Mestrado em Engenharia Informática
Dissertação/Estágio
Relatório Final

Implementação de uma infraestrutura distribuída de virtualização

Pedro Jorge Rosa Magalhães
pjrosa@student.dei.uc.pt

07 de julho de 2016

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Mestrado em Engenharia Informática
Dissertação/Estágio
Relatório Final

Implementação de uma infraestrutura distribuída de virtualização

Pedro Jorge Rosa Magalhães
pjrosa@student.dei.uc.pt

Orientador:
Professor Doutor António Jorge da Costa Granjal

Co-Orientador:
Engenheiro Ricardo José Pessoa Lopes Ruivo

Júri Arguente:
Professor Doutor Marco Paulo Amorim Vieira

Júri Vogal:
Professor Doutor Filipe João Boavida Mendonça Machado de Araújo

07 de julho de 2016

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Agradecimentos

Esta secção é dedicada a todos aqueles que contribuíram de alguma forma para a realização deste estágio.

Primeiramente, agradeço ao orientador Professor Doutor Jorge Granjal e ao co-orientador Engenheiro Ricardo Ruivo, por todo o apoio, confiança, incentivo, orientação, apoio nas decisões e disponibilidade oferecida ao longo deste projeto. O entusiasmo por este projeto permitiu que fosse possível a concretização deste estágio. Quero ainda agradecer ao Professor Doutor Bruno Cabral pela sua ajuda na validação da arquitetura, assim como ao Professor Doutor Tiago Cruz pelo *feedback* dado neste estágio.

Aos meus amigos agradeço o apoio incondicional e a paciência ao longo destes anos. Deixando um agradecimento especial à Rita Ferreira e ao Rodrigo Santos, por me terem acompanhado desde o início da minha vida académica, principalmente, por todo o apoio e paciência dada durante a realização deste estágio.

À minha namorada por todo o seu apoio, dedicação e paciência ao longo deste percurso. Obrigado por sempre acreditares nas minhas capacidades.

Aos meus pais, Jorge Magalhães e Manuela Magalhães, por todo o apoio e incentivo ao longo de todos estes anos, que tornaram possível que este dia fosse concretizado. A toda a minha família agradeço por ter sempre acreditado no meu sucesso.

Resumo

O presente trabalho tem como objetivo planear, implementar e avaliar uma infraestrutura distribuída de virtualização, de forma a responder às limitações do serviço atual de virtualização do Departamento de Engenharia Informática (DEI) da Faculdade de Ciências e Tecnologia da Universidade de Coimbra (FCTUC).

Atualmente, cada vez mais, a virtualização está a tornar-se uma mais valia para responder às necessidades dos utilizadores, dada a crescente necessidade de recursos de computação, armazenamento e comunicação para poderem implementar e validar cenários complexos e reais. Desta forma, o trabalho desenvolvido, no decorrer do estágio, pretendeu evoluir a arquitetura atual do sistema de virtualização do DEI de forma a disponibilizar duas novas funcionalidades chave: integração com *cloud providers* externos e disponibilizar aos utilizadores uma interface que lhes permita gerir todos os seus recursos virtualizados autonomamente.

Palavras-Chave

Cloud-computing, cloud híbrida, virtualização *on-demand*, OpenStack, Integração de sistemas *cloud*

Abstract

The current work aims to plan, implement and evaluate a distributed virtualization infrastructure, so it may respond to the limitations of the Department of Informatics Engineering (DEI-FCTUC) current virtualization service.

More and more, virtualization is becoming a competitive advantage when responding to the needs of users, since there is a significant increase in the use of computation resources, storage and communication in order to implement and validate complex and real scenarios. Therefore, the work developed within the internship is aimed to develop the current architecture of DEI's virtualization system, in order to make available two key functionalities: integration with external *cloud providers* and to make available to users an interface that allows them to manage autonomously their virtualized resources.

Keywords

Cloud-computing, hybrid cloud, virtualization *on-demand*, OpenStack, Cloud integration systems

Índice

Índice de Figuras

Índice de Tabelas

1 Introdução

1.1 Contexto

Um dos serviços mais requisitados no DEI é o sistema de virtualização, sendo usado na maioria dos servidores *core*, tais como o *Domain Name System* (DNS), serviços SMTP, *websites*, entre outros. É igualmente utilizado pelos docentes, investigadores e alunos para a realização dos seus trabalhos com carácter científico e académico. Este sistema é local, visto que toda a infraestrutura está localizada no DEI e, apenas, permite aos utilizadores fazer a gestão das suas máquinas virtuais através de protocolos de terminal remoto como, por exemplo, o *Secure Shell* (SSH) e o *Remote Desktop Protocol* (RDP).

1.2 Motivação

No sistema atual do DEI encontram-se duas grandes limitações: não existe a possibilidade de escalar os recursos para o exterior e os próprios utilizadores não têm qualquer controlo sobre a infraestrutura onde as suas máquinas virtuais se encontram. Como tal, na primeira limitação existe a necessidade de evoluir para um sistema que seja escalável, por forma a aumentar a disponibilidade dos recursos virtuais aos utilizadores do DEI, quando necessário. Relativamente à segunda limitação, é preciso fornecer aos utilizadores uma plataforma que lhes permita controlar os seus recursos virtualizados (servidores e redes).

Este projeto prevê a utilização de recursos localizados num ou mais *providers*, assim como, fornecer uma plataforma uniformizada que permita aos utilizadores gerir os seus recursos (máquinas ou redes virtuais) sempre que necessitarem.

A motivação para a realização deste trabalho foca-se na oferta da possibilidade dos utilizadores terem controlo total da sua infraestrutura. Desta forma, o utilizador consegue gerir os seus recursos sem estar dependente dos administradores de sistemas do DEI. Por exemplo, se o utilizador num fim de semana cometer um erro e fizer *shutdown* à maquina, seria necessário enviar um pedido à equipa de suporte dos Serviços de Informática e Comunicações (SIC) ficando pendente até o inicio da semana. Estas funcionalidades irão libertar os administradores de sistemas de efetuarem tarefas repetitivas, permitindo obter uma maior eficácia nos restantes serviços. Para além disso, é uma tecnologia promissora que ainda se encontra em crescimento, sendo cada vez mais usada.

1.3 Metodologia

Para a realização deste projeto foi necessário escolher o ciclo de vida do desenvolvimento, assim como, uma ferramenta de gestão de projetos, onde seja possível fazer *tracking* das tarefas, utilizando gráficos de *Gantt* e *Program Evaluation and Review Technique* (PERT), permitindo verificar se o projeto estava a decorrer dentro do prazo estimado.

O ciclo de vida considerado no projeto foi o modelo espiral (ver Figura 1.1), que se revelou apropriado, visto que o trabalho foi realizado num sistema complexo e com algum grau de desconhecimento. Durante o decorrer do projeto foram determinados objetivos semanais, que passaram pela identificação e resolução dos problemas. Por fim, efetuou-se o desenvolvimento e os testes planeando a próxima iteração a ser efetuada no projeto.

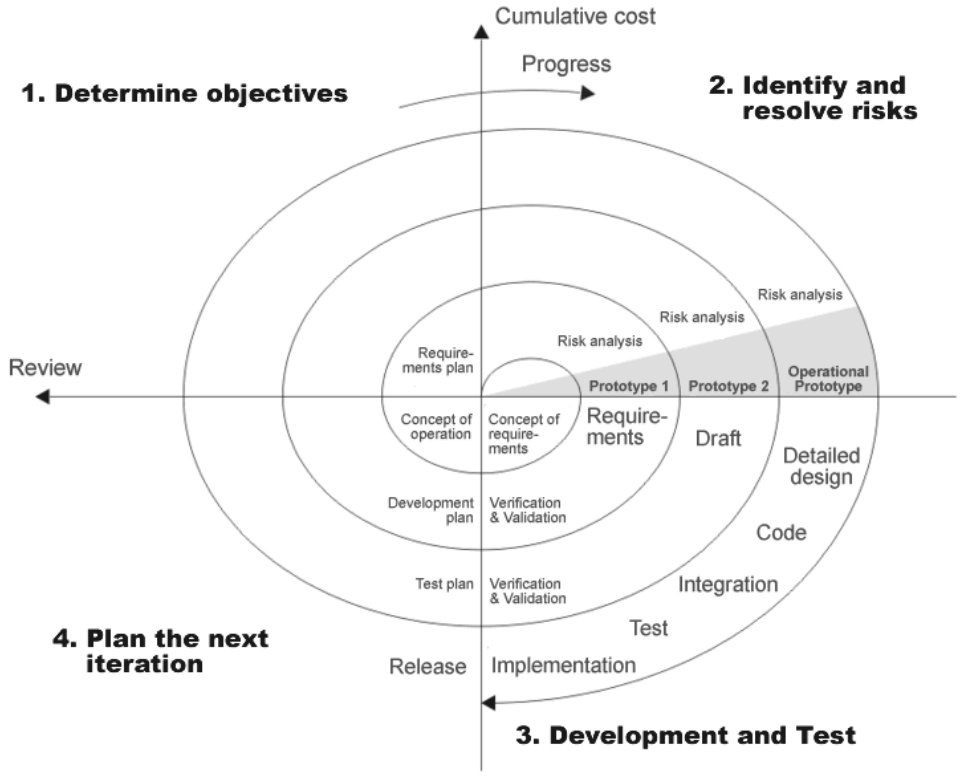


Figura 1.1: Modelo espiral [6]

1.4 Estrutura do Relatório

O presente relatório encontra-se subdividido em cinco Capítulos. O atual Capítulo, 1, apresenta uma breve introdução sobre o tema abordado, os objetivos e a metodologia adotada.

O Capítulo 2 centra-se na apresentação do estado da arte, baseada em pesquisas bibliográficas relacionadas com o conteúdo do presente trabalho - virtualização de sistemas, virtualização de redes e de *frameworks* de virtualização. Por fim, é ainda apresentado

um estudo comparativo das soluções *cloud* existentes de forma a escolher a solução mais adequada às necessidades do projeto.

No Capítulo 3, intitulado como arquitetura de virtualização, é apresentada uma breve introdução da arquitetura atual existente no DEI. Neste Capítulo são definidos os requisitos funcionais e não funcionais que, consequentemente, originaram a proposta da arquitetura. Detalhadamente, é ainda apresentada a tecnologia *cloud* selecionada desdobrada em dois pontos de vista de integração: com vários *datacenters* e com a *cloud* pública. Por fim, são ainda identificados os componentes a desenvolver, assim como as estratégias de avaliação e de validação da arquitetura.

No Capítulo 4 apresenta-se a avaliação preliminar que permite verificar a viabilidade das opções concretizadas na proposta da arquitetura.

O Capítulo 5, denominado por "Desenvolvimento e integração", apresenta todo o trabalho efetuado no decorrer deste estágio, especificando como é que foi feito o *deployment*, assim como o desenvolvimento efetuado neste projeto.

O Capítulo 6 aborda as estratégias adotadas para efetuar a avaliação e validação do sistema, assim como os respetivos resultados.

O capitulo 7 centra-se sobre as conclusões e trabalhos futuros do projeto.

1.5 Planeamento

Nesta secção são apresentados os diagramas de *Gantt* relativamente ao trabalho desenvolvido durante o estágio. Na Figura 1.2 apresenta o diagrama correspondente ao trabalho realizado no primeiro semestre. Inicialmente, focou-se no Estado da Arte, baseado em pesquisas bibliográficas relacionadas com o conteúdo do presente trabalho. Seguidamente, passou-se à conceção da arquitetura de virtualização e, por fim efetuou-se a avaliação preliminar.

ID	Task Name	Duration
1	**Estado da Arte**	**9 wks**
2	Tecnologias de virtualização de sistemas	3 wks
3	Tecnologias de virtualização de redes	3 wks
4	Frameworks de virtualização	3 wks
5	**Arquitetura de virtualização**	**4 wks**
6	Definição de requisitos	2 wks
7	Proposta da arquitetura	4 wks
8	Estudo da solução seleccionada	4 wks
9	Estratégia de avaliação e validação	2 wks
10	**Avaliação Preleminar**	**2 wks**
11	Deployment e testes iniciais	2 wks
12	Integração Xen Server	2 wks
13	Teste de multiplas regiões	2 wks
14	Escrita do relatório	4 wks

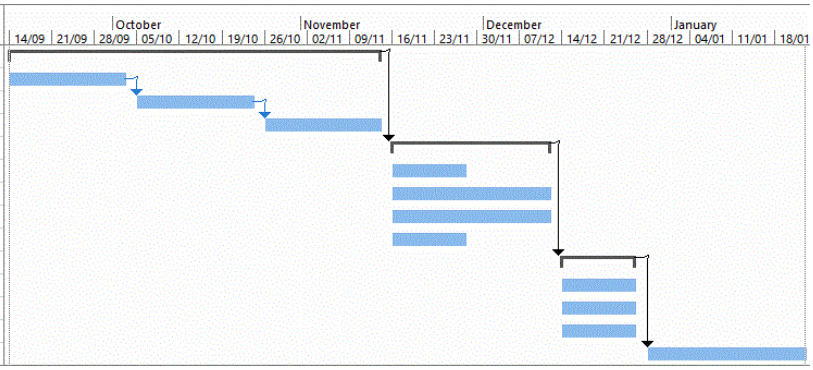


Figura 1.2: Diagrama de *Gantt* do primeiro semestre

Na Figura 1.3 é apresentado o diagrama relativamente ao trabalho concretizado do segundo semestre. Após finalizado a arquitetura do sistema, procedeu-se ao estudo do aprovisionamento e *deployment* automático da infraestrutura *OpenStack*, prosseguindo pelo *deployment*. De seguida, efetuou-se o desenvolvimento dos componentes, e por fim, realizaram-se testes e a escrita do relatório.

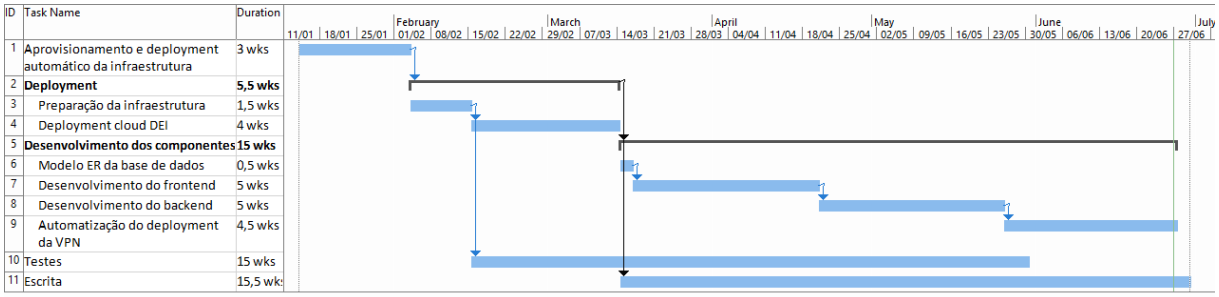


Figura 1.3: Diagrama de *Gantt* do segundo semestre

2 Estado da Arte

2.1 Introdução

Neste capítulo é apresentado um alargado número de trabalhos realizados ao longo dos anos nas diferentes áreas: virtualização de sistemas, virtualização de redes e *frameworks* de virtualização. Relativamente à área de virtualização de sistemas foram identificados os diversos tipos e técnicas de virtualização, *hypervisor* e *container system*, utilizadas nos sistemas atuais. Na área de virtualização de redes são explicados os conceitos de *Network Functions Virtualizations* (NFV) e de *Software Defined Network* (SDN), identificando as suas vantagens. São analisadas diferentes *frameworks* de virtualização, concluindo com uma breve análise comparativa das mesmas.

2.2 Virtualização de sistemas

Atualmente, é possível repartir uma máquina física em várias máquinas virtuais, existindo duas formas diferentes de proceder: *hypervisor* e *container systems*. O *hypervisor* segmenta o *hardware* permitindo correr vários sistemas operativos sobre ele. Enquanto que através de um *container system* (ou virtualização a nível do sistema operativo), o sistema operativo é dividido em múltiplos *containers* partilhando o mesmo *kernel* que o *host* físico.

2.2.1 *Hypervisor*

Os conceitos de máquina virtual (VM, *Virtual Machine*) e monitor de máquina virtual (VMM, *Virtual Machine Monitor*) foram criados pela International Business Machines (IBM) para fornecer *timesharing* de um computador *mainframe*. A IBM define VMM como uma camada de abstração que permite ter uma ou mais máquinas virtuais a correr no mesmo servidor, repartindo dinamicamente e partilhando os recursos de *hardware*, tais como, o CPU (*Central Processing Unit*), a memória RAM (*Random Access Memory*) e os dispositivos de entrada e de saída.[41]

O computador/servidor que contém o *hypervisor*, com uma ou mais máquinas virtuais a correr, denomina-se de máquina *host*, enquanto que as máquinas virtuais são as *guest*. Existem dois tipos de *hypervisors*:

- **Tipo 1 - *Hypervisor* nativo/*bare-metal*:** É um sistema operativo com características e funções de tempo real, que corre diretamente no topo do *hardware*. Deste

modo, torna-se muito mais eficiente a gestão dos recursos do *host*.

- **Tipo 2 - *Hosted hypervisors* (*Hypervisor* hospedado):** Corre como uma aplicação no sistema operativo já existente no computador ou servidor. Isto implica ter uma camada adicional entre as máquinas virtuais e o *hardware*. Este tipo de *hypervisor* é usado quando não é dada demasiada relevância à performance.

O sistema operativo utiliza o modelo *protection ring*, como descrito na Figura 2.1, onde existem várias camadas de segurança hierárquicas desde o mais privilegiado (camada 0) para o menos privilegiado (camada mais alta do *ring*). Este modelo serve para contextualizar os três tipos de virtualização existentes (descritos mais a frente nesta secção) que necessitam destas camadas.

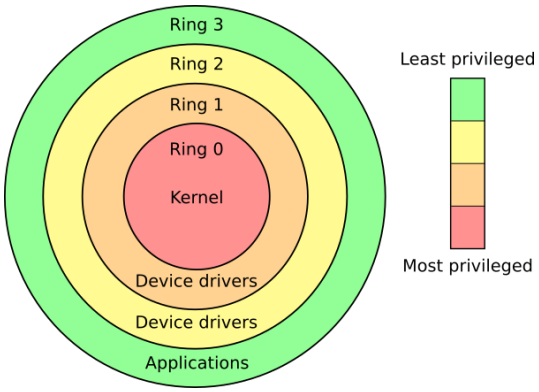


Figura 2.1: Proteção de 4 camadas da arquitetura x86 [48]

Os sistemas operativos por norma têm 4 camadas [14]:

- **Camada 0** - Camada mais confiável, onde o *kernel* do sistema operativo reside. Qualquer processo que esteja a correr neste nível corre em modo privilegiado.
- **Camada 1** - Contém porções não privilegiadas do sistema operativo.
- **Camada 2** - Onde residem as drivers I/O (*Inputs/Outputs*), operações de baixo nível e utilitários.
- **Camada 3** - Camada em que as aplicações e os processos operam, estando a correr em modo de utilizador.

As técnicas de virtualização utilizadas pelos *hypervisors* existentes (como por exemplo, *VMware ESXi*, *Virtualbox* e *Xen*) são as seguintes:

- **Virtualização completa -** Utiliza duas técnicas para realizar a virtualização: a técnica de tradução binária (*bynary translation*) e a técnica de execução direta (*direct execution*), tendo sido implementadas primeiramente pela *VMware* [41]. Como é possível ver na Figura 2.2, a tradução binária é responsável por traduzir o código do *kernel* de modo a substituir as instruções não virtualizáveis[1] em novas sequências de

[1]Instruções que precisam de mais privilégios do que o VMM

instruções que possam ser executadas nas máquinas virtuais. No entanto, a técnica de execução direta faz com que as sequências de instruções sejam executadas com permissões do utilizador diretamente no CPU do *host*. Desta forma, não existe um reconhecimento da máquina virtual que está a ser virtualizada, e não requer qualquer tipo de modificação no sistema operativo, nem necessita de assistência por *hardware* oferecendo um melhor isolamento e segurança.

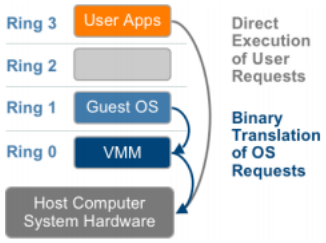


Figura 2.2: Funcionamento da virtualização na arquitetura x86. [43]

- **Virtualização assistida do sistema operativo ou paravirtualização -** Por forma a ser possível a máquina virtual correr utilizando esta técnica é necessário alterar o *kernel* do sistema operativo, substituindo as instruções que não são virtualizáveis por *hypercalls*. Ao ser alterado o *kernel* é possível comunicar diretamente com a camada de virtualização do *hypervisor*, como se mostra na Figura 2.3. Apesar desta técnica oferecer um *overhead* muito baixo face à virtualização completa, tem como desvantagem requerer grandes modificações no *kernel*, estando dependente dos fornecedores dos sistemas operativos.[41]

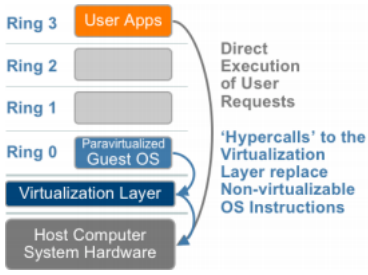


Figura 2.3: Funcionamento da para-virtualização na arquitetura x86. [43]

- **Virtualização assistida por *hardware* -** Devido à popularidade crescente que os sistemas de virtualização têm vindo a conquistar ao longo dos últimos anos, os fabricantes de processadores desenvolveram novas formas de os melhorar. Os CPUs da Intel contêm o *Virtualization Techonology (VT-x)*, e os da AMD, o AMD-V. No VT-x, o estado da máquina virtual é guardado no *virtual machine control structures*, enquanto que no AMD-V o estado da máquina virtual é guardado no *virtual machine control blocks*. A Figura 2.4 mostra esquematicamente como é possível correr o VMM com privilégios, introduzindo uma nova camada de segurança, chamada de Camada -1. Assim, as chamadas ao sistema operativo são intercetadas pelo VMM não sendo necessário recorrer à técnica *binary translation* ou de paravirtualização.

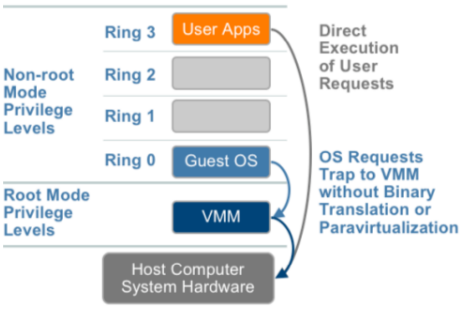


Figura 2.4: Funcionamento da virtualização por *hardware* assistido na arquitetura x86. [43]

2.2.2 Virtualização a nível do sistema operativo (*Container System*)

A virtualização ao nível do sistema operativo, também chamado de *shared kernel* ou *Linux Container*, tem como objetivo a alocação, particionamento e atribuição dos recursos do *host*, tais como CPU, *network I/O*, *bandwidth*, *block I/O* e memória RAM. Desta forma, o *kernel* permite isolar os recursos, protegendo-os de forma a que os processos que se encontram na VM não interfiram com o sistema operativo *host* ou restantes VMs.

Este tipo de virtualização é conseguida através de técnicas que garantem o isolamento de recursos do sistema operativo como, por exemplo, o *chroot*. O *chroot* permite alterar o sistema de ficheiros *root* de um processo, confinando-o do restante sistema, uma vez que só tem acesso ao seu próprio *user-space*.[20] A grande vantagem da virtualização a nível do sistema operativo prende-se ao facto de ser possível a obtenção de uma performance praticamente nativa para cada máquina virtual. Devido à obtenção de uma performance quase nativa, o sistema corre com velocidades nativas (analogamente a uma máquina física) e o número de máquinas virtuais a correr no *host* é superior, comparativamente aos *hypervisors*. No entanto, existem algumas desvantagens na utilização deste tipo de virtualização, uma vez que todas as VMs devem ser compatíveis com o *kernel*. Desta forma, apenas é exequível para sistemas operativos baseados em Unix e/ou Linux. No caso em que ocorra alguma falha no *kernel* (por exemplo, bloquear) todas as instâncias de VMs são afetadas da mesma forma, devido a partilharem o mesmo *kernel*.

2.3 Virtualização de redes

Atualmente, existe uma elevada necessidade de melhorar e incorporar novos serviços para uma infraestrutura de rede, porém, torna-se muito complicado responder a estas necessidades devido ao elevado custo dos equipamentos, espaço, eletricidade e à escassez de profissionais com conhecimentos técnicos dos diversos equipamentos. Além disto, o equipamento físico de rede facilmente é ultrapassado, tornando-se obsoleto, obrigando as empresas a investir em novo *hardware*. Desta forma, as empresas necessitam despender tempo e recursos até o equipamento entrar em produção.

Por forma a combater estes problemas, foi proposta a virtualização destes serviços de

rede através do *Network Function Virtualization* (NFV). A utilização do NFV permite de forma simplificada criar e gerir eficientemente os recursos da rede, tanto os virtualizados como os físicos, através do *Software Defined Network* (SDN), fornecendo uma interface genérica. Assim, os gestores de redes apenas necessitam de saber como são executadas as funcionalidades de rede, aplicando-as em produção de forma uniforme evitando a dependência de fabricantes.

2.3.1 *Network Function Virtualization*

O *Network Function Virtualization* (NFV) muda totalmente o paradigma de como os operadores de rede arquitetam a sua infraestrutura [4], existindo a possibilidade de implementarem funções de rede através das técnicas de virtualização e corre-las num *hardware* comum (por exemplo, um servidor). *Appliances* virtuais são máquinas virtuais pré-configuradas por forma a serem utilizadas num sistema de virtualização. Estas podem ser imediamente instanciadas sem ser necessário recorrer à instalação de novos equipamentos, permitindo mudar as infraestruturas rapidamente. Esta arquitetura permite obter os seguintes benefícios:

- Custo reduzido em equipamentos e eletricidade através da consolidação de vários equipamentos no mesmo *hardware*;
- Diminuição do *time to market* devido à rápida possibilidade de instanciar serviços sem necessidade de comprar *hardware* adicional;
- Os serviços podem ser escalados conforme as necessidades;
- Facilidade do *deployment* remoto;
- Os prazos de entrega do equipamento ficam reduzidos ao tempo de *download*/instalação dos serviços desejados;
- Possibilidade de efetuar testes em arquiteturas de sistemas complexos na mesma infraestrutura;
- Redundância dos serviços, em caso de avaria facilmente se substitui restaurando o último *backup* da *appliance* utilizada em produção.

Desta forma, pretende-se consolidar os equipamentos de redes em apenas um conjunto de servidores, *switches* e *storage*, como é ilustrado na Figura 2.5.

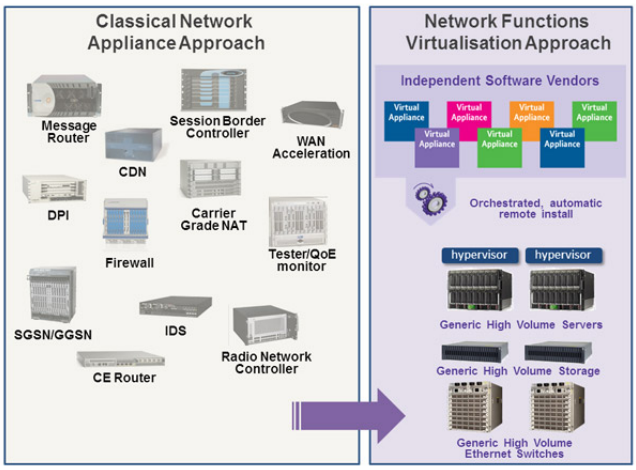


Figura 2.5: Método de rede clássico VS NFV. [8]

2.3.2 *Software Defined Network*

O *Software Defined Network* (SDN) é a separação física entre o plano de controlo de rede (responsável por determinar a forma como os pacotes são encaminhados) e o plano de encaminhamento (responsável por mover os pacotes de entrada para a respetiva saída). O plano de controlo de rede passa a controlar os diversos dispositivos de rede [13], tanto virtuais como físicos, que o suportam, definindo o plano de encaminhamento.

Na Figura 2.6 é possível constatar que a arquitetura do SDN é subdividida em três camadas:

- **Camada de aplicação:** Uma ou mais aplicações encontram-se nesta camada, onde cada uma tem controlo exclusivo de um conjunto de recursos expostos por um ou mais controladores SDN.
- **Camada de controlo:** É responsável por determinar a forma como o tráfego flui, baseado no estado da camada de infraestrutura e pelos requisitos especificados pela camada de aplicação. O modelo, nesta camada, é centralizado, ao invés de ser local ao próprio dispositivo de rede, desta forma a rede mostra-se como um único *switch* lógico para a camada de aplicação.
- **Camada de infraestrutura:** Contém um ou mais equipamentos de rede, onde cada um contém um conjunto de tráfego de encaminhamento ou recursos de processamento de tráfego.

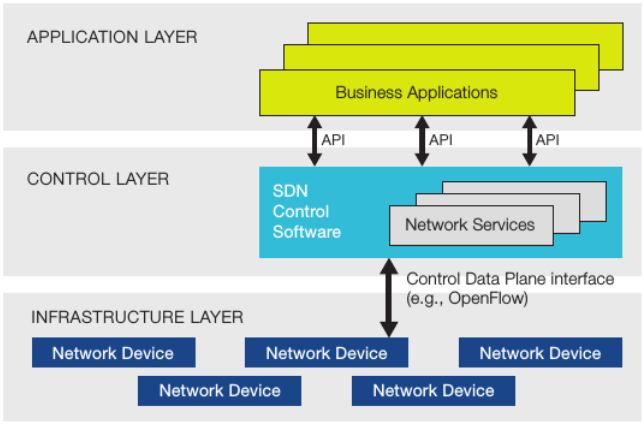
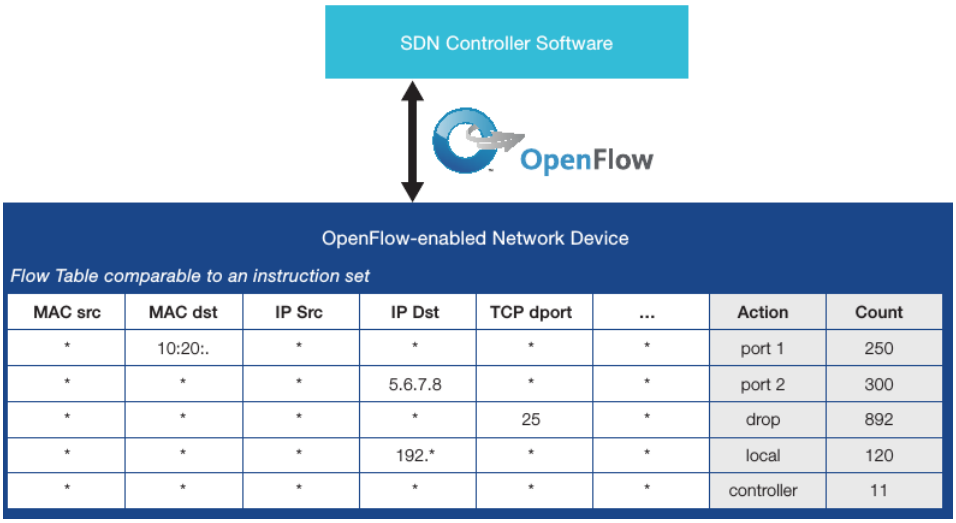


Figura 2.6: Arquitetura SDN. [13]

A partir da arquitetura apresentada, é permitido aos gestores de rede programar a configuração da rede de forma uniforme e abstrata, não sendo necessário configurar individualmente os equipamentos com o *Command-Line Interface* (CLI). É ainda permitido aos gestores gerir e alterar os comportamentos de rede em tempo real com a possibilidade de fazer *deploy* de novas aplicações e serviços nos equipamentos, uma vez que é possível ter controlo global de todos os equipamentos de rede. Esta arquitetura permite utilizar ou implementar *Application Program Interface* (APIs) que tornam possível a utilização de serviços de rede comuns, tais como *routing*, segurança, controlo de acessos, largura de banda, qualidade de serviço, processador e *storage*.

O único *standard* das interfaces de comunicações entre a camada de controlo e a camada de encaminhamento (por exemplo, através de um *router*) do SDN é o *OpenFlow*. O *OpenFlow* permite ter acesso direto à manipulação da camada de encaminhamento dos equipamentos de rede (*switches* e *routers*), tanto físicos como virtuais. A Figura 2.7 apresenta o *OpenFlow* semelhante a um conjunto de instruções do **CPU** [13], onde o protocolo especifica primitivas básicas que podem ser usadas, por uma identidade que reside na camada de aplicação do SDN, de forma a programar o encaminhamento de dados dos equipamentos de rede.



MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Figura 2.7: Exemplo de uma tabela de fluxos comparativamente a um conjunto de instruções. [13]

Na Figura 2.7 são apresentados vários *flows* que representam instruções de controlo. O conceito de *flow* é o tráfego de rede baseado em regras pré-definidas, podendo ser estaticamente ou dinamicamente programáveis através do controlo do *software* SDN. O protocolo do SDN é implementado em ambos os lados das interfaces, ou seja, tanto na camada de infraestrutura como na camada de controlo.
Em resumo, esta abordagem permite ter os seguintes benefícios[13]:

- **Controlo centralizado de vários equipamentos de várias marcas:** O *software* de controlo SDN permite controlar qualquer equipamento de rede que tenha suporte ao protocolo de comunicação *OpenFlow*.
- **Reduzir complexidade através da automação:** Flexibilidade na gestão e automação da rede, com possibilidade de orquestração e provisionamento inteligente.
- **Maior taxa de inovação:** Através da programação/re-programação da rede em tempo real de modo a introduzir novos serviços em horas em vez de semanas/meses, baixando o *time to market*.
- **Aumento da segurança e confiabilidade:** Com a visão global de todos os equipamentos de redes é possível configurá-los, tornando-os consistentes, diminuindo assim possíveis erros de configuração de algum equipamento de rede.
- **Maior controlo fino da rede:** É possível ir ao detalhe das políticas aplicadas na rede à sessão, utilizador, equipamento e ao nível da aplicação.
- **Melhor experiência de utilização:** Através do *quality of service* dinâmico é possível definir a largura de banda automaticamente.

2.4 *Frameworks* de virtualização

2.4.1 *Cloud computing*

Cloud Computing é um modelo que permite o acesso ubíquo e conveniente, através de uma rede, a um conjunto de recursos computacionais partilhados e configuráveis. A partir deste modelo é possível ao utilizador provisionar rapidamente e facilmente recursos virtuais como, por exemplo, redes, servidores, armazenamento, aplicações e serviços sem interação com o *provider* do serviço. O modelo *cloud* é composto por três modelos de serviços e quatro modelos de *deployment*.

Na Figura 2.8 são apresentados os três modelos de serviços, sendo ainda possível verificar que quanto maior é o nível de abstração menor é o nível de controlo que se possui sobre a *cloud*. Os modelos de serviço são:

- ***Software as a Service* (SaaS):** É a capacidade de fornecer ao utilizador aplicações que estejam a correr numa infraestrutura *cloud*. O utilizador não gere nem controla a infraestrutura *cloud* (rede, servidores e sistemas operativos). Um exemplo deste modelo é o Microsoft Office 365 [22].
- ***Platform as a Service* (PaaS):** Neste modelo é dada a possibilidade ao utilizador de fazer *deploy* das suas aplicações criadas/adquiridas na infraestrutura *cloud*, utilizando linguagens de programação, bibliotecas, serviços e ferramentas que sejam suportadas pelo *provider*. O utilizador não gere nem tem qualquer controlo sobre a infraestrutura *cloud* (rede, servidores e sistemas operativos), mas tem o controlo sobre as aplicações que faz *deploy* e pode alterar as configurações das mesmas. Um exemplo deste modelo é o *Heroku* [16].
- ***Infraestructure as a Service* (IaaS):** O utilizador pode fazer aprovisionamento de processamento, armazenamento, redes e outros recursos computacionais fundamentais virtualizados onde o utilizador pode fazer *deploy* e correr *software* arbitrário, tais como, sistemas operativos e aplicações. O utilizador não gere nem controla a infraestrutura *cloud*, reduzindo os custos da manutenção do *datacenter* físico. Um exemplo que suporta este serviço é o *OpenStack* [31].

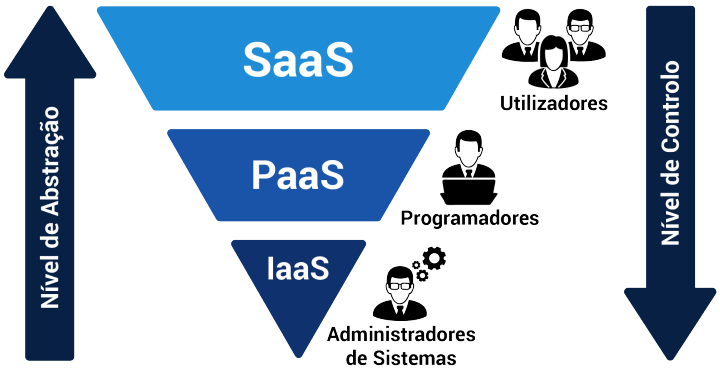


Figura 2.8: Modelos de serviços *cloud*

Na Figura 2.9 são apresentados os modelos de *deployment* que representam as categorias do ambiente *cloud*, permitindo identificar qual é o seu âmbito.

