na *Request* RSpec, o modelo de VNF foi especificado com os parâmetros *image* e *flavor* customizados. O modelo de VNF originalmente utiliza a imagem *cirros* e *flavor m1-tiny*, conforme Apêndice D. Porém, na RSpec a imagem foi alterada para *vnf\_web\_server* e o *flavor* foi alterado para *o2cmf.base*. Para verificar se a customização foi realizada de fato, verificamos o painel de gestão de instâncias, conforme Figura 82. O painel mostra duas instâncias: uma com imagem *nfv\_orchestrator* e *flavor m1.small*, que é a VM de orquestração; e outra com a imagem *vnf\_web\_server* e *flavor o2cmf.base* que é a VDU. Sendo assim, a customização foi realizada com êxito.

Os dados apresentados nas figuras indicam que a implantação e customização do modelo de VNF foram bem sucedidas. Desse modo, comprovamos a efetividade da

Project / Compute / Instances

## Instances

Instance ID

Displaying 2 items

<input type="checkbox"/> Instance Name	Image Name	IP Address	Flavor
<input type="checkbox"/> <a href="#">o2-nnf3ob-fjgn25gefzpd-p6nllaisopfe-VDU1-2Int7sa37qjx</a>	vnf_web_server	192.0.2.24	<a href="#">o2cmf.base</a>
<input type="checkbox"/> <a href="#">orchestrator</a>	nfv_orchestrator	192.0.2.12 Floating IPs: 10.63.1.4	<a href="#">m1.small</a>

Displaying 2 items

Figura 82 – VM de orquestração e VDU de `o2cmf-test-vnf`.

funcionalidade do O2CMF para criação de experimentos com NFV em alto-nível. Essa funcionalidade torna a especificação de VNFs mais “familiar” ao usuário acostumado ao padrão da federação, além de proporcionar maior automação na implantação do experimento. Portanto, o requisito R05 está validado.

## 5.6 Reserva do espaço inteligente

Essa demonstração tem o objetivo de apresentar a contribuição do O2CMF para a realização do *showcase* do FUTEBOL. Ela será composta de duas partes: (i) reserva e provisionamento dos recursos; e (ii) orquestração convergente de experimento. A preparação para o teste consistiu em atualizar o AM e o banco de dados para a versão desenvolvida para o espaço inteligente.

A primeira parte ilustra o processo de preparação do experimento, conforme abordado na Seção 4.3.4. O primeiro passo foi realizar o *login* no jFed utilizando credenciais do Fed4FIRE. Em seguida, iniciou-se a especificação de um novo experimento, utilizando o RSpec Editor. A *Request* RSpec utilizada está disponível no Apêndice F. Essa RSpec contém 6 VNFs que incluem os serviços de controle das câmeras (`camera-gateway`), processamento de imagens (`image-processing`), controle do robô (`robot-controller`), conectividade sem-fio (`sdn-controller` e `mpeg-server`) e fila de mensagens (`rabbit-mq`). Todas essas VNFs são mandatórias para habilitar o funcionamento do espaço inteligente. Ademais, a RSpec contém a VM de orquestração (`orchestrator`), que é mandatória para permitir ao experimentador controlar os recursos. Ao final da RSpec está o espaço inteligente (`ispace`), contendo algumas configurações iniciais para o experimento:

- Taxa inicial de aquisição de imagens de 15 *fps*;
- Trajetória em forma de *lemniscata* com duração de 30 segundos, efetuando 3 voltas;
- Uso de apenas 1 dos *access points*.

Após a especificação dos recursos, foi realizada a reserva e provisionamento do experimento. A Figura 83 mostra o início desse processo, em que o experimento recebeu o nome de *exp22* e teve a duração especificada em 2 horas. Também é possível ver, ao fundo da figura, o RSpec Editor contendo parte da *Request RSpec*.

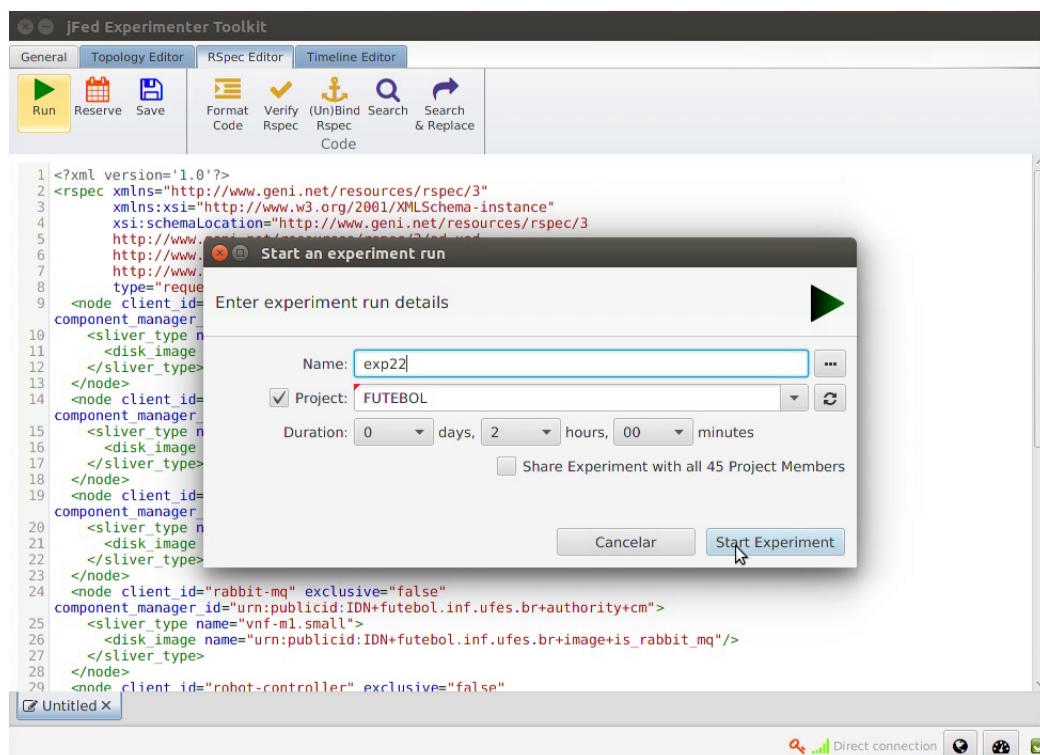


Figura 83 – Reserva dos recursos da nuvem e do espaço inteligente através do jFed.

Aguardamos até os recursos estarem prontos para acesso, conforme Figura 84. A partir desse momento, os recursos ficam sob o controle dos orquestradores. Isso significa que, até que a reserva expire ou o experimentador solicite o encerramento do experimento, o O2CMF não tem qualquer tipo de atuação no experimento.

