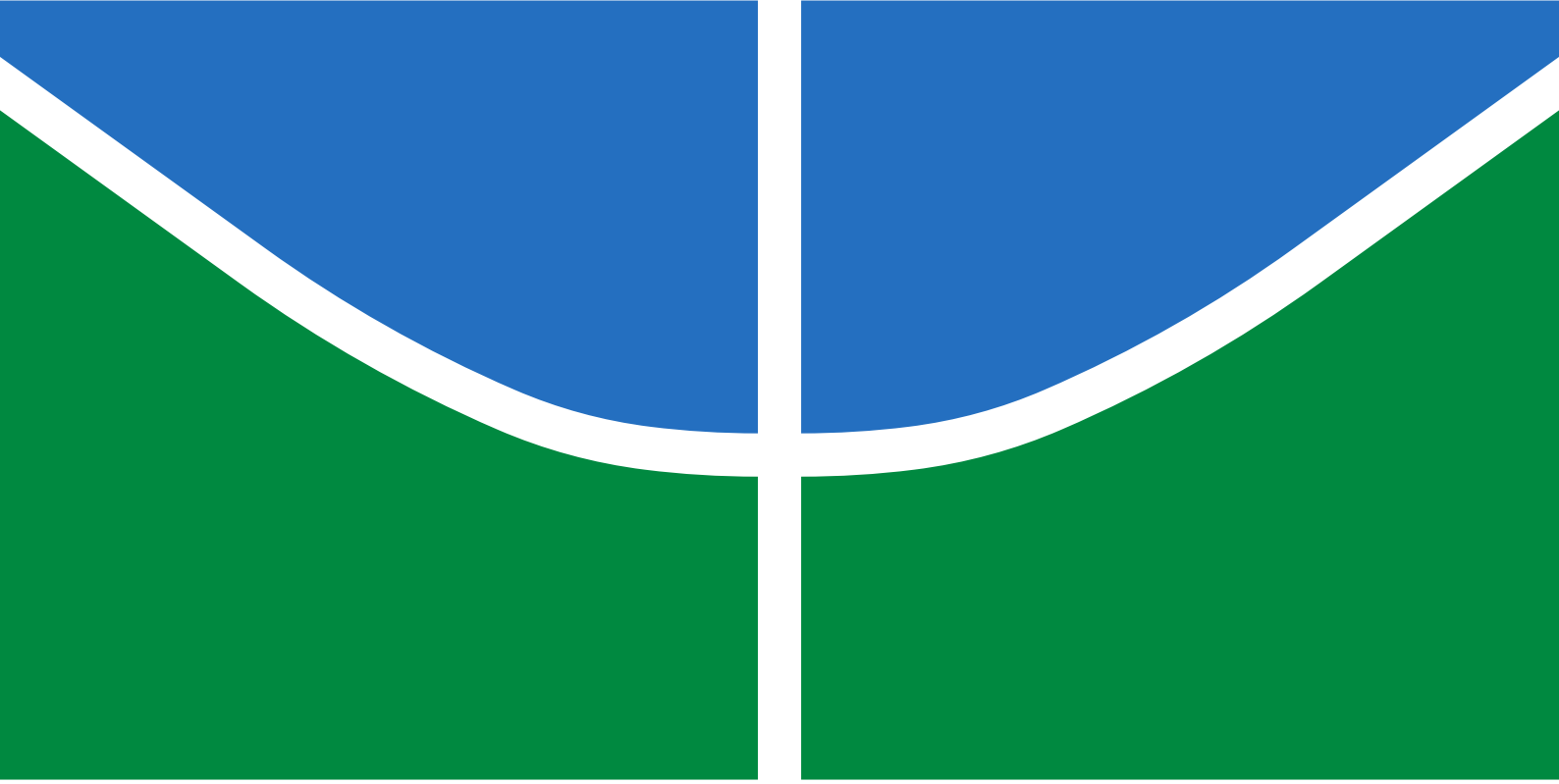


Universidade de Brasília - UnB

Faculdade de tecnologia - FT

Engenharia de Redes de Comunicação

Data center seguro e definido por software: Um estudo de caso para armazenamento seguro e altamente disponível em ambiente de nuvem

Autor: Caio Vitor da Silva Moreira; Rafael Moraes Monteiro

Orientador: Prof. Dr. Rafael Timóteo de Sousa Jr.

Coorientador: Prof. Dr. Robson de Oliveira Albuquerque

Brasília, DF

2019

Caio Vitor da Silva Moreira; Rafael Moraes Monteiro

Data center seguro e definido por software: Um estudo de caso para armazenamento seguro e altamente disponível em ambiente de nuvem

Monografia submetida ao curso de graduação em Engenharia de Redes de Comunicaçãoda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Redes de Comunicação.

Universidade de Brasília - UnB

Faculdade de tecnologia - FT

Orientador: Prof. Dr. Rafael Timóteo de Sousa Jr.