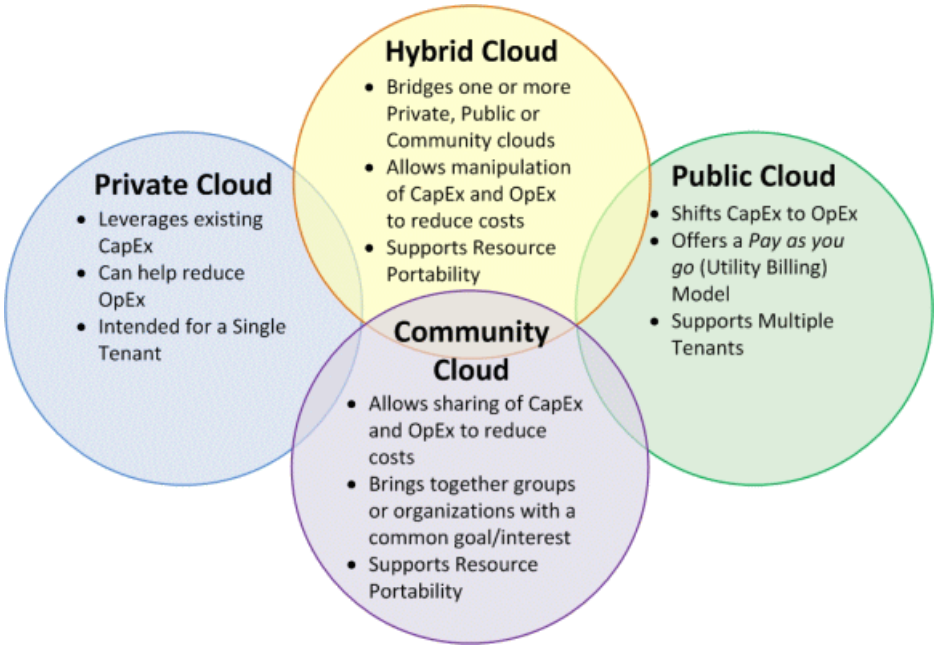


Figura 2.9: Modelos de *deployment cloud* [19]

Como apresenta a Figura 2.9, atualmente, existem quatro modelos de *deployment cloud*:

- ***Cloud* privada:** No modelo de *cloud* privada, a infraestrutura é fornecida para uso exclusivo de uma única organização. Esta organização deverá ser composta por múltiplos tipos de utilizadores (ou consumidores), como por exemplo várias unidades de negócio. A *cloud* pode ser propriedade, gerida ou operada, por esta organização, por terceiros ou pela combinação de ambos, podendo existir dentro e fora do local da organização.

- ***Cloud* comunitária:** Este modelo é fornecido para uso exclusivo de uma comunidade de consumidores de uma organização que partilham interesses comuns, como por exemplo a mesma missão. Pode ser propriedade, gerida e operada, por uma ou mais organizações dessa comunidade, por terceiros, ou pela combinação de ambos, podendo existir dentro ou fora do local da comunidade.

- ***Cloud* pública:** Neste modelo a infraestrutura é aberta ao público. Pode ser propriedade, gerida e operada, por: uma organização comercial; a nível académico; uma organização governamental; ou uma combinação destas. Esta existe nas instalações do fornecedor de *clouds*.

- ***Cloud* híbrida:** É composta por duas ou mais infraestruturas *cloud* distintas (privada, comunitária ou pública) que permanecem como entidades únicas. No entanto, estão interligadas por uma tecnologia padrão ou proprietária que permite a portabilidade de dados e aplicações como, por exemplo, a partilha de equilíbrio de cargas entre *clouds*.

2.4.2 Soluções Comerciais

Nesta secção, a escolha das soluções de virtualização apresentadas para a análise foram baseadas num estudo realizado pelo grupo de investigação Info-Tech [15]. Optou-se por selecionar as quatro soluções mais relevantes: *VMware vCloud Suite*, *Abiquo True Hybrid Cloud*, *Flexiant Cloud Orchestrator* e o *xStream Cloud Management*.

2.4.2.1 *VMware vCloud Suite*

O *VMware vCloud Suite*[45] é uma ferramenta integrada que incorpora três plataformas de gestão: *VMware vSphere*, *VMware Site Recovery* e *VMware vRealize Cloud*, fornecendo uma *cloud* privada baseada no *vSphere*. *O VMware vCloud Suite* pode ainda ser estendido para uma *cloud* híbrida através da compra dos produtos *vRealize Suite*[47] e *vCloud Air*[44]. O *vCloud Air* não contém NFV, sendo necessário adquirir o add-on *VMware NSX*[46]. No entanto, o *VMware Cloud Suite* integra, de forma gratuita, a API do *OpenStack* que permite aos utilizadores ligar os componentes do *VMware* (*vSphere*, NSX) por forma a fazer *deploy*, gerir e correr a infraestrutura *OpenStack*. Esta solução tem ainda o suporte para estender os serviços *cloud* para o *vCloud Air*, *Amazon Web Services* e *Microsoft Azure*.

2.4.2.2 *Abiquo True Hybrid Cloud*

A solução de virtualização *Abiquo True Hybrid Cloud*[1] permite integrar e gerir *clouds* públicas e privadas em ambientes homogéneos, tendo o suporte de vários *hypervisors*, tais como o *VMware ESX*, *Hyper-V*, *KVM* (*Kernel-based Virtual Machine*), *Oracle VM* e *Xen*. Esta solução é compatível com os seguintes fornecedores de *clouds* públicas: *Amazon AWS*, *Rackspace*, *Google Compute Engine*, *HP Cloud*, *ElasticHosts* e *DigitalOcean*.

2.4.2.3 *Flexiant Cloud Orchestrator*

O *Flexiant Cloud Orchestrator*[12] tem como suporte vários *hypervisors*: *KVM*, *Xen*, *PCS*[2], *VMware* e *Hyper-V*. No entanto, deve ter-se em conta a edição selecionada para os *hypervisors VMware* e *Hyper-V*, uma vez que dependem da mesma. Com a utilização desta solução é possível ter como principais funcionalidades: *live migration*, *live recovery*, *firewall* integrada como serviço, *load balancing* como serviço e alta disponibilidade.

2.4.2.4 *xStream Cloud Management*

O *xStream Cloud Management*[42] possibilita a implementação tanto num ambiente empresarial como num *service provider* por forma a criar uma *cloud* privada ou uma *cloud* pública, respetivamente. O *KVM*, *Xen* e *VMware ESXi* são *hypervisors* suportados pela *xStream Cloud Managament*. Além disto, possibilita confinar o acesso a determinadas funções, gestão de quotas de recursos, suporte à API da *Amazon EC2* entre outros.

[2] *Parallels Cloud Server* que se chama agora *Virtuozzo*

2.4.3 Soluções *Open-Source*

Nesta secção apresenta-se uma breve análise das quatro plataformas *cloud open-source* mais relevantes: *OpenStack*, *HP Helion Eucalyptus*, *CloudStack* e *OpenNebula*.

2.4.3.1 *OpenStack*

O *OpenStack*[31] é uma plataforma de *cloud computing* que controla um conjunto de recursos computacionais, *storage* e de rede num *datacenter*, tudo gerido através de uma *dashboard* que permite aos administradores terem controlo da infraestrutura e aos utilizadores a possibilidade de fazer gestão da sua própria infraestrutura. Algumas vantagens são:

- Endereços IP Flutuantes;
- Grupos de Segurança;
- Controlo de acesso baseado em privilégios;
- Quotas de utilizadores e projetos;
- Suporte a vários modos de rede;
- *Dashboard* com suporte total para aprovisionamento automático;
- Capacidades de *Object Storage*[3] e *Block Storage*[4].

2.4.3.2 *HP Helion Eucalyptus*

O *HP Helion Eucalyptus*[10] é uma plataforma *open-source* que permite construir *clouds* privadas e híbridas com o principal objetivo de disponibilizar uma interface compatível com a API dos diversos serviços disponíveis pela *Amazon Web Services (AWS)*. O sistema é desenvolvido utilizando *Java* e *C*. Contém algumas características[11], tais como:

- Compatibilidade com as seguintes APIs da *Amazon Web Services*: EC2, EBS, S3, IAM, *Auto Scaling*, ELB e *CloudWatch*;
- Acesso baseado em funções, gestão de quotas e contabilidade de recursos;
- Suporte ao *hypervisor KVM*;
- *Microsoft Windows* e *Linux* como sistema operativo *guest*.

[3]Não é necessário estar agregado a um sistema operativo, consiste em dados do objeto e *metadata* que pode ser acessível diretamente através de APIs ou pelo protocolo HTTP/HTTPS.

[4]Fornece um tamanho fixo *raw* de capacidade de armazenamento. Pode ser agregado a uma instância como se fosse um disco físico. Exemplos de *block storage*: SAN, iSCSI e discos locais.

2.4.3.3 *CloudStack*

O *CloudStack*[9] é um projeto de nível superior da *Apache Software Foundation* (ASF). Este projeto teve como objetivo desenvolver um *software open-source* por forma a fazer *deployment* de uma *cloud* pública e privada utilizando o modelo de serviço IaaS. É um projeto desenvolvido em *Java* que contém as seguintes funcionalidades:

- Suporta os seguintes *hypervisors*: *Xen*, *LXC*, *KVM*, *Hyper-V* e/ou *VMware ESXi* com *vSphere*;
- Fornece uma API nativa;
- Tem interface web para gestão da *cloud*;
- Tem uma API compatível com a *Amazon* S3/EC2;
- Gere as instâncias de armazenamento que estão a correr nos *hypervisors* (armazenamento primário), assim como *templates*, *snapshots* e imagens ISO (armazenamento secundário);
- Orquestração de serviços de rede desde a camada de ligação (L2) até aos serviços da *layer* aplicacional (L7), tais como DHCP (*Dynamic Host Configuration Protocol*), *firewall*, VPN, entre outros;
- Contabilização dos recursos de rede, computação e de armazenamento;
- Separação de contas de utilizador através de *Multi-tenancy* (instâncias independentes que residem num ambiente partilhado encontrando-se separados logicamente).

2.4.3.4 *OpenNebula*

O *OpenNebula*[26] foi lançado em março de 2008 estando licenciado de acordo com a *Apache License* (versão 2). Esta plataforma permite gerir uma *cloud* utilizando um *deployment* privado, público ou híbrido, usando o modelo *IaaS*. Este projeto encontra-se desenvolvido em várias linguagens de programação: C++, C, *Ruby*, *Java*, *Shell Script*, *Lex* e *Yacc*. Seguidamente destacam-se algumas funcionalidades desta plataforma:

- Compatibilidade com várias APIs: *Amazon Elastic Compute Cloud 2* (AWS EC2), *Amazon Elastic Block Store* (AWS EBS), *Open Cloud Computing Interface* (OGF OCCI);
- Interface *web* distinta para utilizadores e administradores;
- *OpenNebula market place* - contém um catálogo com várias *virtual appliances* prontas a correr no ambiente *OpenNebula*;
- ACLs (*Access Control Lists*) para a alocação de recursos;
- Gestão das quotas dos utilizadores para limitar os recursos computacionais, tanto de *storage* como de rede;
- Arquitetura de alta disponibilidade;

- Suporte nativo a *clouds* híbridas com conectores para o AWS, *SoftLayer* e *Azure*;
- Suporte aos seguintes *hypervisors*: *Xen*, KVM, *VMware ESXi*, *VMware vCenter*.

2.4.4 Avaliação de soluções

De modo a selecionar a plataforma *cloud* que mais se adequa à realidade vivida no DEI analisaram-se as funcionalidades com maior relevância nas soluções referidas nas secções 2.4.2 e 2.4.3.

As características essenciais que foram identificadas são o tipo de licenciamento, *hypervisors* suportados, *cloud providers* suportados, funcionalidades de redes, mecanismos de gestão de utilizadores, robustez e por fim os aspetos de segurança.

2.4.4.1 Tipo de licenciamento

Existem dois tipos de licenciamento: gratuitos (*open-source*) e pagos. No licenciamento gratuito, as soluções tendem a ter uma maior comunidade, uma vez que todos os utilizadores podem contribuir com alterações ao código-fonte, como, por exemplo, a correção de erros de forma mais eficiente e a adição de novas funcionalidades. Permite ao utilizador fazer *upgrades*, assim como, alterar o *software* conforme as suas necessidades sem qualquer custo adicional.

No licenciamento pago não é permitido ao utilizador alterar, com facilidade, o código-fonte do software. No entanto, tem a grande vantagem de oferecer um suporte de 24/7, onde um ou mais especialistas técnicos estão disponíveis para resolver os problemas que possam ocorrer na plataforma *cloud* em produção. A Tabela 2.1 apresenta o tipo de licenciamento para as diversas plataforma *cloud*.

Tipo de licenciamento	
	Gratuito
CloudStack	**Sim**
HP Helion Eucalyptus	**Sim**
OpenNebula	**Sim**
OpenStack	**Sim**
Abiquo True Hybrid Cloud	Não
Flexiant Cloud Orchestrator	Não
VMware vCloud Suite	Não
xStream Cloud Management	Não

Tabela 2.1: Tipo de licenciamento das diversas plataformas *cloud*

2.4.4.2 *Hypervisors* suportados

Atualmente, devido à rápida evolução tecnológica, pode existir a necessidade de alterar a arquitetura de uma infraestrutura para responder às necessidades atuais. Como tal, é necessário utilizar uma plataforma *cloud* que suporte o máximo de *hypervisors*, por forma a permitir uma migração com maior facilidade. Na Tabela 2.2 é apresentada a lista de *hypervisors* suportados pelas plataformas *cloud*.

Hypervisors suportados							
Soluções Open Source	**KVM**	**LXC**	**Xen**	**VMware ESXi**	**Hyper-V**	**Docker**	**Virtuozzo**
CloudStack	**Sim**	**Sim**	**Sim**	Parcial	**Sim**	Não	Não
HP Helion Eucalyptus	**Sim**	Não	Não	Não	Não	Não	Não
OpenNebula	**Sim**	Não	**Sim**	**Sim**	Não	Não	**Sim**
OpenStack	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**
Soluções comerciais							
Abiquo True Hybrid Cloud	**Sim**	Não	**Sim**	**Sim**	**Sim**	Não	Não
Flexiant Cloud Orchestrator	**Sim**	Não	**Sim**	Parcial	**Sim**	Não	Não
VMware vCloud Suite	Não	Não	**Sim**	**Sim**	**Sim**	Não	Não
xStream Cloud Management	**Sim**	Não	**Sim**	**Sim**	Não	Não	Não

Tabela 2.2: Comparativo dos *hypervisors* suportados pelas plataformas *cloud*

O *CloudStack* suporta parcialmente o *VMware ESXi*, isto deve-se ao facto de ser necessário ter o *VMware vCenter* e/ou o *VMware vSphere*. Por sua vez, o *Flexiant Cloud Orchestrator* apenas necessita de ter o *VMware vCenter*.

2.4.4.3 Suporte API dos *cloud providers*

Um dos grandes desafios da *cloud* é a interoperabilidade entre as diversas plataformas *cloud* existentes. Nesta secção são apresentados os serviços dos três maiores *cloud providers*[2]: *Amazon Web Services* (AWS), *Rackspace cloud* e *Google Compute Engine* (GCE).

O *Amazon EC2* é um serviço *web* que fornece uma capacidade elástica de computação na *cloud*, o *Amazon S3* é um serviço de *object storage* e o *Amazon EBS* é um serviço de *block storage*.

Na Tabela 2.3 são apresentadas as APIs suportadas pelas plataformas *clouds*.

Cloud providers suportados				
Soluções Open Source	**AWS EC2/S3**	**AWS EBS**	**Rackspace**	**GCE**
CloudStack	**Sim**	Não	Não	**Sim**
HP Helion Eucalyptus	**Sim**	**Sim**	Não	Não
OpenNebula	**Sim**	**Sim**	Não	Não
OpenStack	**Sim**	Não	**Sim**	**Sim**
Soluções comerciais				
Abiquo True Hybrid Cloud	**Sim**	Não	**Sim**	**Sim**
Flexiant Cloud Orchestrator	N/D	N/D	N/D	N/D
VMware vCloud Suite	N/D	N/D	N/D	N/D
xStream Cloud Management	**Sim**	Não	Não	Não

Tabela 2.3: Comparativo das APIs suportadas pelas plataformas *clouds*

2.4.4.4 Funcionalidades de redes

Nesta secção são apresentadas diferentes plataformas *clouds* (CP, *Cloud Platforms*) que suportam SDN, VLANS ou NFV. Como é possível observar a partir da Tabela 2.4, o *HP Eucalyptus* é a única *cloud* incapaz de suportar a tecnologia SDN. Relativamente

ao NFV, apenas o *OpenStack* tem suporte direto[36]. Enquanto que o *CloudStack* tem suporte parcial devido aos *plugins* de terceiros.

Funcionalidades de Rede			
Soluções Open Source	**SDN**	**VLANS**	**NFV**
CloudStack	**Sim**	**Sim**	Parcial
HP Helion Eucalyptus	Não	**Sim**	Não
OpenNebula	**Sim**	**Sim**	Não
OpenStack	**Sim**	**Sim**	**Sim**
Soluções comerciais			
Abiquo True Hybrid Cloud	Parcial	**Sim**	N/D
Flexiant Cloud Orchestrator	**Sim**	**Sim**	Não
VMware vCloud Suite	Parcial	**Sim**	Parcial
xStream Cloud Management	Não	N/D	Não

Tabela 2.4: Comparativo das funcionalidades de redes suportadas pelas plataformas *clouds*

A Tabela 2.5 apresenta a lista de vários *plugins* que podem ser integrados no *OpenStack* e/ou no *CloudStack*. Relativamente às soluções comerciais, o *VMware vCloud Suite* não tem suporte tanto para SDN como para NFV. Contudo, ambos têm um *software* adicional (licenciamento comercial), o *VMware NSX*, que permite adicionar estas funcionalidades à solução *VMware vCloud Suite*. No caso do *Abiquo True Hybrid Cloud* é possível ter disponível o SDN através do software *CohesiveFT*.

	OPNFV[38]	**OpenContrail[25]**	**Midonet[23]**	**VMware NSX[46]**
CloudStack	Não	**Sim**	**Sim**	**Sim**
OpenStack	**Sim**	**Sim**	**Sim**	**Sim**

Tabela 2.5: *Plugins* NFV suportados pelas plataformas *cloud*

2.4.4.5 Mecanismos de gestão de utilizadores

Existem três tipos de mecanismos de gestão de utilizadores: i) quotas de recursos, ii) delegação de privilégios e iii) integração com *Lightweight Directory Access Protocol* (LDAP).

i) quotas de recursos

A possibilidade de definir quotas de recursos para os utilizadores é uma funcionalidade pretendida nos mecanismos de gestão de utilizadores, como por exemplo, definir o número total de instâncias, quantidade de RAM máxima e número total de vCPUs.

ii) delegação de privilégios

Na delegação de privilégios, o utilizador, ao pertencer a um projeto, pode não ter permissões para criar novas instâncias, contudo, pode ter a possibilidade de as controlar, como, por exemplo, fazer *shutdown* e/ou *reboot*. Entende-se como "projeto" um espaço de trabalho onde está um *datacenter* virtual (recursos virtuais), que podem ser geridos por um ou mais utilizadores.

iii) integração com LDAP

A integração com o LDAP é importante, uma vez que permite ter uma gestão centralizada dos utilizadores, assim como a possibilidade de suportar mecanismos de *single sign-on*.

A Tabela 2.6 apresenta os mecanismos de gestão de utilizadores oferecidas pelas plataformas *cloud*, mostrando quais as soluções que suportam as funcionalidades descritas anteriormente.

Mecanismos de gestão de contas			
Soluções Open Source	**Quotas de recursos**	**Delegação de privilégios**	**Integração com LDAP**
CloudStack	**Sim**	**Sim**	**Sim**
HP Helion Eucalyptus	**Sim**	**Sim**	**Sim**
OpenNebula	**Sim**	**Sim**	**Sim**
OpenStack	**Sim**	**Sim**	**Sim**
Soluções comerciais			
Abiquo True Hybrid Cloud	**Sim**	**Sim**	**Sim**
Flexiant Cloud Orchestrator	**Sim**	**Sim**	Não
VMware vCloud Suite	**Sim**	**Sim**	**Sim**
xStream Cloud Management	N/D	N/D	N/D

Tabela 2.6: Funcionalidades de mecanismos de gestão de utilizadores oferecidas pelas plataformas *cloud*

2.4.4.6 Robustez

Na robustez são utilizados frequentemente mecanismos de balanceamento de carga (*Load Balancing*), que permitem aumentar o *throughput* de dados, promovendo a escalabilidade. Um exemplo de escalabilidade é o caso de através de um endereço IP, existir a possibilidade de distribuir os pedidos por vários servidores SMTP (*Simple Mail Transfer Protocol*). Assim, é possível enviar uma maior quantidade de e-mails, distribuindo a sua carga. Outra característica igualmente importante na robustez é a alta disponibilidade (*High Avaiability*, HA), sendo necessária por forma a garantir que em caso de falha de um ou mais nós, os serviços *core* continuam em funcionamento. Desta forma, os utilizadores não detetam qualquer falha que possa ter ocorrido nos serviços *core*. A monitorização dos serviços é bastante importante, visto que permite saber o estado atual do sistema, detetando possíveis falhas que possam ocorrer num ou mais serviços *core*. A Tabela 2.7 apresenta as *cloud platforms* que suportam as funcionalidades de robustez exigidas.

Mecanismos de robustez			
	Clustering		
Soluções Open Source	***High Avaiability***	***Load Balancing***	**Monitorização dos serviços**
CloudStack	Sim	Sim	N/D
HP Helion Eucalyptus	Sim	Sim	Sim
OpenNebula	Sim	Sim	Sim
OpenStack	Sim	Sim	Sim
Soluções Comerciais			
Abiquo True Hybrid Cloud	Parcial	Sim	Sim
Flexiant Cloud Orchestrator	Sim	Sim	Sim
VMware vCloud Suite	Parcial	Parcial	Sim
xStream Cloud Management	N/D	N/D	Sim

Tabela 2.7: Funcionalidades de robustez oferecidas pelas plataformas *cloud*

2.4.4.7 Segurança

A área de segurança é um dos grandes desafios num sistema *cloud*. É importante existir confidencialidade dos utilizadores através da encriptação dos dados a nível de armazenamento e de comunicações, principalmente em *clouds* híbridas. No caso das *clouds* híbridas é necessário garantir que toda a informação trocada entre a *cloud* privada e *cloud* pública não é decifrada por utilizadores mal intencionados.

É importante existir um isolamento de VMs, por forma a garantir que uma instância não tenha possibilidade de escalar os seus privilégios com o intuito de ter acesso ao *hypervisor*. O sistema deve ser auditável de forma a ter um registo de todas as ações efetuadas pelos utilizadores. Deste modo, caso ocorra algum problema, é possível verificar o histórico do sistema, permitindo assim ter acesso às alterações realizadas pelo utilizador. O suporte de uma infraestrutura de chaves públicas *(Public Key Infraestructure, PKI)* é um mecanismo de segurança de maneira a garantir que as comunicações efetuadas sejam autênticas e privadas.

A Tabela 2.8 apresenta as várias funcionalidades de segurança suportadas pelas diferentes plataformas *cloud*.

Segurança					
	Confidencialidade através da encriptação de dados				
Soluções Open Source	**Armazenamento**	**Comunicações**	**Isolamento de VMs**	**Auditável**	**Suporte PKI**
CloudStack	N/D	N/D	N/D	**Sim**	Não
HP Helion Eucalyptus	N/D	N/D	N/D	**Sim**	Não
OpenNebula	N/D	N/D	N/D	**Sim**	Não
OpenStack	**Sim**	**Sim**	N/D	**Sim**	**Sim**
Soluções Comerciais					
Abiquo True Hybrid Cloud	N/D	N/D	N/D	**Sim**	Não
Flexiant Cloud Orchestrator	N/D	N/D	N/D	N/D	N/D
VMware vCloud Suite	N/D	N/D	N/D	**Sim**	Não
xStream Cloud Management	**Sim**	N/D	N/D	**Sim**	Não

Tabela 2.8: Funcionalidades de segurança suportadas pelas plataformas *cloud*

2.4.5 Estudo comparativo

As Tabelas 2.9 e 2.10 foram construídas face aos resultados das tabelas anteriores, onde foram enumeradas as plataformas por ordem de pontos, sendo que para cada parcial foram atribuídos 0.5 pontos e por cada sim, foi atribuído 1 ponto. Para as características que tenham sido classificadas com 'N/D' foi também atribuído 1 ponto, assumindo que a plataforma poderá ter essa característica no limite.

Soluções Open-Source - Estudo Comparativo					
	CloudStack	**HP Helion Eucalyptus**	**OpenNebula**	**OpenStack**	**Total**
Gratuito	1	1	1	1	**1**
Hypervisors	4.5	1	4	7	**7**
Cloud Providers Suportados	2	2	2	3	**4**
Funcionalidades de Redes	2.5	1	2	3	**3**
Mecanismos de Gestão de Contas	3	3	3	3	**3**
Robustez	2	3	3	3	**3**
Segurança	1	1	1	4	**5**
Não definido	4	3	3	1	
Total	16	12	16	24	**26**
Total c/ N/D	20	15	19	25	

Tabela 2.9: Conclusão dos resultados das funcionalidades oferecidas pelas soluções *open-source* analisadas

Soluções Comerciais - Estudo Comparativo					
	Abiquo True Hybrid Cloud	**Flexiant Cloud Orchestrator**	**VMware vCloud Suite**	**xStream Cloud Management**	**Total**
Gratuito	0	0	0	0	**1**
Hypervisors	4	3.5	3	3	**7**
Cloud Providers Suportados	3	N/D	N/D	1	**4**
Funcionalidades de Redes	1.5	2	2	0	**3**
Mecanismos de Gestão de Contas	3	2	3	N/D	**3**
Robustez	2.5	3	2	1	**3**
Segurança	1	N/D	1	2	**5**
Não definidos	4	9	7	8	
Total	15	10.5	11	7	**26**
Total c/ N/D	19	19.5	18	15	

Tabela 2.10: Conclusão dos resultados das funcionalidades oferecidas pelas soluções comerciais analisadas

Na realização das análises comparativas anteriormente, optou-se por não utilizar uma análise multi-critério, tendo sido atribuídos pesos iguais para todas as funcionalidades. Procedeu-se desta forma, uma vez que todos os critérios foram considerados igualmente importantes na seleção da plataforma.

Os resultados obtidos de todas as funcionalidades pretendidas (Total c/ N/D), a solução a optar é o *OpenStack* (obteve 1ºlugar) devido a este ter uma margem de 5 pontos face à segunda melhor plataforma *cloud Cloudstack* (obteve 2ºlugar) para o nosso cenário.

2.5 Conclusão

A partir do estudo realizado no presente capítulo foi possível analisar quais as melhores opções para a implementação de um sistema *cloud* desta complexidade. Verificou-se que

um *hypervisor* do tipo 1, que utilize a técnica de para-virtualização ou *hardware* assistido, garante uma melhor performance face a outras soluções. No entanto, pode recorrer-se aos *Linux Containers* para alguns tipos de *deployment* adequados, como por exemplo, *web servers* em *Linux*.

Com o NFV é possível utilizar *hardware* comum (servidores) para correr *appliances* virtuais que forneçam funcionalidades de rede sem necessitar de fazer alterações a nível infraestrutural. Enquanto que o SDN simplifica a forma como os equipamentos de rede (físicos e virtuais) são controlados através da separação física entre o plano de controlo e o plano de encaminhamento.

Após uma análise aos três modelos de serviços (SaaS, PaaS e IaaS), verificou-se que o modelo que mais se adequa à solução pretendida é o IaaS. O IaaS permite ao utilizador fazer o aprovisionamento dos seus recursos computacionais, efetuando o aprovisionamento do sistema operativo desejado, com a possibilidade de fazer *deployment* das aplicações. Por sua vez, dos quatro modelos de *deployment* analisados, concluiu-se que o modelo que mais se adequa à realidade vivida no DEI é a *cloud* híbrida. Desta forma, é possível ter recursos no *datacenter* físico do DEI, existindo a possibilidade de os expandir para *clouds* exteriores às do DEI.

Por fim, efetuou-se um estudo comparativo entre as quatro soluções comerciais e as quatro soluções gratuitas (*open-source*). A solução que mais se adequou foi o *OpenStack*, uma vez que apresenta um maior número de funcionalidades pretendidas para o projeto. No entanto, por não ser uma solução comercial, não existe qualquer tipo de suporte 24/7 dias, tendo apenas o apoio da comunidade *open-source*.

3 Arquitetura de virtualização

3.1 Introdução

Neste capítulo é descrita a arquitetura atual do sistema de virtualização do DEI. A presente arquitetura apresenta algumas limitações que, por sua vez, motivam a existência de um sistema alternativo, a saber:

- Impossibilidade de estender os recursos de virtualização para *providers* externos, não sendo possível escalar a infraestrutura do DEI quando está na sua capacidade máxima;
- Não existe possibilidade de virtualizar redes, ou seja, o utilizador fica limitado à rede definida pelos administradores de sistemas do DEI;
- Os utilizadores não têm qualquer controlo sobre a infraestrutura, tendo apenas acesso às máquinas virtuais remotamente. Não existe a possibilidade de controlar a máquina virtual localmente, ou seja, no caso de efetuarem uma configuração errada perdem o acesso à máquina virtual (por exemplo, bloquear o acesso SSH na *firewall*).

Após a identificação das limitações da arquitetura atual, efetuou-se o levantamento dos requisitos funcionais e dos atributos de qualidade necessários para a nova proposta de sistema. No presente capítulo é apresentada uma proposta de arquitetura, a descrição da tecnologia *cloud* selecionada e os detalhes das diversas vistas de integração existentes. São ainda abordados os componentes necessários a desenvolver, de forma a cumprir os requisitos definidos.

3.2 Arquitetura atual

O sistema de virtualização do DEI é suportado por cinco servidores, quatro *switches*, três *storages* e três zonas de redes, como é possível verificar na Figura 3.1.

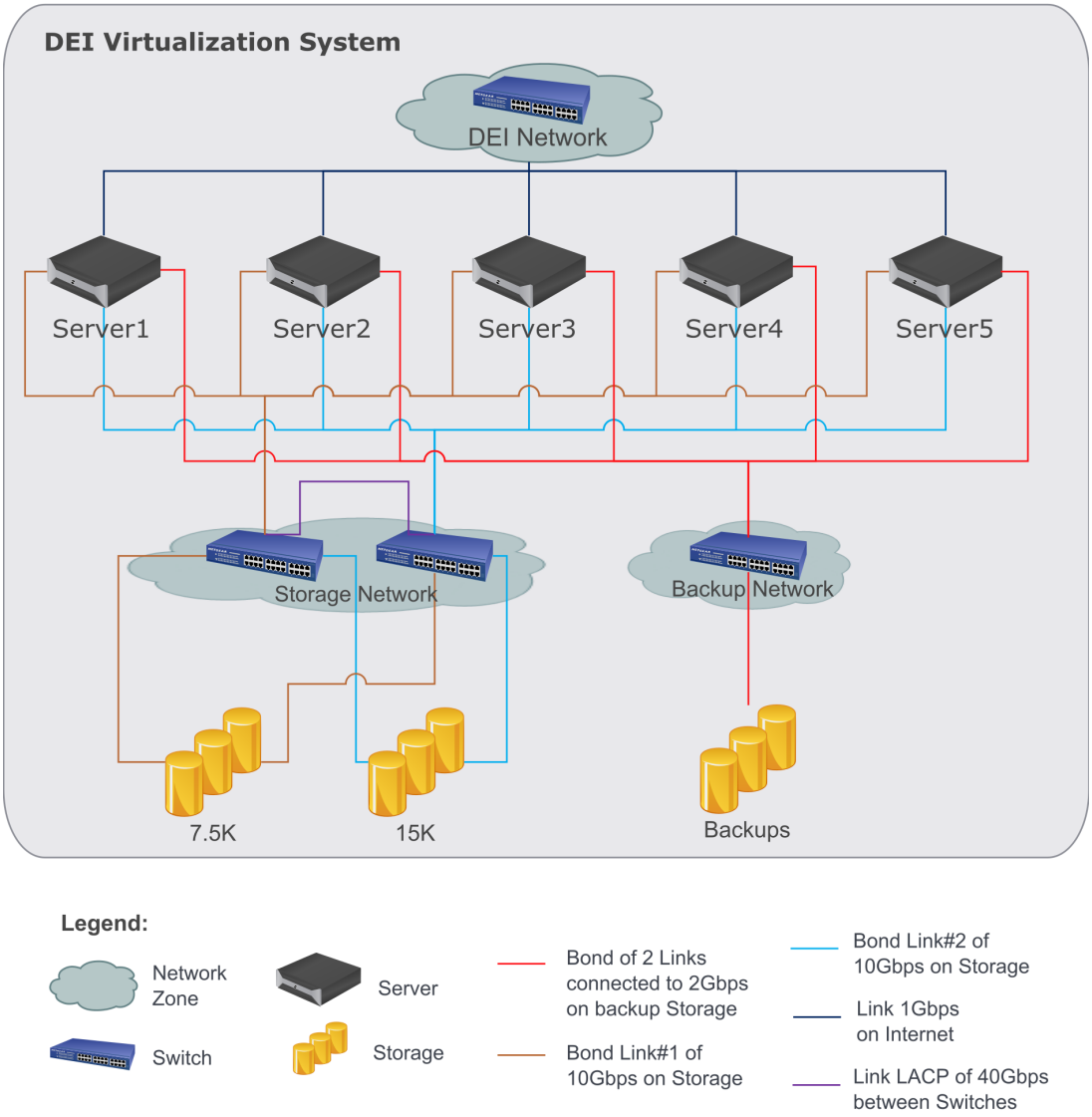


Figura 3.1: Visão de alto nível da arquitetura atual

Os servidores estão ligados de forma redundante à *storage* através de dois *switches* que, por sua vez, se encontram na zona de rede das *storages*, utilizando um *link* de 10Gbps. De forma análoga, os servidores estão ligados através de três *links* a 1Gbps (um para a rede interna do DEI e dois para a rede de *backup*). Desta forma, há garantia que existe, pelo menos, um caminho válido desde o emissor até o recetor caso ocorra uma falha de um dos componentes (*switch* ou placa de rede).

Relativamente às *storages*, tanto a de 7.5k como a de 15k rotações, encontram-se ligadas de forma redundante aos *switches* que se encontram na zona de rede das *storages*. A *storage* de *backup* encontra-se ligada ao *switch* de *backups* com dois *links* de 1Gbps.

Atualmente, os servidores de virtualização do DEI possuem o *hypervisor Xen*, sendo a gestão dos recursos controlada pelos administradores de sistemas (ver Figura 3.2). A

gestão de recursos passa pela configuração de rede (VLANS, *bonding*), configuração de *storages* (repositórios ISO e *internet Small Computer System Interface*, iSCSI) e a configuração das máquinas virtuais, definindo os metadados.

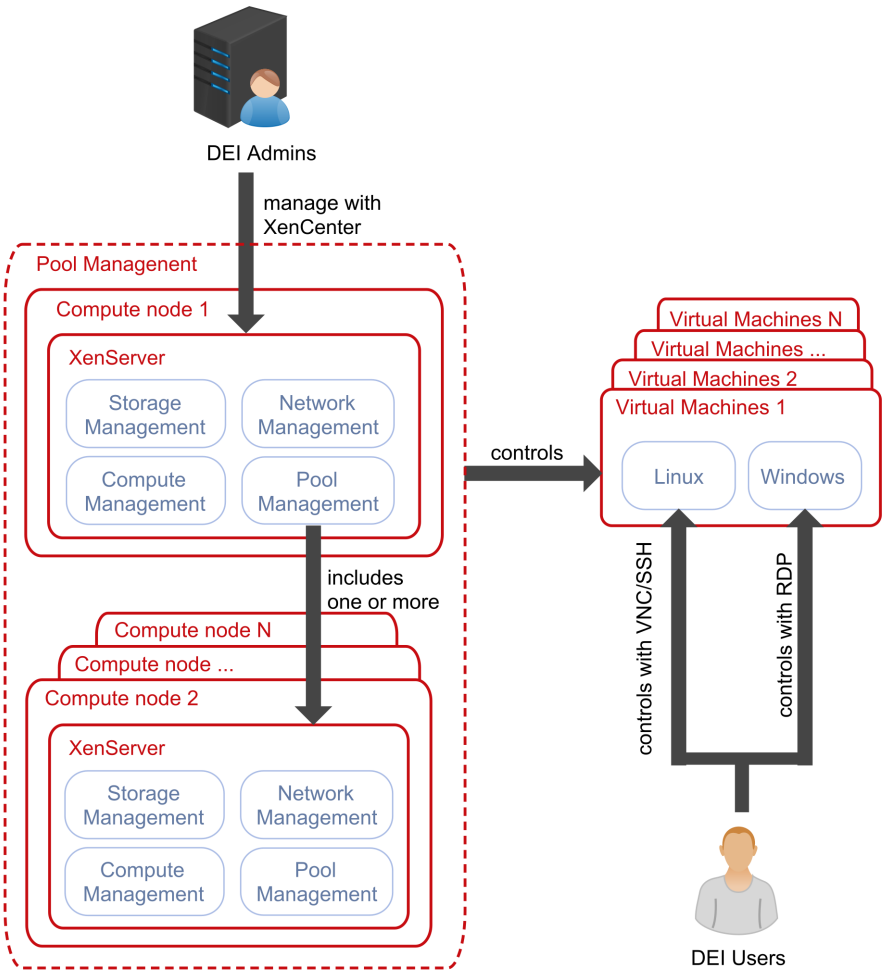


Figura 3.2: Diagrama lógico da arquitetura atual

Na presente arquitetura é possível efetuar a migração de máquinas virtuais entre os servidores sem que haja qualquer *downtime* associado, uma vez que os cinco servidores se encontram interligados entre si. Os utilizadores do DEI apenas têm acesso às máquinas virtuais através de um protocolo terminal remoto, não existindo a possibilidade de alterar o *hardware* existente. Por exemplo, para acederem ao *Windows* é necessário usar o RDP (*Remote Desktop Protocol*) e ao utilizar o *Linux* é possível efetuar o acesso a partir do SSH (*Secure Shell*), VNC (*Virtual Network Computing*) ou outro *software* semelhante.

3.3 Requisitos funcionais

No DEI foram identificados cinco atores diferentes, que irão utilizar a plataforma *cloud* que são: alunos, investigadores, docentes, administradores de sistema e funcionários. Os

requisitos funcionais foram retirados em forma de *user stories*, tendo sido divididos em duas categorias: requisitos funcionais do *OpenStack* e requisitos funcionais do componente a desenvolver[1].