Após estabelecer uma conexão SSH com a VM *orchestrator*, o usuário pode iniciar uma *dashboard* que reúne clientes dos orquestradores. Desse modo, a *dashboard* permite configurar recursos de diferentes domínios (robótica, nuvem e rede sem-fio). Além disso, ela possibilita executar o deslocamento do robô e coletar métricas.

A Figura 85 exibe o painel de controle dos recursos de robótica. Nesse painel é possível configurar a taxa de aquisição das câmeras, a trajetória do robô, a quantidade de voltas e a duração. Embora essas configurações possam ser feitas via RSpec, elas podem

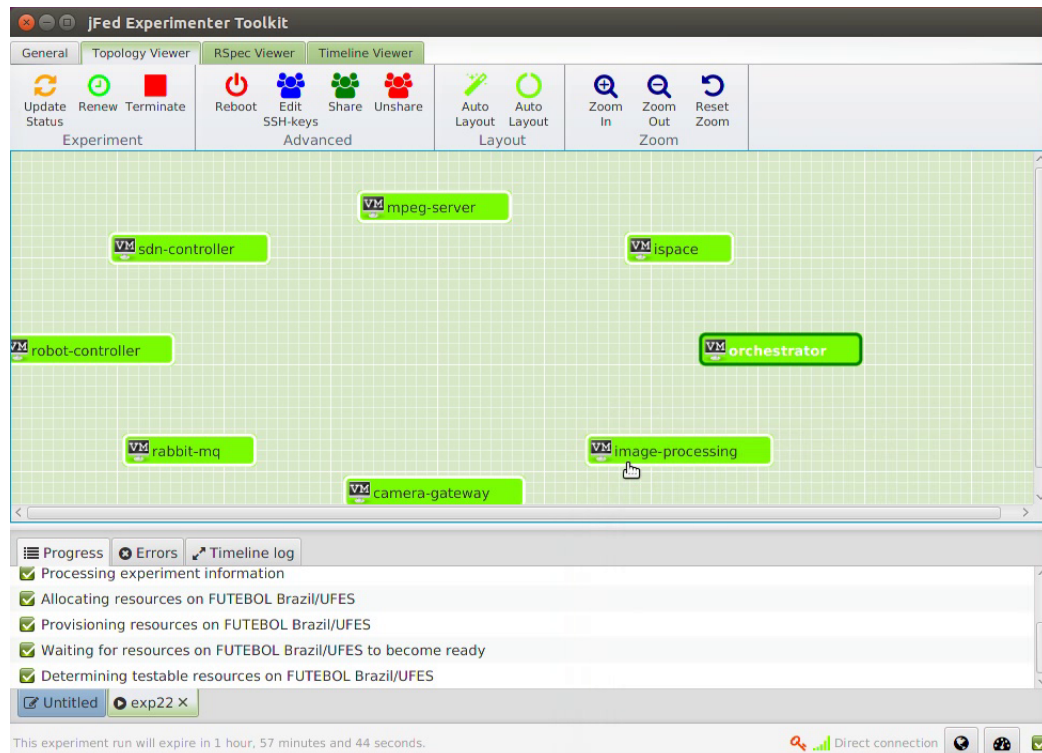


Figura 84 – Experimento provisionado.

ser modificadas na *dashboard* para dar maior liberdade ao experimentador. A presença dessas configurações na *Request* RSpec têm o objetivo de estabelecer valores *default* para o experimento, o que acaba favorecendo a reprodução deste. O painel de robótica também permite acompanhar a trajetória efetuada pelo robô, sua velocidade, o erro na trajetória e as imagens das câmeras.

No painel de controle da nuvem, exibido na Figura 86, é possível acompanhar a utilização de memória e CPU das VNFs e ainda ativar ou desativar a funcionalidade de *auto-scaling*. Já no painel de controle da comunicação sem-fio, exibido na Figura 87, é possível configurar a frequência e o canal das estações, além do mecanismo de *handover* (utilizado quando há mais de uma estação habilitada). Também são fornecidas métricas referentes à largura de banda.

Após configurar os recursos, o experimentador inicia o experimento (no painel da robótica). A partir de então, ocorre o seguinte *loop*: As imagens são capturadas pelas câmeras e enviadas por **camera-gateway** para uma fila de mensagens (fornecida por **rabbit-mq**). O serviço **image-processing** consome as imagens da fila, identifica o robô e publica em uma fila os pontos onde o robô se localiza. O serviço **robot-controller** consome essas mensagens e publica comandos para regular direção e aceleração, que são consumidos pelo robô. A coleta de métricas é feita de modo paralelo a esse *loop* de controle, assim como o controle da conectividade sem-fio.

Ao final da execução do deslocamento do robô, é solicitado o encerramento do





Figura 85 – Painel de orquestração dos recursos de robótica.



Figura 86 – Painel de orquestração dos recursos da nuvem.

experimento através do jFed. Após a liberação dos recursos, verificamos o painel de *logs* do jFed (Figura 88). Nele podemos observar que todas as interações com o AM e a *Clearinghouse* foram bem sucedidas.



Figura 87 – Pannel de orquestração dos recursos de comunicação sem-fio.

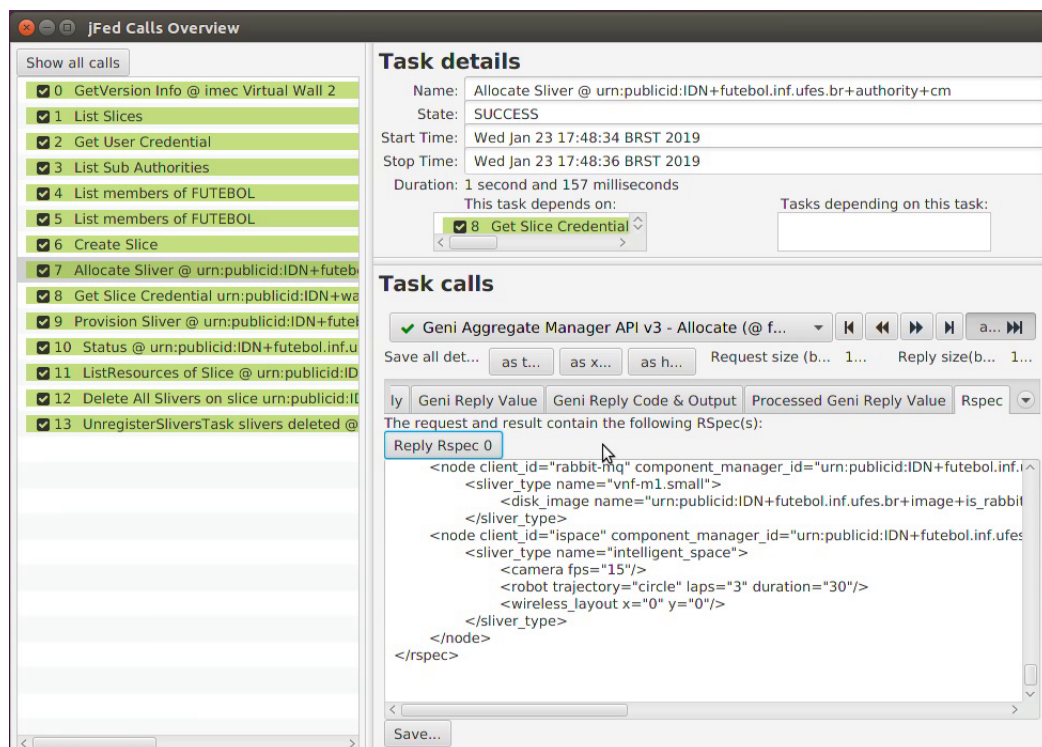


Figura 88 – Log de chamadas realizadas pelo jFed.

Através dessa demonstração, conseguimos validar a integração de NFV com o espaço inteligente. Embora a atuação do O2CMF seja restrita à reserva e provisionamento dos recursos, ele teve o papel de habilitar essa integração. Desse modo, o O2CMF trabalhou



---

expondo os recursos do espaço inteligente para a federação e promovendo a convergência de NFV para um domínio de aplicação específico.



## 6 Conclusão

Os CMFs estabelecidos anteriormente tiveram papel fundamental na definição da estrutura da federação (GCF) e no desenvolvimento de mecanismos para experimentação em redes ópticas (OCF) e sem fio (OMF). Contudo, o surgimento de novas tecnologias em redes, como NFV, gera a necessidade de evoluir a plataforma de experimentação federada, atualizando a infraestrutura e os mecanismos de controle. No contexto de NFV, essa evolução representa a introdução de novos recursos no catálogo de serviço (como a VNF) e a inclusão das funcionalidades de orquestração de funções de rede virtuais. Além disso, tornou-se necessário o aprimoramento de mecanismos de virtualização, isolamento entre experimentos e medição.

A proposta do O2CMF é habilitar a experimentação federada em NFV, mantendo a compatibilidade com a estrutura legada da federação. Para isso, ele oferece aos usuários do Fed4FIRE um catálogo que permite a criação de experimentos na nuvem empregando recursos de NFV ou VMs simples. O catálogo dispõe de modelos de VNF, opções de imagem de disco e configuração de hardware, agregando maior grau de escolha dos requisitos do experimento. Se o usuário quiser mais simplicidade, ele pode descrever na reserva quais VMs ou VNFs devem ser instanciadas e o O2CMF cuidará da configuração completa do experimento. Entretanto, se o usuário quiser mais controle de seu experimento, ele pode reservar recursos “brutos” da nuvem (CPU, memória e disco) para criar VNFs customizadas. Dessa forma, o O2CMF é capaz de alcançar diferentes perfis de usuários, provendo automação e programabilidade.

A arquitetura do O2CMF é composta de três entidades: o AM, o orquestrador e a nuvem. A nuvem foi a base para o desenvolvimento da estrutura arquitetural de NFV. Ela é responsável pela gestão da infraestrutura virtual e provê mecanismos para medição e isolamento. Além disso, a nuvem possibilita ao operador um controle fino da infraestrutura. O orquestrador permite ao usuário controlar de forma plena o ciclo de vida de VNFs, habilitando o monitoramento e o redimensionamento dinâmico de recursos. O AM é responsável por expor à federação os recursos do *testbed*, além de controlar as reservas e o acesso do usuário.

A implementação do O2CMF foi dividida em etapas, cada uma delas gerando uma versão. A versão inicial permitiu ao usuário criar uma topologia usando VMs e conectividade LAN. A segunda versão evoluiu para permitir também a criação de VNFs personalizadas a partir da reserva de um *slice* de recursos “brutos” da nuvem (CPU, memória e disco). A terceira versão permite ao usuário instanciar VNFs a partir de modelos disponíveis no catálogo. A última versão integra NFV aos domínios da robótica e comunicação sem fio.

Dessa forma, foi possível adaptar o modelo de dados e interações da federação para o modelo de NFV, criando uma plataforma que permite que duas especificações distintas (SFA e NFV) sejam interoperáveis.

Após a implementação, foram realizados testes para validar a aderência do O2CMF à especificação, apresentada na Seção 4.1. Ao todo, foram realizados seis testes abrangendo cada uma das versões e requisitos do O2CMF: (i) Definição de política para gestão do *testbed*; (ii) Reserva e provisionamento; (iii) Isolamento entre experimentos; (iv) Criação de VNF através de *script* TOSCA; (v) Criação de VNF através da RSpec; (vi) Reserva do espaço inteligente. Assim, foram cobertas funcionalidades de gestão do *testbed*, compatibilidade com o Fed4FIRE, segurança, orquestração e integração com outros domínios (robótica e redes sem-fio).

Os testes de validação demonstraram que o O2CMF segue a especificação, cumprindo com o objetivo de habilitar a experimentação em NFV no ambiente da federação. O O2CMF foi capaz de aliar a API originária do GCF à flexibilidade e robustez da nuvem, agregando mecanismos de controle do experimento em alto nível (tal como o OMF). Além disso, ele conseguiu oferecer flexibilidade ao público especialista, sem deixar de lado a simplicidade necessária para promover a inclusão de iniciantes ou público de outras áreas. Também foi demonstrado que é possível integrar NFV com outros domínios. A integração de ferramentas de orquestração de redes convergentes (integrando o domínio óptico ao sem fio) é um trabalho em andamento que visa aumentar a programabilidade dos recursos do *testbed*. Esse esforço abre caminho para integração de NFV em contextos como 5G, Indústria 4.0, *Cloud Robotics* e *Smart Cities*. Por último, mas não menos relevante, o O2CMF contribuiu com a inclusão no CMF de mecanismos de segurança e gestão do *testbed*, a fim facilitar do trabalho do operador.

O O2CMF foi utilizado inicialmente pela UFES, para a implantação de um *testbed* do projeto FUTEBOL. Por meio desse *testbed*, o O2CMF tem apoiado o desenvolvimento de atividades de pesquisa e educação em redes. Além de colaborar na realização do trabalho (DOMINICINI et al., 2018), o O2CMF já viabilizou a realização de atividades práticas das seguintes iniciativas:

- Minicurso “Uma Introdução Prática à Criação e Orquestração de VNFs em Nuvens” realizado no IV JACEE<sup>1</sup>.
- Minicurso “Teoria e Prática na Virtualização de Funções de Redes em Ambiente de Nuvem”<sup>2</sup> realizado no SBrT 2018<sup>3</sup>.

---

<sup>1</sup> <<http://www.jacee.ufes.br/>>

<sup>2</sup> Material disponível em <<http://nerds.ufes.br/tutorials>>

<sup>3</sup> <<http://www.sbrt.org.br/sbrt2018/>>

- Minicurso “Virtualização de funções de redes na nuvem” realizado no ENCOMP 2018<sup>4</sup>.