3.3.1 *OpenStack*

Enquanto aluno, eu quero:

- Acesso a uma plataforma *cloud* que me permita instanciar máquinas virtuais para me auxiliarem na realização das unidades curriculares;
- Instalar outros sistemas operativos (imagens) para além dos que estão disponibilizados na plataforma;
- Aceder às seguintes funcionalidades de rede virtualizadas através do NFV: *firewall*, *load balancers*, *routers* e *networks*;
- Poder ter uma ou mais redes com vários *routers* e *networks* para ambientes mais complexos;
- Partilhar o meu projeto com colegas de grupos. De modo, a todos terem controlo sobre a infraestrutura na qual estamos a trabalhar;
- Aceder às instâncias localmente sem recorrer ao serviço SSH ou *Remote Desktop* no caso de perder acesso à rede;
- Configurar de forma automática a minha chave pública em novas instâncias, por forma a facilitar o acesso remoto;
- Controlar várias regiões, ou seja, gerir os diversos recursos, que se encontram tanto dentro do *datacenter*, localizado no DEI, como num *datacenter* do DEI, localizado num *provider* externo, utilizando a mesma interface;
- Interligar as minhas instâncias na mesma rede interna.

Enquanto investigador, eu quero:

- Acesso a uma plataforma *cloud* que me permita instanciar máquinas virtuais para os projetos de investigação;
- Instalar outros sistemas operativos (imagens) para além dos que estão disponibilizados na plataforma;
- Partilhar o meu projeto (*datacenter* virtual) com outros investigadores/docentes;
- Pertencer a um ou mais projetos na plataforma;

[1]Estas funcionalidades não são atualmente suportados pelo *OpenStack*. Como tal, motivam o desenvolvimento de um novo componente (DEIPubCloud) durante o trabalho do estágio, como se descreve mais à frente no relatório.

- Aceder às instâncias localmente sem recorrer ao serviço SSH ou *Remote Desktop* no caso de perder acesso à rede;
- Configurar de forma automática a minha chave pública em novas instâncias, por forma a facilitar o acesso remoto;
- Aceder às seguintes funcionalidades de rede virtualizadas através do NFV: *firewall*, *load balancers*, *routers* e *networks*;
- Poder ter uma ou mais redes com vários *routers* e *networks* para ambientes mais complexos;
- Poder criar volumes e associá-los a instâncias;
- Controlar várias regiões, ou seja, gerir os diversos recursos, que se encontram tanto dentro do *datacenter*, localizado no DEI, como num *datacenter* do DEI, localizado num *provider* externo, utilizando a mesma interface;
- Interligar as minhas instâncias na mesma rede interna.

Enquanto docente, eu quero:

- Acesso a uma plataforma *cloud* que me permita instanciar máquinas virtuais para fornecer aos alunos;
- Instalar outros sistemas operativos (imagens) para além dos que estão disponibilizados pela plataforma;
- Aceder às instâncias localmente sem recorrer ao serviço SSH ou *Remote Desktop* no caso de perder acesso à rede;
- Configurar de forma automática a minha chave pública em novas instâncias, por forma a facilitar o acesso remoto;
- Aceder às seguintes funcionalidades de rede virtualizadas através do NFV: *firewall*, *load balancers*, *routers* e *networks*;
- Poder ter uma ou mais redes com vários *routers* e *networks* para ambientes mais complexos;
- Poder criar volumes e associá-los a instâncias;
- Controlar várias regiões, ou seja, gerir os diversos recursos, que se encontram tanto dentro do *datacenter*, localizado no DEI, assim como num *datacenter* do DEI, localizado num *provider* externo, utilizando a mesma interface;
- Interligar as minhas instâncias na mesma rede interna.

Enquanto administrador, eu quero:

- Visibilidade geral do sistema, onde possa ver a totalidade de recursos disponíveis;
- Criar/Editar/Apagar planos de capacidade de computação, memória e capacidade de armazenamento que os utilizadores possam escolher quando efetuarem *deployment* das instâncias;
- Criar/Editar/Apagar papéis para os utilizadores do sistema (administrador, aluno, professor e funcionários);
- Criar/Editar/Apagar projetos associando-o aos utilizadores impondo limites de recursos;
- Criar/Editar/Apagar utilizadores que usem o sistema;
- Criar/Editar/Apagar papeis aos utilizadores;
- Acesso a todas as instâncias localmente de forma a resolver eventuais problemas que possam ocorrer;
- Ser notificado sempre que ocorra um erro na plataforma *cloud* ou quando os recursos atingem os limites definidos;
- Limitar o número de IOPS (*Input/Output Operations Per Second*) máximo que um projeto/utilizador possa usar;
- Controlar várias regiões, ou seja, gerir os diversos recursos, que se encontram tanto dentro do *datacenter*, localizado no DEI, assim como num *datacenter* do DEI, localizado num *provider* externo, utilizando a mesma interface.
- Interligar as minhas instâncias na mesma rede interna;

Como um Funcionário, eu quero:

- Aceder às minhas instâncias tanto dentro como fora do local de trabalho.

3.3.2 Componente a desenvolver (DEIPubCloud)

Enquanto investigador, eu quero:

- **ID 1:** Gerir os recursos localizados na *cloud* pública pela mesma interface que uso para gerir os recursos localmente;
- **ID 2:** Que as instâncias criadas na *cloud* pública comuniquem com as instâncias que estão localizadas na rede em que estou a trabalhar;
- **ID 3:** Uma visão geral das instâncias que tenho a correr, com informações relevantes, tais como, o endereço IP local (do DEI) e público, e o *provider* onde a máquina está instanciada.

Enquanto docente, eu quero:

- **ID 1:** Gerir os recursos localizados na *cloud* pública pela mesma interface que uso para gerir os recursos localmente.
- **ID 2:** Que as instâncias criadas na *cloud* pública comuniquem com as instâncias que estão localizadas na rede em que estou a trabalhar;
- **ID 3:** Uma visão geral das instâncias que tenho a correr, com informações relevantes, tais como, o endereço IP local (do DEI) e público, e o *provider* onde a máquina está instanciada.

Enquanto administrador, eu quero:

- **ID 4:** Gerir os *providers* disponíveis no módulo, definindo o valor máximo mensal que se pode gastar nesse *provider*, não sendo possível excede-lo;
- **ID 5:** Gerir as imagens dos sistemas operativos disponíveis de cada *provider*;
- **ID 6:** Gerir os planos de cada *provider*;
- **ID 7:** Gerir os projetos que têm acesso à *cloud* pública, definindo os respetivos *providers*/planos que podem criar, assim como a quantidade total que cada projeto possa ter.
- **ID 8:** Acesso a estimativas de gastos que ocorrem no mês corrente, assumindo que todas as instâncias ativas de momento tenham sido criadas desde o início do mês;
- **ID 9:** Guardar e visualizar o histórico da alteração de preços que um dado plano possa ter;
- **ID 10:** Visualizar todas as instâncias ativas e o respetivo custo que têm desde a sua criação;
- **ID 11:** Visualizar todas as instâncias criadas desde que o módulo entrou em funcionamento, visualizando o custo das instâncias;
- **ID 12:** Que apenas os utilizadores que têm o papel *deipubcloud* possam aceder ao módulo.

3.4 Atributos de qualidade

A seguir identificam-se os atributos de qualidade que devem ser garantidos pela implementação da arquitetura proposta.

Atributo de Qualidade Disponibilidade	Componente controlador
ID:	AQ#1
Fonte do estimulo:	Sistema
Estímulo:	Ocorreu um erro num servidor de controlo
Ambiente:	Sistema em produção
Artefacto:	Componente controlador
Resposta:	O sistema continua a funcionar normalmente
Medir resposta:	Não se verificam interrupções no serviço correspondente

Atributo de Qualidade Disponibilidade	Componente *network*
ID:	AQ#2
Fonte do estimulo:	Sistema
Estímulo:	Ocorreu um erro num servidor de *network*
Ambiente:	Sistema em produção
Artefacto:	Componente de *network*
Resposta:	O sistema continua a funcionar normalmente
Medir resposta:	Não se verificam interrupções no serviço correspondente

Atributo de Qualidade Disponibilidade	Componente *storage*
ID:	AQ#3
Fonte do estimulo:	Sistema
Estímulo:	Ocorreu um erro num servidor de *storage*
Ambiente:	Sistema em produção
Artefacto:	Componente de *storage*
Resposta:	O sistema continua a funcionar normalmente
Medir resposta:	Não se verificam interrupções no serviço correspondente

Atributo de Qualidade Extensibilidade	*Clouds* públicas
ID:	AQ#4
Fonte do estimulo:	Administrador de sistemas
Estímulo:	Adicionar/editar uma *cloud* pública
Ambiente:	Em manutenção
Artefacto:	Componente da respetiva *cloud* pública
Resposta:	O componente entra em produção
Medir resposta:	Após a modificação o sistema não sofre nenhum *down-time*.

Atributo de Qualidade Extensibilidade	*Clouds* privadas
ID:	AQ#5
Fonte do estimulo:	Administrador de sistemas
Estímulo:	Adicionar/apagar uma *cloud* privada que se encontra num *provider*
Ambiente:	Em manutenção
Artefacto:	Sistema
Resposta:	O componente entra em produção
Medir resposta:	Após a modificação o sistema não sofre nenhum *down-time*.

Atributo de Qualidade Extensibilidade	Suporte de sistemas de virtualização
ID:	AQ#6
Fonte do estimulo:	Administrador de Sistemas
Estímulo:	Adicionar/remover um novo sistema de virtualização
Ambiente:	Em manutenção
Artefacto:	Sistema
Resposta:	O componente entra em produção
Medir resposta:	Após a modificação o sistema não sofre nenhum *down-time*.

Atributo de Qualidade Segurança	Comunicações
ID:	AQ#7
Fonte do estimulo:	Sistema, Utilizador
Estímulo:	Os componentes comunicam entre si
Artefacto:	Componentes core do sistema, Dados
Ambiente:	Sistema em produção
Resposta:	Encripta todas as comunicações trocadas entre os componentes
Medir resposta	Entidades não envolvidas na comunicação verificam que a mesma está encriptada

Atributo de Qualidade Segurança	Volumes
ID:	AQ#8
Fonte do estimulo:	Utilizadores
Estímulo:	Utilizador não autorizado tenta aceder a um volume que não lhe pertence
Artefacto:	Componente responsável pelo armazenamento de volumes
Ambiente:	Sistema em produção
Resposta:	O acesso ao volume é bloqueado
Medir resposta:	Os acessos não são disponibilizados

Atributo de Qualidade Segurança	Área pessoal do utilizador
ID:	AQ#9
Fonte do estimulo:	Utilizadores
Estímulo:	Utilizador tenta aceder de forma não autorizada como se fosse outro utilizador
Artefacto:	Web UI
Ambiente:	Sistema em produção
Resposta:	O acesso indevido é bloqueado
Medir resposta:	Os acessos não são disponibilizados

Atributo de Qualidade Segurança	Dados
ID:	AQ#10
Fonte do estimulo:	Utilizadores
Estímulo:	Utilizador mal intencionado tenta captar tráfego que circula na rede *cloud*
Ambiente:	Sistema em produção
Artefacto:	Controlador de *network*
Resposta:	Acede ao tráfego da sua rede
Medir resposta:	Não se verifica tráfego de outros utilizadores

Atributo de Qualidade Robustez	Dados
ID:	AQ#11
Fonte do estimulo:	Sistema
Estímulo:	Uma falha ocorre na *storage*
Ambiente:	Sistema em produção
Artefacto:	Componente responsável pelo armazenamento de dados
Resposta:	Sistema continua a funcionar normalmente
Medir resposta:	Não há dados perdidos e é possível continuar a efetuar operações na *storage*

Atributo de Qualidade Performance	Criação/eliminação de instâncias
ID:	AQ#12
Fonte do estimulo:	Utilizadores
Estímulo:	Criam/eliminam instâncias de virtualização
Ambiente:	Sistema em produção
Artefacto:	Componente responsável pela gestão de recursos virtuais
Resposta:	Os pedidos são processados normalmente
Medir resposta:	Num sistema em produção com 50 pedidos, todos são processados

Atributo de Qualidade Performance	*Boot/restart/stop* de instâncias
ID:	AQ#13
Fonte do estimulo:	Utilizadores
Estímulo:	Fazem *boot/restart/stop* das instâncias de virtualização
Ambiente:	Sistema em produção
Artefacto:	Componente responsável pela gestão de recursos virtuais
Resposta:	Os pedidos são processados normalmente
Medir resposta:	Num sistema em produção com 50 pedidos, todos são processados

Atributo de Qualidade Auditabilidade	Componentes do Sistema
ID:	AQ#14
Fonte do estimulo:	Administrador de Sistemas
Estímulo:	Observar o estado do sistema *cloud*
Ambiente:	Sistema em desenvolvimento e em produção
Artefacto:	Componentes do sistema
Resposta:	Diagnostico final com a informação do estado dos componentes do sistema
Medir resposta:	Verificam-se quais são os componentes em falha, assim como quem realizou as operações no sistema

3.5 Proposta de arquitetura

A Figura 3.3 apresenta a arquitetura idealizada para a implementação de uma *cloud* híbrida no DEI. O *deployer node* é um servidor responsável por realizar a automatização do *deployment* da infraestrutura privada, localizada tanto no DEI como num *provider* externo. É utilizada uma ferramenta de *deployment* responsável pela configuração automática dos serviços *OpenStack*.

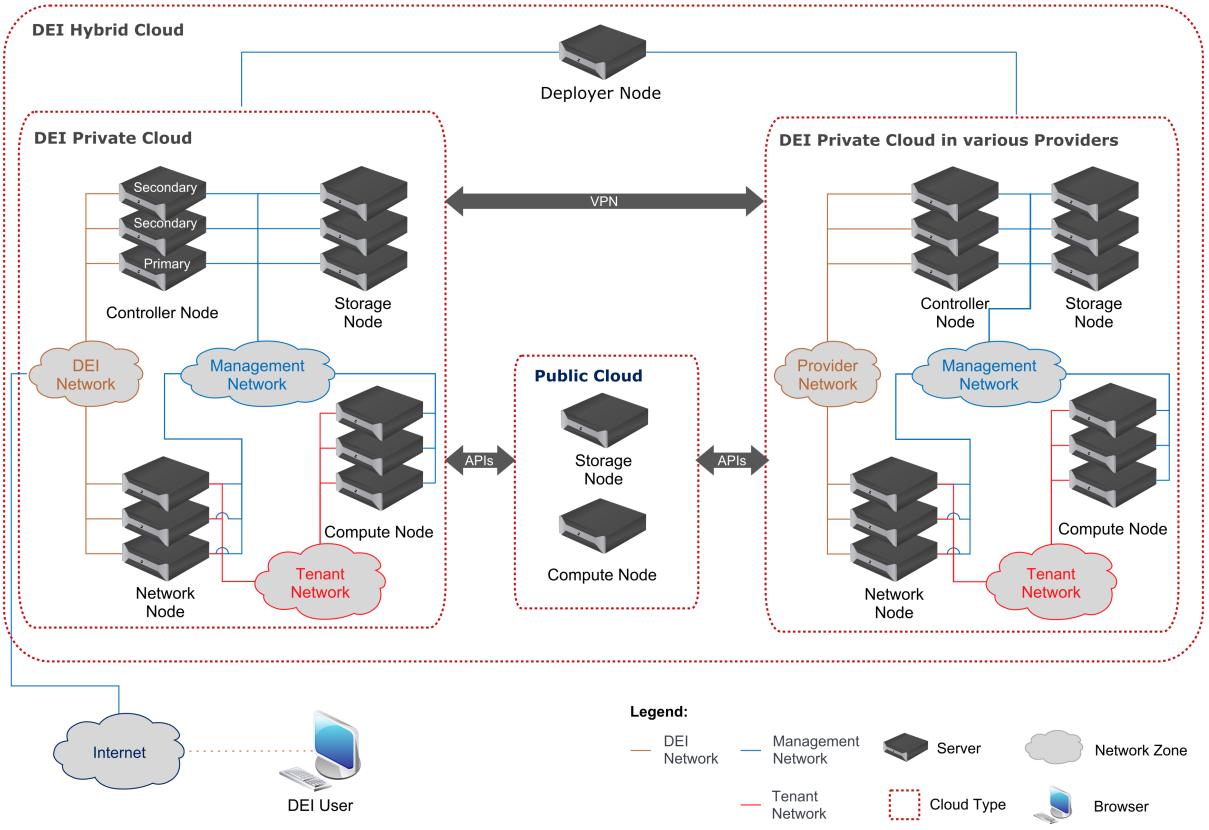


Figura 3.3: Visão de alto nível da arquitetura idealizada

O conjunto dos servidores de controlo é responsável por fornecer uma interface Web para gerir a *cloud*, controlar os recursos localizados na *cloud* pública, gerir projetos, utilizadores, volumes e recursos de rede e os *hypervisors*.

É necessário ainda um conjunto de servidores de *storage* onde é fornecido *block storage* por forma a ser utilizado nos servidores *compute*, assim como, disponibilizar armazenamento de *object storage*. Os servidores de *network* são responsáveis por fornecer as funcionalidades de rede virtualizadas aos servidores de *compute* que correm as máquinas virtuais dos utilizadores.

Na Figura 3.4 verifica-se os componentes necessários para o funcionamento dos respetivos serviços de *storage*, *network*, *compute* e de controlo.

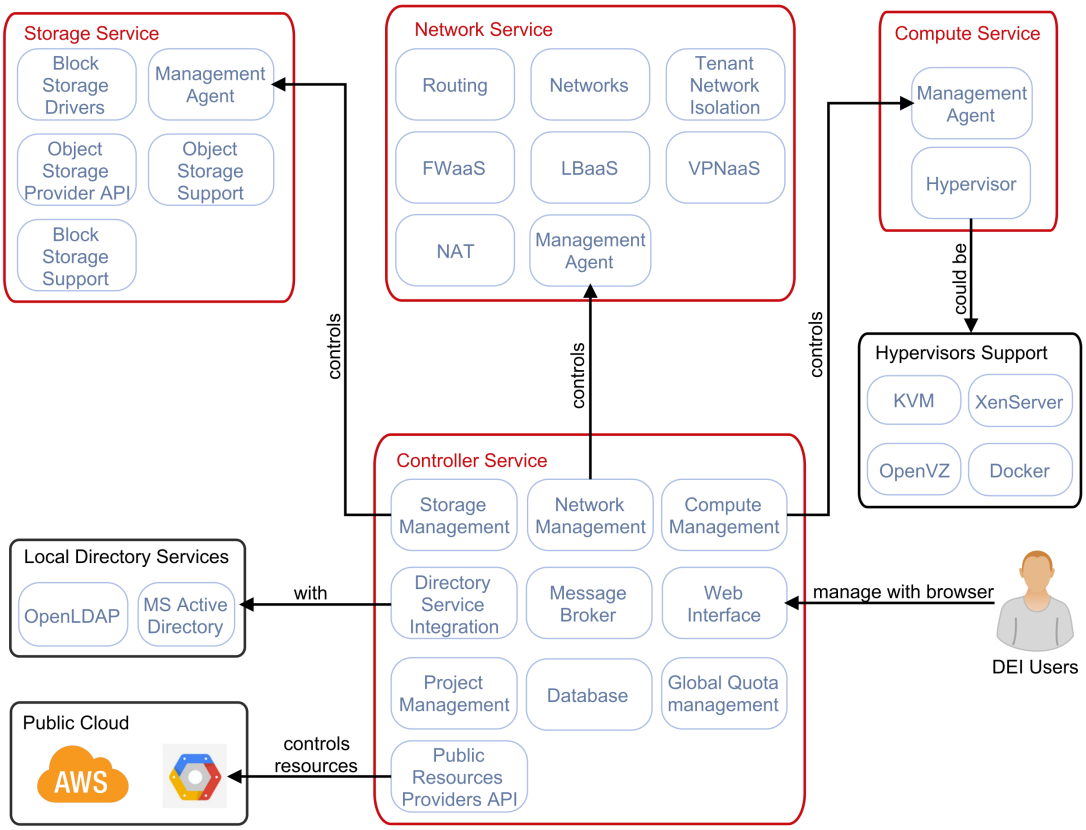


Figura 3.4: Diagrama lógico da arquitetura idealizada

As Tabelas 3.1, 3.2, 3.3 e 3.4 descrevem os componentes que se encontram dentro dos serviços definidos na Figura 3.4.

Controller Service	
Componente	**Descrição**
Compute Management	*Interface* que permita controlar o serviço de *compute*, por exemplo: criar/editar/eliminar VMs, verificar os recursos disponíveis na máquina que corre o serviço *compute*.
Network Management	*Interface* que permita controlar o serviço de *network*, por exemplo: adicionar/editar/eliminar interfaces de rede a VMs/redes/routers, fazer NAT (Network Address Translation).
Storage Management	*Interface* que permita controlar o serviço de *storage*
Directory Service Integration	Componente que permita integrar o sistema de autenticação com um serviço *LDAP* existente, por exemplo: *OpenLDAP*, *Microsoft Active Directory*.
Web Interface	Uma interface Web que permita os utilizadores do DEI gerirem a sua *cloud*
Public Resources Providers API	Controlar os recursos *compute* localizados nas *clouds* públicas, por exemplo: AWS, *Google Compute Engine*.
Projects Management	Controlar os projetos associando-os aos utilizadores com gestão de quotas e privilégios.
Message Broker	Facilitar a interoperabilidade entre os diversos serviços da *cloud*, coordenando as operações e a informação do estado dos serviços.
Database	Guarda todas os dados relativos à plataforma *cloud*.
Global Quota Management	Guarda a informação de todos os recursos que estão livres/ocupados da infraestrutura *cloud*, tais como, os recursos *compute*, *networking*, *storage* e os recursos alocados na *cloud* pública.

Tabela 3.1: Componentes do serviço de controlo

Compute Service	
Componente	**Descrição**
Hypervisor	Sistema de virtualização que corre as VMs na *cloud*, por exemplo: *KVM*, *OpenVZ*, *Xen* e *Docker*.
Management Agent	Interface que recebe instruções de controlo do serviço.

Tabela 3.2: Componentes do serviço *compute*

Network Service	
Componente	**Descrição**
Routing	Componente responsável pelo encaminhamento dos pedidos de rede na 3º camada.
Networks	Componente responsável pela gestão das *networks* dos utilizadores.
Tenant Network Isolation	Componente responsável pela separação física da rede dos diversos projetos, sendo possível existir a mesma rede em projetos diferentes.
NAT	Componente responsável por fazer as VMs terem acesso à Internet que esteja na rede privada.
Management Agent	Interface que recebe instruções de controlo do serviço.
VPNaaS (Virtual Private Network as a Service)	Componente que permite utilizar *software* VPN como um serviço sem a necessidade de realizar configurações complexas.
LBaaS (Load Balancer as a Service)	Componente que permite utilizar *software load balancer* como um serviço sem a necessidade de realizar configurações complexas.
FWaaS (Firewall as a Service)	Componente que permite utilizar *software* de *firewall* como um serviço sem a necessidade de realizar configurações complexas.

Tabela 3.3: Componentes do serviço *network*

Storage Service	
Componente	**Descrição**
Block Storage Drivers	Componente que contém diversos *drivers* que permitam utilizar *storages* físicas.
Management Agent	Interface que recebe instruções de controlo do serviço.
Block Storage Support	Componente que guarda os dados em blocos, através de volumes.
Object Storage Support	Componente que guardar os dados como objetos.
Object Storage Provider API	Componente que permitir gerir dados que estejam numa *cloud* pública.

Tabela 3.4: Componentes do serviço *storage*

3.6 Tecnologia *cloud* selecionada

Como referido na secção 2.4.5, "Estudo comparativo", a tecnologia com mais funcionalidades implementadas para o objetivo pretendido com este trabalho é o *OpenStack*. Por este motivo será a solução adotada no projeto. O *OpenStack*[31] é uma plataforma de *cloud computing* que controla um conjunto de recursos computacionais, *storage* e de rede num *datacenter*. Através da *dashboard* do *OpenStack* é permitido aos administradores terem controlo da infraestrutura e aos utilizadores a possibilidade de fazer a gestão da infraestrutura. O *OpenStack* é composto por vários componentes que em conjunto formam a *cloud*, como mostra a Figura 3.5.

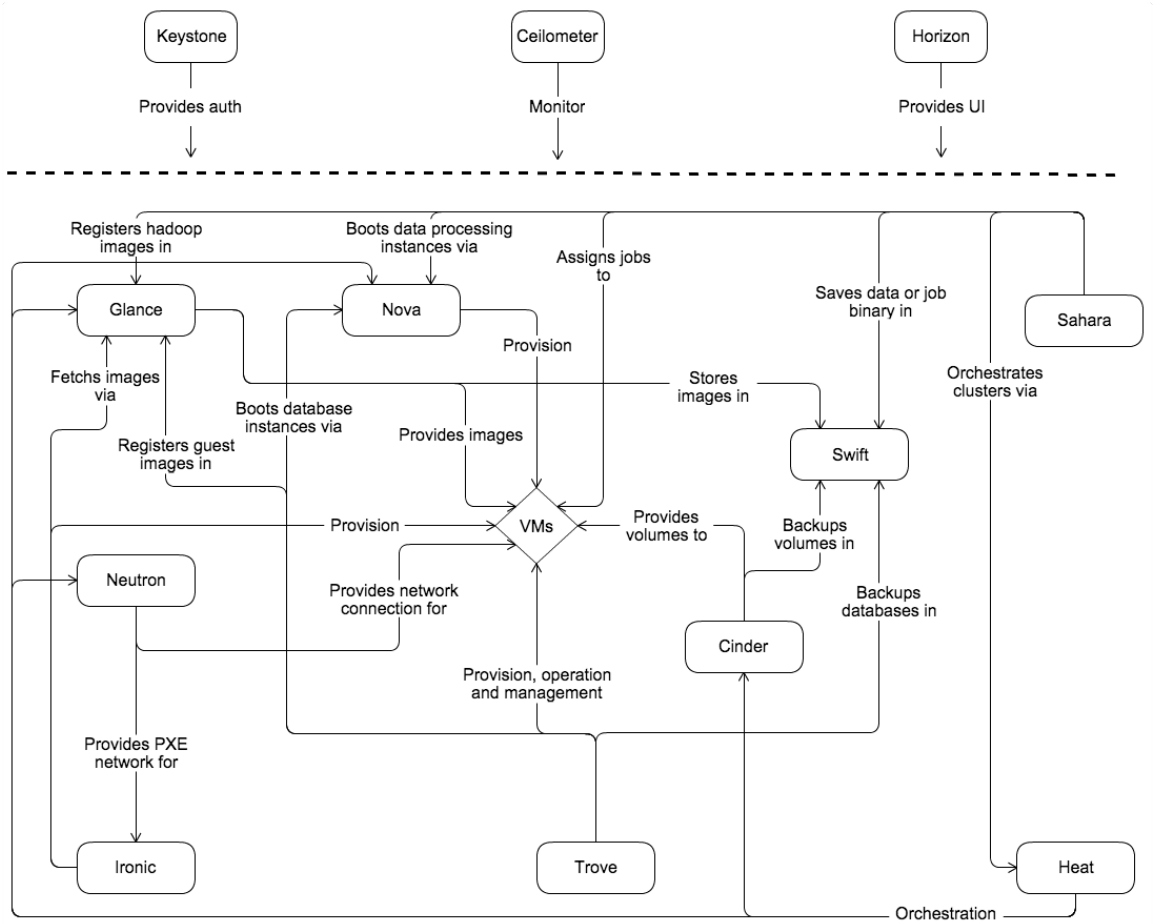


Figura 3.5: Diagrama conceptual da arquitetura do *OpenStack*[33]

O componente ***KeyStone***[30] (*OpenStack Identity Service*) fornece serviços de autenticação, gestão de utilizadores e respetivos privilégios. É um serviço crucial que está na base da autenticação e verificação entre todos os serviços *cloud* do *OpenStack*. A autenticação dos utilizadores e *tenants* é efetuada enviando uma autorização válida do *token* entre todos os serviços, sendo, posteriormente, usado para a autenticação e verificação dos privilégios do utilizador.

O ***Celiometer*** permite recolher e guardar os dados de utilização dos recursos físicos e virtuais, localizados na *cloud*, sendo possível realizar a análise e executar ações pré-definidas quando um ou mais eventos definidos acontecem.

O ***Horizon*** (*OpenStack DashBoard*) é uma aplicação Web desenvolvida em *Django* que permite aos utilizadores e administradores efetuarem a gestão da *cloud*, através de um *browser*.

O ***Glance*** (*OpenStack Image Service*) é um serviço que permite registar, descobrir e obter imagens (ISO, VHD) de máquinas virtuais por forma a utilizar no ambiente *OpenStack*, dando a possibilidade do próprio utilizador enviar as suas próprias imagens. Estas podem ser guardadas em diversas locais, desde um *filesystem* local até a um *filesystem* distribuído

como, por exemplo, o *OpenStack Object Storage*.

O ***Nova*** (*OpenStack Compute*) é o componente responsável pelo *compute*. Este componente permite controlar vários *hypervisors* suportados onde correm várias instâncias que são distribuídas por diversos *hosts* físicos.

O ***Sahara*** (*Elastic Map Reduce*) fornece aos utilizadores a possibilidade de fazer aprovisionamento automático e gestão de *clusters* como o *Hadoop*, *Spark*, *Storm* ou outra *framework* que permita realizar o processamento distribuído de vários *datasets*. Este aprovisionamento permite especificar vários parâmetros como a versão, topologia do *cluster* e detalhes do *hardware* dos nós, efetuando um *deployment* sem ser necessário efetuar configurações complexas.

O ***Neutron*** (*OpenStack Networking*), componente que tem o serviço de *Networking* SDN, fornece uma API para definir a conectividade e o endereçamento *cloud* dos utilizadores. A arquitetura do *Neutron* é modular, sendo possível adicionar *plugins*, tais como FWaaS, LBaaS e VPNaaS.

O ***Swift*** é o componente que fornece o *Object Storage*. Este serviço permite criar um *cluster* de armazenamento de dados, baseado em objetos, sendo otimizado para *multi-tenancy* e alta concorrência.

O ***Cinder*** é o componente que fornece *Block Storage*. Os dados guardados nas instâncias não são persistentes, ou seja, quando uma instância for terminada, qualquer alteração efetuada no disco local é perdida. Este componente permite associar um ou mais volumes a uma instância, mesmo que esta seja eliminada, o volume permanece disponível, podendo ser alocado para outra instância.

O ***Ironic*** (*OpenStack Bare-Metal Provisioning*) integra no *OpenStack* com a funcionalidade de realizar aprovisionamentos de *bare-metals*. Por defeito, utiliza o PXE (*Preboot Execution Environment*) e o IPMI (*Intelligent Platform Management Interface*) de forma a efetuar o aprovisionamento através do *boot* pela rede e de controlar o estado da máquina (ligada/desligada). Suporta ainda *plugins* de diversos fabricantes, por forma a suportar funcionalidades adicionais.

O **Trove** fornece uma *database as a service* e permite aos utilizadores usar funcionalidades de base de dados relacionais ou não relacionais sem efetuarem qualquer tipo de configuração. Este componente permite fazer a gestão e aprovisionamento de instâncias conforme o necessário. Suporta bases de dados como o *Percona XtraDB cluster*, *MongoDB* e *MemCached*.

Por fim, o ***Heat*** (*OpenStack Orchestration*) é um serviço que permite efetuar a orquestração, através de *templates* que descrevem a infraestrutura em forma de texto, permitindo realizar um *deployment* automático dos recursos.

Após efetuada a introdução aos componentes envolvidos na arquitetura conceptual do

OpenStack, descritos na Figura 3.5, nem todos são necessários para a realização da arquitetura proposta. Nas Tabelas 3.5, 3.6, 3.6 e 3.8 apresentam o mapeamento dos componentes definidos na arquitetura idealizada.

Mapeamento do serviço de controlo com o *OpenStack*	
Componente da Arquitetura	**Componente do OpenStack**
Compute Management	*Nova*
Network Management	*Neutron*
Storage Management	*Swift*, *Cinder*
Directory Service Integration	*KeyStone*
Web Interface	*Horizon*
Public Resources Providers API	(Componente não existente)
Projects Management	*KeyStone*
Message Broker	*RabbitMQ*
Database	*MySQL*
Global Quota Management	*Nova*, *Neutron*, *Cinder*

Tabela 3.5: Mapeamento do serviço de controlo com o *OpenStack*

Mapeamento do serviço compute com o *OpenStack*	
Componente da Arquitetura	**Componente do *OpenStack***
Management Agent	*Nova-compute*
Hypervisor	*KVM* ou *Xen*

Tabela 3.6: Mapeamento do serviço de *compute* com o *OpenStack*

Mapeamento do serviço *network* com o *OpenStack*	
Componente da Arquitetura	**Componente do *OpenStack***
Routing	*Neutron*
Networks	
Tenant Network Isolation	
NAT	
Management Agent	
VPNaaS	*Neutron-vpnaas*
LBaaS	*Neutron-lbaas*
FWaaS	*Neutron-fwaas*

Tabela 3.7: Mapeamento do serviço de *network* com o *OpenStack*

Mapeamento do serviço *storage* com o *OpenStack*	
Componente da Arquitetura	**Componente do *OpenStack***
Block Storage Driver	*Cinder*
Management Agent	*Cinder*, *Swift*
Block Storage Support	*Cinder*
Object Storage Support	*Swift*
Object Storage Provider	Falta implementar

Tabela 3.8: Mapeamento do serviço de *storage* com o *OpenStack*

A escolha do *MySQL* como base de dados e do *RabbitMQ* como *Message Broker*, deve-se ao facto destas tecnologias terem sido utilizadas e testadas pelos administradores de sistemas do DEI, onde provaram ser soluções bastantes robustas. Para além disto, são as tecnologias que mais têm suporte pela comunidade do *OpenStack*.

3.7 Integração de *datacenters*

Nesta integração pretende-se utilizar mais que um *datacenter* físico, distribuídos geograficamente num *provider* externo, estando interligados com o *datacenter* do DEI. Desta forma, garantimos que a infraestrutura tenha possibilidade de ser escalável, utilizando *bare-metals*, assim como a possibilidade de diminuir as latências para situações específicas.

Como é possível verificar na Figura 3.6 os *datacenters* encontram-se com uma configuração semelhante, à exceção dos nós controladores, onde apenas o controlador do DEI tem o serviço *KeyStone* e *Horizon*. Esta arquitetura permite centralizar todos os dados dos utilizadores, projetos e privilégios que estes têm sobre a plataforma, assim como, acederem a ambas as regiões na mesma interface Web. A comunicação entre os componentes de um *datacenter* e outro são efetuados através de uma ligação VPN persistente.

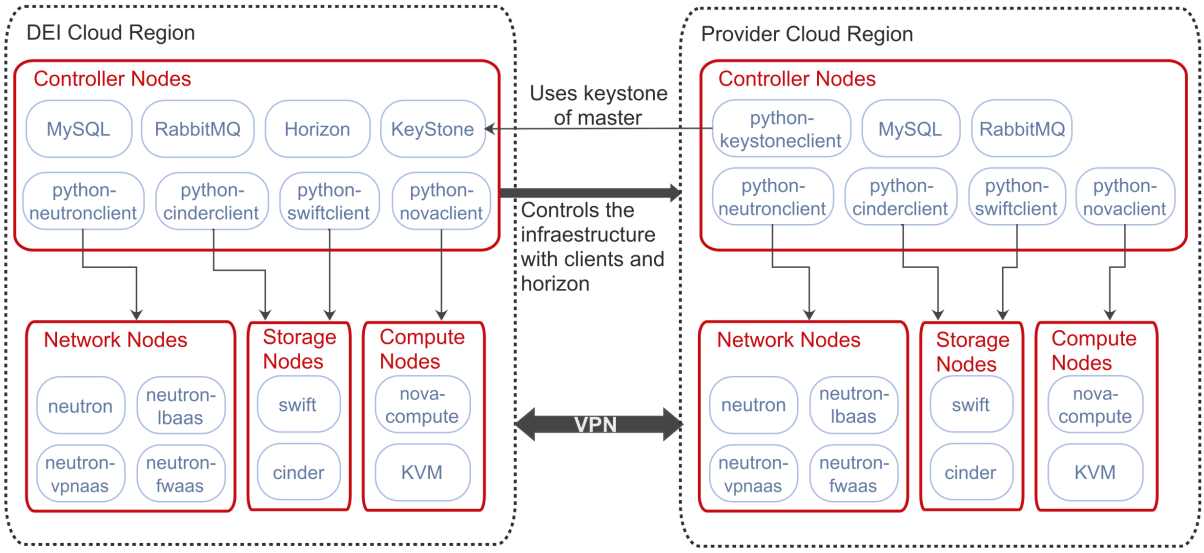


Figura 3.6: Diagrama lógico da arquitetura proposta

Existe a possibilidade de integração do *datacenter* do DEI com outras universidades que tenham a plataforma *OpenStack*, através do *Keystone-to-Keystone federation*[39]. Através desta funcionalidade, é possível aos utilizadores do DEI a autenticação na plataforma *cloud* habitual e ter acesso automático aos recursos de outra região, externa do domínio do DEI.

Para além disto, é necessário seleccionar componentes externos ao *OpenStack*. Um conjunto de ferramentas que tratam da automatização da infraestrutura *OpenStack* como, por exemplo, o *Cobbler* por forma a tratar do aprovisionamento dos servidores e o *Puppet*, *Chef* ou *Ansible* para fazer o *deployment* dos componentes.

Por forma a garantir a redundância dos serviços *core* são utilizados mecanismos de *clustering* do *Linux* como, por exemplo, o *keepalived* ou *pacemaker*. É ainda necessário um

cluster de base de dados (*Galera Cluster*), distribuindo a carga dos servidores num modo ativo/ativo. Estes mecanismos de *clustering* são recomendados e suportados pelo *OpenStack*, desta forma, opta-se por este tipo de soluções de alta disponibilidade.

3.8 Integração com a *cloud* pública

Nesta integração pretende-se utilizar os serviços *cloud* existentes (*Amazon Web Services* e a *DigitalOcean*), por forma a se obter mais *storage* e recursos computacionais. Para efetuar a integração é necessário verificar se o *provider* desejado suporta operações de gestão via API. Após a escolha de um ou mais *providers*, é possível utilizar os recursos *cloud* públicos, existindo dois cenários:

1. Sem integração com a rede do utilizador.
2. Com integração na rede do utilizador, ou seja, as instâncias localizadas na *cloud* pública conseguem comunicar com as que se encontram na *cloud* privada.