Nesse contexto, o O2CMF usa as características de nuvem para prover um ambiente de laboratório rapidamente implantável e facilmente reproduzível. No site do *testbed* UFES <<http://futebol.inf.ufes.br/>> há tutoriais para orientar os usuários no uso dos recursos oferecidos para experimentação. Isso inclui a descrição de cada um dos *flavors* e imagens disponíveis, assim como orientações para a criação de VNFs usando TOSCA.

O O2CMF também dispõe de uma *Wiki* <<https://gitlab.com/futebol/O2CMF/wikis/home>> para apoiar a implantação de outros *testbeds*. Na *Wiki* há instruções sobre a instalação, teste e extensão do O2CMF. O intuito dessa documentação é possibilitar que outras instituições façam como a Universidade de Bristol, que implantou recentemente um *testbed* utilizando o O2CMF.

Como trabalhos futuros, apontamos algumas oportunidades de melhoria. Foi identificada a necessidade de integrar de forma direta o OpenFlow ao O2CMF. Atualmente, a integração com o OpenFlow se restringe a aplicações de terceiros que compõem o orquestrador do espaço inteligente. No entanto, seria desejável incluir o OpenFlow internamente no O2CMF. Dessa forma, seria possível oferecer uma maior programabilidade da rede, além de alcançar um público maior. Outro ponto de melhoria é a integração do O2CMF a outras tecnologias, além das utilizadas pelo espaço inteligente, como meio de fomentar a inovação através de NFV. Também é desejável, unificar as diferentes versões do O2CMF, de modo a permitir, por exemplo, a utilização do espaço inteligente em conjunto com VNFs customizadas. Essa melhoria ampliaria as possibilidades de uso tanto do O2CMF como do espaço inteligente. Por fim, é sugerida a integração do O2CMF ao serviço de *stitching* da federação. Esse serviço é responsável por configurar e prover circuitos virtuais entre *testbeds*. A integração habilitaria o O2CMF a participar de experimentos envolvendo outros *testbeds* federados.

---

<sup>4</sup> <<http://encomp.com.br/>>



## Referências

- AGUIAR, E. *RPT4 - Relatório de teste da funcionalidade de escalabilidade de VNFs*. [S.l.], 2017. Citado na página 46.
- AGUIAR, E.; CORREA, J. H. G. M.; TAVARES, R. F. *RPT6: Relatório do teste da funcionalidade de VDU-Auto Scaling*. [S.l.], 2018. Citado 2 vezes nas páginas 105 e 106.
- AGUIAR, E.; TAVARES, R. F. *RPT1 - Relatório de detalhamento do Pacote de Trabalho 1*. [S.l.], 2017. Citado na página 45.
- AGUIAR, E.; TAVARES, R. F. *RPT2: Relatório de teste da funcionalidade de gerência do ciclo de vida de VNFs*. [S.l.], 2017. Citado 3 vezes nas páginas 11, 46 e 47.
- AGUIAR, E.; TAVARES, R. F. *RPT3 - Relatório de teste da funcionalidade de monitoramento de VNFs*. [S.l.], 2017. Citado na página 48.
- AMORIM, B. et al. *Arquitetura*. 2018. <<http://www.land.ufrj.br/~bamorim/redes2/openstack/arquitetura.html>>. Acesso em: 17-10-2018. Citado 2 vezes nas páginas 11 e 39.
- ANDREWS, J. G. et al. What will 5G be? *IEEE Journal on Selected Areas in Communications*, v. 32, n. 6, p. 1065–1082, 2014. ISSN 0733-8716. Disponível em: <<http://ieeexplore.ieee.org/ielx7/49/6862080/06824752.pdf?tp={&}arnumber=6824752{&}isnumber=6>>. Citado 2 vezes nas páginas 21 e 25.
- BECUE, P. et al. *Project Deliverable D5.1: Design of experimentation tools*. [S.l.], 2015. Citado 4 vezes nas páginas 11, 30, 31 e 32.
- BERMAN, M. et al. Geni: A federated testbed for innovative network experiments. *Computer Networks*, Elsevier, v. 61, p. 5–23, 2014. Citado 3 vezes nas páginas 29, 31 e 34.
- BERMAN, M. et al. Future internets escape the simulator. *Communications of the ACM*, ACM, New York, NY, USA, v. 58, n. 6, p. 78–89, maio 2015. ISSN 0001-0782. Disponível em: <<http://doi.acm.org/10.1145/2699392>>. Citado na página 21.
- CALED, D. et al. The next generation of the fibre software architecture. *Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF – SBRC)*, v. 8, n. 1/2017, 2017. ISSN 2595-2692. Disponível em: <<http://ojs.sbc.org.br/index.php/wpeif/article/view/2601>>. Citado na página 34.
- CERAVOLO, I. A. et al. Proposta e implementação de um framework de controle para testbeds federados que integram nuvem e sdn. *Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF - SBRC)*, v. 8, n. 1/2017, 2017. ISSN 2595-2692. Disponível em: <<http://portaldeconteudo.sbc.org.br/index.php/wpeif/article/view/2605>>. Citado na página 63.
- CERAVOLO, I. A. et al. O2cmf: Experiment-as-a-service for agile fed4fire deployment of programmable nfv. In: *Optical Fiber Communication Conference*. Optical Society of America, 2018. p. Tu3D.13. Disponível em: <<http://www.osapublishing.org/abstract.cfm?URI=OFC-2018-Tu3D.13>>. Citado na página 66.

- DOMINICINI, C. k. et al. Enabling experimental research through converged orchestration of optical, wireless, and cloud domains. In: *Proceedings of the 27th European Conference on Networks and Communications (EUCNC)*, 18-21 June 2018, Ljubljana, Slovenia. [S.l.: s.n.], 2018. p. 370–371. Citado na página 118.
- ETSI NFV ISG. *NFV 002 V1.2.1. Network Functions Virtualisation (NFV): Architectural Framework*. 2014. Citado 2 vezes nas páginas 44 e 55.
- FED4FIRE. *About Fed4FIRE*. 2012. <<https://old.fed4fire.eu/the-project/>>. Acesso em: 30-09-2018. Citado na página 28.
- FED4FIRE. *Add Your Facility*. 2012. <<https://old.fed4fire.eu/add-your-facility/>>. Acesso em: 30-09-2018. Citado 3 vezes nas páginas 28, 53 e 55.
- FED4FIRE. *jFed*. 2012. <<https://old.fed4fire.eu/jfed/>>. Acesso em: 30-09-2018. Citado 2 vezes nas páginas 11 e 28.
- FED4FIRE. *Run Your Experiment*. 2012. <<https://old.fed4fire.eu/run-your-experiment/>>. Acesso em: 30-09-2018. Citado na página 29.
- FED4FIRE. *Federation AM API RSpec*. 2014. <<https://fed4fire-testbeds.ilabt.iminds.be/asciidoc/rspec.html>>. Acesso em: 30-09-2018. Citado na página 58.
- FED4FIRE+. *About Fed4FIRE+*. 2017. <<https://www.fed4fire.eu/the-project/>>. Acesso em: 30-09-2018. Citado na página 28.
- FIBRE. *OMF Control Monitoring Framework*. 2018. <<http://www.fibre-ict.eu/index.php/cmfmf/omf>>. Acesso em: 15-09-2018. Citado 2 vezes nas páginas 11 e 34.
- FUGA CLOUD. *Technical overview*. 2018. <<https://fuga.cloud/documentation/>>. Acesso em: 17-10-2018. Citado 2 vezes nas páginas 11 e 42.
- FUTEBOL. *Concept and Approach – FUTEBOL*. 2016. <<http://www.ict-futebol.org.br/index.php/about/concept-and-approach/>>. Acesso em: 15-09-2018. Citado 3 vezes nas páginas 11, 26 e 27.
- FUTEBOL. *Objectives – FUTEBOL*. 2016. <<http://www.ict-futebol.org.br/index.php/about/objectives/>>. Acesso em: 15-09-2018. Citado na página 26.
- GENI. *GENI Aggregate Manager API Version 3*. 2014. <[http://groups.geni.net/geni/wiki/GAPI\\_AM\\_API\\_V3](http://groups.geni.net/geni/wiki/GAPI_AM_API_V3)>. Acesso em: 05-10-2018. Citado na página 32.
- GENI. *GENI Aggregate Manager API Version 3 Common Concepts*. 2014. <[http://groups.geni.net/geni/wiki/GAPI\\_AM\\_API\\_V3/CommonConcepts](http://groups.geni.net/geni/wiki/GAPI_AM_API_V3/CommonConcepts)>. Acesso em: 05-10-2018. Citado 2 vezes nas páginas 11 e 32.
- GENI. *Clearinghouse*. 2015. <<http://groups.geni.net/geni/wiki/GeniClearinghouse>>. Acesso em: 05-10-2018. Citado na página 30.
- GENI. *GENI API: Certificates*. 2016. <<http://groups.geni.net/geni/wiki/GeniApiCertificates>>. Acesso em: 05-10-2018. Citado na página 33.
- GENI. *GENI API Identifiers*. 2016. <<http://groups.geni.net/geni/wiki/GeniApiIdentifiers>>. Acesso em: 05-10-2018. Citado na página 33.



- GENI. *GENI Credentials*. 2017. <<http://groups.geni.net/geni/wiki/GeniApiCredentials>>. Acesso em: 05-10-2018. Citado na página 33.
- HAMMAD, A. et al. *D4.1: FUTEBOL Control Framework Design*. [S.l.], 2016. Citado 5 vezes nas páginas 11, 27, 29, 52 e 53.
- HAMMAD, A. et al. *D4.2: Implementation and validation of control framework*. [S.l.], 2017. Citado na página 101.
- HUANG, T. et al. A survey on large-scale software defined networking (sdn) testbeds: Approaches and challenges. *IEEE Communications Surveys Tutorials*, v. 19, n. 2, p. 891–917, Secondquarter 2017. ISSN 1553-877X. Citado na página 35.
- JFED. *jFed*. 2012. <<https://jfed.ilabt.imec.be/>>. Acesso em: 30-09-2018. Citado na página 28.
- MARQUES, P. et al. *D2.2: Specification of FUTEBOL showcases*. [S.l.], 2017. Citado 2 vezes nas páginas 12 e 71.
- MARTINELLO, M. et al. *D4.3: Documented control framework*. [S.l.], 2018. Citado na página 53.
- MARTINEZ, V. G. et al. Ultra reliable communication for robot mobility enabled by sdn splitting of wifi functions. In: *IEEE Symposium on Computers and Communications*. [S.l.]: IEEE Symposium on Computers and Communications, 2018. Citado na página 71.
- MELL, P.; GRANCE, T. The nist definition of cloud computing. *National institute of standards and technology*, v. 53, n. 6, p. 50, 2009. Citado 4 vezes nas páginas 37, 38, 51 e 55.
- MIJUMBI, R. et al. Management and orchestration challenges in network functions virtualization. *IEEE Communications Magazine*, IEEE, v. 54, n. 1, p. 98–105, 2016. Citado 3 vezes nas páginas 11, 44 e 45.
- MIJUMBI, R. et al. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys Tutorials*, v. 18, n. 1, p. 236–262, Firstquarter 2016. ISSN 1553-877X. Citado 6 vezes nas páginas 11, 22, 43, 44, 51 e 55.
- MIRANTIS. *Configuring Floating IP addresses for Networking in OpenStack Public and Private Clouds*. 2012. <<https://www.mirantis.com/blog/configuring-floating-ip-addresses-networking-openstack-public-private-clouds/>>. Acesso em: 05-12-2018. Citado 3 vezes nas páginas 11, 42 e 62.
- OPENSTACK. *Scenario: Classic with Open vSwitch*. 2016. <<http://docs.openstack.org/liberty/networking-guide/scenario-classic-ovs.html>>. Acesso em: 26-12-2016. Citado 2 vezes nas páginas 11 e 41.
- OPENSTACK. *Alarm monitoring framework*. 2018. <[https://docs.openstack.org/tacker/latest/user/alarm\\_monitoring\\_usage\\_guide.html](https://docs.openstack.org/tacker/latest/user/alarm_monitoring_usage_guide.html)>. Acesso em: 05-12-2018. Citado na página 47.
- OPENSTACK. *Create and manage roles*. 2018. <<https://docs.openstack.org/horizon/latest/admin/admin-manage-roles.html>>. Acesso em: 17-10-2018. Citado na página 40.