Para a realização do primeiro cenário, é necessário proceder à implementação de um componente que permita a utilização da API do *provider*, por forma a efetuar o aprovisionamento das instâncias, através da plataforma Web.

No segundo cenário, para além do componente referido anteriormente, é necessário planear uma instalação e configuração automática de uma instância que faça a ligação entre a rede do *provider* ao *datacenter* do utilizador, através do serviço VPNaaS. Para além disto, é necessário aceder às instâncias virtuais do utilizador, por forma a definir as rotas de encaminhamento necessárias para efetuar as comunicações na rede do DEI.

Para ambos os cenários é necessário estender o projeto *Horizon* de forma aos utilizadores poderem controlar os seus recursos *cloud* públicos na interface Web. Optou-se pelo segundo cenário, pois é uma solução mais economicamente viável face à solução do *datacenter*, assim como é possível instanciar máquinas na *cloud* pública como se se encontrassem na rede do DEI. Para além disto, trata-se de uma solução que cumpre com os requisitos iniciais do projeto, suportando virtualização de redes e transparência nas comunicações entre as máquinas virtuais de um utilizador.

3.9 Componentes a desenvolver

Na arquitetura proposta (Figura 3.4), o *OpenStack* tem em falta dois componentes: *Public Resources Providers API* e o *Object Storage Provider*. Devido aos objetivos do Estágio, bem como ao tempo disponível para desenvolvimento no planeamento dos trabalhos, optou-se por desenvolver o módulo *Public Resources Providers*, chamado de *DEIPubCloud*.

Devido a inexistência destes componentes no *OpenStack*, foi necessário desenvolve-los por forma a cumprir os seguintes objetivos fundamentais:

- Controlar várias regiões, ou seja, gerir os diversos recursos que se encontram tanto dentro do *datacenter* do DEI como num *provider* público utilizando a mesma interface.
- Interligar as instâncias na mesma rede do DEI.

O componente *DEIPubCloud* permite implementar o 2º Cenário descrito na secção 3.8, e, como tal, é responsável por adicionar o suporte ao controlo de instâncias virtuais localizadas na *cloud* pública. Este componente também permite integrar as instâncias que estão na *cloud* pública com os recursos locais, através de uma ligação VPN. Para isto, é necessário estender o projeto *Horizon*, criando *dashboards* adicionais, de forma a permitir mais funcionalidades na interface *Web*. Desta forma, usa-se sempre a mesma plataforma para efetuar a gestão de todos os *datacenters*. A utilização deste componente tem dois tipos de utilizadores:

- Administradores - têm acesso a uma área administrativa no *Horizon* que lhes permite fazer a gestão dos *cloud providers*; definir quais os utilizadores que têm acesso a utilizar esta funcionalidade, definindo-lhes quotas; controlar as máquinas virtuais de todos os utilizadores que estejam registados no sistema.
- Utilizadores - podem gerir as suas máquinas virtuais na *cloud*; interligar as suas máquinas virtuais ligadas na *cloud* pública com o *datacenter* do DEI.

A Figura 3.7 apresenta o contexto do sistema final, onde através do *OpenStack* se pretende gerir instâncias na *cloud* pública, com a possibilidade de as integrar no *datacenter* do DEI através de uma VPN.

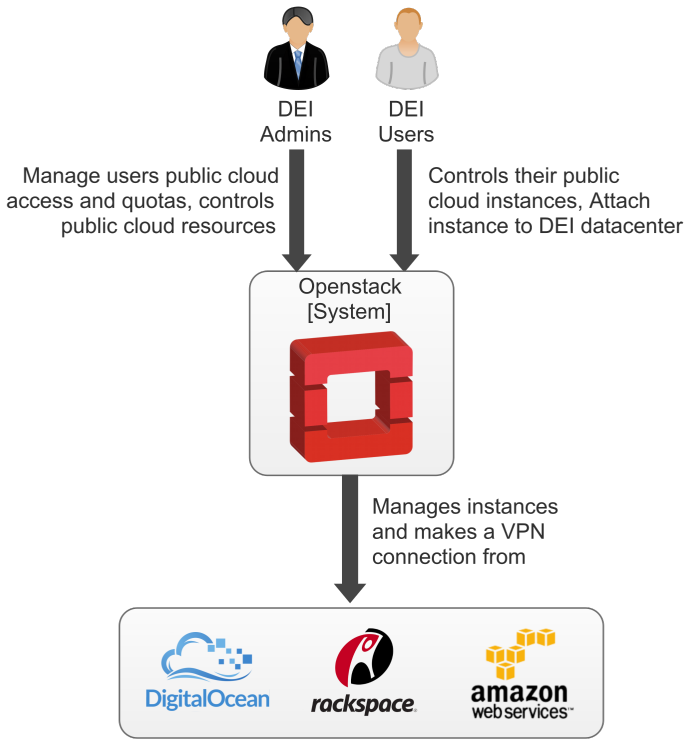


Figura 3.7: Diagrama de contexto do módulo a implementar

A Figura 3.8 apresenta os *containers* do *OpenStack* que são utilizados nesta arquitetura, onde o *Horizon* é o componente que é estendido.

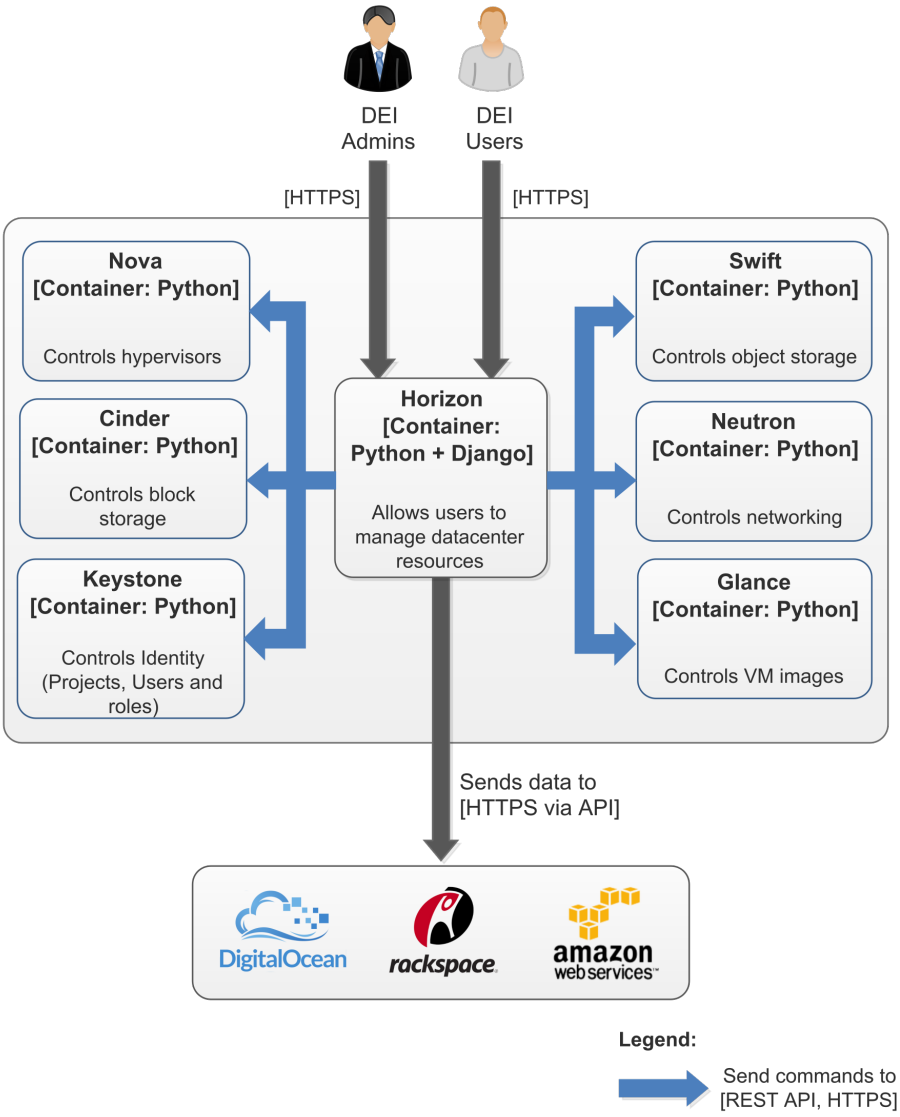


Figura 3.8: Diagrama de *containers* do componente *Horizon* do DEI

Como é possível verificar na Figura 3.9, o componente *Horizon* inclui quatro *dashboards*: *Project dashboard*, *Admin dashboard*, *Identity dashboard* e *Settings dashboard*. Neste trabalho, criou-se um novo *dashboard*, chamado DEIPubCloud Dashboard - Frontend, que fornece uma interface Web para ambos os utilizadores gerirem os recursos públicos *cloud*.

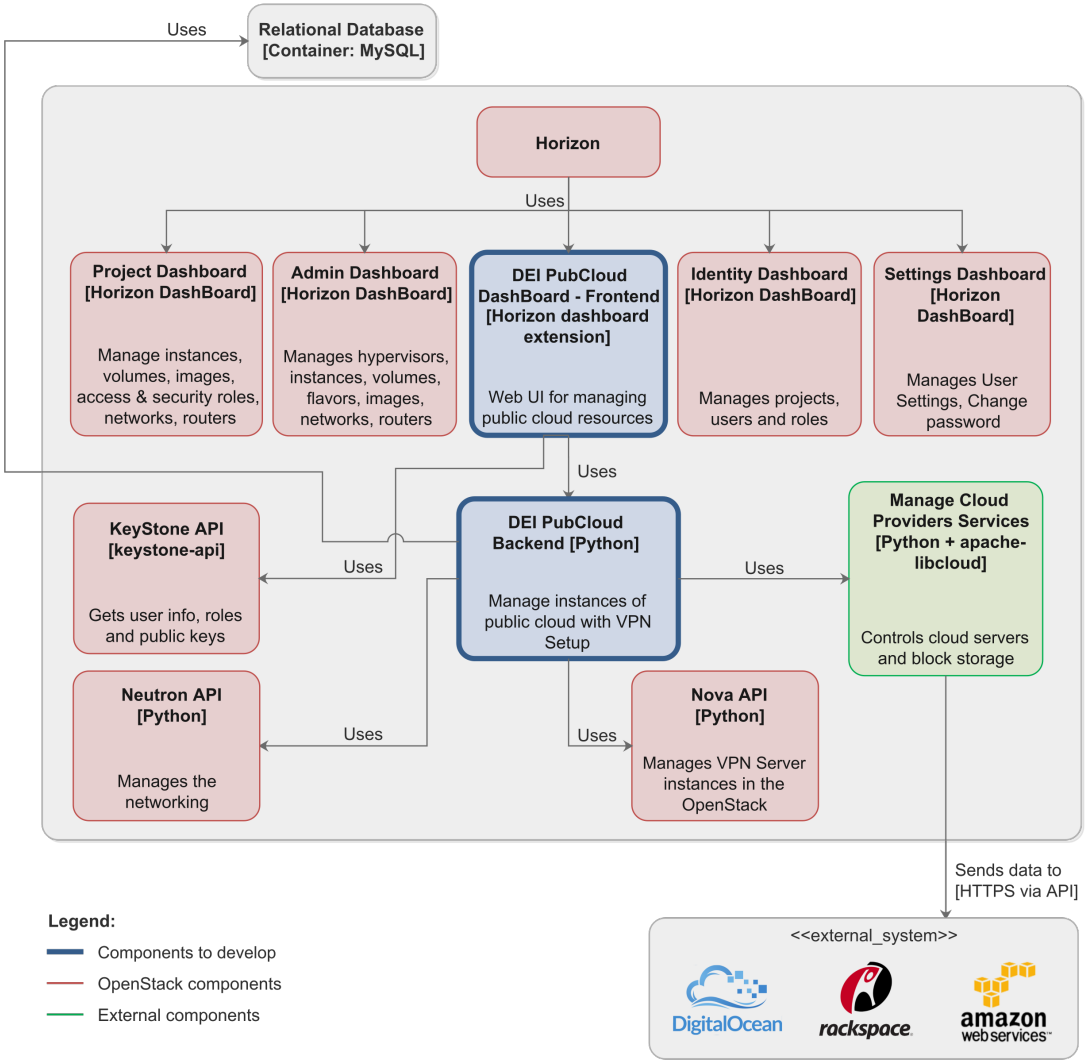


Figura 3.9: Diagrama de componentes do projeto *Horizon*

Foi desenvolvido o componente *DEIPubCloud Dashboard - Frontend*, assim como o **DEIPubCloud Backend** que trata de todo o negócio lógico das interações dos utilizadores com as *clouds* públicas. Este componente utiliza ainda outros componentes, sendo:

- ***Keystone*:** através deste componente é possível identificar se o utilizador tem privilégios para gerir os recursos que se encontram na *cloud* pública, assim como obter a chave pública do utilizador que é exportada nas instâncias criadas;
- **DEIPubCloud *Backend*:** gere toda a parte lógica do módulo, sendo também responsável por interagir com vários *cloud providers*, utilizando uma única API. Desta forma, é utilizada a biblioteca *apache-libcloud*[3] escrita em *python* que interage com os *cloud providers* mais usados, através da sua API única;
- ***Neutron*:** é necessário para efetuar as configurações de rede necessárias no projeto, de forma ao serviço VPN funcionar;
- ***Nova*:** responsável por criar as instâncias com o servidor VPN.

3.10 Conclusão

Nesta secção foi possível identificar quais as limitações da arquitetura atual que levaram à necessidade de um novo sistema. Foi descrita a arquitetura de virtualização atual do DEI que falha no sentido em que os utilizadores estão dependentes dos administradores de sistemas, por forma a poderem obter recursos virtuais, e além disso, não têm possibilidade de criar as suas próprias redes, estando limitados às redes existentes do DEI.

Após o levantamento dos requisitos funcionais e não funcionais foi proposta uma arquitetura que corresponde às necessidades apresentadas. De seguida, foi realizada uma descrição da tecnologia selecionada (*OpenStack*) descrevendo os seus componentes e a forma como se relacionam com a arquitetura proposta (Secção 3.5). A nova arquitetura permite duas estratégias: integração com um ou mais *datacenters* físicos e integração com *cloud providers*.

Na primeira integração, os *datacenters* são interligados através de uma ligação VPN, partilhando todos o mesmo serviço *KeyStone* e o *Horizon* que se encontram no controlador principal. Quanto à segunda integração, é necessário escolher os *providers* que suportam o controlo dos recursos via API, por forma a ser possível o controlo através do componente desenvolvido para o sistema.

Foram ainda abordados os componentes necessários a desenvolver, onde o *DEIPubCloud* é responsável pela lógica da gestão pública da *cloud*, assim como pelas instâncias virtuais que se encontram localizadas no *provider*.

4 Avaliação preliminar

4.1 Introdução

Este capítulo descreve uma prova de conceito da tecnologia selecionada, de modo a verificar se alguns componentes e funcionalidades descritas são possíveis de realizar na arquitetura definida. O componente selecionado foi o *hypervisor Xen*, onde foram efetuados testes para verificar se este cumpre os requisitos definidos, quando integrado com o *OpenStack*. Para além deste componente, foi testada a funcionalidade de ter existir várias regiões com o serviço *KeyStone* e *Horizon* partilhado, por forma a verificar se esta funcionalidade permite alcançar o cenário desejado.

4.2 *Deployment* inicial

De forma a verificar o funcionamento da plataforma selecionada e efetuar uma prova de conceito, recorreu-se ao *software Fuel*[35] que permite fazer *deployment* e gestão de infraestruturas *OpenStack* de uma forma simples, rápida e automática. A Figura 4.1 ilustra a arquitetura do *Fuel*:

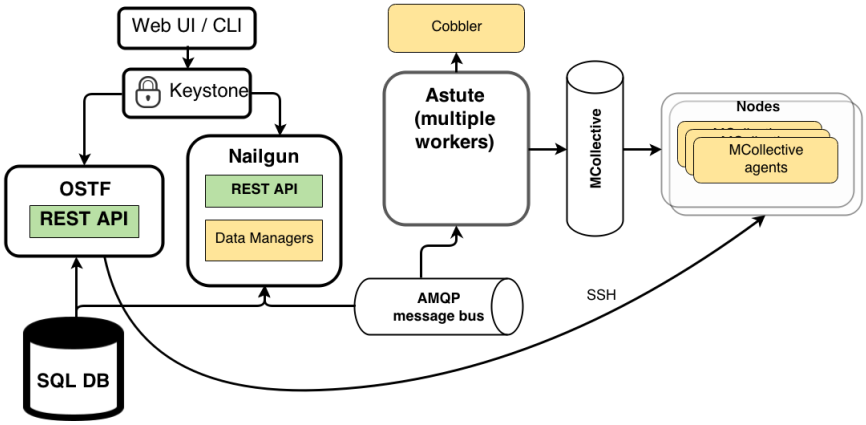


Figura 4.1: Arquitetura *Fuel* [35]

Como podemos verificar na Figura 4.1, esta consiste em vários componentes independentes, onde vários componentes são específicos do *Fuel* e outros de serviços de terceiros (*Puppet*, *Mcollective* entre outros). Os componentes desta arquitetura são[35]:

- ***Web UI*:** Aplicação web, escrita em *Javascript*, que utiliza as *frameworks bootstrap*[7] e *backbone*[5] que permite fazer o *deployment*/gestão das diversas infraestruturas do *OpenStack*.
- **Nailgun:** Componente *core* do *Fuel*, implementa uma API REST (*Representational State Transfer*). Gere a configuração dos volumes dos discos, assim como as configurações de rede e outras propriedades necessárias para fazer o *deployment* de uma infraestrutura *OpenStack*. Utiliza uma lógica de orquestração sequencial para fazer o aprovisionamento na ordem correta. Utiliza ainda uma base de dados por forma a guardar os dados relativos aos *deployments* e o serviço AMQP (*Advanced Message Queuing Protocol*) para interagir com os *workers*.
- **Astute:** representa os *Nailgun workers* e tem a função de correr determinadas ações fornecidas pelo *Nailgun*. É uma camada que encapsula todos os detalhes de interação com uma variedade de serviços, tais como, o *Cobbler*, *Puppet* e *shell scripts*. Fornece ainda uma interface assíncrona para esses serviços. Este componente troca de informações com o *Nailgun* através do AMQP.
- **Cobbler:** É usado como serviço de aprovisionamento.
- **Puppet:** É o único serviço de *deployment* utilizado pelo *Fuel* atualmente.
- **Mcollective agents:** Permite efetuar determinadas tarefas, tais como a formatação de discos e testes à conectividade de rede.
- **OSTF (OpenStack Testing Framework, or Health Check):** É um componente separado do *Fuel*, que é facilmente removido e reutilizado sem o *Fuel*. Efetua verificações do *OpenStack* após o *deployment* estar concluído. O seu principal objetivo é verificar o máximo de funcionalidades que estão a funcionar corretamente.

Após esta análise, construiu-se um *testbed* composto por três máquinas físicas: um servidor com o *Fuel*; um serivdor com o controlador, *storage* e os mecanismos de monitorização *cloud*; e um servidor de computação que contém o *hypervisor Xen*.

4.3 Integração com *Xen*

Ao testar a integração com o *Xen* utilizou-se um *plugin* do *Fuel* "*Citrix XenServer Plugin*"[24], permitindo efetuar o *deployment* de um ou mais servidores *compute* que contenha o *hypervisor Xen*. A criação de uma máquina virtual no *hypervisor* foi responsável pelo controlo do *Xen*, através do *Host Internal Management Network*[18], permitindo comunicar com o *dom0* (*hypervisor*) a partir de um *domU* (máquina virtual criada anteriormente).

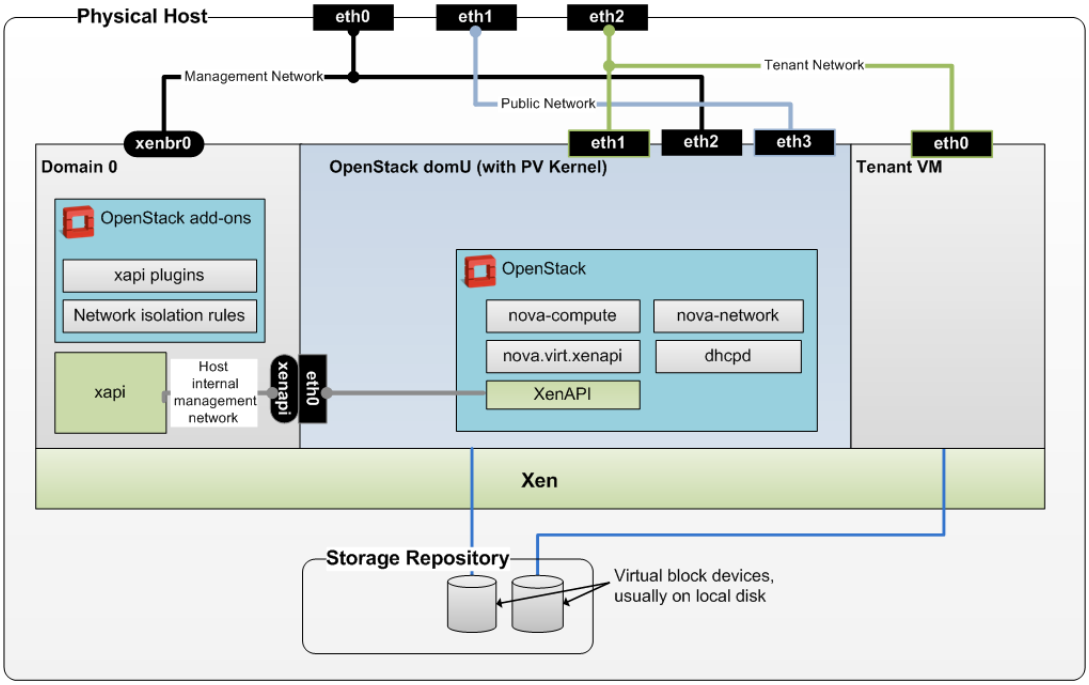


Figura 4.2: Arquitetura do *Xen* no *OpenStack* [37]

No decorrer dos testes das funcionalidades base do *OpenStack*, concluiu-se que o uso do *Xen* apresenta várias limitações face ao pretendido. Uma das limitações é o facto de não suportar o *SR (Storage Repository)* do tipo LVM (*Logical Volume Manager*), que, atualmente, é utilizado em produção[17], apenas permitindo o tipo *EXT3* (*Third Extended Filesystem*). Para além desta limitação, o *Xen* não suporta discos que se encontram no formato *qcow2*, sendo este um formato de imagem *cloud*, fornecido pelo *CentOS*.

Existindo estas limitações e não sendo possível cumprir um dos requisitos funcionais sobre a limitação do número de IOPS que uma instância possa ter, o *hypervisor* que está integrado na arquitetura definida passa a ser o *KVM*, uma vez que suporta todas as funcionalidades pretendidas.[28]

4.4 Integração com várias regiões

A integração com várias regiões foi testada utilizando o *DevStack*[27], permitindo construir infraestruturas *OpenStack* numa instalação de raiz do *Ubuntu* ou *Fedora*. Foram criados dois servidores com 8Gb de RAM com o sistema operativo *Ubuntu*, onde no primeiro servidor foi instalado os serviços base do *OpenStack*, enquanto que no segundo, foram instalados os serviços base à exceção do *Horizon* e do *KeyStone*, pois apenas foi instalado o *KeyStone client* para utilizar o *KeyStone* do primeiro servidor.

Nas Figuras 4.3 e 4.3 é possível verificar que o projeto de demonstração tem quotas diferentes nas duas regiões, enquanto que na segunda região (simulando um *datacenter* num *provider* externo) o projeto não tem qualquer possibilidade de criar recursos.

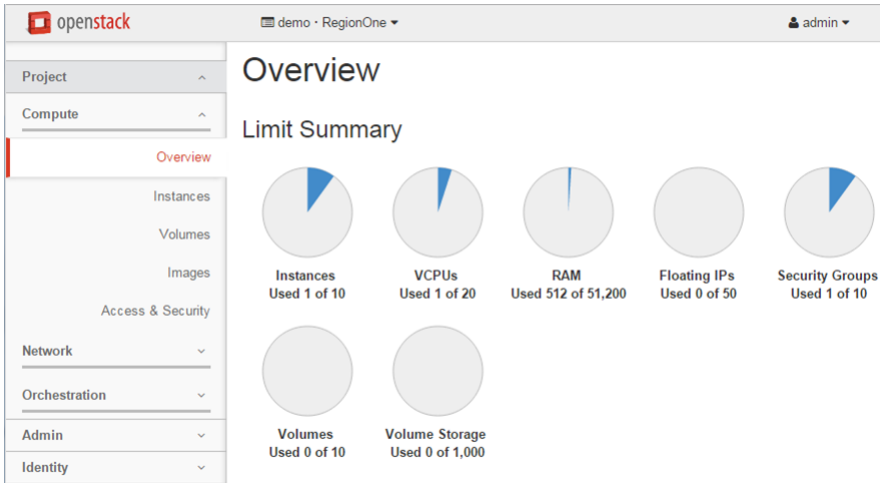


Figura 4.3: *Dashboard* da primeira região

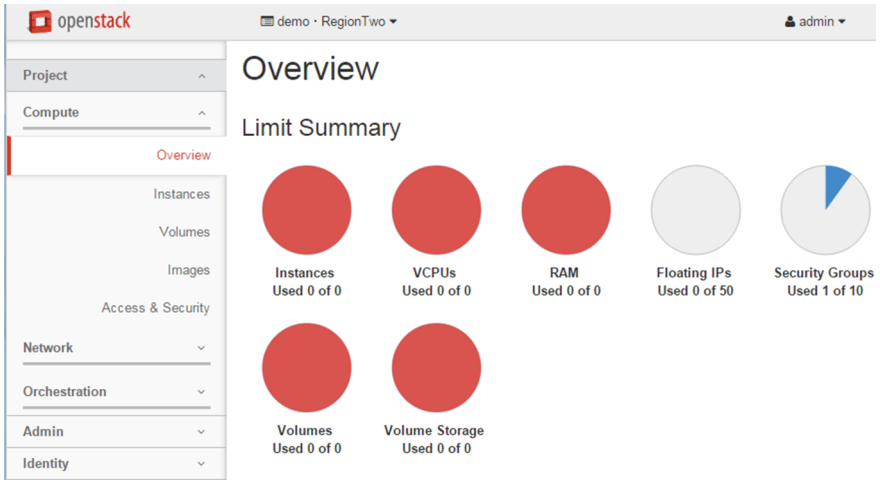


Figura 4.4: *Dashboard* da segunda região

4.5 Conclusão

Após a análise realizada foi possível verificar que o *Xen*, atualmente, não se adequa para a arquitetura definida devido as suas limitações. Tal como descrito anteriormente, a utilização deste *hypervisor* pelo *OpenStack* apresenta limitações ao nível dos *filesystems* suportados e da gestão de informação de armazenamento. Por estas razões, optou-se por utilizar o *hypervisor KVM*, tratando-se de um *hypervisor* mais robusto, testado e excelente suporte total no *OpenStack Nova* [29].

O teste de integração das várias regiões permitiu verificar que é possível utilizar um serviço *KeyStone* partilhado, assim como definir quotas de recursos específicos para cada região.

5 Desenvolvimento e integração

5.1 Introdução

Neste capítulo irá ser descrito todo o processo de desenvolvimento e integração efetuado durante o estágio. Primeiramente passou-se à instalação de todo o equipamento necessário, desde a instalação e configuração dos servidores, equipamentos de rede, assim como a alteração das redes existentes no DEI. De seguida foi realizado o *deployment* do *OpenStack* utilizando o projeto *openstack-ansible*, tendo sido um processo continuo desde o inicio até ao final do projeto. Com o *OpenStack* a funcionar, foram realizados diversos *mockups*, tendo servido de base para o desenvolvimento do módulo. Seguidamente foi efetuado o desenho da base-de-dados, assim como a construção do *frontend* e o *backend*. Para finalizar este capítulo é abordado o *workflow* das tarefas mais complexas deste módulo.

5.2 *Deployment* do *OpenStack*

Numa primeira abordagem do estágio procedeu-se à instalação do equipamento no *datacenter* servindo de suporte à plataforma *cloud* (*OpenStack*) utilizada no projeto. O cenário do *deployment* é composto por cinco servidores (*deployment node*, *controller node*, *network node*, *compute node* e *storage node*) e um *switch* de 1Gbps de ligação. A Figura 5.1 representa os diversos serviços que cada *node* suporta, assim como realça os componentes que foram desenvolvidos e modificados no decorrer do estágio. Os *nodes* asseguram os seguintes papeis:

- ***Deployment node*** - Responsável por todo o *deployment* infraestrutural do *OpenStack*, assim como dos servidores/clientes VPN que efetuam a interligação das instâncias da *cloud* pública com o *datacenter* do DEI.
- ***Controller node*** - Responsável por correr todos os serviços infraestruturais do *OpenStack*, assim como os serviços que são externos ao *OpenStack*, tais como o galera (*cluster MariaDB*), *memcached* e o *rsyslog* (centralização de *logs*).
- ***Compute node*** - Responsável por correr todas as máquinas virtuais através do *hypervisor KVM*.
- ***Storage node*** - Responsável por disponibilizar os volumes para armazenamento de dados aos *compute nodes*.
- ***Network node*** - Responsável por fornecer todos os serviços de *networking* (SDN) aos projetos do *OpenStack*.

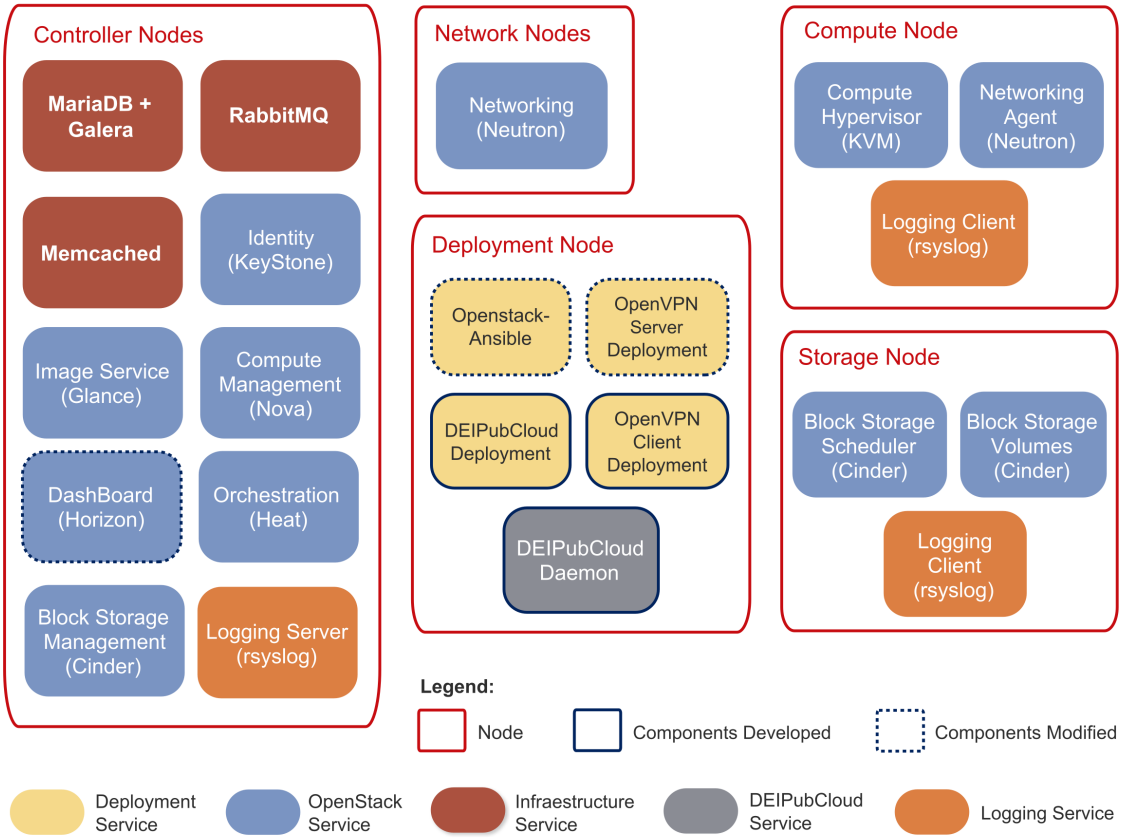


Figura 5.1: Visão geral dos serviços *OpenStack* contidos nos *nodes*

Para a escolha da plataforma de *configuration management* a utilizar, foram analisadas as três maiores plataformas que dominam atualmente o mercado: o *Puppet*, o *Chef* e o *Ansible*. Apesar de serem muito semelhantes e existirem projetos do *OpenStack* nestas plataformas, para efetuar o *deployment*, foram utilizados dois critérios que levaram à escolha do *Ansible*: i) o facto do *openstack-ansible*[32] ser muito ativo, possuir excelente documentação, assim como apresentar um grande suporte por parte da comunidade e ii) ser *agent-less* (apenas é necessário que o servidor tenha o serviço *SSH* com o *python* instalado, não sendo necessário efetuar configurações adicionais. Desta forma, só é necessário manter atualizado o serviço *SSH*, diminuindo os números de serviços desnecessários a correr no servidor).

Após finalizada a instalação dos equipamentos e verificado os serviços que cada *node* iria ter, procedeu-se à configuração do *switch*, onde foram definidas as vLANS utilizadas pelos *nodes*, assim como a passagem das vLANS do DEI para este *switch*. A plataforma *cloud* passou a ter acesso à rede do DEI, como por exemplo, à rede do CISUC (*Centre for Informatics and Systems*).

Finalizando a configuração do *switch*, procedeu-se à instalação e configuração do sistema operativo, *Ubuntu* 14.04, em todos os servidores utilizados neste ambiente. A escolha do *Ubuntu* deveu-se ao facto de ser o sistema operativo mais suportado pelo projeto *openstack-ansible*. Por fim, com o *datacenter* devidamente instalado e todos os recursos

configurados (servidores e *switches*), procedeu-se à configuração dos *nodes*, sendo possível passar a dispor da infraestrutura necessária ao suporte da arquitetura de virtualização, tal como ilustra a Figura 5.2.

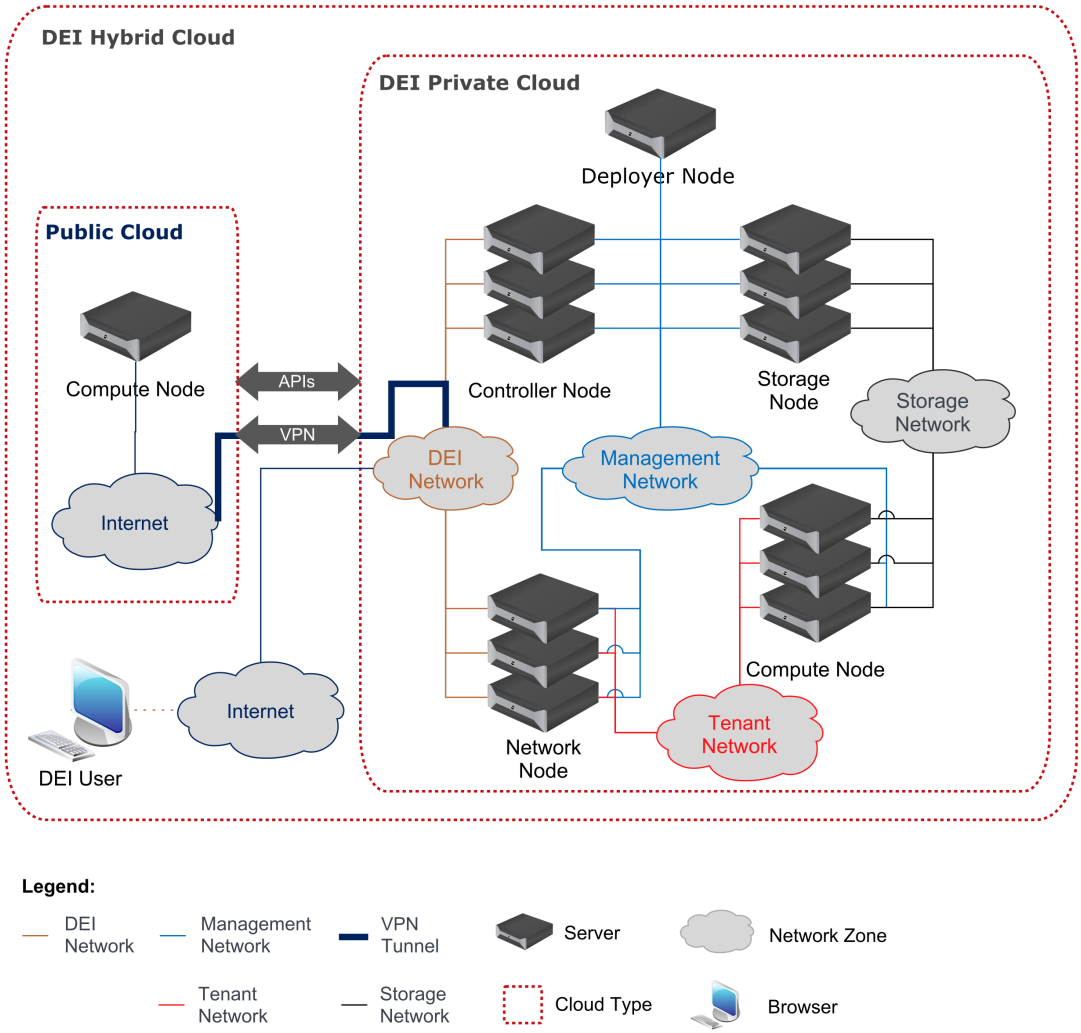


Figura 5.2: Visão de alto nível da arquitetura desenvolvida

A versão do *OpenStack deployed* foi, inicialmente, a *Liberty* (lançada em outubro de 2015), tendo sido efetuado *upgrade* para a última versão *Mitaka* (lançada em abril de 2016). Uma das razões que obrigou a este *update* no decorrer do estágio foi o facto do projeto *openstack-ansible* ter iniciado o suporte ao serviço VPNaaS a partir desta última versão. O VPNaaS é um serviço necessário para o módulo desenvolvido, sendo responsável por garantir a integração das instâncias remotas na rede do DEI, desta forma, é possível comunicar com a rede interna do DEI através da *cloud* pública e vice-versa. Por forma a ser possível a utilização da funcionalidade VPNaaS, foi necessário recorrer às funcionalidades *deployment* do projeto *openstack-ansible*. No entanto, como este projeto, à data da realização do trabalho, não dispunha desta funcionalidade completamente desenvolvida, foi necessário recorrer a várias alterações ao projeto, tal como o Anexo A detalha. Na

realidade, estas alterações foram introduzidas de forma continua ao longo do trabalho de desenvolvimento, à medida que foram detetados problemas no suporte ao *deployment* de componentes do *OpenStack*. Estas alterações foram aceites no projeto *openstack-ansible*, passando a fazer parte da sua versão atual e futuras versões.