- OPENSTACK. *Flavors*. 2018. <<https://docs.openstack.org/nova/latest/user/flavors.html>>. Acesso em: 17-10-2018. Citado 3 vezes nas páginas 41, 80 e 81.
- OPENSTACK. *Get images*. 2018. <<https://docs.openstack.org/image-guide/obtain-images.html>>. Acesso em: 17-10-2018. Citado 2 vezes nas páginas 81 e 82.
- OPENSTACK. *Manage IP addresses*. 2018. <<https://docs.openstack.org/ocata/user-guide/cli-manage-ip-addresses.html>>. Acesso em: 17-10-2018. Citado na página 42.
- OPENSTACK. *Manage projects, users, and roles*. 2018. <<https://docs.openstack.org/keystone/pike/admin/cli-manage-projects-users-and-roles.html>>. Acesso em: 17-10-2018. Citado na página 40.
- OPENSTACK. *Metadata Definition Concepts*. 2018. <<https://docs.openstack.org/glance/pike/user/metadefs-concepts.html>>. Acesso em: 17-10-2018. Citado na página 63.
- OPENSTACK. *Open Source software for creating private and public clouds*. 2018. <<https://www.openstack.org/>>. Acesso em: 17-10-2018. Citado na página 39.
- OPENSTACK. *OpenStack Virtual Machine Image Guide*. 2018. <<https://docs.openstack.org/image-guide/>>. Acesso em: 17-10-2018. Citado na página 82.
- OPENSTACK. *Quotas*. 2018. <<https://wiki.openstack.org/wiki/Quotas>>. Acesso em: 17-10-2018. Citado na página 63.
- OPENSTACK. *Set environment variables using the OpenStack RC file*. 2018. <[https://docs.openstack.org/zh\\_CN/user-guide/common/cli-set-environment-variables-using-openstack-rc.html](https://docs.openstack.org/zh_CN/user-guide/common/cli-set-environment-variables-using-openstack-rc.html)>. Acesso em: 17-10-2018. Citado na página 42.
- OPENSTACK. *Software – SDKs*. 2018. <<https://www.openstack.org/software/project-navigator/sdks>>. Acesso em: 17-10-2018. Citado na página 42.
- OPENSTACK. *Tacker*. 2018. <<https://wiki.openstack.org/wiki/Tacker>>. Acesso em: 05-12-2018. Citado 4 vezes nas páginas 11, 45, 46 e 55.
- OPENSTACK. *tosca-vnfd-alarm-respawn.yaml*. 2018. <<https://github.com/openstack/tacker/blob/master/samples/tosca-templates/vnfd/tosca-vnfd-alarm-respawn.yaml>>. Acesso em: 05-12-2018. Citado na página 145.
- OPENSTACK. *tosca-vnfd-alarm-scale.yaml*. 2018. <<https://github.com/openstack/tacker/blob/master/samples/tosca-templates/vnfd/tosca-vnfd-alarm-scale.yaml>>. Acesso em: 05-12-2018. Citado na página 143.
- OPENSTACK. *VNF Descriptor Template Guide*. 2018. <[https://docs.openstack.org/tacker/latest/contributor/vnfd\\_template\\_description.html](https://docs.openstack.org/tacker/latest/contributor/vnfd_template_description.html)>. Acesso em: 05-12-2018. Citado na página 46.
- OPENSTACK. *Welcome to the Heat documentation!* 2018. <<https://docs.openstack.org/heat/latest/>>. Acesso em: 17-10-2018. Citado na página 40.
- OPENSTACK. *InstanceResourceQuota*. 2019. <<https://wiki.openstack.org/wiki/InstanceResourceQuota>>. Acesso em: 05-01-2019. Citado na página 84.

- OPENSTACK. *Networking services*. 2019. <<https://docs.openstack.org/security-guide/networking/services.html>>. Acesso em: 05-01-2019. Citado na página 101.
- OSS LINE. *What is Network Function Virtualization?* 2017. <<http://www.ossline.com/2017/09/what-is-network-function-virtualization-nfv.html>>. Acesso em: 05-12-2018. Citado 2 vezes nas páginas 11 e 43.
- PETERSON, L. et al. *Slice Federation Architecture 2.0*. 2010. <<http://groups.geni.net/geni/wiki/SliceFedArch>>. Acesso em: 05-10-2018. Citado 3 vezes nas páginas 29, 30 e 31.
- RAKOTOARIVELO, T. et al. Omf: a control and management framework for networking testbeds. *ACM SIGOPS Operating Systems Review*, ACM, v. 43, n. 4, p. 54–59, 2010. Citado 2 vezes nas páginas 33 e 34.
- RED HAT. *Chapter 1. Components*. 2018. <[https://access.redhat.com/documentation/en-us/red\\_hat\\_openstack\\_platform/8/html/architecture\\_guide/components](https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/8/html/architecture_guide/components)>. Acesso em: 17-10-2018. Citado 3 vezes nas páginas 11, 40 e 41.
- RED HAT. *Introdução ao OpenStack*. 2018. <<https://www.redhat.com/pt-br/topics/openstack>>. Acesso em: 17-10-2018. Citado 2 vezes nas páginas 39 e 55.
- REICHERT, C. *MWC 2018: Intel and Huawei showcase 5G interoperability*. 2018. <<https://www.zdnet.com/article/mwc-2018-intel-and-huawei-showcase-5g-interoperability/>>. Acesso em: 15-09-2018. Citado 2 vezes nas páginas 11 e 25.
- SALMITO, T. et al. Fibre: an international testbed for future internet experimentation. In: *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC*. [S.l.: s.n.], 2014. p. p-969. Citado 2 vezes nas páginas 34 e 35.
- SAN ENTHUSIAST. *Introduction to Cloud Computing*. 2016. <<https://sanenthusiast.com/tag/nist-definition-of-cloud-computing/>>. Acesso em: 17-10-2018. Citado 2 vezes nas páginas 11 e 38.
- SANTOS, P. et al. Monitoramento de recursos para aplicações de robótica em espaços inteligentes baseados em uma nuvem openstack. In: *Workshop Em Clouds E Aplicações (WCGA - SBRC)*. [S.l.]: Workshop Em Clouds E Aplicações (WCGA - SBRC), 2018. Citado na página 71.
- SOUSA, N. F. S. de et al. Network service orchestration: A survey. *arXiv preprint arXiv:1803.06596*, 2018. Citado na página 22.
- SUNE, M. et al. Design and implementation of the ofelia fp7 facility: The european openflow testbed. *Comput. Netw.*, Elsevier North-Holland, Inc., New York, NY, USA, v. 61, p. 132–150, mar. 2014. ISSN 1389-1286. Disponível em: <<http://dx.doi.org/10.1016/j.bjp.2013.10.015>>. Citado na página 34.
- VANDENBERGHE, W. et al. Architecture for the heterogeneous federation of future internet experimentation facilities. In: IEEE. *Future Network and Mobile Summit (FutureNetworkSummit)*, 2013. [S.l.], 2013. p. 1–11. Citado 3 vezes nas páginas 11, 21 e 22.



## Apêndices



# APÊNDICE A – Request RSpec contendo duas VMs

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <rspec xmlns="http://www.geni.net/resources/rspec/3"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://www.geni.net/resources/rspec/3
5     http://www.geni.net/resources/rspec/3/ad.xsd
6     http://www.geni.net/resources/rspec/ext/opstate/1
7     http://www.geni.net/resources/rspec/ext/opstate/1/ad.xsd"
8   type="request">
9   <node client_id="vm1" exclusive="false"
10     component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
11     <sliver_type name="vm-m1.small">
12       <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu"/>
13     </sliver_type>
14     <interface client_id="vm1:eth0">
15       <ip address="10.255.255.111" netmask="255.255.255.0" type="ipv4"/>
16     </interface>
17   </node>
18   <node client_id="vm2" exclusive="false"
19     component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
20     <sliver_type name="vm-m1.small">
21       <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu"/>
22     </sliver_type>
23     <interface client_id="vm2:eth0">
24       <ip address="10.255.255.112" netmask="255.255.255.0" type="ipv4"/>
25     </interface>
26   </node>
27   <node client_id="bridge0" exclusive="false"
28     component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
29     <sliver_type name="bridge"/>
30     <interface client_id="bridge0:eth0"/>
31     <interface client_id="bridge0:eth1"/>
32   </node>
33   <link client_id="link0">
34     <component_manager name="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm"/>
35     <interface_ref client_id="vm1:eth0"/>
36     <interface_ref client_id="bridge0:eth0"/>
37     <link_type name="lan"/>
38   </link>
39   <link client_id="link1">
40     <component_manager name="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm"/>
41     <interface_ref client_id="vm2:eth0"/>
42     <interface_ref client_id="bridge0:eth1"/>
43     <link_type name="lan"/>
44   </link>
45 </rspec>

```





## APÊNDICE B – *Request* RSpec

### requisitando uma porção dos recursos da nuvem e a VM de orquestração

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <rspec xmlns="http://www.geni.net/resources/rspec/3"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://www.geni.net/resources/rspec/3
5     http://www.geni.net/resources/rspec/3/ad.xsd
6     http://www.geni.net/resources/rspec/ext/opstate/1
7     http://www.geni.net/resources/rspec/ext/opstate/1/ad.xsd"
8     type="request">
9   <vim client_id="my-project" exclusive="false"
10     component_manager_id="urn:publicid:IDN+futebolufes+authority+cm">
11     <sliver_type name="tenant">
12       <properties disk="80" ram="6" vcpu="4"/>
13     </sliver_type>
14   </vim>
15   <node client_id="orchestrator" exclusive="false"
16     component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
17     <sliver_type name="vm-m1.small">
18       <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+nfv_orchestrator"/>
19     </sliver_type>
20   </node>
21 </rspec>
```



# APÊNDICE C – *Script* TOSCA descrevendo servidor Web

```

1  tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0
2  description: 02CMF Test - Web Server
3
4  metadata:
5    template_name: vnfd-web-server
6
7  topology_template:
8    node_templates:
9      VDU1:
10       type: tosca.nodes.nfv.VDU.Tacker
11       capabilities:
12         nfv_compute:
13           properties:
14             disk_size: 20 GB
15             mem_size: 1024 MB
16             num_cpus: 1
17       properties:
18         image: vnf_web_server
19         mgmt_driver: noop
20         availability_zone: nova
21         metadata: {metering.server_group: SG1}
22
23      CP1:
24       type: tosca.nodes.nfv.CP.Tacker
25       properties:
26         management: true
27         anti_spoofing_protection: false
28       requirements:
29         - virtualLink:
30           node: VL1
31         - virtualBinding:
32           node: VDU1
33
34      VL1:
35       type: tosca.nodes.nfv.VL
36       properties:
37         network_name: selfservice
38         vendor: Tacker
39
40  policies:
41    - SP1:
42      type: tosca.policies.tacker.Scaling
43      targets: [VDU1]
44      properties:
45        increment: 1
46        cooldown: 120
47        min_instances: 1
48        max_instances: 3

```

```
49     default_instances: 1
50
51 - vdu_cpu_usage_monitoring_policy:
52     type: tosca.policies.tacker.Alarming
53     triggers:
54         vdu_hcpu_usage_scaling_out:
55             event_type:
56                 type: tosca.events.resource.utilization
57                 implementation: ceilometer
58             metric: cpu_util
59             condition:
60                 threshold: 70
61                 constraint: utilization greater_than 70%
62                 granularity: 5
63                 evaluations: 1
64                 aggregation_method: mean
65                 resource_type: instance
66                 comparison_operator: gt
67             metadata: SG1
68             action: [SP1]
```

# APÊNDICE D – Modelo de VNF

## disponibilizado no catálogo

```

1  tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0
2  description: Generic VNF for O2CMF tests
3
4  metadata:
5    template_name: o2cmf-test-vnf
6
7  topology_template:
8    node_templates:
9      VDU1:
10       type: tosca.nodes.nfv.VDU.Tacker
11       properties:
12         image: cirros
13         flavor: m1.tiny
14         mgmt_driver: noop
15         availability_zone: nova
16         metadata: {metering.server_group: SG1}
17
18      CP1:
19       type: tosca.nodes.nfv.CP.Tacker
20       properties:
21         management: true
22         anti_spoofing_protection: false
23       requirements:
24         - virtualLink:
25           node: VL1
26         - virtualBinding:
27           node: VDU1
28
29      VL1:
30       type: tosca.nodes.nfv.VL
31       properties:
32         network_name: selfservice
33         vendor: Tacker
34
35  policies:
36    - SP1:
37      type: tosca.policies.tacker.Scaling
38      targets: [VDU1]
39      properties:
40        increment: 1
41        cooldown: 120
42        min_instances: 1
43        max_instances: 3
44        default_instances: 1
45
46    - vdu_cpu_usage_monitoring_policy:
47      type: tosca.policies.tacker.Alarming
48      triggers:

```

```
49     vdu_hcpu_usage_scaling_out:
50         event_type:
51             type: toska.events.resource.utilization
52             implementation: ceilometer
53         metric: cpu_util
54         condition:
55             threshold: 70
56             constraint: utilization greater_than 70%
57             granularity: 5
58             evaluations: 1
59             aggregation_method: mean
60             resource_type: instance
61             comparison_operator: gt
62         metadata: SG1
63         action: [SP1]
```

## APÊNDICE E – *Request* RSpec para criação de VNF a partir de um modelo do catálogo

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <rspec xmlns="http://www.geni.net/resources/rspec/3"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://www.geni.net/resources/rspec/3
5     http://www.geni.net/resources/rspec/3/ad.xsd
6     http://www.geni.net/resources/rspec/ext/opstate/1
7     http://www.geni.net/resources/rspec/ext/opstate/1/ad.xsd"
8     type="request">
9   <node client_id="orchestrator" exclusive="false"
10       component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
11     <sliver_type name="vm-m1.small">
12       <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+nfv_orchestrator"/>
13     </sliver_type>
14   </node>
15   <nfv client_id="my_vnf" exclusive="false"
16       component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
17     <sliver_type name="vnf">
18       <template name="o2cmf-test-vnf">
19         <parameter name="image" value="vnf_web_server"/>
20         <parameter name="flavor" value="o2cmf.base"/>
21       </template>
22     </sliver_type>
23   </nfv>
24 </rspec>
```





## APÊNDICE F – Request RSpec contendo o espaço inteligente e seus serviços na nuvem