5.3 DEIPubCloud *Module*

O DEIPubCloud é um módulo que foi desenvolvido durante o estágio, permitindo estender a *cloud* privada do DEI para a *cloud* pública. Este módulo é dividido em dois componentes: i) *frontend*, extensão para o projeto *Horizon*, que permite aos utilizadores do DEI criarem as suas instâncias na *cloud* pública; ii) *backend*, responsável pela lógica necessária para efetuar a criação das instâncias, assim como a configuração dos servidores VPNs, que irá permitir às instâncias da *cloud* comunicarem com o *datacenter* do DEI.

Inicialmente, realizaram-se protótipos de baixo nível entre os *stakeholders* do projeto (ver Anexo B), por forma a obter-se uma melhor compreensão do funcionamento e da interação com o sistema em desenvolvimento. Estes protótipos permitiram verificar o *workflow* do módulo, sendo possível efetuar alterações na esquematização, com o objetivo de melhorar o desenvolvimento dos seus artefactos.

5.3.1 Base de dados

Após a construção de todos os ecrãs da aplicação, procedeu-se ao desenvolvimento do modelo entidade-relacionamento da base de dados, apresentado na Figura 5.3, com a descrição na Tabela 5.1. Esta base de dados, a ser suportada pelo componente *MariaDB+Galera* (ver Figura 5.1), serve o propósito de alojar toda a informação relativa a projetos, *providers*, planos e respetivas alterações de preços, imagens disponíveis em cada *provider* e informação relativa a quotas de recursos.

Na construção da base de dados foram tidos em conta alguns princípios fundamentais necessários para o módulo. Desta forma, no desenho da base de dados, garantiram-se as seguintes propriedades:

- Cada projeto pode ter:
  - Um servidor VPN por cada *provider*;
  - Um ou mais planos de um ou mais *providers*;
  - Uma ou mais instâncias.
- Guardar o histórico de preços de cada plano dos *providers*;
- Guardar o custo total de cada instância desde a data de criação até à data atual ou final;
- Definição de um limite de custos global de cada *provider*;
- Possibilidade de existir *providers* iguais, desde que sejam de regiões diferentes;

- Delimitar a quantidade de planos de um dado *provider* que um projeto possa ter;
- Guardar as imagens que cada *provider* disponibiliza.

Tabela	Descrição
images	Regista todas as imagens (sistemas operativos) que são disponibilizadas pelos *providers* externos da *cloud* pública.
plans	Guarda todos os planos existentes (por exemplo: nome, memória RAM e espaço em disco) de cada *provider*.
plans_price	É responsável por guardar todos os preços daquele plano, permitindo ter não só um histórico das alterações de preço, assim como, obter um cálculo mais preciso dos gastos de cada máquina virtual desde a sua criação.
provider_driver	É responsável por definir quais são os *providers* suportados pelo módulo desenvolvido.
providers	Guarda a informação de todos os *providers* que podem ser utilizados no módulo. **Nota:** podem existir vários *providers* iguais, desde que sejam de diferentes regiões.
projects	Guarda os projetos que podem utilizar este módulo. O ID do projeto é igual ao ID que se encontra no *OpenStack*.
projects_plans	Guarda a quantidade de planos que um projeto pode criar.
virtual_machines	Guarda a informação das máquinas virtuais criadas na plataforma *cloud* selecionada, incluindo o custo total desde a sua criação.
VPN	Guarda a informação das máquinas virtuais criadas no *OpenStack* que servem como VPN *server*.

Tabela 5.1: Descrição das tabelas do modelo de base de dados

Na terminologia do *OpenStack*, um projeto representa a unidade base de propriedade de um utilizador do sistema, sendo que todos os recursos disponíveis do *OpenStack* devem ser propriedade de um projeto existente[34]. Desta forma, consideramos um projeto como um *datacenter* virtual composto por um ou mais utilizadores. Da mesma forma que um plano assume as características de cada instância, como por exemplo, existir um plano de 512MB de RAM com 20GB de espaço em disco e outro de 1GB de RAM com 30GB de espaço em disco.

Com a base de dados definida, seguiu-se para o desenvolvimento da camada lógica aplicacional, onde o desenvolvimento do componente foi realizado em *python*, por esta ser a linguagem base do *OpenStack*.

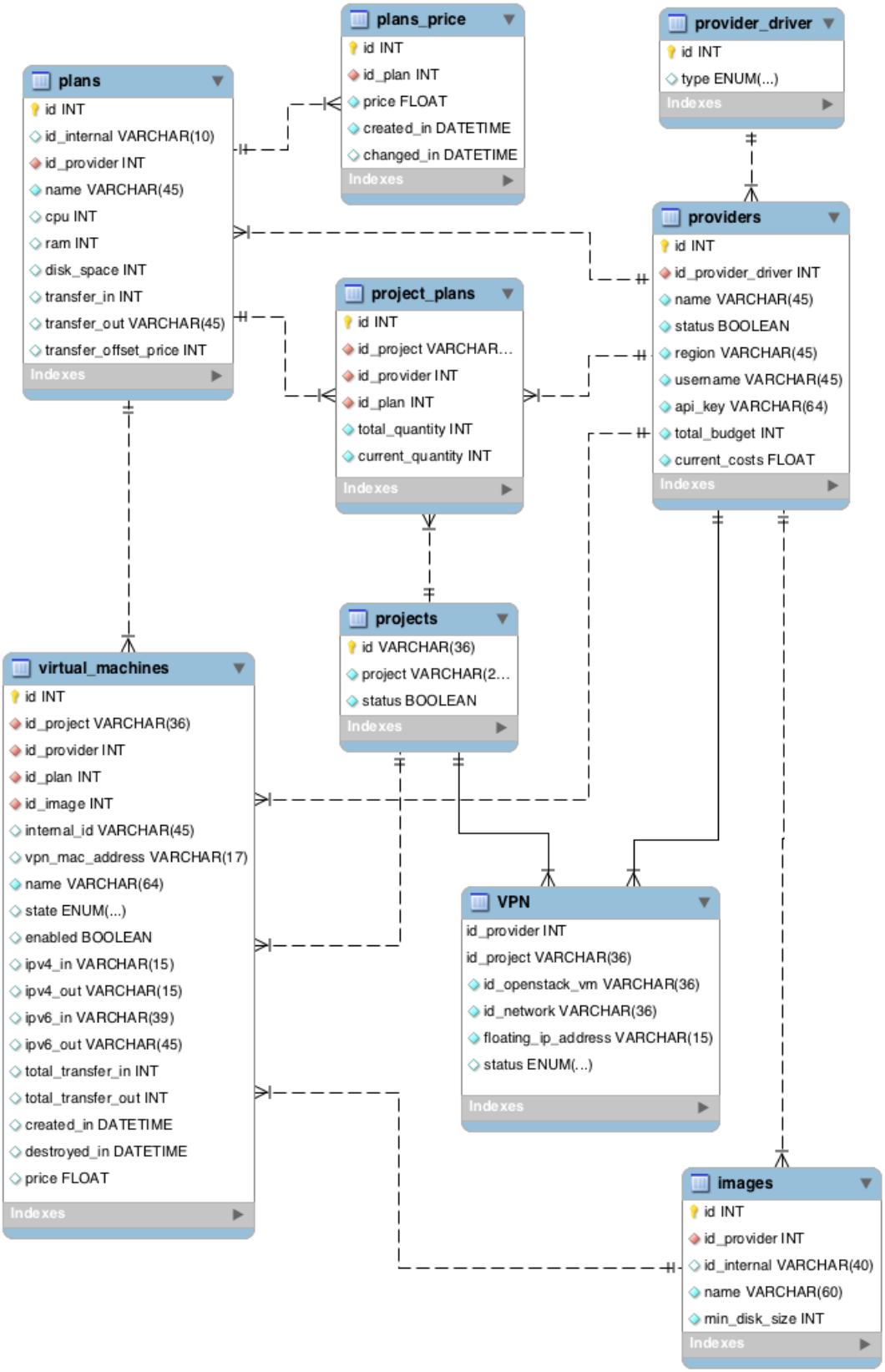


Figura 5.3: Base de dados do *software DEIPubCloud backend*

5.3.2 *DEIPubCloud backend*

Após a verificação dos requisitos necessários para o desenvolvimento do módulo, chegou-se à conclusão da necessidade de um ORM (*Object-relational mapping*) por forma a permitir que o acesso à base de dados seja mais abstrato. Desta forma, é possível alterar o motor de base de dados e a aplicação continua a funcionar sem a necessidade de efetuar alterações, devido ao ORM ser responsável pela escrita SQL (*Structured Query Language*), específica de cada motor de base de dados. O ORM escolhido foi o *SQLAlchemy* por este ser utilizado em vários serviços do *OpenStack*. Além disto, definiu-se o nível de isolamento da base de dados para o tipo *serializable*. Desta forma, garantiu-se que caso exista colisão ao alterar o mesmo registo, por transações diferentes, a base de dados fique num estado consistente.

A Figura 5.4 apresenta a interação que os componentes do módulo desenvolvido assumem perante as bibliotecas externas, assim como a interação que estas fazem com os componentes externos. Como é possível verificar na Figura 5.4, o módulo é executado em dois servidores distintos: *Horizon* e *Deployer*. No servidor *Horizon* o *Scheduler* é utilizado para enviar pedidos assíncronos através do *Celery*. Enquanto que no servidor *Deployer*, corre o *DEIPubCloud Celery Daemon*, sendo responsável por executar os pedidos assíncronos recebidos. O serviço *DEIPubCloud Celery Daemon* ao ser iniciado, permite que sejam criados processos em número adequado à configuração de *hardware* do servidor. No caso particular deste *deployment*, são criados oito processos, permitindo executar até oito tarefas em simultâneo, minimizando desta forma o tempo de espera dos utilizadores.

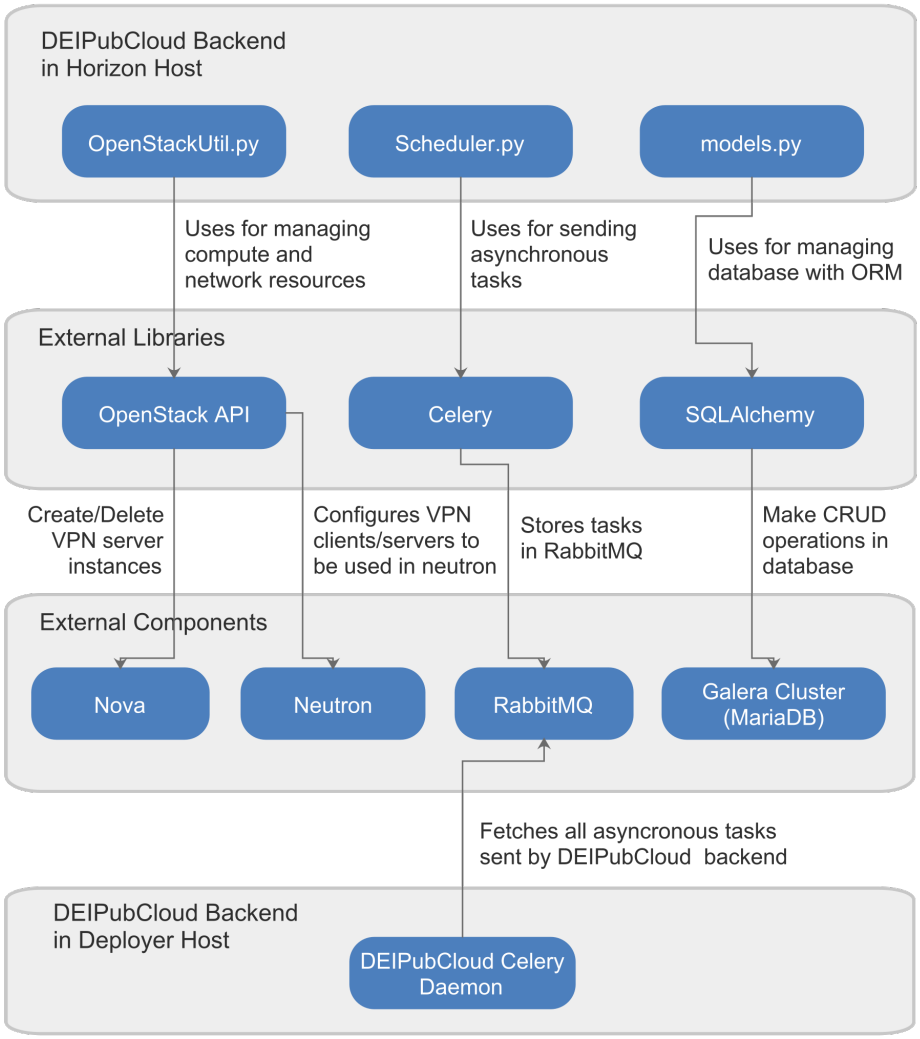


Figura 5.4: Interação do módulo com os componentes externos

Relativamente às operações realizadas de forma assíncrona, foi escolhido um módulo que permitisse ter uma *distributed task queue*, denominado por *Celery*. Para que o *Celery* funcione é necessário utilizar um *Message Broker*, uma vez que permite o envio e receção de mensagens. O *Message Broker* selecionado foi o *RabbitMQ* devido a ser utilizado pelos serviços *core* do *OpenStack*. O *Celery* foi configurado para utilizar três servidores *RabbitMQ*, recorrendo à estratégia de *failover round-robin*. Desta forma, caso o servidor *RabbitMQ* ao qual ele esteja ligado fique indisponível, é utilizado o próximo servidor disponível que estiver na lista de servidores.

Para a interação com os *providers* da *cloud* pública, o módulo escolhido foi o *apache-libcloud*, uma vez que a biblioteca possui uma documentação muito detalhada, tendo tido um grande desenvolvimento pela parte da comunidade, assim como um grande suporte dos principais *providers* existentes.

Inicialmente para efetuar a interligação dos *datacenters*, era para ser utilizado o VPNaaS fornecido pelo *OpenStack*, mas após a configuração do serviço, assim como ter-se efetuado alguns testes, não foi possível utilizar essa funcionalidade, porque o VPNaaS só suporta ligações *site-to-site* e não *road warriors*, desta forma apenas seria possível ligar um cliente ao serviço VPN configurado. Dado esta limitação ao nível da plataforma, foi necessário alterar a estratégia que passou pelo desenvolvimento de um VPNaaS próprio e independente do já existente do *OpenStack*. Este novo serviço, permite ter um servidor VPN por cada projeto, que suportasse uma ou mais instâncias que fossem criadas na *cloud* pública.

De forma a efetuar o *deployment* dos servidores/clientes VPN e do módulo DEIPubCloud foi utilizado o *Ansible*. No entanto, para os servidores/clientes VPN o *software* instalado é o *OpenVPN*. Os ficheiros de configuração do *ansible*, denominados por *playbooks*, definem uma lista de tarefas (*plays* na linguagem *Ansible*) que cada servidor necessita efetuar por forma a ficar no estado desejado. Os *playbooks* são escritos na linguagem YAML (*Yet Another Markup Language*), sendo possível incluir outros *playbooks*. É ainda possível alterar as variáveis por forma a que o mesmo *playbook* possa correr diferentes cenários em diferentes servidores. Neste caso foram desenvolvidos três *playbooks*:

1. ***DEIPubCloud Deployment***: Permite efetuar o *deployment* automático do módulo *DEIPubCloud* desenvolvido para a infraestrutura do *OpenStack*, alterando em conformidade a configuração dos servidores *Horizon*, assim como do *deployment node*. Para além de atualizar o módulo nos serviços, adiciona ainda o *branding* do DEI UC na plataforma *Horizon*. Este *playbook* é executado sempre que forem efetuadas alterações tanto no módulo como no *playbook*;

2. ***OpenVPN Server Deployment***: Responsável pelo *deployment* do servidor *OpenVPN*, baseado num projeto *open-source* existente [21], tendo sido alterado para funcionar no sistema operativo *CentOS* e adequado para a realidade do DEI. Cria o servidor *OpenVPN* configurando um certificado único de forma a garantir que apenas os utilizadores do projeto possam aceder ao seu servidor VPN. Este *playbook* é usado quando o utilizador cria um servidor VPN;

3. ***OpenVPN Client Deployment***: Responsável por efetuar o *deployment* do cliente *OpenVPN*. Obtém os ficheiros de configuração do cliente *OpenVPN* no servidor *OpenVPN*, enviando-os para a instância criada na *cloud* pública. Este *playbook* é utilizado para configurar as instâncias criadas na *cloud* pública.

Após escolhidos os componentes do sistema, foi desenvolvido um diagrama de classes que permitiu visualizar a interação entre as classes do *DEIPubCloud backend* (ver Figura 5.5). O *backend* é responsável pela lógica do módulo, onde é efetuada a gestão dos recursos na *cloud* pública, sendo este utilizado pelo *frontend*, de forma a realizar as operações submetidas pelos utilizadores.

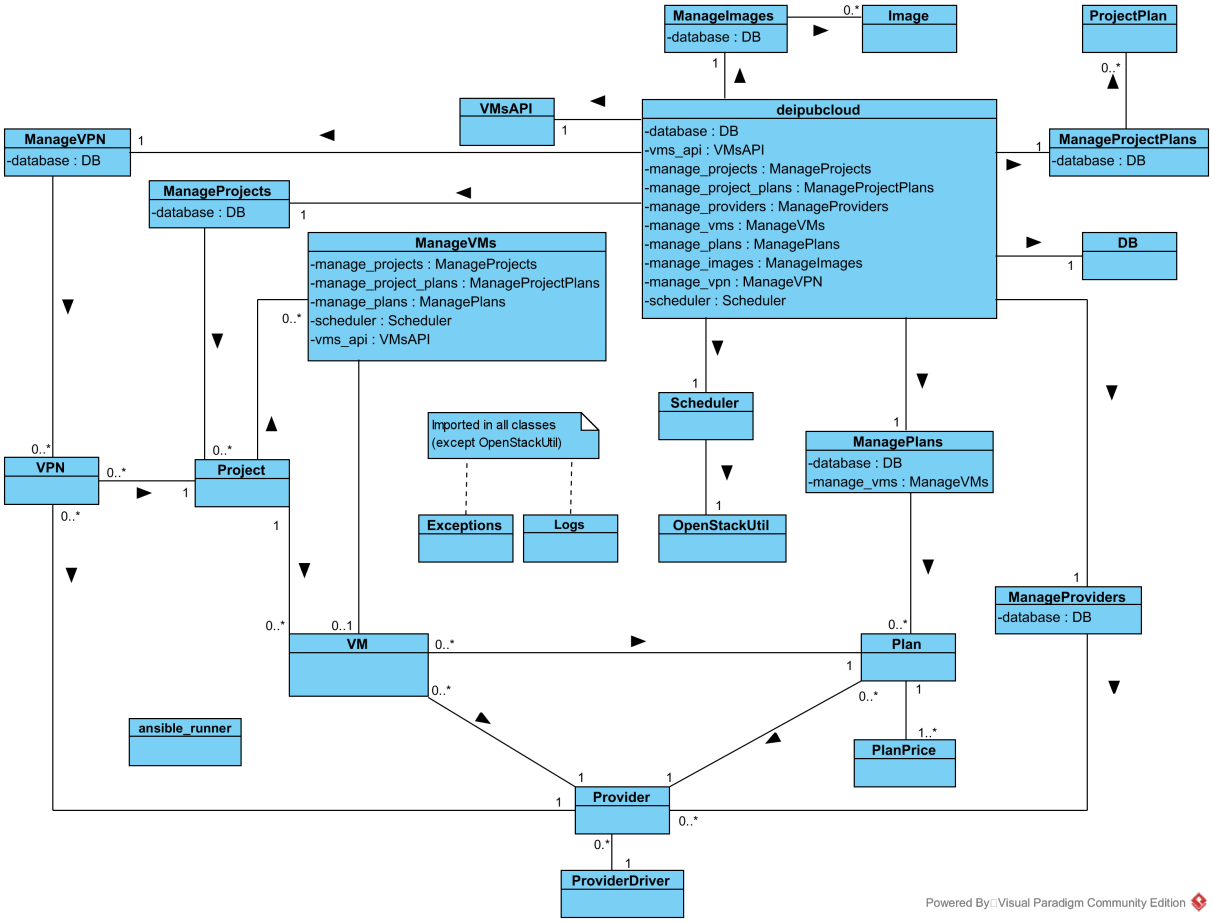


Figura 5.5: Diagrama de classes do DEIPubCloud (diagrama completo no Anexo F)

A Tabela 5.2 mostra a descrição das classes apresentadas na Figura 5.5, assim como as relações existentes entre elas.

Classe	Descrição
deipubcloud	Constrói todas as classes e define as suas relações através da injeção de dependências.
exceptions	Classe que tem as exceções que a aplicação pode gerar.
ManageImages	Efetua a gestão das imagens, ou seja, dos sistemas operativos suportados pelo *provider*.
ManagePlans	Efetua a gestão dos planos e alterações de preço ocorridos nos mesmo.
ManageProviders	Efetua a gestão dos *providers* e trata da importação dos planos e imagens do *provider* através das suas *API*, utilizando o *apache-libcloud*.
ManageProjectPlans	Efetua a gestão de planos dos projetos.
ManageProjects	Efetua a gestão dos projetos.
ManageVMs	Efetua a gestão das máquinas virtuais, incluindo as interações com os *providers*. É igualmente responsável por fazer o aprovisionamento automático dos servidores e clientes VPN.
ManageVPN	Efetua a gestão e *deployment* dos servidores *VPNs*.
models	Contém todas as classes que são mapeadas com as tabelas da base de dados, tais como a *VPN*, *Project*, *VM*, *Provider*, *ProviderDriver*, *Plan*, *PlanPrice*, *Image* e *ProjectPlan*.
Scheduler	Contém as funções que são executadas de forma assíncrona.
DB	É responsável pela gestão da ligação à base de dados, a filtragem dos dados (aceitando dados válidos na classe), assim como efetuar diversas operações como, por exemplo, *joins*.
VMsAPI	É responsável por obter o *driver* de um dado *provider*.
OpenStackUtil	É responsável por executar tarefas nos serviços *OpenStack*, utilizando as respetivas *APIs*.
Logs	É responsável pela identificação do utilizador que realizou determinadas operações num projeto. Regista ainda erros que possam ocorrer no módulo, de forma a facilitar o processo de *debug*. Existem quatro ficheiros de *logs*: i) *operations.log*: todas as operações são executadas pelos utilizadores; ii) *errors.log*: todos os erros que ocorrem no módulo, ao executar determinada tarefa; iii) *debug.log*: *traceback* dos erros ocorridos e iv) *ansible.log*: todos os erros relacionados com o *deployment* do *Ansible*.
ansible_runner	Responsável por correr os *playbooks* do *Ansible*.

Tabela 5.2: Descrição das classes da camada lógica da aplicação

5.3.3 *DEIPubCloud frontend*

Como referido anteriormente, o *frontend* do componente DEIPubCloud foi desenvolvido para a plataforma *Horizon* do *OpenStack*. O *frontend* recorre às *roles* do projeto, *KeyStone*, garantindo que só os utilizadores que pertençam ao grupo *deipubcloud* possam aceder ao módulo desenvolvido. Este módulo é composto por dois grupos de *panels*:

- ***Compute*** - acessível a todos os utilizadores que tenham o estejam autorizados a utilizar o *panel Compute*. É composto pelos seguintes *panels*:
  - **VM *Management*** - o utilizador tem a visibilidade global de todas as suas máquinas virtuais que estão instanciadas na *cloud* pública. Permite ainda geri-las, podendo remover ou adicionar novas instâncias, assim como visualizar o custo que as instâncias têm desde que a data de criação;
  - **VPN *Management*** - permite ao utilizador criar um servidor VPN local, com o objetivo de expandir a rede do DEI na *cloud* pública. Esta rede é escolhida pelo utilizador, podendo ser uma rede *privada* ou existente do DEI (por exemplo, rede CISUC).

- ***Admin*** - é acessível a todos os administradores da plataforma *OpenStack*, sendo composto pelos seguintes *panels*:
  - ***Overview*** - permite verificar todas as máquinas virtuais em atividade, as despesas gastas desde a primeira instância criada e obter uma expectativa total de custos mensais (assumindo que uma instância criada é mantida durante um mês);
  - ***Projects Management*** - gestão dos planos que um projeto tem acesso;
  - ***Providers Management*** - gestão dos *providers* das *clouds* públicas, definindo o limite máximo de custos por *provider*. É ainda possível sincronizar os *plans* e *images* diretamente dos *providers*, via API;
  - ***Plans*** - gestão dos planos existentes de cada *provider*;
  - ***Images*** - gestão das imagens existentes de cada *provider*.

No *frontend* também foram desenvolvidos testes funcionais (ver Anexo D), sendo possível verificar se alguma funcionalidade passou a ter algum comportamento inesperado, no caso de ser efetuada alguma alteração, tanto no *frontend* como no *backend*.

5.3.4 *Workflow* aplicacional

Esta secção descreve as ações mais complexas do *backend* do módulo desenvolvido, especificando os passos, por forma a realizar determinada tarefa autonomamente. Foram identificadas duas ações complexas no *backend*: a criação de instâncias na *cloud* pública e a criação do servidor VPN local com *deployment* automático do serviço *OpenVPN*.

Criação de instâncias na *cloud* pública

Na realização da criação de instâncias na *cloud* pública, foi necessário recorrer ao *Neutron* por forma a criar uma porta virtual na rede estendida (por exemplo, a rede CISUC), guardando o *mac-address* e endereço IP gerado. Considera-se uma porta virtual como uma porta de um *switch* protegida via *mac-address*. Desta forma, garante-se que os pedidos DHCP do cliente VPN não são ignorados pelo *Neutron*, uma vez que, por defeito, bloqueia os pacotes de rede com endereços *mac-address* desconhecidos.

De modo a permitir a gestão de recursos criados no *provider* da *cloud* pública, foi necessário gerar uma *API Key* no *provider* externo selecionado. Desta forma, foi possível controlar os recursos através da biblioteca *apache-libcloud*. Quando a instância é criada no *provider*, é necessário esperar que a mesma efetue o *boot* completo, assim como obter o seu endereço IP público. Por sua vez, são enviados pedidos para verificar o estado da instância, de 3 em 3 segundos, num período máximo de 600 segundos.

Para finalizar o processo de criação, através do endereço IP público obtido, acedeu-se à instância realizando as seguintes operações: i) instalação do cliente *OpenVPN*; ii) transferência das configurações do servidor VPN para o cliente VPN; iii) configuração do serviço

DHCP para não substituir o *default gateway*; iv) definir o *mac-address* (obtido no *Neutron*) na interface TAP do serviço *OpenVPN* e v) configuração das rotas para que o tráfego dirigido à rede do DEI seja entregue no *router* do *Neutron*.

Após efetuada a preparação do sistema, procedeu-se à automatização dos processos validados anteriormente, efetuando os seguintes passos (ver Figura 5.6 e 5.7):

1. O utilizador através do *Horizon*, acede ao *panel VM Management* da *DashBoard* do módulo DEIPubCloud. Ao criar uma instância, deve-se preencher o formulário de criação, especificando o *provider*, o plano, a imagem desejada (por exemplo, CentOS), o nome, e a chave pública (guardadas no serviço *KeyStone*). A chave pública é importada na instância, permitindo ao utilizador efetuar a autenticação no serviço SSH, utilizando a sua chave privada.

2. O servidor recebe os dados enviados pelo utilizador, executando as seguintes tarefas: i) envia um pedido REST ao serviço *Neutron*, por forma a criar uma porta virtual na mesma rede que o servidor VPN; ii) com a porta criada, tanto o endereço IP como o endereço *mac-address* são guardados na base de dados; iii) é verificado se o utilizador pode criar mais instâncias, verificando a sua quota; iv) é verificado se o *budget* total definido do *provider* não é ultrapassado.
   Caso não exista nenhuma restrição é enviado um pedido assíncrono para criar a instância no *provider* escolhido;

3. O serviço *RabbitMQ* recebe a *task* no *virtual host deipubcloud* com os argumentos enviados, neste caso, o ID da instância (ID gerado automaticamente na base de dados) e a chave pública.

4. O *celery daemon* consome a *task* que se encontra no *virtual host deipubcloud*.

5. A *task* de criação de uma instância obtém a informação da instância que foi guardada na base de dados, de maneira a obter o ID do plano, a imagem e a localização. Um dos problemas do *provider cloud* utilizado é o facto de não permitir a injeção de chaves públicas diretamente, sendo necessário importar as mesmas para o *website*. Desta maneira, é utilizado o módulo *apache-libcloud* para tratar da criação da instância. Seguidamente é verificado se a chave pública selecionada pelo utilizador se encontra no *provider*, utilizando o ID da chave do site. Caso não exista, a mesma será importada. Posteriormente é enviado um pedido de criação da instância, com a respetiva chave pública selecionada pelo utilizador. Nesse momento aguarda-se que a instância tenha efetuado o *boot* completo, num período máximo de 600 segundos. Caso a instância seja criada com sucesso (*boot* completo), o estado da instância passa a *provisioning*, caso contrário, fica no estado *failed*, acabando o *workflow*. Por fim, é executado o *playbook* responsável pelo *deployment* do cliente VPN. Este *playbook* executa as seguintes operações:

   - **No servidor VPN:** efetua *download* das configurações do cliente VPN para o *deployment node*, utilizando uma pasta temporária única (com o seu ID). Desta forma, garante-se que o método possa ser executado concorrentemente.
   - **No cliente VPN:**

- Instalação do cliente *OpenVPN*;
- Transferência dos ficheiros de configuração do *OpenVPN* do *deployment node* para o cliente VPN;
- Configuração das rotas por forma ao tráfego dirigido à rede do DEI seja entregue no *router* do DEI;
- Configuração da interface TAP, definindo o *mac-address* obtido no *Neutron*;
- Configuração do serviço *DHCPD* por forma a impedir que este substitua o *default gateway* definido pelo *router* do *Neutron*.

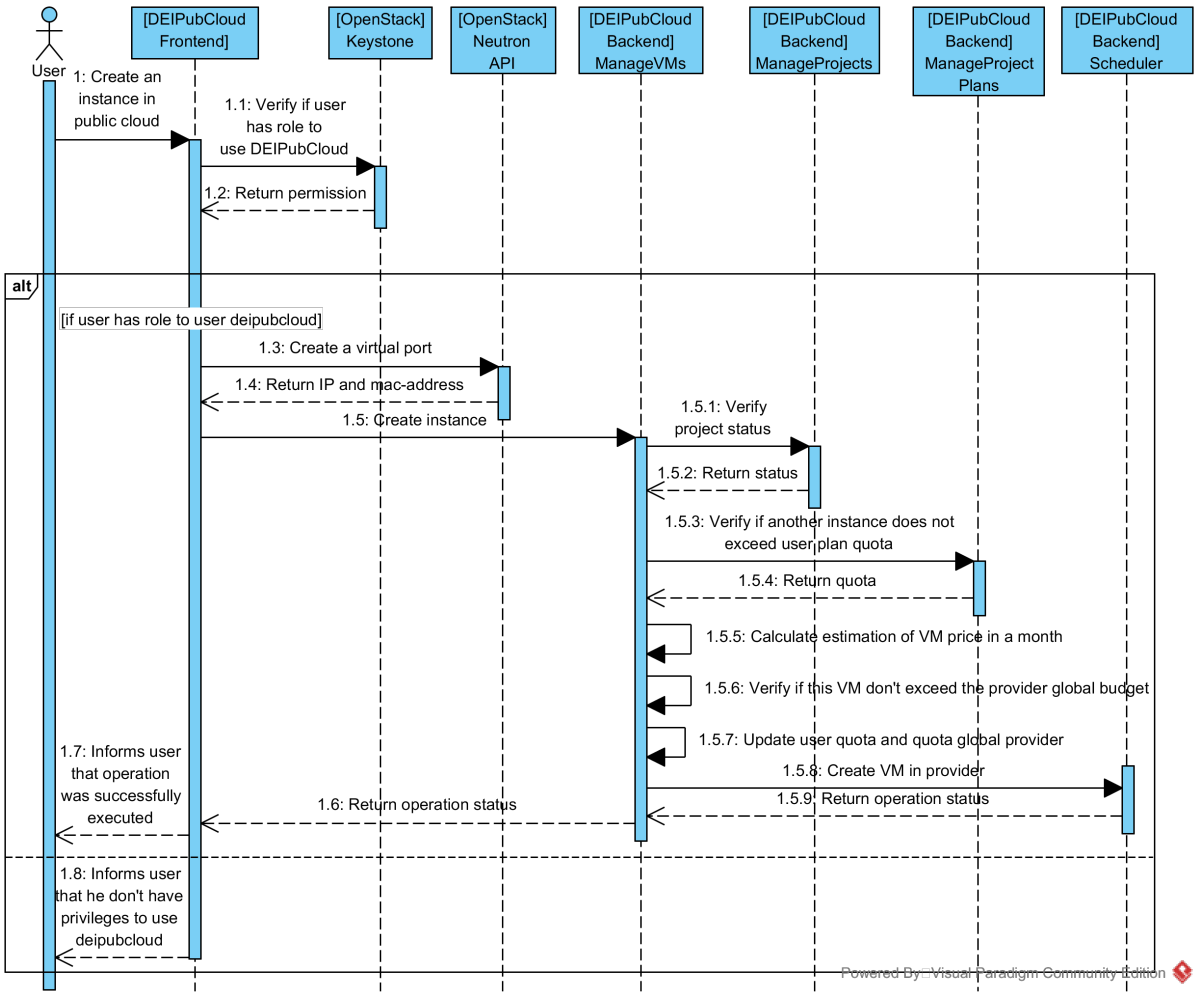


Figura 5.6: *Workflow* da criação de uma instância no lado do *frontend*

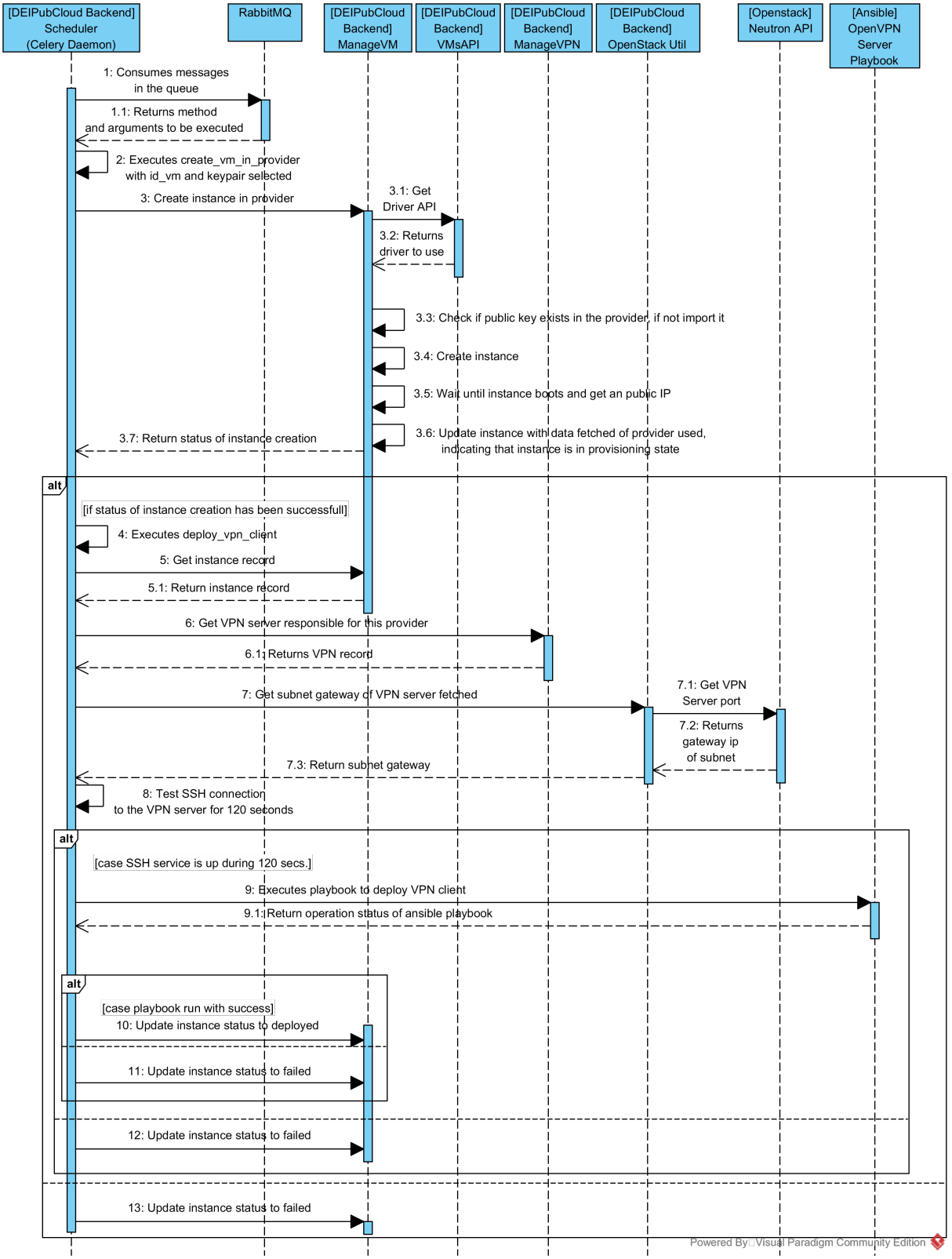


Figura 5.7: *Workflow* da criação de uma instância no lado do *backend*

Criação do servidor VPN local com *deployment* automático do serviço *OpenVPN*

O módulo desenvolvido é responsável por permitir a integração da *cloud* privada com a *cloud* pública. De forma a ser possível, é necessário existir um servidor VPN em cada projeto, com privilégios para instanciar recursos na *cloud* pública. Na criação do servidor VPN, o utilizador escolhe qual a rede que deseja estender para a *cloud* pública, escolhendo entre uma rede privada (existente no seu projeto) ou uma rede do DEI (por exemplo, rede do CISUC). Após a criação do servidor, garante-se que cada instância na *cloud* pública possa ligar-se à VPN, permitindo à instância obter IP automaticamente na rede estendida, caso o servidor DHCP se encontre ativo. A Figura 5.8 ilustra um exemplo real do funcionamento do módulo para dois projetos distintos na *cloud*.