```

1 <?xml version='1.0'?>
2 <rspec xmlns="http://www.geni.net/resources/rspec/3"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://www.geni.net/resources/rspec/3
5     http://www.geni.net/resources/rspec/3/ad.xsd
6     http://www.geni.net/resources/rspec/ext/opstate/1
7     http://www.geni.net/resources/rspec/ext/opstate/1/ad.xsd"
8     type="request">
9   <node client_id="orchestrator" exclusive="false" component_manager_id="urn:publicid:IDN+
10     futebol.inf.ufes.br+authority+cm">
11     <sliver_type name="vm-m1.small">
12       <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+experiment_orchestrator"/>
13     </sliver_type>
14   </node>
15   <node client_id="image-processing" exclusive="false" component_manager_id="urn:publicid:IDN+
16     futebol.inf.ufes.br+authority+cm">
17     <sliver_type name="vnf-m1.small">
18       <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_image_processing"/>
19     </sliver_type>
20   </node>
21   <node client_id="camera-gateway" exclusive="false" component_manager_id="urn:publicid:IDN+
22     futebol.inf.ufes.br+authority+cm">
23     <sliver_type name="vnf-m1.small">
24       <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_camera_gateway"/>
25     </sliver_type>
26   </node>
27   <node client_id="rabbit-mq" exclusive="false" component_manager_id="urn:publicid:IDN+futebol.
28     inf.ufes.br+authority+cm">
29     <sliver_type name="vnf-m1.small">
30       <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_rabbit_mq"/>
31     </sliver_type>
32   </node>
33   <node client_id="robot-controller" exclusive="false" component_manager_id="urn:publicid:IDN+
34     futebol.inf.ufes.br+authority+cm">
35     <sliver_type name="vnf-m1.small">
36       <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_robot_controller"/>
37     </sliver_type>
38   </node>
39   <node client_id="sdn-controller" exclusive="false" component_manager_id="urn:publicid:IDN+
40     futebol.inf.ufes.br+authority+cm">
41     <sliver_type name="vnf-m1.small">
42       <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_sdn_controller"/>
43     </sliver_type>
44   </node>
45   <node client_id="mpeg-server" exclusive="false" component_manager_id="urn:publicid:IDN+futebol.
46     inf.ufes.br+authority+cm">
47     <sliver_type name="vnf-m1.small">
48       <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+is_mpeg_server"/>
49     </sliver_type>
50   </node>
51 </rspec>

```

```
42     </sliver_type>
43 </node>
44 <node client_id="ispace" exclusive="false" component_manager_id="urn:publicid:IDN+futebol.inf.
    ufes.br+authority+cm">
45   <sliver_type name="intelligent_space">
46     <camera fps="15"/>
47     <robot duration="30" laps="3" trajectory="8"/>
48     <wireless_layout x="0" y="0"/>
49   </sliver_type>
50 </node>
51 </rspec>
```

## Anexos



# ANEXO A – Modelo de VNF contendo políticas de *scaling* e monitoramento

Modelo de VNF obtido de ([OPENSTACK, 2018o](#)):

```

1  tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0
2  description: Demo example
3
4  metadata:
5    template_name: sample-tosca-vnfd
6
7  topology_template:
8    node_templates:
9      VDU1:
10       type: tosca.nodes.nfv.VDU.Tacker
11       capabilities:
12         nfv_compute:
13           properties:
14             disk_size: 1 GB
15             mem_size: 512 MB
16             num_cpus: 2
17       properties:
18         image: cirros-0.4.0-x86_64-disk
19         mgmt_driver: noop
20         availability_zone: nova
21         metadata: {metering.server_group: SG1}
22
23      CP1:
24       type: tosca.nodes.nfv.CP.Tacker
25       properties:
26         management: true
27         anti_spoofing_protection: false
28       requirements:
29         - virtualLink:
30           node: VL1
31         - virtualBinding:
32           node: VDU1
33
34      VL1:
35       type: tosca.nodes.nfv.VL
36       properties:
37         network_name: net_mgmt
38         vendor: Tacker
39
40  policies:
41    - SP1:
42       type: tosca.policies.tacker.Scaling
43       targets: [VDU1]
44       properties:
45         increment: 1
46         cooldown: 120
47         min_instances: 1

```

```
48     max_instances: 3
49     default_instances: 1
50
51 - vdu_cpu_usage_monitoring_policy:
52     type: tosca.policies.tacker.Alarming
53     triggers:
54         vdu_hcpu_usage_scaling_out:
55             event_type:
56                 type: tosca.events.resource.utilization
57                 implementation: ceilometer
58             metric: cpu_util
59             condition:
60                 threshold: 80
61                 constraint: utilization greater_than 80%
62                 granularity: 60
63                 evaluations: 1
64                 aggregation_method: mean
65                 resource_type: instance
66                 comparison_operator: gt
67             metadata: SG1
68             action: [SP1]
69
70         vdu_lcpu_usage_scaling_in:
71             event_type:
72                 type: tosca.events.resource.utilization
73                 implementation: ceilometer
74             metric: cpu_util
75             condition:
76                 threshold: 10
77                 constraint: utilization less_than 10%
78                 granularity: 60
79                 evaluations: 1
80                 aggregation_method: mean
81                 resource_type: instance
82                 comparison_operator: lt
83             metadata: SG1
84             action: [SP1]
```

## ANEXO B – Modelo de VNF contendo política de monitoramento

Modelo de VNF obtido de ([OPENSTACK, 2018n](#)):

```

1  tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0
2  description: Demo example
3
4  metadata:
5    template_name: sample-tosca-vnfd
6
7  topology_template:
8    node_templates:
9      VDU1:
10       type: tosca.nodes.nfv.VDU.Tacker
11       capabilities:
12         nfv_compute:
13           properties:
14             disk_size: 1 GB
15             mem_size: 512 MB
16             num_cpus: 2
17       properties:
18         image: cirros-0.4.0-x86_64-disk
19         mgmt_driver: noop
20         availability_zone: nova
21         metadata: {metering.server_group: VDU1}
22
23      CP1:
24       type: tosca.nodes.nfv.CP.Tacker
25       properties:
26         management: true
27         anti_spoofing_protection: false
28       requirements:
29         - virtualLink:
30           node: VL1
31         - virtualBinding:
32           node: VDU1
33
34      VL1:
35       type: tosca.nodes.nfv.VL
36       properties:
37         network_name: net_mgmt
38         vendor: Tacker
39
40  policies:
41    - vdu1_cpu_usage_monitoring_policy:
42      type: tosca.policies.tacker.Alarming
43      triggers:
44        vdu_hcpu_usage_respanning:
45          event_type:
46            type: tosca.events.resource.utilization
47            implementation: ceilometer

```

```
48     metric: cpu_util
49     condition:
50         threshold: 50
51         constraint: utilization greater_than 50%
52         granularity: 300
53         evaluations: 1
54         aggregation_method: mean
55         resource_type: instance
56         comparison_operator: gt
57     metadata: VDU1
58     action: [respawn]
```