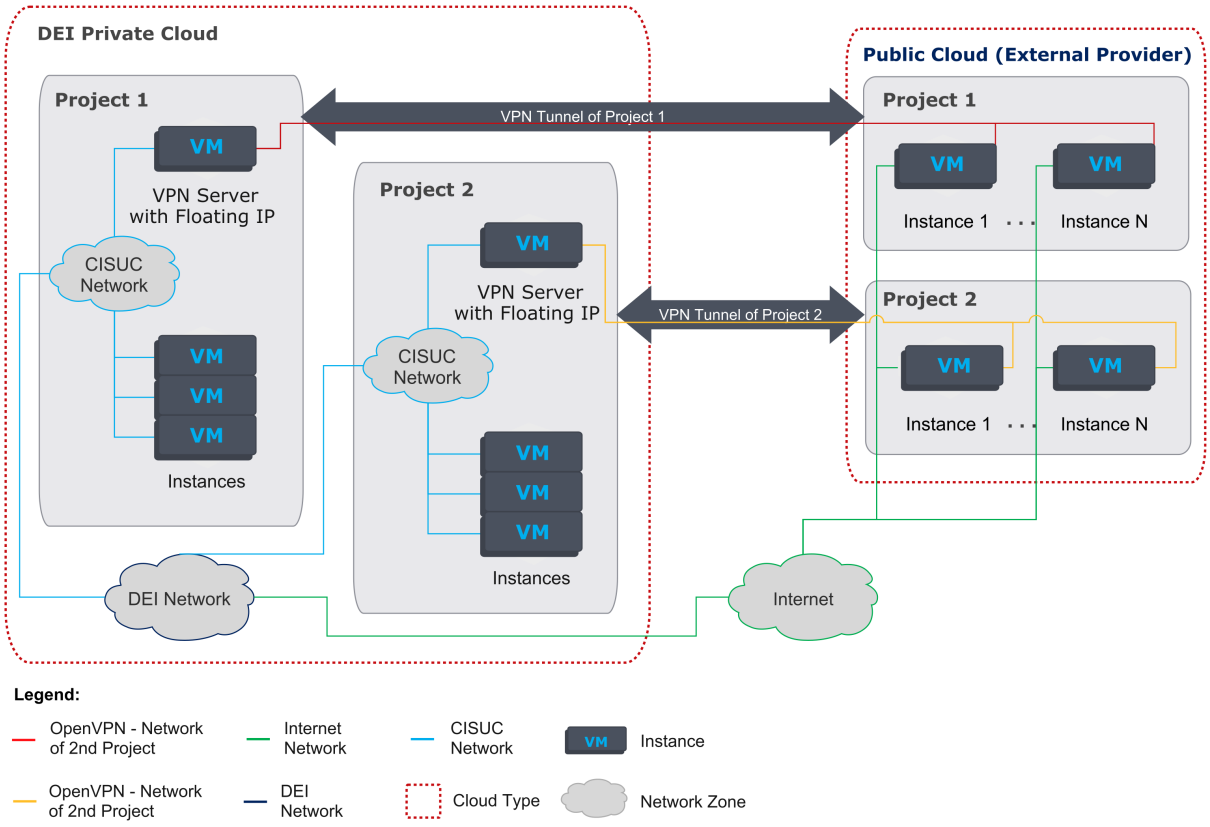


Figura 5.8: Exemplo de utilização do módulo DEIPubCloud com dois projetos distintos na *cloud*

Para efetuar o processo de criação do servidor VPN local, criou-se uma instância no *OpenStack* através do *Horizon*, escolhendo a rede de investigação CISUC. Após a instância criada, e ter efetuado o *boot* completo, acedeu-se à máquina via *SSH*, procedendo-se à instalação e configuração do *OpenVPN*.

Após efetuada a configuração, foi necessário expor a máquina virtual à Internet, sendo criado e associado um *floating ip* à máquina virtual já existente, possibilitando a comu-

nicação das instâncias da *cloud* pública com o servidor VPN.

Após terminados os passos descritos anteriormente, ao efetuar uma ligação OpenVPN para o servidor criado, através do *floating ip*, apenas é possível efetuar ligações do servidor para o cliente, não sendo possível comunicar com os restantes serviços ou dispositivos localizados na rede do CISUC. Este comportamento ocorre devido ao *Neutron*, que, por defeito, rejeita todos os pacotes que não tenham o mesmo *mac-address* de origem da instância. Como tal, é necessário desativar o *port security* do servidor VPN, por forma a ser possível efetuar as comunicações com todos os servidores localizados no CISUC.

Após efetuada a preparação do sistema, procedeu-se à automatização dos processos validados anteriormente, efetuando os seguintes passos (ver Figura 5.9 e 5.10):

1. O utilizador através do *Horizon*, acede ao *panel VPN Management* da *DashBoard* do módulo DEI PubCloud, onde utiliza o formulário de criação de um servidor VPN, especificando qual a rede que deseja estender, o *provider* que vai está associado ao serviço VPN e o *floating ip*;

2. O servidor recebe os dados enviados pelo formulário (via método *POST*), executando as seguintes tarefas: i) envia um pedido REST ao serviço *Nova* para efetuar a criação de uma instância; ii) guarda os dados do servidor VPN na base de dados e iii) invoca o método de criação da VPN local, através do módulo *DeiPubCloud backend*. Este método é invocado de forma assíncrona, sendo enviado uma *task* para o servidor *RabbitMQ*;

3. O *Scheduler (Celery Daemon)* consome todas as *tasks* que são enviadas para a *queue*(deipubcloud) do módulo.

4. Após a receção do pedido de criação da VPN local, são executadas as seguintes operações:

    - O *floating ip* escolhido no 1º passo é associado à instância que foi criada através do *Nova*;
    - O *port security* do NIC (*Network Interface Controller*) é desativado, uma vez que, torna possível aos clientes VPN comunicarem com os *hosts* que se encontrem na rede estendida.
    - Durante 120 segundos a aplicação testa a ligação ao serviço *SSH* da instância criada no *Nova*. Caso não seja possível efetuar a ligação ao serviço SSH, o método termina e regista na base de dados a falha no processo de *deployment*.
    - Caso o passo anterior seja realizado com sucesso, o *playbook* de *deployment* do VPN *Server* é executado através do *ansbile*. No final da execução do *playbook*, é guardado na base de dados o estado da operação.

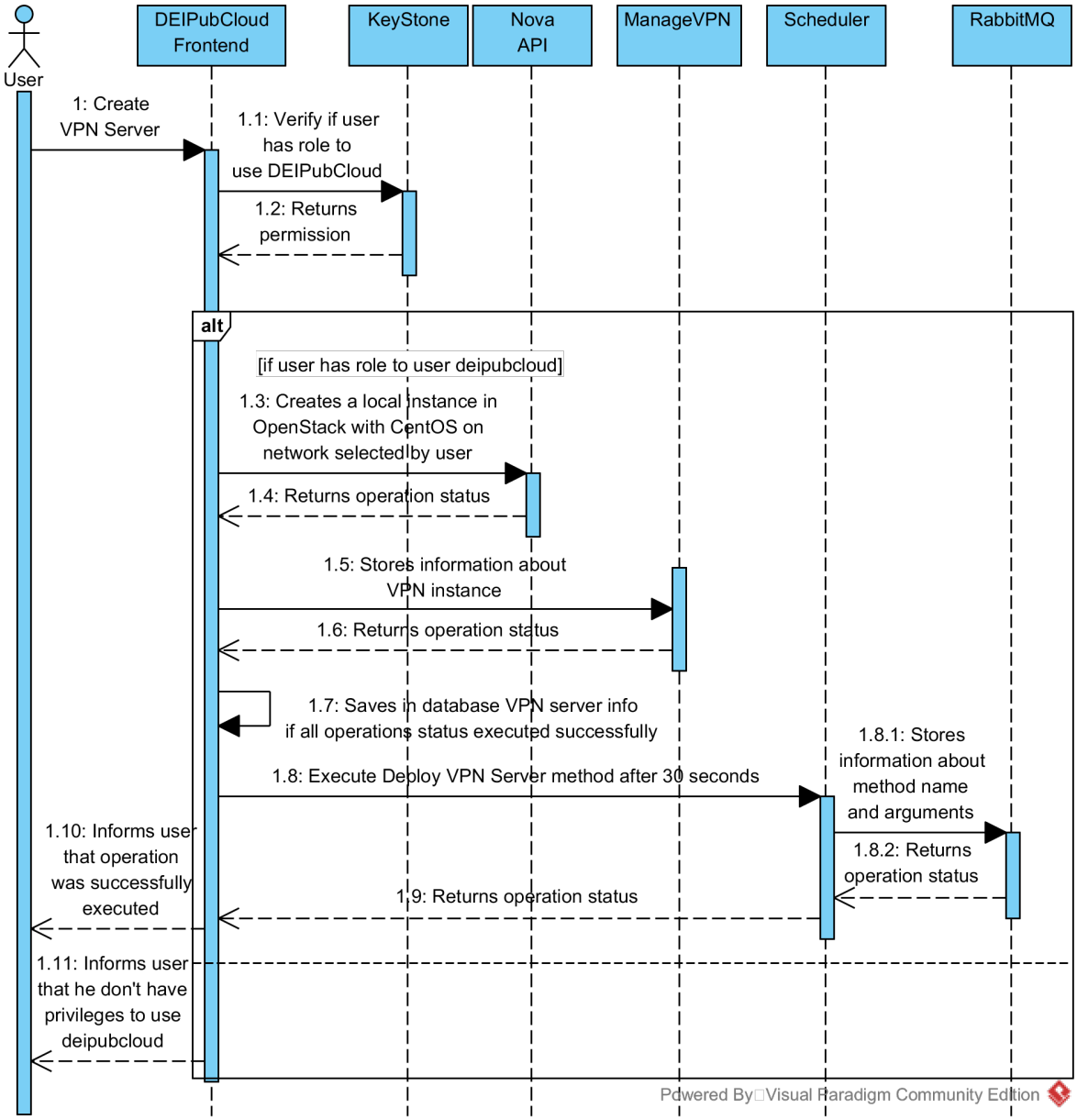


Figura 5.9: *Workflow* da criação do servidor VPN no lado do *frontend*

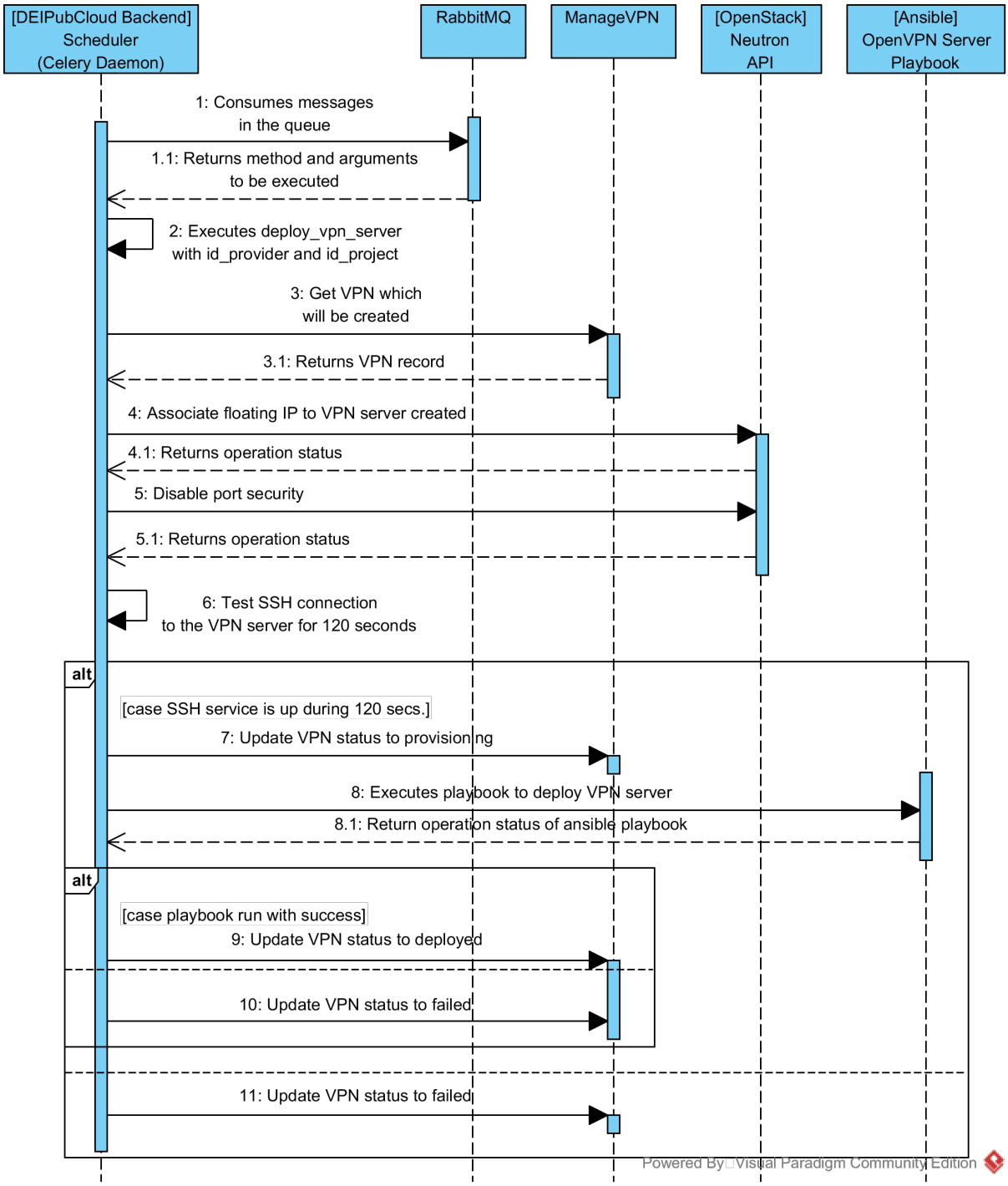


Figura 5.10: *Workflow* da criação do servidor VPN no lado do *backend*

5.4 Conclusão

Com o *openstack-ansible* foi possível efetuar o *deployment* da plataforma *OpenStack* com sucesso. O desenvolvimento do módulo DEIPubCloud permitiu efetuar a integração com a *cloud* pública, sendo este um dos requisitos principais deste projeto, não exis-

tindo no *OpenStack*. Este módulo passou por um desenvolvimento completo, passando pela realização de *mockups*, da base-de-dados, criação do *frontend* e da parte lógica do módulo (*backend*). Na realização da integração de *clouds* foram necessárias efetuar várias alterações ao nível do sistema operativo, assim como do *Neutron*, que permitiram às instâncias da *cloud* pública comunicar tanto com as redes do utilizador como com as redes do DEI. Por fim, foi possível ter uma instalação do *OpenStack* funcional com a possibilidade de integração na *cloud* pública.

O trabalho descrito no presente capítulo permitiu alcançar os objetivos inicialmente delineados para o novo sistema de virtualização, em particular ao nível da integração transparente de *datacenters* e da possibilidade dos utilizadores terem controlo sobre os seus recursos virtualizados. Tal como anteriormente referido, o módulo desenvolvido permitiu igualmente inovar ao adaptar o *OpenStack* para a integração com *cloud providers* externos. De referir que as contribuições que foram efetuadas no projeto *openstack-ansible* fazem parte do código oficial deste projeto, onde permitiu que o *deployment* da funcionalidade VPNaaS passa-se a ser possível.

6 Avaliação e validação

6.1 Estratégia adotada

Para a avaliação e validação do trabalho desenvolvido, foram efetuados testes funcionais, de forma a validar os requisitos funcionais, que podem ser consultados no Anexo C e Anexo D. No que se refere aos atributos de qualidade, foram selecionados os atributos definidos na Secção 3.4, que estão relacionados com o módulo desenvolvido, deixando de parte a análise dos atributos que apenas dizem respeito à plataforma *OpenStack*. Como tal, selecionaram-se os seguintes atributos de qualidade:

- AQ#1: Dentro do componente controlador, escolheu-se analisar o serviço *RabbitMQ* por forma a testar o comportamento do módulo face à falha deste serviço. O foco neste serviço, fica a dever-se ao facto deste, no contexto do controlador, ser o ponto de falha do módulo desenvolvido, já que é responsável pelo envio e receção de tarefas a realizar na *cloud* pública. Para avaliar a redundância deste serviço foram criados três *containers* com o *RabbitMQ*. Desta forma, é possível verificar se o módulo desenvolvido consegue recuperar de eventuais falhas que possam ocorrer.

- AQ#4: Neste atributo de qualidade, passa por verificar a possibilidade de adicionar/remover novos *providers* sem afetar o sistema que se encontra em produção.

- AQ#7: A estratégia neste caso será verificar se o tráfego trocado entre os componentes do sistema é efetuado de forma encriptada.

- AQ#9: Neste atributo a estratégia é analisar se um utilizador não autorizado consegue aceder ao módulo desenvolvido sem ter os devidos privilégios.

- AQ#10: Observar se todo o tráfego trocado entre o utilizador e o *browser* é encriptado.

- AQ#12 e AQ#13: Verificar se todos os pedidos são processados com sucesso no sistema.

- AQ#14: Verificar se a disponibilidade de informação relativa ao normal funcionamento ou situações de falha do sistema, com a possibilidade de identificar os utilizadores e projetos associados.

6.2 Avaliação e validação

No AQ#1 desenvolveu-se um *script* (ver *script* 1 no Anexo E) responsável por efetuar operações assíncronas de criação de instâncias na *cloud* pública, com um intervalo de um segundo, por forma a simular uma falha. Durante a execução do *script*, injetou-se uma falha no *container rabbitmq* que continha o *virtual host* primário do módulo, desligando o serviço *rabbitmq-server*. Após efetuar a simulação da falha, analisou-se o *deployer node*, verificando a continuidade do processamento das tarefas existentes na *queue*. Durante o processamento foi ainda observado se existe alguma tarefa que não tenha sido executada com sucesso. O teste foi realizado com base na criação de 10 instâncias, tendo sido injetado a falha no serviço *RabbitMQ* durante a criação da 3º instância, onde se verificou que o serviço *celery daemon* recuperou de forma rápida, tendo os 10 pedidos sido completados com sucesso.

No AQ#4, devido à arquitetura deste sistema, qualquer alteração efetuada nos *providers*, o sistema irá estar sempre em funcionamento, desde que as credenciais sejam válidas. Isto deve-se ao facto dos *providers* serem utilizados nas operações de gestão das instâncias (criar, reiniciar e/ou apagar). Após a criação das mesmas, apenas é necessário o seu endereço IP público ou privado para as gerir. Mesmo que se efetue a alteração das credenciais do *provider* erradamente, não impede o normal funcionamento das instâncias que já tenham sido criadas. Caso seja necessário adicionar um tipo de *provider* no código-fonte, quando o processo é finalizado, são executados os testes funcionais. No caso em que os testes são executados com sucesso, o módulo é atualizado automaticamente, sem qualquer impacto ao nível de *downtime*, não sendo percecionado pelo utilizador final. Para analisar este atributo, foi efetuado um *reload* no serviço *apache* antes de finalizar o pedido de criação de uma instância, onde se verificou que o pedido foi processado normalmente. O mecanismo de "*reload*" do *apache* (denominado por *graceful restart*) garante que os filhos do processo *apache* sejam reiniciados, apenas quando estes não processarem nenhum pedido HTTP, sendo reiniciados com as novas alterações efetuadas[40].

No AQ#7 e no AQ#10, os atributos são garantidos através do uso SSL no HAProxy, sendo o *software* responsável por efetuar o balanceamento de carga dos serviços, utilizando comunicações seguras. Desta forma, tanto os *endpoints* internos como externos são efetuados utilizando o protocolo HTTPS. Para a verificação deste atributo, foi utilizado o *tcpdump* para observar se o tráfego efetivamente era encriptado, onde se obteve sucesso na validação.

O AQ#9 é verificado através do *Horizon* em conjunto com o *KeyStone* do *OpenStack*. Através dos *panels* do *Horizon*, foi definido quais os utilizadores que assumem privilégios para aceder a cada um dos *panels* (por exemplo, *VM Management* e *Images Management*), existindo dois tipos de permissão: i) os administradores têm controlo total sobre os recursos localizados na *cloud* pública e ii) os utilizadores (com o *role deipubcloud*) têm privilégios para aceder ao módulo desenvolvido (ver Figura 5.6). Desta forma, para validar este atributo de qualidade, foram realizados testes funcionais no *frontend* (ver Anexo D), por forma a garantir que apenas os utilizadores autorizados possam aceder ao módulo desenvolvido, consoante os seus privilégios.

No AQ#12 foi igualmente criado um *script* (ver *script* 2 no Anexo E) responsável por criar um conjunto de instâncias na *cloud* pública. Foi verificado se as instâncias foram corretamente criadas e se se encontravam acessíveis a partir da rede interna do DEI, utilizando um teste de conectividade. Para além disto, o *script* testa ainda a eliminação de todas as instâncias, verificando se na base de dados se encontram zero instâncias ativas. Este *script* foi dividido em diferentes *batches*, onde testou-se primeiramente a criação/configuração correta de 15, 25 e 50 instâncias, assim como a eliminação de 15, 25 e 50 instâncias, tendo sido todas criadas e eliminadas com sucesso para todos os *batches*.

No AQ#13 utilizou-se o *script* 3 (ver Anexo E) onde, primeiramente, efetuou-se o *reboot* de todas as instâncias criadas no *script 2*. O *script E* foi igualmente executado em 15, 25 e 50 instâncias, onde todas foram reiniciadas com sucesso. Neste atributo de qualidade apenas foi testado o *reboot* devido à limitação da biblioteca *apache-libcloud*, visto que, atualmente, apenas suporta a funcionalidade *reboot*.

No AQ#14 ao executar os testes funcionais, verificaram-se se os ficheiros de *logs* eram escritos corretamente, verificando a escrita de *logs* à medida que estes foram efetuados. A título de exemplo, ao tentar criar uma instância sem ter quota suficiente, verifica-se que nos *logs* a operação é negada, devido a ter ultrapassado a quota atribuída.

7 Conclusões e Trabalhos Futuros

7.1 Conclusões

O presente trabalho teve como principal objetivo implementar e validar um sistema distribuído de virtualização para o DEI, que permitisse resolver duas das maiores limitações do sistema atual: i) impossibilidade de escalar recursos para o exterior e ii) a impossibilidade dos utilizadores terem controlo sobre os seus recursos na infraestrutura de virtualização.

Tendo como base a arquitetura atual do DEI, efetuou-se o levantamento de requisitos funcionais, não funcionais e a realização da arquitetura do sistema idealizado pretendido. A arquitetura foi mapeada com os componentes já existentes da plataforma *cloud*, o *OpenStack*. Neste mapeamento foi identificado um componente em falta, que permitisse estender os recursos para a *cloud* pública, existindo integração com a rede do utilizador ou com uma rede do DEI. Neste sentido, foi desenhado um módulo para o *OpenStack* que fornecesse esta funcionalidade, onde se efetuou o ciclo de desenvolvimento. O módulo desenvolvido, nas suas componentes *frontend* e *backend*, bem como as restantes contribuições no contexto da arquitetura proposta e validada no presente estágio, permitiram evoluir o *OpenStack* no sentido de suportar integração transparente entre *clouds* privadas e públicas, ao mesmo tempo permitindo responder aos principais desafios deste trabalho.

Após a realização do trabalho, foi possível concluir que a adoção da plataforma *OpenStack* revelou-se capaz de responder às necessidades pretendidas para um sistema distribuído de virtualização do DEI. Verificou-se ainda que é uma plataforma extensível, ao ponto de desenvolver módulos adicionais que interagem com o sistema, assim como se revelou uma alternativa para substituir o sistema de virtualização atual do DEI. Deste modo, os utilizadores são capazes de simular ambientes próximos da realidade, através das funcionalidades que a plataforma permite. Além disto, a plataforma encontra-se em desenvolvimento constante, sendo lançada uma nova versão a cada 6 meses e com uma grande comunidade *open-source* muito ativa.

O trabalho desenvolvido foi bastante abrangente e completo, desde a instalação e configuração dos equipamentos de suporte à infraestrutura, até ao desenvolvimento e *deployment* do módulo desenvolvido. Envolveu muito esforço a nível de integração tecnológica, desde à utilização de ferramentas de *deployment* automático, utilização de *Message Brokers* com mecanismos de *failover*, conhecimentos a nível de administração de sistemas entre outros. Foi um trabalho gratificante, pois não só foca muitos aspetos da engenharia

de redes e engenharia do *software*, mas também permitiu realizar diversas contribuições: i) contribuição para o *deployment* de um novo serviço de virtualização do DEI a entrar em funcionamento brevemente e ii) contribuições para projetos de referência do *OpenStack*, como é o caso do *openstack-ansible*, onde foi possível efetuar alterações no projeto que permitiram trazer melhorias neste projeto.

7.2 Trabalhos Futuros

O projeto desenvolvido aponta para resultados interessantes relativamente à integração da *cloud* do DEI com a *cloud* pública. Contudo, apesar do trabalho desenvolvido permitir ter num futuro imediato um novo serviço de virtualização no DEI, justificando-se o desenvolvimento de trabalhos complementares. Primeiramente, é necessário dar continuidade ao projeto *openstack-ansible*, de modo a ativar as restantes funcionalidades da plataforma, nomeadamente o LBaaS e o FWaaS.

Considera-se, contudo relevante, investir numa análise mais aprofundada nos restantes projetos do *OpenStack* como, por exemplo, o *Cinder* e *Swift*. Desta forma, é possível existir um maior contacto com estes serviços, com o objetivo de identificar as suas limitações e mecanismos de redundância.

Por forma a dar continuidade ao projeto é necessário investir em continuamente na integração de novas funcionalidades. Numa próxima versão, poderá-se integrar a solução PKI do DEI na infraestrutura *OpenStack*, permitindo automação no processo de gestão dos certificados X509, tendo a possibilidade de controlar outros aspetos.

8 Anexos

Anexo A. Contribuições no projeto *openstack ansible*

Como referido no secção 5.2, durante o *deployment* do *OpenStack* efetuei algumas contribuições no projeto, de forma a ser possível utilizar o serviço VPNaaS. As contribuições efetuadas por mim podem ser visualizadas para no projeto podem ser visualizadas no seguinte url: https://review.openstack.org/#/q/owner:pjm .

Neste anexo vou listar as contribuições efetuadas ao longo deste semestre, identificado o ID da alteração, a qual projeto se refere, a respetiva descrição, se a alteração foi *backported* para uma *branch stable* ou não, a data da proposta, assim como a data de aceitação. As contribuições efetuadas neste semestre, foram as seguintes:

- ***Change ID*:** 297179
  - **Projeto:** openstack-ansible
  - **Descrição:** Foi o meu primeiro *commit*, em que consistiu em juntar toda a documentação dos *network services* (LBaaS, VPNaaS e FWaaS) num só ficheiro.
  - ***Backported*:** Não
  - **Data da proposta:** 24 de Março de 2016
  - **Data de aceitação:** 26 de Março de 2016
  - **URL:** https://review.openstack.org/#/c/297179/
- ***Change ID*:** 307322
  - **Projeto:** openstack-ansible
  - **Descrição:** Correção de um *typo* na documentação dos *network services*
  - ***Backported*:** Para a *branch stable/mitaka* (*Change ID*: 307659)
  - **Data da proposta:** 18 de Abril de 2016
  - **Data de aceitação:** 19 de Abril de 2016
  - **URL:** https://review.openstack.org/#/c/307322/ https://review.openstack.org/#/c/307659/
- ***Change ID*:** 308252

  - **Projeto:** openstack-ansible-os_neutron
  - **Descrição:** Correção do *service_provider* na configuração do VPNaaS. Sem este *patch*, o serviço *Neutron* deixava de funcionar porque estava a utilizar uma classe que não existia.
  - ***Backported*:** Para a *branch stable/mitaka* (*Change ID*: 308342)
  - **Data da proposta:** 20 de Abril de 2016
  - **Data de aceitação:** 20 de Abril de 2016
  - **URL:** https://review.openstack.org/#/c/308252/ https://review.openstack.org/#/c/308342/

- ***Change ID*:** 310382
  - **Projeto:** openstack-ansible-os_neutron
  - **Descrição:** Este *patch* tem várias correções:
    1. O nome do serviço "*neutron-vpnaas-agent*" está errado, o nome do executável é "*neutron-vpn-agent*". Sem esta alteração, ao executar o *init-script* do *neutron-vpn-agent*, o *script* iria utilizar um executável que não existia, desta forma o serviço nunca iria ser executado;
    2. Correção da *classpath* do *neutron_driver_vpnaas*, substituição da biblioteca *deprecated* (que deixará de ser suportada na nova *release*) pela nova *classpath*;
    3. Adição do *init-script* do serviço VPNaaS, responsável por correr o agente como um serviço.
    4. No ficheiro de configuração do VPNaaS faltava adicionar a *external_network_bridge*, assim como a *interface_driver* a ser utilizada pelo serviço.
  - ***Backported*:** Para a *branch stable/mitaka* (*Change ID*: 312457)
  - **Data da proposta:** 27 de Abril de 2016
  - **Data de aceitação:** 04 de Maio de 2016
  - **URL:** https://review.openstack.org/#/c/310382/ https://review.openstack.org/#/c/312457/

- ***Change ID*:** 319668
  - **Projeto:** openstack-ansible-os_neutron
  - **Descrição:** Este *patch* irá substituir a *classpath deprecated* do FWaaS pela nova, desta maneira garante-se que nas próximas *releases* o serviço continua a funcionar.
  - ***Backported*:** Para a *branch stable/mitaka* (*Change ID*: 320065)
  - **Data da proposta:** 22 de Maio de 2016
  - **Data de aceitação:** 23 de Maio de 2016
  - **URL:** https://review.openstack.org/#/c/319668/ https://review.openstack.org/#/c/320065/

- ***Change ID*:** 321648
  - **Projeto:** openstack-ansible
  - **Descrição:** Este *patch* foi feito na documentação, que especifica quais são os módulos do *kernel* que o *bare-metal* necessita para poder utilizar o VPNaaS, ativando os módulos necessários para o IPSec funcionar. Esta alteração indica ao utilizador quais são as alterações que ele deve executar nos seus ficheiros de configuração, assim como os *playbooks* que necessita de executar para ter este serviço ativo.
  - ***Backported*:** Para a *branch stable/mitaka* (*Change ID*: 322254)
  - **Data da proposta:** 26 de Maio de 2016
  - **Data de aceitação:** 26 de Maio de 2016
  - **URL:** https://review.openstack.org/#/c/321648/ https://review.openstack.org/#/c/322254/
- ***Change ID*:** 321819
  - **Projeto:** openstack-ansible-openstack_hosts
  - **Descrição:** Este *patch* não foi realizado por mim, foi realizado pelo Ala Raddauoui na *branch master*. No entanto, eu fiz *backport* deste *commit* para a *branch stable/mitaka*, desta maneira a alteração anterior (321648) funciona na versão *mitaka*, visto que este *commit* tem as *tasks* necessárias para importar os módulos do *kernel* no *bare-metal*.
  - ***Backported*:** Este *patch* já é um *backport*.
  - **Data da proposta:** 26 de Maio de 2016
  - **Data de aceitação:** 27 de Maio de 2016
  - **URL:** https://review.openstack.org/#/c/321819/

Anexo B. *Mockups* do módulo desenvolvido

Este anexo apresenta os diversos *mockups* correspondestes ao *frontend* do módulo desenvolvido.

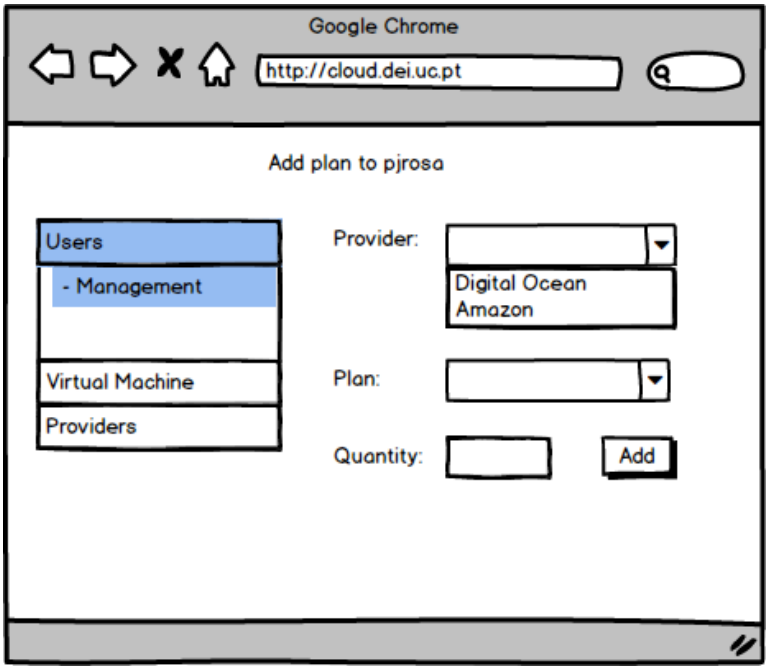


Figura 1: Adicionar um plano ao utilizador pjrosa

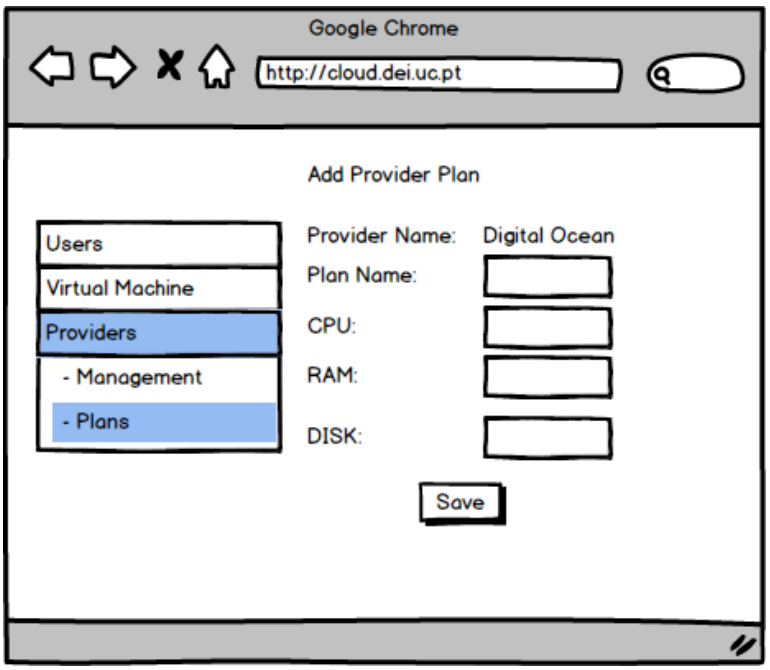


Figura 2: Adicionar um plano a um *provider*

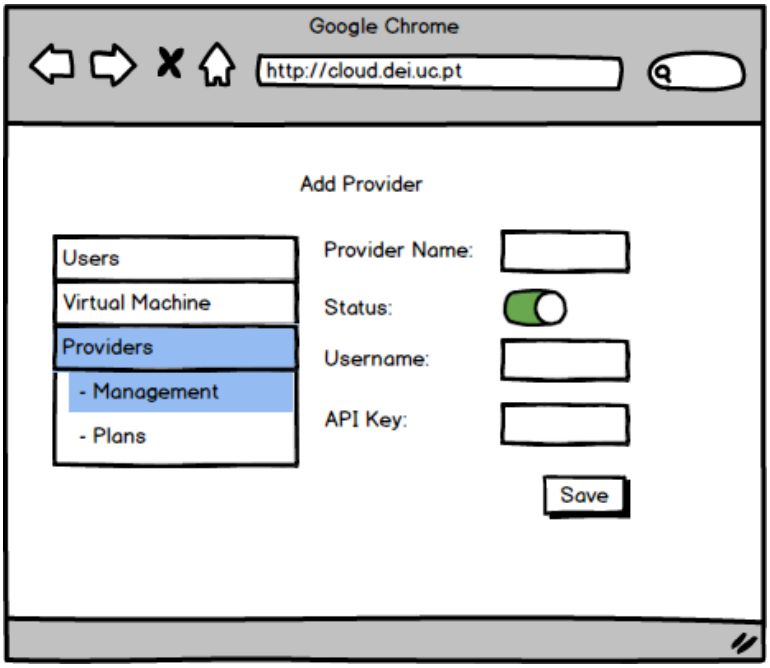


Figura 3: Adicionar um *provider*

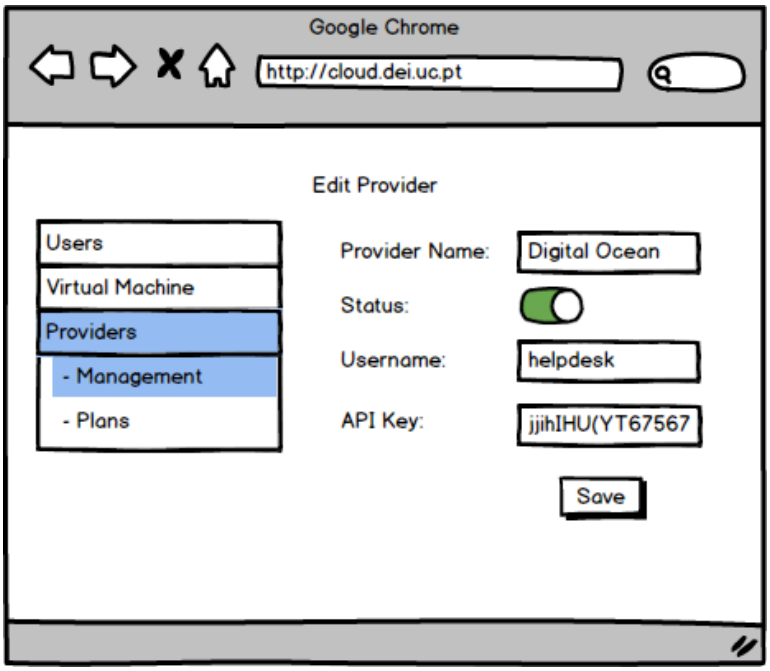


Figura 4: Editar um *provider*

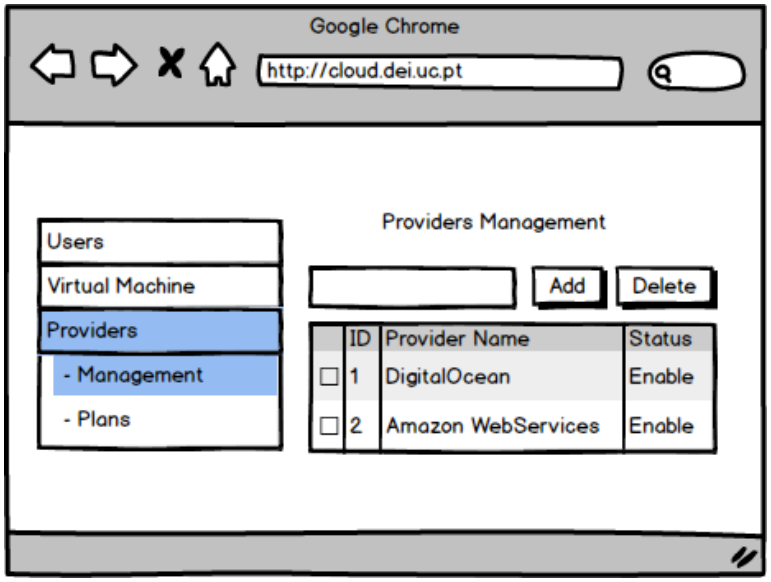


Figura 5: Visão global da gestão dos *providers*

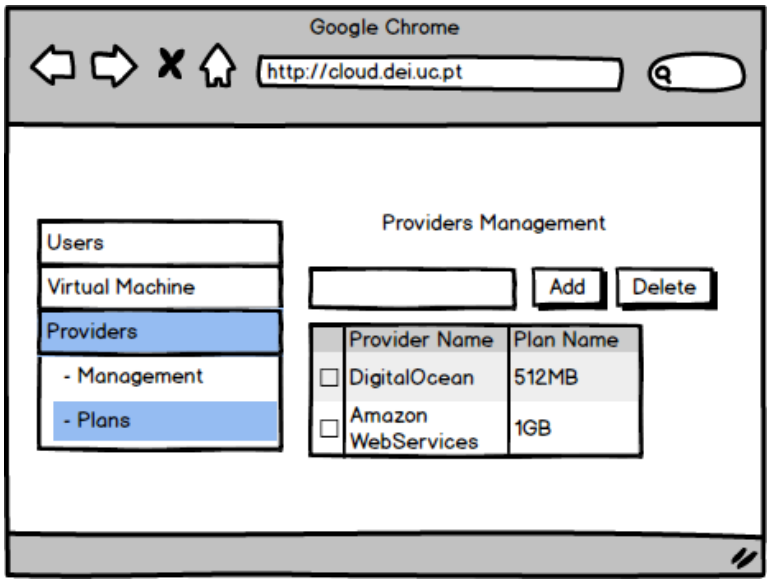


Figura 6: Visão global da gestão dos planos de um *provider*

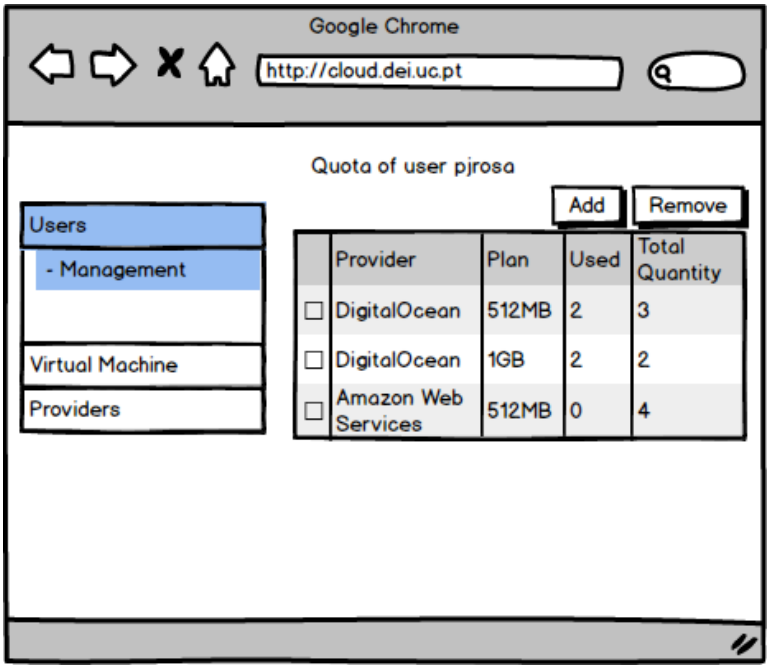


Figura 7: *Quota* do utilizador pjrosa

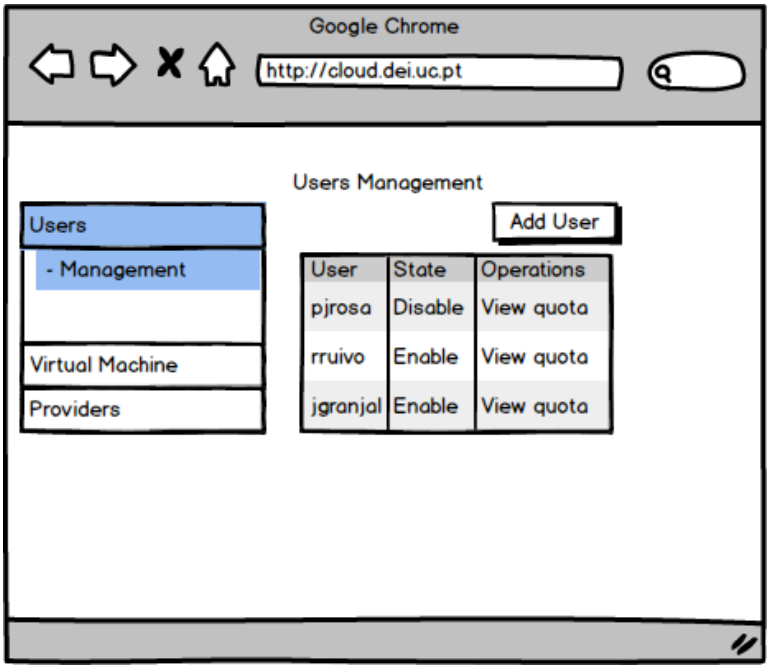


Figura 8: Gestão de quotas de utilizadores

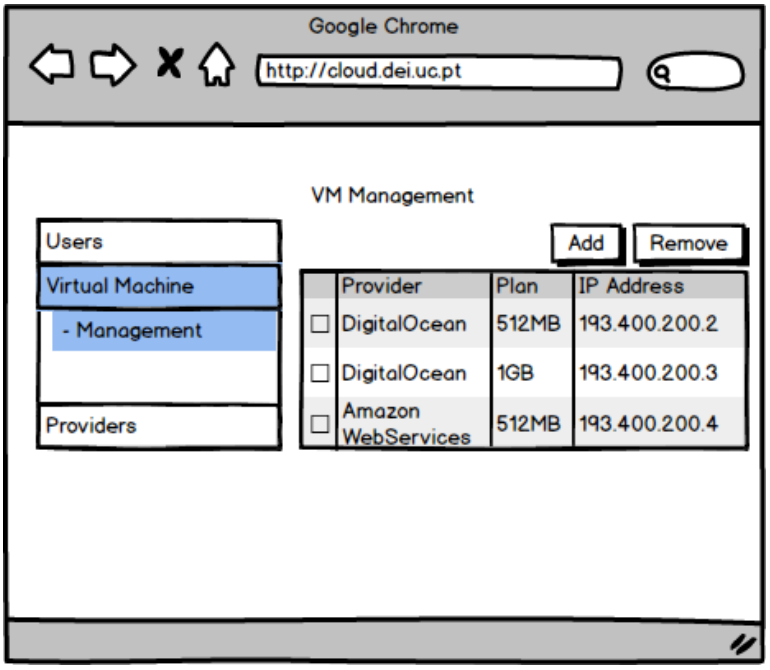


Figura 9: Gestão de instâncias de um utilizador

Anexo C. Testes funcionais: *backend*

No ínicio do desenvolvimento, após criada uma classe como, por exemplo, o *ManageProjects*, foram desenvolvidos testes para todos os métodos existentes dessa classe, tendo sido executados sempre que existiu alguma alteração no módulo, utilizando uma base de dados limpa. De seguida, é apresentada a lista de testes executados com a respetiva descrição, que são:

- **Classe:** *ManageImages*; Garante o requisito funcional com o ID 5.
  1. **Método a ser testado:** *ManageImages.get_image()*
     - **Nome do método:** test_get_image
     - **Métodos dependentes:** test_create_image
     - **Descrição:** Este método obtém uma imagem registada na base de dados, comparando alguns dos atributos, nomeadamente, o nome da imagem, nome do *provider* e o ID do *provider*.
     - **Testes efetuados:**
       (a) ***Input:*** id=1
       ***Output* esperado:** Objeto *image*, com o name="ubuntu", id_provider="1"e o provider.name="DigitalOcean".
       Resultado: Passou
  2. **Método a ser testado:** *ManageImages.create_image()*
     - **Nome do método:** manage_providers
     - **Métodos dependentes:** *ManageProviders.create_provider()*
     - **Descrição:** Antes de efetuar o teste é criado um *provider* válido para ser utilizado na criação de uma *image*. Este método testa a criação de uma imagem. Com o primeiro teste verificou-se a criação de uma imagem com os valores todos válidos. Enquanto que o segundo teste permite observar se ao adicionar uma *foreign key* inválida é atribuída uma exceção.
     - **Testes efetuados:**
       (a) ***Input:*** Todos os campos da classe *Image* válidos.
       ***Output* esperado:** *True*
       Resultado: Passou
       (b) ***Input:*** Todos os campos da classe *Image* válidos com a exceção do *id_provider*, onde *id_provider* utiliza uma *foreignkey* inválida.
       ***Output* esperado:** exceptions.IntegrityError
       Resultado: Passou
  3. **Método a ser testado:** *ManageImages.update_image()*
     - **Nome do método:** test_update_image
     - **Métodos dependentes:** test_create_image
     - **Descrição:** Antes de efetuar os testes, é criada uma imagem válida. Este método verifica se a alteração de uma imagem foi bem sucedida.

- **Testes efetuados:**
  (a) **Método:** update_image
  Input: id=1, name='centos'
  ***Output* esperado:** *True*
  Resultado: Passou
  (b) **Método:** get_image
  Input: id=1, name='centos'
  ***Output* esperado:** Objeto imagem com o nome *centos*
  Resultado: Passou

4. **Método a ser testado:** *ManageImages.remove_image()*
   - **Nome do método:** test_remove_image
   - **Métodos dependentes:** test_create_image
   - **Descrição:** Antes de efetuar os testes, é criada uma imagem válida. Este teste verifica se a eliminação de uma imagem foi bem sucedida.
   - **Testes efetuados:**
     (a) **Método:** delete_image
     Input: id=1
     ***Output* esperado:** True
     Resultado: Passou
     (b) **Método:** get_image
     Input: id=1
     ***Output* esperado:** exceptions.NoResultFound
     Resultado: Passou
5. **Método a ser testado:** *ManageImages.get_images()*
   - **Nome do método:** test_get_images
   - **Métodos dependentes:** test_create_image
   - **Descrição:** Antes de efetuar os testes, é criada uma imagem válida. Este teste verifica se o número total de imagens, que se encontram na base de dados, são devolvidos corretamente.
   - **Testes efetuados:**
     (a) **Método:** len(get_images())
     Input: Não recebe argumentos
     ***Output* esperado:** 0
     Resultado: Passou
     (b) **Método:** test_create_image()
     Input: Não recebe argumentos.
     ***Output* esperado:** Não ocorre nenhuma exceção.
     Resultado: Passou
     (c) **Método:** len(get_image())
     Input: Nenhum
     ***Output* esperado:** 1
     Resultado: Passou

- **Classe:** *ManagePlans*; Garante o requisito funcional com o ID 6.

1. **Método a ser testado:** *ManagePlans.get_plan()*
   - **Nome do método:** test_get_plan
   - **Métodos dependentes:** test_create_plan
   - **Descrição:** Este método irá obtém um *provider* registado na base de dados, comparando alguns dos atributos, nomeadamente, o ID e o nome do *provider*.
   - **Testes efetuados:**
     (a) ***Input:*** id=1
     ***Output* esperado:** Objeto *plan*, com o name="512MB"e id_provider="1".
     Resultado: Passou
2. **Método a ser testado:** *ManagePlans.create_plan()*
   - **Nome do método:** manage_plans.create_plan()
   - **Métodos dependentes:** *ManageProviders.create_provider()*
   - **Descrição:** Antes de efetuar o teste são criados dois *provider* válidos. Este método testa a criação de um *provider*. O primeiro teste verifica a criação de dois *providers* com todos os valores válidos. O segundo teste, verifica se não existe nenhum plano na base de dados. Por fim, são criados dois planos válidos e é verificado se existem dois planos registados na base de dados.
   - **Testes efetuados:**
     (a) **Método:** ManageProviders.create_provider()
     Input: Todos os campos da classe *provider* válidos.
     ***Output* esperado:** *True*
     Resultado: Passou
     (b) **Método:** ManageProviders.create_provider()
     Input: Todos os campos da classe *provider* válidos, diferentes do *provider* criado anteriormente.
     ***Output* esperado:** *True*
     Resultado: Passou
     (c) **Método:** ManagePlans.get_plans()
     Input: Sem dados no input
     ***Output* esperado:** 0
     Resultado: Passou
     (d) **Método:** ManagePlans.create_plan()
     Input: Todos os campos da classe *plan* válidos.
     ***Output* esperado:** True
     Resultado: Passou
     (e) **Método:** ManagePlans.create_plan()
     Input: Todos os campos da classe *plan* válidos, diferentes do *plan* criado anteriormente.
     ***Output* esperado:** True
     Resultado: Passou

3. **Método a ser testado:** *ManagePlans.update_plan()*
   - **Nome do método:** test_update_plan
   - **Métodos dependentes:** test_create_plan
   - **Descrição:** Antes de efetuar os testes, é criado um *provider* válido. Este método verifica se a alteração do *provider* foi bem sucedida, inclusive se a alteração do preço é bem efetuada, confirmando se o preço devolvido pelo atributo *price* é o último valor que foi atualizado.
   - **Testes efetuados:**
     (a) **Método:** get_plan
     Input: id=1
     ***Output* esperado:** Objeto *plan* com o cpu=1
     Resultado: Passou
     (b) **Método:** update_plan
     Input: id=1, cpu=2
     ***Output* esperado:** *True*
     Resultado: Passou
     (c) **Método:** get_plan().plan_price.price
     Input: id=1
     ***Output* esperado:** 5
     Resultado: Passou
     (d) **Método:** update_plan()
     Input: id=1, price=0.005
     ***Output* esperado:** True
     Resultado: Passou
     (e) **Método:** get_plan()
     Input: id=1, price=0.005
     ***Output* esperado:** Objeto *plan* com o plan_price.price=0.005
     Resultado: Passou
4. **Método a ser testado:** *ManagePlans.remove_plan()*
   - **Nome do método:** test_remove_plan
   - **Métodos dependentes:** test_create_plan
   - **Descrição:** Antes de efetuar os testes, é criado um *plan* válido. Este teste verifica se a eliminação de um *plan* foi bem sucedida.
   - **Testes efetuados:**
     (a) **Método:** len(get_plans())
     Input: Sem argumentos de input
     ***Output* esperado:** 2
     Resultado: Passou
     (b) **Método:** delete_plan()
     Input: id=1
     ***Output* esperado:** *True*
     Resultado: Passou
     (c) **Método:** len(get_plans())

Input: Sem argumentos de input
***Output* esperado:** 1
Resultado: Passou

(d) **Método:** delete_plan()
Input: id=1
***Output* esperado:** exceptions.NoResultFound
Resultado: Passou

5. **Método a ser testado:** *ManagePlans.get_plans()*
   - **Nome do método:** test_get_plans
   - **Métodos dependentes:** test_create_plan
   - **Descrição:** Antes de efetuar os testes, são criados dois *providers* válidos. Este teste verifica se o número total de imagens que se encontra na base de dados são devolvidos corretamente.
   - **Testes efetuados:**
     (a) **Método:** len(get_plans())
     Input: Não recebe argumentos
     ***Output* esperado:** 2
     Resultado: Passou

- **Classe:** *ManageProjectPlans*; Garante o requisito funcional com o ID 7.

1. **Método a ser testado:** *ManageProjectPlans.get_project_plan()*
   - **Nome do método:** test_get_project_plan
   - **Métodos dependentes:** test_create_project_plan
   - **Descrição:** Este método obtém um *project_plan* registado na base de dados, comparando a quantidade total que um plano possa ter.
   - **Testes efetuados:**
     (a) **Método:** ManageProjectPlans.get_project_plan().total_quantity
     Input: id_project="a", id_provider=1, id_plan=1
     ***Output* esperado:** 1
     Resultado: Passou
2. **Método a ser testado:** *ManageProjectPlans.create_project_plan()*
   - **Nome do método:** test_create_project_plan
   - **Métodos dependentes:** *ManageProjectPlans.create_provider(), ManageProjectPlans.create_plan(), ManageProjects.create_project()*
   - **Descrição:** Antes de efetuar o teste foi criado um *provider*, *plan* e *project* válidos. Foi ainda criado um *project_plan* com dados válidos.
   - **Testes efetuados:**
     (a) **Método:** ManageProjectPlans.create_provider()
     Input: Todos os campos da classe *project_plan* válidos.
     ***Output* esperado:** *True*
     Resultado: Passou
3. **Método a ser testado:** *ManageProjectPlans.update_project_plan()*

- **Nome do método:** test_update_project_plan
- **Métodos dependentes:** test_create_project_plan
- **Descrição:** Antes de efetuar os testes, foi necessário executar a função test_create_project_plan. Este método verifica se a alteração do *project_plan* foi bem sucedida.
- **Testes efetuados:**
  (a) **Método:** get_project_plan().total_quantity
  Input: id=1
  ***Output* esperado:** 1
  Resultado: Passou
  (b) **Método:** update_project_plan
  Input: id=1, total_quantity=2
  ***Output* esperado:** *True*
  Resultado: Passou
  (c) **Método:** get_project_plan().total_quantity
  Input: id=1
  ***Output* esperado:** 2
  Resultado: Passou

4. **Método a ser testado:** *ManageProjectPlans.remove_project_plan()*
   - **Nome do método:** test_remove_project_plan
   - **Métodos dependentes:** test_create_project_plan
   - **Descrição:** Antes de efetuar os testes, é utilizado a função test_create_project_plan. Este teste verifica se a eliminação de um *project_plan* é bem sucedida.
   - **Testes efetuados:**
     (a) **Método:** delete_project_plan()
     Input: id=1
     ***Output* esperado:** *True*
     Resultado: Passou
     (b) **Método:** delete_project_plan()
     Input: id=1
     ***Output* esperado:** exceptions.NoResultFound
     Resultado: Passou
5. **Método a ser testado:** *ManageProjectPlans.get_project_plans()*
   - **Nome do método:** test_get_project_plans
   - **Métodos dependentes:** test_create_project_plan
   - **Descrição:** Antes de efetuar os testes, é executado o método test_create_project_plan. Este teste verifica se o número total de *project_plans*, que se encontram na base de dados são devolvidos corretamente.
   - **Testes efetuados:**
     (a) **Método:** len(get_project_plans())
     Input: Não recebe argumentos
     ***Output* esperado:** 1
     Resultado: Passou

- **Classe:** *ManageProjects*; Garante o requisito funcional com o ID 7.
  1. **Método a ser testado:** *ManageProjects.create_project()*
     - **Nome do método:** test_create_project
     - **Métodos dependentes:** *ManageProjects.create_project()*
     - **Descrição:** São criados três projectos com todos os campos da classe *project* válidos.
     - **Testes efetuados:**
       (a) **Método:** ManageProjects.create_project()
       Input: Todos os campos da classe *project* válidos.
       ***Output* esperado:** *True*
       Resultado: Passou
       (b) **Método:** len(ManageProjects.get_projects())
       Input: Todos os campos da classe *project* válidos.
       ***Output* esperado:** 3
       Resultado: Passou
  2. **Método a ser testado:** *ManageProjects.get_project()*
     - **Nome do método:** test_get_project
     - **Métodos dependentes:** test_create_project
     - **Descrição:** Este método verifica se os três projetos foram criados corretamente, analisando o nome do projeto.
     - **Testes efetuados:**
       (a) **Método:** ManageProjects.get_project().project
       Input: id="a"
       ***Output* esperado:** project1
       Resultado: Passou
       (b) **Método:** ManageProjects.get_project().project
       Input: id="b"
       ***Output* esperado:** project2
       Resultado: Passou
       (c) **Método:** ManageProjects.get_project().project
       Input: id="c"
       ***Output* esperado:** project3
       Resultado: Passou
       (d) **Método:** ManageProjects.get_project().project
       Input: id="d"
       ***Output* esperado:** exceptions.NoResultFound
       Resultado: Passou
  3. **Método a ser testado:** *ManageProjects.update_project()*
     - **Nome do método:** test_update_project
     - **Métodos dependentes:** test_create_project
     - **Descrição:** Antes de efetuar os testes, é necessário executar a função test_create_project. Este método verifica se a alteração do *project* foi bem sucedida.

- **Testes efetuados:**
  (a) **Método:** get_project()
  Input: id='a'
  ***Output* esperado:** project1
  Resultado: Passou
  (b) **Método:** update_project
  Input: id='a', project='project1_modified'
  ***Output* esperado:** *True*
  Resultado: Passou
  (c) **Método:** get_project()
  Input: id='a'
  ***Output* esperado:** project1_modified
  Resultado: Passou

4. **Método a ser testado:** *ManageProjects.remove_project()*
   - **Nome do método:** test_remove_project
   - **Métodos dependentes:** test_create_project
   - **Descrição:** Antes de efetuar os testes, é utilizado a função test_create_project. Este teste verifica se a eliminação de um *project* foi bem sucedida.
   - **Testes efetuados:**
     (a) **Método:** len(ManageProjects.get_projects())
     Input: Sem argumentos
     ***Output* esperado:** 3
     Resultado: Passou
     (b) **Método:** delete_project()
     Input: id='b'
     ***Output* esperado:** *True*
     Resultado: Passou
     (c) **Método:** delete_project()
     Input: id='c'
     ***Output* esperado:** *True*
     Resultado: Passou
     (d) **Método:** len(ManageProjects.get_projects())
     Input: Sem argumentos
     ***Output* esperado:** 1
     Resultado: Passou
     (e) **Método:** delete_project()
     Input: id='c'
     ***Output* esperado:** exceptions.NoResultFound
     Resultado: Passou

5. **Método a ser testado:** *ManageProjects.get_projects()*
   - **Nome do método:** test_get_projects
   - **Métodos dependentes:** test_create_project

- **Descrição:** Antes de efetuar os testes é executado o método test_create_project. Este teste irá verificar se o total número de registos *project* que se encontram na base de dados são devolvidos corretamente.
- **Testes efetuados:**
  (a) **Método:** len(get_projects())
  Input: Não recebe argumentos
  ***Output* esperado:** 3
  Resultado: Passou

- **Classe:** *ManageProviders*; Garante o requisito funcional com o ID 4.

1. **Método a ser testado:** *ManageProviders.create_provider()*
   - **Nome do método:** test_create_provider
   - **Métodos dependentes:** Nenhum
   - **Descrição:** São criados dois *provider*, verificando se é posssível ter *providers* duplicados com o mesmo nome.
   - **Testes efetuados:**
     (a) **Método:** len(ManageProviders.get_providers())
     Input: Verifica se não existe nenhum *provider* registado na base de dados.
     ***Output* esperado:** 0
     Resultado: Passou
     (b) **Método:** ManageProviders.create_provider()
     Input: Criação de um provider com todos os campos da classe *provider* válidos.
     ***Output* esperado:** *True*
     Resultado: Passou
     (c) **Método:** ManageProviders.create_provider()
     Input: Criação de um *provider* com todos os campos da classe *provider* válidos.
     ***Output* esperado:** *True*
     Resultado: Passou
     (d) **Método:** len(ManageProviders.get_providers())
     Input: Verifica se o número total de *providers* corresponde aos *providers* criados anteriormente.
     ***Output* esperado:** 2
     Resultado: Passou
     (e) **Método:** ManageProviders.create_provider()
     Input: Criação de um *provider* com todos os campos da classe *provider* iguais a um dos *providers* criados anteriormente.
     ***Output* esperado:** exceptions.IntegrityError
     Resultado: Passou
2. **Método a ser testado:** *ManageProviders.get_provider_type()*
   - **Nome do método:** test_get_provider_type

- **Métodos dependentes:** test_create_provider
- **Descrição:** Assume-se que a função test_create_provider é executada antes de se iniciarem os testes. Este método verificar se é possível obter os dois *providers* suportados pelo nosso módulo.
- **Testes efetuados:**
  (a) **Método:** ManageProviders.get_provider_type().type
  Input: id=1
  ***Output* esperado:** digitalocean
  Resultado: Passou
  (b) **Método:** ManageProviders.get_provider_type().type
  Input: id=2
  ***Output* esperado:** aws
  Resultado: Passou

3. **Método a ser testado:** *ManageProviders.get_provider()*
   - **Nome do método:** test_get_provider
   - **Métodos dependentes:** test_create_provider
   - **Descrição:** Assume-se que a função test_create_provider é executada antes de se iniciar os testes. Este método verifica se é possível obter os dois *providers* suportados pelo nosso módulo.
   - **Testes efetuados:**
     (a) **Método:** ManageProviders.get_provider().name
     Input: id=1
     ***Output* esperado:** DigitalOcean
     Resultado: Passou
     (b) **Método:** ManageProviders.get_provider().name
     Input: id=2
     ***Output* esperado:** AWS
     Resultado: Passou
4. **Método a ser testado:** *ManageProviders.update_provider()*
   - **Nome do método:** test_update_provider
   - **Métodos dependentes:** test_create_provider
   - **Descrição:** Antes de efetuar os testes, é necessário executar a função test_create_provider. Este método verifica se a alteração do *provider* foi bem sucedida.
   - **Testes efetuados:**
     (a) **Método:** get_provider().api_key
     Input: id=1
     ***Output* esperado:** dei_key
     Resultado: Passou
     (b) **Método:** update_provider
     Input: id='1', api_key='dei_key2'
     ***Output* esperado:** *True*
     Resultado: Passou

(c) **Método:** get_provider().api_key
Input: id=1
***Output* esperado:** dei_key2
Resultado: Passou

5. **Método a ser testado:** *ManageProviders.remove_provider()*
   - **Nome do método:** test_remove_provider
   - **Métodos dependentes:** test_create_provider
   - **Descrição:** Antes de efetuar os testes, é utilizado a função test_create_provider. Este teste verifica se a eliminação de um *provider* foi bem sucedida, assim como indica se a eliminação de um *provider* inexistente gera uma exceção correta.
   - **Testes efetuados:**
     (a) **Método:** len(ManageProviders.get_providers())
     Input: Sem argumentos
     ***Output* esperado:** 2
     Resultado: Passou
     (b) **Método:** delete_provider()
     Input: id=1
     ***Output* esperado:** *True*
     Resultado: Passou
     (c) **Método:** delete_provider()
     Input: id=3
     ***Output* esperado:** exceptions.NoResultFound
     Resultado: Passou
     (d) **Método:** len(ManageProviders.get_providers())
     Input: Sem argumentos
     ***Output* esperado:** 1
     Resultado: Passou
6. **Método a ser testado:** *ManageProviders.get_providers()*
   - **Nome do método:** test_get_providers
   - **Métodos dependentes:** test_create_provider
   - **Descrição:** Antes de efetuar os testes é executado o método test_create_provider. Este teste verifica se o número total de registos *providers* que se encontram na base de dados são devolvidos corretamente.
   - **Testes efetuados:**
     (a) **Método:** len(get_providers())
     Input: Não recebe argumentos
     ***Output* esperado:** 2
     Resultado: Passou
7. **Método a ser testado:** *ManageProviders.get_providers_types()*
   - **Nome do método:** test_get_providers_types
   - **Métodos dependentes:** Nenhum

- **Descrição:** Este teste verifica se o número total de registos *providers_types* que se encontram na base de dados são devolvidos corretamente.
- **Testes efetuados:**
  (a) **Método:** len(get_providers_types())
  Input: Não recebe argumentos
  ***Output* esperado:** 2
  Resultado: Passou

8. **Método a ser testado:** *ManageProviders.get_plans_from_api()*
   - **Nome do método:** test_get_plans_from_api
   - **Métodos dependentes:** Nenhum
   - **Descrição:** Este teste verifica se o método consegue obter informações dos planos à API do *DigitalOcean*.
   - **Testes efetuados:**
     (a) **Método:** create_provider
     Input: Todos os campos válidos da classe *provider*
     ***Output* esperado:** *True*
     Resultado: Passou
     (b) **Método:** get_plans_from_api
     Input: id=1
     ***Output* esperado:** *True*
     Resultado: Passou

- **Classe:** *ManageVMs*; Garante o requisito funcional com o ID 1, ID 2 e ID 3.

1. **Método a ser testado:** *ManageVMs.get_vm()*
   - **Nome do método:** test_get_vm
   - **Métodos dependentes:** test_create_vm
   - **Descrição:** Este método irá obter uma vm registada na base de dados, comparando alguns dos atributos, nomeadamente o nome da VM.
   - **Testes efetuados:**
     (a) ***Input:*** id=1
     ***Output* esperado:** Objeto *VM*, com o name="cenas.cloud.dei.uc.pt"
     Resultado: Passou
2. **Método a ser testado:** *ManageVMs.create_vm()*
   - **Nome do método:** test_create_vm
   - **Métodos dependentes:**
   - **Descrição:** Este método assume que já existem dois plano e um plano associado a um projeto. Este método realiza diversos testes: i) cria uma VM válida; ii) cria uma VM válida, mas como não tem mais recursos para alocar, irá gerar uma excepção e iii) cria uma VM válida, mas num plano que exceda o limite global definido.
   - **Testes efetuados:**

(a) ***Input:*** Todos os dados válidos de uma VM
***Output* esperado:** True
Resultado: Passou

(a) ***Input:*** Todos os dados válidos de uma VM
***Output* esperado:** exceptions.NoResourcesAvailable
Resultado: Passou

(a) ***Input:*** Todos os dados válidos de uma VM, mas num plano que exceda o limite global definido do provider
***Output* esperado:** exceptions.ExceededProviderBudget
Resultado: Passou

3. **Método a ser testado:** *ManageVMs.calculate_vms_prices()*
   - **Nome do método:** test_calculate_vms_prices
   - **Métodos dependentes:** Nenhum
   - **Descrição:** Este método cria uma VM com data de um mês atrás, onde foi efetuada a alteração de preço do plano uma vez, em que o custo inicial era de 0.0744 euros por hora (30 dias atrás), e a segunda passado 15 dias passou a ser 0.0999 euros.
   - **Testes efetuados:**
     (a) ***Input:*** Todos os dados válidos de uma VM com a data de um mês atrás
     ***Output* esperado:** (15*24*0.00744 + 15*24*0.0999) = 38.64
     Resultado: Passou
4. **Método a ser testado:** *ManageVMs.calculate_vm_monthly_price()*
   - **Nome do método:** test_calculate_vm_monthly_price
   - **Métodos dependentes:** test_create_vm
   - **Descrição:** Este método irá verificar o cálculo do custo mensal de uma VM. Este teste assume que o plano 1 tem um custo de 1.5 euros por hora.
   - **Testes efetuados:**
     (a) ***Input:*** id_plan=1
     ***Output* esperado:** $(1.5 * 24h * 30) = 1080$
     Resultado: Passou
5. **Método a ser testado:** *ManageVMs.delete_vm()*
   - **Nome do método:** test_delete_vm
   - **Métodos dependentes:** test_create_vm
   - **Descrição:** Este método irá verificar se uma VM é apagada corretamente.
   - **Testes efetuados:**
     (a) ***Input:*** id=1
     ***Output* esperado:** True
     Resultado: Passou

Anexo D. Testes funcionais: *frontend*

Para validar que o *frontend* do módulo obtenha o comportamento desejado, foram realizados testes funcionais que permitiram verificar se alguma alteração efetuada no *frontend*, assim como no *backend*, tenha afetado o funcionamento normal do módulo. Estes testes foram executados num servidor de desenvolvimento, utilizando o *devstack*[27] de forma a não interferir com o sistema que se encontrava em produção. Estes testes, permitiram ainda verificar se as alterações efetuadas ao longo do desenvolvimento alteraram o módulo de forma a realizar comportamentos indesejados. De seguida, é apresentada a lista de testes executados, e a respetiva descrição:

- ***Panel* a ser testado:** *Images*

  1. **Nome do método:** test_user_access
     Descrição: É verificado se um utilizador sem privilégios de administrador consegue ter acesso ao módulo desenvolvido.
     Testes efetuados:
     (a) ***Input:*** Utilizador normal (sem privilégios de administrador) acede à página inicial do *panel Images Management*.
     ***Output* esperado:** *Redirect* para a página de *login*, de forma a obrigar o utilizador autenticar-se com uma conta de administrador.
     Resultado: Passou
  2. **Nome do método:** test_image_index
     Descrição: É verificado se um administrador consegue visualizar as imagens disponíveis na página inicial.
     Testes efetuados:
     (a) ***Input:*** Administrador acede à página inicial do *panel Management Images*.
     ***Output* esperado:** O número de imagens apresentadas pelo *frontend* ser igual ao número de imagens que se encontram registadas na base de dados.
     Resultado: Passou
  3. **Nome do método:** test_create_image_invalid_parameters
     Descrição: É verificado se um administrador consegue adicionar uma imagem com parâmetros inválidos.
     Testes efetuados:
     (a) ***Input:*** Administrador tenta adicionar uma imagem com todos os campos vazios.
     ***Output* esperado:** O formulário apresenta erros, indicando que há campos em falta.
     Resultado: Passou
     (b) ***Input:*** Administrador tenta adicionar uma imagem com o campo *min_disk_size* inválido.

***Output* esperado:** O formulário apresentar um erro, indicando que o valor deverá ser do tipo inteiro e não uma *string*.
Resultado: Passou

(c) ***Input:*** Administrador tenta adicionar um imagem com um *id_provider* inexistente.
***Output* esperado:** O formulário apresenta um erro, indicando que o *id_provider* selecionado não é válido.
Resultado: Passou

4. **Nome do método:** test_create_image_valid_parameters
Descrição: É verificado se um administrador consegue adicionar uma imagem com parâmetros válidos.
Testes efetuados:

(a) ***Input:*** Administrador tenta adicionar uma imagem com todos os campos válidos.
***Output* esperado:** O formulário não apresentar erros.
Resultado: Passou

5. **Nome do método:** test_update_image
Descrição: Antes de executar o teste, é criada uma imagem, utilizando o método anterior (test_create_image_valid_parameter). Neste teste é verificado se um *update* efetuado numa imagem é efetuado com sucesso.
Testes efetuados:

(a) ***Input:*** É atualizado a imagem alterando o nome da imagem de 'Ubuntu' para 'CentOS'.
***Output* esperado:** Não ocorreram erros no formulário e o nome da imagem é alterado na base de dados.
Resultado: Passou

6. **Nome do método:** test_delete_image
Descrição: Antes de executar o teste, é criada uma imagem, utilizando o método anterior (test_create_image_valid_parameter). Neste teste é verificado se uma imagem é apagada com sucesso.
Testes efetuados:

(a) ***Input:*** É apagado a imagem com o id=1.
***Output* esperado:** Não ocorrem erros no formulário e a tabela imagem não contém nenhum registo de imagens.
Resultado: Passou

- ***Panel* a ser testado:** *Plans*

1. **Nome do método:** test_user_access
Descrição: É verificado se um utilizador sem privilégios de administrador consegue ter acesso ao *panel Plans Management*
Testes efetuados:

(a) ***Input:*** Utilizador normal (sem privilégios de administrador) acede à página inicial do *panel Plans Management*.
***Output* esperado:** *Redirect* para a página de *login*, de forma a obrigar o utilizador autenticar-se com uma conta de administrador.
Resultado: Passou

2. **Nome do método:** test_plan_index

   Descrição: É verificado se um administrador consegue visualizar os planos disponíveis na página inicial.

   Testes efetuados:

   (a) ***Input:*** Administrador acede à página inicial do *panel Management Plans*.
   ***Output* esperado:** O número de planos apresentadas pelo *frontend* ser igual ao número de planos que se encontram registados na base de dados.
   Resultado: Passou

3. **Nome do método:** test_planprice_history_index

   Descrição: É verificado se um administrador consegue visualizar o histórico de preços de um dado plano.

   Testes efetuados:

   (a) ***Input:*** Administrador acede à página do histórico de preços de um plano.
   ***Output* esperado:** O número do histórico de preços de um dado plano apresentadas pelo *frontend* ser igual ao número de histórico de preços que se encontram registados na base de dados.
   Resultado: Passou

4. **Nome do método:** test_create_plan_invalid_parameters

   Descrição: É verificado se um administrador consegue adicionar um plano com parâmetros inválidos.

   Testes efetuados:

   (a) ***Input:*** Administrador tenta adicionar um plano com todos os campos vazios.
   ***Output* esperado:** O formulário apresenta erros, indicando que há campos em falta.
   Resultado: Passou

   (b) ***Input:*** Administrador tenta adicionar um plano com um *id_provider* inexistente.
   ***Output* esperado:** O formulário apresenta um erro, indicando que o *id_provider* selecionado não é válido.
   Resultado: Passou

5. **Nome do método:** test_create_plan_valid_parameters

   Descrição: É verificado se um administrador consegue adicionar um plano com parâmetros válidos.

   Testes efetuados:

   (a) ***Input:*** Administrador tenta adicionar um plano com todos os campos válidos.

***Output* esperado:** O formulário não apresenta erros.
Resultado: Passou

6. **Nome do método:** test_update_plan

   Descrição: Antes de executar o teste, é criada um plano, utilizando o método anterior (test_create_plan_valid_parameter). Neste teste é verificado se um *update* efetuado num plano é efetuado com sucesso.

   Testes efetuados:

   (a) ***Input:*** É atualizado a plano alterando o seu preço de '0.05' para '0.10'.
   ***Output* esperado:** Não ocorrem erros no formulário e o preço do plano é alterado na base de dados.
   Resultado: Passou

7. **Nome do método:** test_delete_plan

   Descrição: Antes de executar o teste, é criada um plano, utilizando o método anterior (test_create_plan_valid_parameter). Neste teste é verificado se um plano é apagada com sucesso.

   Testes efetuados:

   (a) ***Input:*** É apagado a plano com o id=1.
   ***Output* esperado:** Não ocorrem erros no formulário e a tabela plano não contém nenhum registo de planos.
   Resultado: Passou

- ***Panel* a ser testado:** *Projects*

1. **Nome do método:** test_user_access

   Descrição: É verificado se um utilizador sem privilégios de administrador consegue ter acesso ao *panel Projects Management*.

   Testes efetuados:

   (a) ***Input:*** Utilizador normal (sem privilégios de administrador) acede à página inicial do *panel Projects Management*.
   ***Output* esperado:** *Redirect* para a página de *login*, de forma a obrigar o utilizador autenticar-se com uma conta de administrador.
   Resultado: Passou

2. **Nome do método:** test_project_index

   Descrição: É verificado se um administrador consegue visualizar os projetos disponíveis na página inicial.

   Testes efetuados:

   (a) ***Input:*** Administrador acede à página inicial do *panel Management Projects*.
   ***Output* esperado:** O número de projetos apresentadas pelo *frontend* deve ser igual ao número de projetos que se encontram registados na base de dados.
   Resultado: Passou

3. **Nome do método:** test_create_project_invalid_parameters

   Descrição: É verificado se um administrador consegue adicionar um projeto com parâmetros inválidos.

   Testes efetuados:

   (a) ***Input:*** Administrador tenta adicionar um projeto com todos os campos vazios.
   ***Output* esperado:** O formulário apresenta erros, indicando que há campos em falta.
   Resultado: Passou

   (b) ***Input:*** Administrador tenta adicionar um projeto com um *id_project* inexistente.
   ***Output* esperado:** O formulário apresenta um erro, indicando que o *id_project* selecionado não é válido.
   Resultado: Passou

4. **Nome do método:** test_create_project_valid_parameters

   Descrição: É verificado se um administrador consegue adicionar um projeto com parâmetros válidos.

   Testes efetuados:

   (a) ***Input:*** Administrador tenta adicionar um projeto com todos os campos válidos.
   ***Output* esperado:** O formulário não apresenta erros.
   Resultado: Passou

5. **Nome do método:** test_update_project

   Descrição: Antes de executar o teste, é criada um projeto, utilizando o método anterior (test_create_project_valid_parameter). Neste teste é verificado se um *update* efetuado num projeto é efetuado com sucesso.

   Testes efetuados:

   (a) ***Input:*** É atualizado o projeto alterando o seu *status* de *True* para *False*.
   ***Output* esperado:** Não ocorrem erros no formulário e o *status* do projeto é alterado na base de dados.
   Resultado: Passou

6. **Nome do método:** test_delete_project

   Descrição: Antes de executar o teste, é criada um projeto, utilizando o método anterior (test_create_project_valid_parameter). Neste teste é verificado se um projeto é apagada com sucesso.

   Testes efetuados:

   (a) ***Input:*** É apagado a projeto com o id=1.
   ***Output* esperado:** Não ocorrem erros no formulário e a tabela projeto não contém nenhum registo.
   Resultado: Passou

- ***Panel* a ser testado:** *Providers*

1. **Nome do método:** test_user_access

   Descrição: É verificado se um utilizador sem privilégios de administrador consegue ter acesso ao *panel Providers Management*

   Testes efetuados:

   (a) ***Input:*** Utilizador normal (sem privilégios de administrador) acede à página inicial do *panel Providers Management*.
   ***Output* esperado:** *Redirect* para a página de *login*, de forma a obrigar o utilizador autenticar-se com uma conta de administrador.
   Resultado: Passou

2. **Nome do método:** test_provider_index

   Descrição: É verificado se um administrador consegue visualizar os *providers* disponíveis na página inicial.

   Testes efetuados:

   (a) ***Input:*** Administrador acede à página inicial do *panel Management Providers*.
   ***Output* esperado:** O número de *providers* apresentados pelo *frontend* deve ser igual ao número de *providers* que se encontram registados na base de dados.
   Resultado: Passou

3. **Nome do método:** test_create_provider_invalid_parameters

   Descrição: É verificado se um administrador consegue adicionar um *provider* com parâmetros inválidos.

   Testes efetuados:

   (a) ***Input:*** Administrador tenta adicionar uma *provider* com todos os campos vazios.
   ***Output* esperado:** O formulário apresenta erros, indicando que há campos em falta.
   Resultado: Passou

   (b) ***Input:*** Administrador tenta adicionar um *provider* com um *id_provider_type* inexistente.
   ***Output* esperado:** O formulário apresenta um erro, indicando que o *id_provider_type* selecionado não é válido.
   Resultado: Passou

   (c) ***Input:*** Administrador tenta adicionar um *provider* com um *total_budget* inválido.
   ***Output* esperado:** O formulário apresenta um erro, indicando que o *total_budget* não é válido.
   Resultado: Passou

4. **Nome do método:** test_create_provider_valid_parameters

   Descrição: É verificado se um administrador consegue adicionar um *provider* com parâmetros válidos.

   Testes efetuados:

(a) ***Input:*** Administrador tenta adicionar uma *provider* com todos os campos válidos.
***Output* esperado:** O formulário não apresenta erros e o *provider* será guardado na base de dados.
Resultado: Passou

5. **Nome do método:** test_update_provider
Descrição: Antes de executar o teste, é criada um *provider*, utilizando o método anterior (test_create_provider_valid_parameter). Neste teste é verificado se um *update* efetuado num *provider* é efetuado com sucesso.
Testes efetuados:
(a) ***Input:*** É atualizado o *provider* alterando o seu *nome* de Teste para Teste3.
***Output* esperado:** Não ocorrem erros no formulário e o *nome* do *provider* é alterado na base de dados.
Resultado: Passou
6. **Nome do método:** test_delete_provider
Descrição: Antes de executar o teste, é criada uma *provider*, utilizando o método anterior (test_create_provider_valid_parameter). Neste teste é verificado se uma *provider* é apagada com sucesso.
Testes efetuados:
(a) ***Input:*** É apagado a *provider* com o id=1.
***Output* esperado:** Não ocorrem erros no formulário e a tabela *provider* não contém nenhum registo.
Resultado: Passou

- ***Panel* a ser testado:** *VMs*

1. **Nome do método:** test_user_access
Descrição: É verificado se um utilizador sem a *role deipubcloud* consegue aceder ao *panel VMs Management*
Testes efetuados:
(a) ***Input:*** Utilizador normal (sem *deipubcloud*) acede à página inicial do *panel VMs Management*.
***Output* esperado:** *Redirect* para a página de *login*, de forma a obrigar o utilizador autenticar-se com uma conta que tenha esse *role*.
Resultado: Passou
2. **Nome do método:** test_provider_index
Descrição: É verificado se um utilizador consegue visualizar as *vms* disponíveis na página inicial.
Testes efetuados:
(a) ***Input:*** Utilizador com a *role deipubcloud* acede à página inicial do *panel Management VMs*.
***Output* esperado:** O número de vms apresentados pelo *frontend* deve ser igual ao número de vms do projeto do utilizador, que se encontram registados na base de dados.
Resultado: Passou

- ***Panel* a ser testado:** *VPN Servers*

  1. **Nome do método:** test_user_access

     Descrição: É verificado se um utilizador sem o *deipubcloud* consegue aceder ao *panel VPN Servers Management*

     Testes efetuados:

     (a) ***Input:*** Utilizador normal (sem *deipubcloud*) acede à página inicial do *panel VPN Servers Management*.
     ***Output* esperado:** *Redirect* para a página de *login*, de forma a obrigar o utilizador autenticar-se com uma conta que tenha esse *role*.
     Resultado: Passou

  2. **Nome do método:** test_provider_index

     Descrição: É verificado se um utilizador consegue visualizar os servidores VPNs disponíveis na página inicial.

     Testes efetuados:

     (a) ***Input:*** Utilizador com a *role deipubcloud* acede à página inicial do *panel Management VPN Servers*.
     ***Output* esperado:** O número de servidores VPN apresentados pelo *frontend* deve ser igual ao número de servidores VPN do projeto do utilizador, que se encontram registados na base de dados.
     Resultado: Passou

Anexo E. *Scripts* de validação utilizados

Para efetuar a validação dos AQ#1, AQ#12 e AQ#13, foram realizados os seguintes *scripts*, que permitiram verificar se os respetivos atributos de qualidade definidos, estão a ser garantidos pelo módulo desenvolvido.

Script 1: Validação do AQ#1

```
from DEIPubCloudResources import deipubcloud
from datetime import datetime
from time import sleep

results = []

total = 1
success = 0
failed = 0


class VMTest:
    async_result = None
    completed = False

    def __init__(self, name, async_result):
        self.name = name
        self.async_result = async_result
        self.completed = False
        self.start_time = datetime.now()
        self.end_time = None

for act in xrange(0, total):
    vm_id = act+1

    result = deipubcloud.manage_vms.create_vm(id_project="41068
    a5aca964561b13ff42d646fb54c",
                                              id_provider=1,
                                              id_plan=1,
                                              id_image=16,
                                              name='vm{}'.format(vm_id),
                                              keypair=None,
                                              )

    results.append(VMTest(vm_id, result))
    print "Sended request to create VM number {}".format(vm_id)
    sleep(1)

print "All VMs created, waiting for results..."

while (success+failed) < total:
    for act in results:
        if not act.completed:
            if act.async_result.ready():
                if act.async_result.get():
```

```
                    success += 1
                else:
                    failed += 1

                act.completed = True
                act.end_time = datetime.now()

print "VMs created successfully: {}".format(success)
print "VMs failed: {}".format(failed)

for act in results:
    execution_time = (act.end_time - act.start_time).total_seconds()
    print('VM {}: {}'.format(act.name, execution_time))
```

Script 2: Validação do AQ#12

```
from DEIPubCloudResources import deipubcloud
from datetime import datetime
import commands
from sys import argv, exit
from time import sleep


class VMTest:
    async_result = None
    completed = False

    def __init__(self, name, async_result):
        self.name = name
        self.async_result = async_result
        self.completed = False
        self.start_time = datetime.now()
        self.end_time = None


def get_results(results_name, results):
    print("Waiting for results of {}".format(results_name))
    success = 0
    failed = 0
    total = len(results)
    while (success + failed) < total:
        for act in results:
            if not act.completed:
                if act.async_result.ready():
                    if act.async_result.get():
                        success += 1
                    else:
                        failed += 1

                    act.completed = True
                    act.end_time = datetime.now()

    print("Number of success operations: {}".format(success))
    print("Number of failed operations: {}".format(failed))

    for act in results:
```

```
41          execution_time = (act.end_time - act.start_time).total_seconds()
42          print('VM {}: {}'.format(act.name, execution_time))
43
44      print("End or results of {}".format(results_name))
45
46
47  def create_vms(total_vms):
48      for act in xrange(0, total_vms):
49          vm_id = act + 1
50
51          result = deipubcloud.manage_vms.create_vm(id_project="41068
        a5aca964561b13ff42d646fb54c",
52                                                    id_provider=1,
53                                                    id_plan=1,
54                                                    id_image=16,
55                                                    name='vm{}'.format(vm_id)
        ,
56                                                    keypair=None,
57                                                    )
58
59          results.append(VMTest(vm_id, result))
60          print "Sended request to create VM number {}".format(vm_id)
61
62      get_results('Create VMs', results)
63      vms_with_vpn_corrected_configuration = 0
64      vms_with_vpn_failed_configuration = 0
65
66      vms = deipubcloud.manage_vms.get_vms(enabled=True)
67      sleep(10)
68
69      for vm in vms:
70          status, result = commands.getstatusoutput("ping -c1 " + vm.ipv4_in)
71
72          if status == 0:
73              vms_with_vpn_corrected_configuration += 1
74          else:
75              vms_with_vpn_failed_configuration += 1
76
77      print "Successfuly ping to internal DEI IP: {}".format(
        vms_with_vpn_corrected_configuration)
78      print "Failed ping to internal DEI IP: {}".format(
        vms_with_vpn_failed_configuration)
79
80
81  def delete_vms():
82      results = []
83      vms = deipubcloud.manage_vms.get_vms(enabled=True)
84
85      for vm in vms:
86          result = deipubcloud.scheduler.destroy_vm_in_provider.delay(vm.id)
87          results.append(VMTest(vm.id, result))
88          print "Sended request to delete VM number {}".format(vm.id)
89
90      get_results('Delete VM', results)
91      total_vms_enabled = len(deipubcloud.manage_vms.get_vms(enabled=True))
92
```

```
93      if total_vms_enabled == 0:
94          print 'VMs deleted with success'
95      else:
96          print "There's still {} enabled".format(total_vms_enabled)
97
98  if len(argv) < 3:
99      print "Usage: {} number_of_vms_to_use type_of_test".format(argv[0])
100     print "type_of_test could be:"
101     print "\tcreate_vms"
102     print "\tdelete_vms"
103     exit(0)
104
105 results = []
106 success = 0
107 failed = 0
108 total = 0
109
110 try:
111     total = int(argv[1])
112 except ValueError:
113     print "Error! Second argument must be a number!"
114     exit(0)
115
116 type_of_test = argv[2]
117
118 if type_of_test == 'create_vms':
119     create_vms(total)
120 else:
121     delete_vms()
```

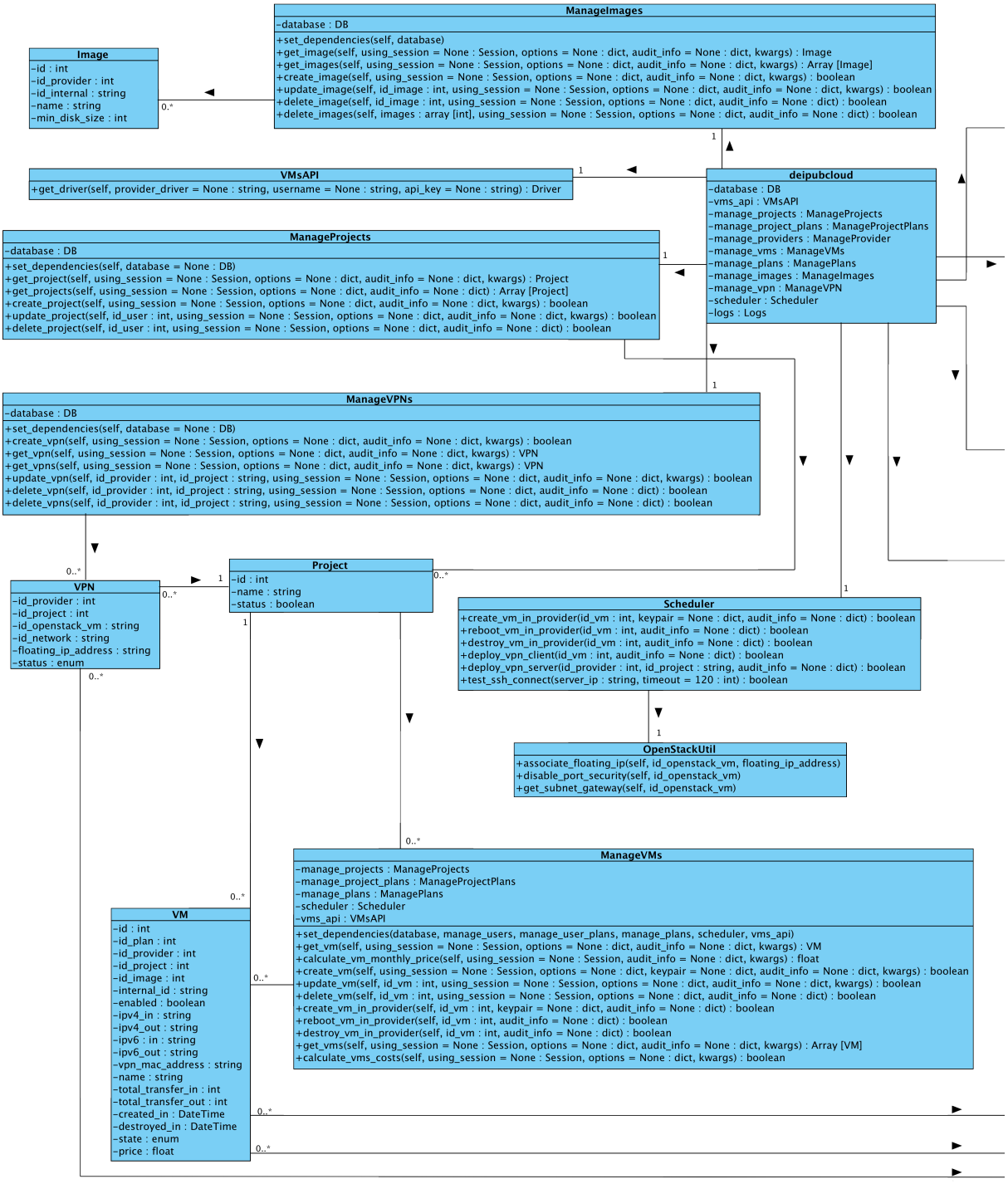
Script 3: Validação do AQ#13

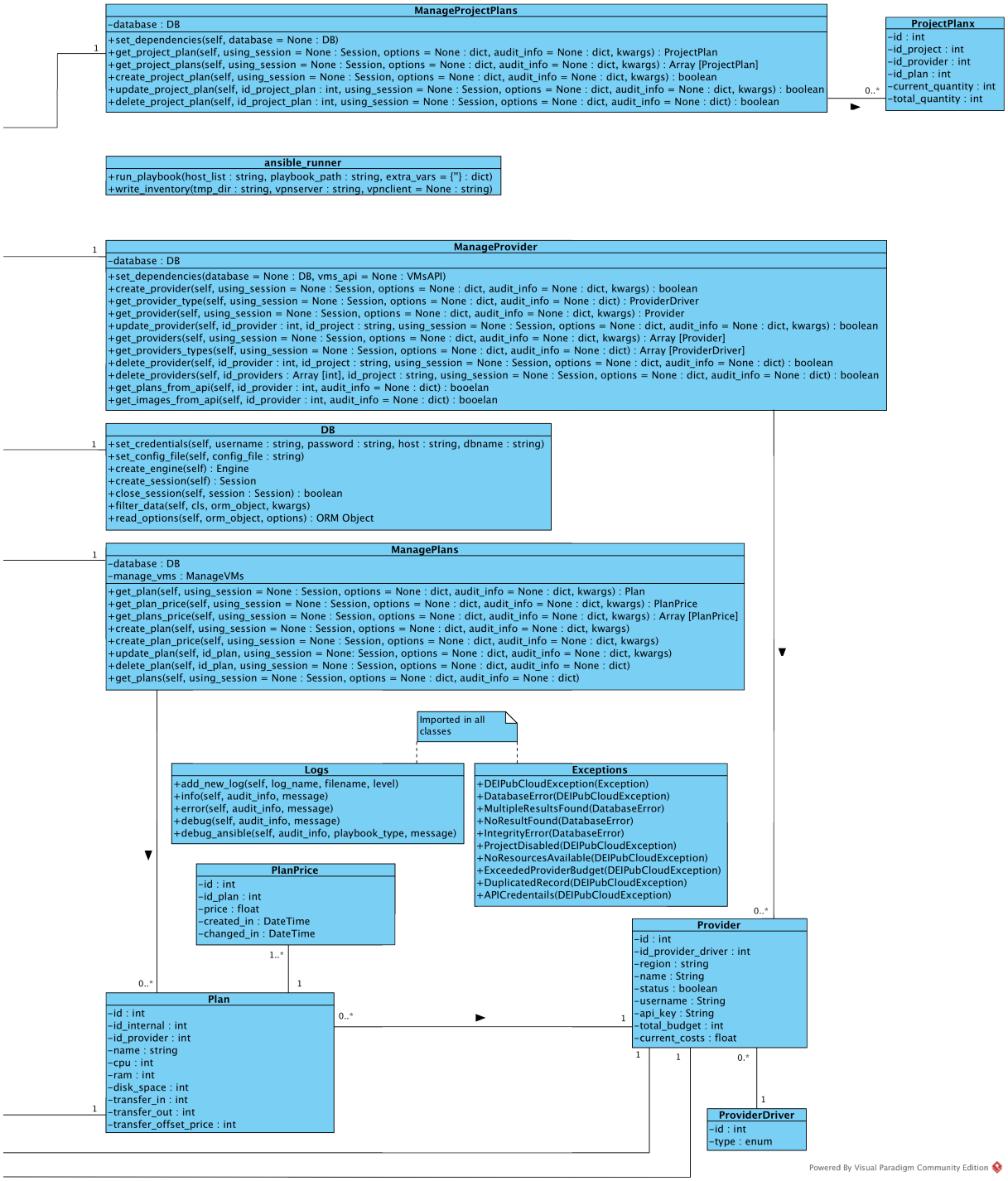
```
1  from DEIPubCloudResources import deipubcloud
2  from datetime import datetime
3  import commands
4
5  results = []
6
7
8  class VMTest:
9      async_result = None
10     completed = False
11
12     def __init__(self, name, async_result):
13         self.name = name
14         self.async_result = async_result
15         self.completed = False
16         self.start_time = datetime.now()
17         self.end_time = None
18
19
20 def get_results(results_name, results):
21     print("Waiting for results of {}".format(results_name))
22     success = 0
23     failed = 0
24     total = len(results)
```

```
25      while (success + failed) < total:
26          for act in results:
27              if not act.completed:
28                  if act.async_result.ready():
29                      if act.async_result.get():
30                          success += 1
31                      else:
32                          failed += 1
33
34                      act.completed = True
35                      act.end_time = datetime.now()
36
37      print("Number of success operations: {}".format(success))
38      print("Number of failed operations: {}".format(failed))
39
40      for act in results:
41          execution_time = (act.end_time - act.start_time).total_seconds()
42          print('VM {}: {}'.format(act.name, execution_time))
43
44      print("End or results of {}".format(results_name))
45
46  vms = deipubcloud.manage_vms.get_vms(enabled=True)
47
48  for vm in vms:
49      result = deipubcloud.manage_vms.reboot_vm(vm.id)
50
51      results.append(VMTest(vm.id, result))
52      print "Sended request to reboot VM number {}".format(vm.id)
53
54  get_results('Rebbot VM', results)
```

Anexo F. Diagrama completo de classes do *backend* do módulo *DEIPubCloud*

Este anexo representa o diagrama completo de classes do componente *backend* do módulo desenvolvido, sendo este o diagrama completo da Figura 5.5.




ManageProjectPlans
-database : DB
+set_dependencies(self, database = None : DB)
+get_project_plan(self, using_session = None : Session, options = None : dict, audit_info = None : dict, kwargs) : ProjectPlan
+get_project_plans(self, using_session = None : Session, options = None : dict, audit_info = None : dict, kwargs) : Array [ProjectPlan]
+create_project_plan(self, using_session = None : Session, options = None : dict, audit_info = None : dict, kwargs) : boolean
+update_project_plan(self, id_project_plan : int, using_session = None : Session, options = None : dict, audit_info = None : dict, kwargs) : boolean
+delete_project_plan(self, id_project_plan : int, using_session = None : Session, options = None : dict, audit_info = None : dict) : boolean
ProjectPlanx
-id : int
-id_project : int
-id_provider : int
-id_plan : int
-current_quantity : int
-total_quantity : int
ansible_runner
+run_playbook(host_list : string, playbook_path : string, extra_vars = {''} : dict)
+write_inventory(tmp_dir : string, vpnserver : string, vpnclient = None : string)
ManageProvider
-database : DB
+set_dependencies(database = None : DB, vms_api = None : VMsAPI)
+create_provider(self, using_session = None : Session, options = None : dict, audit_info = None : dict, kwargs) : boolean
+get_provider_type(self, using_session = None : Session, options = None : dict, audit_info = None : dict) : ProviderDriver
+get_provider(self, using_session = None : Session, options = None : dict, audit_info = None : dict, kwargs) : Provider
+update_provider(self, id_provider : int, id_project : string, using_session = None : Session, options = None : dict, audit_info = None : dict, kwargs) : boolean
+get_providers(self, using_session = None : Session, options = None : dict, audit_info = None : dict, kwargs) : Array [Provider]
+get_providers_types(self, using_session = None : Session, options = None : dict, audit_info = None : dict) : Array [ProviderDriver]
+delete_provider(self, id_provider : int, id_project : string, using_session = None : Session, options = None : dict, audit_info = None : dict) : boolean
+delete_providers(self, id_providers : Array [int], id_project : string, using_session = None : Session, options = None : dict, audit_info = None : dict) : boolean
+get_plans_from_api(self, id_provider : int, audit_info = None : dict) : booelan
+get_images_from_api(self, id_provider : int, audit_info = None : dict) : booelan
DB
+set_credentials(self, username : string, password : string, host : string, dbname : string)
+set_config_file(self, config_file : string)
+create_engine(self) : Engine
+create_session(self) : Session
+close_session(self, session : Session) : boolean
+filter_data(self, cls, orm_object, kwargs)
+read_options(self, orm_object, options) : ORM Object
ManagePlans
-database : DB
-manage_vms : ManageVMs
+get_plan(self, using_session = None : Session, options = None : dict, audit_info = None : dict, kwargs) : Plan
+get_plan_price(self, using_session = None : Session, options = None : dict, audit_info = None : dict, kwargs) : PlanPrice
+get_plans_price(self, using_session = None : Session, options = None : dict, audit_info = None : dict, kwargs) : Array [PlanPrice]
+create_plan(self, using_session = None : Session, options = None : dict, audit_info = None : dict, kwargs)
+create_plan_price(self, using_session = None : Session, options = None : dict, audit_info = None : dict, kwargs)
+update_plan(self, id_plan, using_session = None: Session, options = None : dict, audit_info = None : dict, kwargs)
+delete_plan(self, id_plan, using_session = None : Session, options = None : dict, audit_info = None : dict)
+get_plans(self, using_session = None : Session, options = None : dict, audit_info = None : dict)
Imported in all classes
Logs
+add_new_log(self, log_name, filename, level)
+info(self, audit_info, message)
+error(self, audit_info, message)
+debug(self, audit_info, message)
+debug_ansible(self, audit_info, playbook_type, message)
Exceptions
+DEIPubCloudException(Exception)
+DatabaseError(DEIPubCloudException)
+MultipleResultsFound(DatabaseError)
+NoResultFound(DatabaseError)
+IntegrityError(DatabaseError)
+ProjectDisabled(DEIPubCloudException)
+NoResourcesAvailable(DEIPubCloudException)
+ExceededProviderBudget(DEIPubCloudException)
+DuplicatedRecord(DEIPubCloudException)
+APICredentails(DEIPubCloudException)
PlanPrice
-id : int
-id_plan : int
-price : float
-created_in : DateTime
-changed_in : DateTime
Plan
-id : int
-id_internal : int
-id_provider : int
-name : string
-cpu : int
-ram : int
-disk_space : int
-transfer_in : int
-transfer_out : int
-transfer_offset_price : int
Provider
-id : int
-id_provider_driver : int
-region : string
-name : String
-status : boolean
-username : String
-api_key : String
-total_budget : int
-current_costs : float
ProviderDriver
-id : int
-type : enum
Powered By Visual Paradigm Community Edition

Siglas

AMQP	Advanced Message Queuing Protocol.
API	Application Program Interface.
AWS	Amazon Web Services.
AWS VPC	Amazon Web Services Virtual Private Cloud.
CISUC	Centre for Informatics and Systems.
CLI	Command-Line Interface.
CP	Cloud Platform.
CPU	Central Processing Unit.
DEI	Departamento de Engenharia Informática.
DHCP	Dynamic Host Configuration Protocol.
EXT3	Third Extended Filesystem.
FWaaS	Firewall as a Service.
GCE	Google Compute Engine.
HA	High Avaiability.
IaaS	Infraestructure as a Service.
IBM	International Business Machines.
IOPS	Input/Output Operations Per Second.
IPMI	Intelligent Platform Management Interface.
iSCSI	internet Small Computer System Interface.
KVM	Kernel-based Virtual Machine.

LACP Link Aggregation Control Protocol.

LBaaS Load Balancer as a Service.

LDAP Lightweight Directory Access Protocol.

LVM Logical Volume Manager.

NAT Network Address Translation.

NFV Network Function Virtualization.

NIC Network Interface Controller.

ORM Object-relational mapping.

OSTF OpenStack Testing Framework.

PaaS Platform as a Service.

PCS Parallels Cloud Server.

PERT Program Evaluation and Review Technique.

PKI Public Key Infraestructure.

PXE Preboot Execution Environment.

RAM Random Access Memory.

RDP Remote Desktop Protocol.

REST Representational state transfer.

SaaS Software as a Service.

SDN Software Defined Network.

SIC Serviço de Informática e Comunicações.

SMTP Simple Mail Transfer Protocol.

SQL Structured Query Language.

SR Storage Repository.

SSH Secure Shell.

VLAN Virtual Local Area Network.

VM Virtual Machine.

VMM Virtual Machine Monitor.

VNC Virtual Network Computing.

VPN Virtual Private Network.

VPNaaS VPN as a Service.

VTx Virtualization Technology.

YAML Yet Another Markup Language.

Bibliografia

[1] Abiquo. *True Hybrid Cloud - Abiquo Hybrid Cloud Software*. URL: http://www.openstack.org/ (acedido em 26/11/2015).

[2] et al. Ang Li Xiaowei Yang. «CloudCmp: Comparing Public Cloud Providers» (setembro de 2011). URL: https://users.cs.duke.edu/~xwy/publications/cloudcmp-imc10.pdf.

[3] Apache. *Apache Libcloud*. URL: https://libcloud.apache.org/ (acedido em 05/01/2016).

[4] Century Link AT&T BT et al. «Network Functions Virtualisation». *SDN and OpenFlow World Congress* (outubro de 2012). URL: http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[5] Backbone. *Backbone.js*. URL: http://backbonejs.org/ (acedido em 19/12/2015).

[6] Bernie Thompson. *Boehm's Spiral Revisited — Lean Software Engineering*. (acedido em 24/02/2016). URL: http://leansoftwareengineering.com/2008/05/05/boehms-spiral-revisited/.

[7] Bootstrap. *Bootstrap, a sleek, intuitive, and powerful mobile first front-end framework for faster and easier web development*. URL: http://getbootstrap.com/ (acedido em 19/12/2015).

[8] BT. *Network Functions Virtualisation (NFV)*. 2015. URL: http://www.globalservices.bt.com/uk/en/point-of-view/nfv (acedido em 23/10/2015).

[9] CloudStack. *CloudStack*. URL: https://cloudstack.apache.org/ (acedido em 25/10/2015).

[10] Eucalyptus. *Eucalyptus*. URL: https://www.eucalyptus.com/ (acedido em 25/10/2015).

[11] Eucalyptus. *Eucalyptus*. URL: https://www.eucalyptus.com/sites/all/files/ds-hp-helion-eucalyptus.pdf (acedido em 25/10/2015).

[12] Flexiant. *Flexiant Cloud Orchestrator - Software Features Tour*. URL: https://www.flexiant.com/flexiant-cloud-orchestrator/ (acedido em 26/11/2015).

[13] Open Networking Foundation. «Software-Defined Networking: The New Norm for Networks». *ONF White Paper* (abril de 2012). URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[14] Michael Gregg. *CISSP Exam Cram*. 3ª ed. Pearson IT Certification, nov. de 2012.

[15] Info-Tech Research Group. *Vendor Landscape: Cloud Management Solutions*. URL: https://www.citrix.com/content/dam/citrix/en_us/documents/products-solutions/vendor-landscape-cloud-management-solutions.pdf (acedido em 16/09/2015).

[16] Heroku. *Build apps on Heroku: the innovative PaaS & leading dev experience*. URL: https://www.heroku.com/platform (acedido em 26/11/2015).

[17] HuanXie. *Integrating XenServer, RDO and Neutron - Virtualization Blog*. URL: http://xenserver.org/blog/entry/integrating-xenserver-rdo-and-neutron.html (acedido em 18/12/2015).

[18] huazhihao. *citrix-openstack/xencenter-himn-plugin*. URL: https://github.com/citrix-openstack/xencenter-himn-plugin (acedido em 18/12/2015).

[19] IBM. *Cloud deployment models*. (acedido em 20/12/2015). URL: http://www.ibm.com/developerworks/websphere/techjournal/1206_dejesus/1206_dejesus.html.

[20] Amy Newman Kenneth Hess. *Practical Virtualization Solutions: Virtualization from the Trenches*. Prentice Hall/Pearson Education, 2010.

[21] Kirill Klenov. *Ansible role to install and configure OpenVPN server*. URL: https://github.com/Stouts/Stouts.openvpn (acedido em 05/04/2016).

[22] Microsoft. *SaaS Solutions — Microsoft Cloud Hub*. URL: https://technet.microsoft.com/en-us/cloud/gg697163.aspx (acedido em 26/11/2015).

[23] midonet. *midonet*. URL: http://www.midokura.com/midonet/ (acedido em 30/10/2015).

[24] Mirantis. *Fuel Plugins — OpenStack Plugins — Mirantis*. URL: https://www.mirantis.com/products/openstack-drivers-and-plugins/fuel-plugins/ (acedido em 18/12/2015).

[25] opencontrail. *opencontrail*. URL: http://www.opencontrail.org/ (acedido em 30/10/2015).

[26] OpenNebula. *OpenNebula*. 2015. URL: http://opennebula.org/ (acedido em 25/10/2015).

[27] OpenStack. *DevStack - an OpenStack Community Production*. URL: http://docs.openstack.org/developer/devstack/ (acedido em 10/12/2015).

[28] OpenStack. *Feature Support Matrix - nova 13.0.0.0b2.dev595 documentation*. URL: http://docs.openstack.org/developer/nova/support-matrix.html (acedido em 18/12/2015).

[29] OpenStack. *HypervisorSupportMatrix - OpenStack*. URL: https://wiki.openstack.org/wiki/HypervisorSupportMatrix (acedido em 18/12/2015).

[30] OpenStack. *Keystone - OpenStack*. URL: https://wiki.openstack.org/wiki/Keystone (acedido em 03/01/2016).

[31] OpenStack. *OpenStack*. URL: http://www.openstack.org/ (acedido em 26/11/2015).

[32] OpenStack. *OpenStack Ansible*. URL: http://docs.openstack.org/developer/openstack-ansible/ (acedido em 05/02/2016).

[33] OpenStack. *OpenStack Docs: Conceptual architecture*. (acedido em 03/01/2016). URL: http://docs.openstack.org/admin-guide-cloud/common/get_started_conceptual_architecture.html.

[34] OpenStack. *OpenStack Docs: Glossary*. URL: http://docs.openstack.org/ops-guide/common/glossary.html#term-project (acedido em 01/02/2016).

[35] OpenStack. *OpenStack - Fuel*. URL: https://wiki.openstack.org/wiki/Fuel (acedido em 18/12/2015).

[36] OpenStack. *OpenStack Tacker*. URL: https://wiki.openstack.org/wiki/Tacker (acedido em 30/10/2015).

[37] OpenStack. *XenServer/XenAndXenServer - OpenStack*. URL: https://wiki.openstack.org/wiki/XenServer/XenAndXenServer (acedido em 27/12/2015).

[38] OPNFV. *OPNFV*. URL: https://www.opnfv.org/ (acedido em 30/10/2015).

[39] Rackspace. *Keystone-to-Keystone Federation with the openstack-ansible Project*. URL: https://developer.rackspace.com/blog/keystone-to-keystone-federation-with-openstack-ansible/ (acedido em 05/12/2015).

[40] Kai Seidler. *Urban legends: Apache reload(ed) (Oswald@Work)*. URL: https://blogs.oracle.com/oswald/entry/urban_legends_apache_reload_ed (acedido em 05/06/2016).

[41] Wei Chen, Hongyi Lu, Li Shen et al. «A Novel Hardware Assisted Full Virtualization Technique». *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference* (nov. de 2008). URL: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4709159.

[42] virtustream. *xStream Cloud Management — Virtustream*. URL: http://www.virtustream.com/cloud-software/xstream (acedido em 26/11/2015).

[43] VMware. «Understanding Full Virtualization, Paravirtualization, and Hardware Assist» (nov. de 2007). URL: http://www.vmware.com/resources/techresources/1008.

[44] VMware. *vCloud Air Infrastructure as a Service (IaaS) Hybrid Cloud*. URL: http://www.vmware.com/cloud-services/infrastructure (acedido em 30/10/2015).

[45] VMware. *vCloud Suite*. 2015. URL: https://www.vmware.com/products/vcloud-suite/features.html (acedido em 23/10/2015).

[46] VMware. *VMware NSX*. URL: https://www.vmware.com/products/nsx (acedido em 30/10/2015).

[47] VMware. *vRealize Suite Cloud Management Platform: VMware*. URL: http://www.vmware.com/products/vrealize-suite/ (acedido em 20/12/2015).

[48] Wikipedia, the free encyclopedia. *Protection ring*. (acedido em 26/11/2015). URL: https://en.wikipedia.org/wiki/Protection_ring#/media/File:Priv_rings.svg.