Coorientador: Prof. Dr. Robson de Oliveira Albuquerque

Brasília, DF

2019

Caio Vitor da Silva Moreira; Rafael Moraes Monteiro

Data center seguro e definido por software: Um estudo de caso para armazenamento seguro e altamente disponível em ambiente de nuvem/ Caio Vitor da Silva Moreira; Rafael Moraes Monteiro. – Brasília, DF, 2019-

101 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Rafael Timóteo de Sousa Jr.

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade de tecnologia - FT , 2019.

1. engenharia de redes de comunicação. 2. software defined networks (SDN). 3. data center. 4. virtualização. I. Prof. Dr. Rafael Timóteo de Sousa Jr. . II. Universidade de Brasília. III. Faculdade de Tecnologia - FT. IV. Data center seguro e definido por software: Um estudo de caso para armazenamento seguro e altamente disponível em ambiente de nuvem

CDU 02:141:005.6

Caio Vitor da Silva Moreira; Rafael Moraes Monteiro

Data center seguro e definido por software: Um estudo de caso para armazenamento seguro e altamente disponível em ambiente de nuvem

Monografia submetida ao curso de graduação em Engenharia de Redes de Comunicaçãoda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Redes de Comunicação.

Brasília, DF, 13 de Dezembro de 2019:

Prof. Dr. Rafael Timóteo de Sousa Jr.
Orientador

Prof. Dr. Robson de Oliveira Albuquerque
Convidado 1

Prof. MsC. Francisco Lopes de Caldas Filho
Convidado 2

Prof. Dr. Alexandre Nery Solon
Convidado 3

Brasília, DF
2019

Agradecimentos

Agradeço primeiramente aos meus pais, que sempre se preocuparam em proporcionar condições para o meu crescimento profissional e, sobretudo, me proporcionaram amor incondicional. Agradeço também às orientações e conselhos dos líderes do projeto UIoT, Francisco, Lucas e Rafael Timóteo, e a imensa paciência de nosso coorientador, Robson. Agradeço também ao meu amigo Rafael, que se dedicou de corpo e alma ao projeto, me ajudou nos momentos difíceis, e sem o qual eu não teria chegado até aqui.

Caio Moreira

O primeiro agradecimento será a minha mãe, pessoa que mais amo no mundo, tendo por mim um grande amor e carinho, me ajudando e apoiando nos momentos felizes e importantes de minha vida. Obrigado por tudo mãe! Agradeço de coração ao meu grande amigo Caio, sempre me suportando, me ensinando a ser um ser humano melhor à cada dia. Não conseguiria ter realizado esse trabalho sem sua ajuda, por isso, obrigado por ser tão bom comigo e me ajudado a crescer. Agradeço aos meus orientadores e conselheiros Rafael Timóteo de Sousa Jr., Lucas Martins, Francisco Lopes e Robson Albuquerque, em especial aos dois últimos por toda a atenção e paciência que tiveram durante o projeto. Os últimos agradecimentos são aos meus amigos e familiares que tem um grande carinho e amor por mim, muito obrigado à todos.

Rafael Moraes

Este projeto contou com apoio da Agência brasileira de pesquisa, desenvolvimento e inovação Fundação de Apoio à Pesquisa do Distrito Federal FAPDF (Projetos UIoT 0193.001366/2016 e SSDDC 0193.001365/2016), bem como do Gabinete de Segurança Institucional da Presidência da República (TED 002/2017) e do Laboratório LATITUDE/UnB (Projeto SDN 23106.099441/2016-43), sendo de extrema importância para o desenvolvimento deste trabalho.

Resumo

Na última década, observou-se um crescimento no tamanho e complexidade de sistemas computacionais especializados no tratamento de dados, os data centers. O aumento na complexidade, principalmente, levou administradores de tecnologia da informação a repensarem o design de data centers. Modelos tradicionais vinculavam a computação e os dados de aplicações à servidores e sistemas de armazenamento específicos. Tal rigidez no provisionamento acarreta em um dimensionamento muitas vezes não acurado, tornando o data center dispendioso e energeticamente ineficiente.

No entanto, o cenário de data centers inflexíveis começa a mudar com o avanço de técnicas de virtualização de computação, seguida de avanços no estudo das redes definidas por software. O desacoplamento do plano de controle do hardware presente em um data center começa a possibilitar a implementação de softwares que tomam as decisões que antes cabiam ao administrador. Por isso, definiu-se o *Software-defined data center - SDDC* (data center definido por software).

Desta forma, este trabalho aplica conceitos de *Software-defined data center* - SDDC em um ambiente de testes e faz a implantação de uma nuvem privada, definindo e provendo serviços de *Infrastructure as a Service* - IaaS (Infraestrutura como serviço). Com a IaaS implementada, é proposto um estudo de caso para solucionar um problema de armazenamento seguro e distribuído em ambientes de nuvem privada.

Utilizou-se a plataforma OpenStack para virtualização do data center e disponibilização de IaaS. Com esse cenário executado nos dois data centers, o *software* GlusterFS foi utilizado para construção de um volume NAS cifrado e disponibilizado na rede. Uma VPN foi implantada, com o *software* Wireguard, para garantir segurança no transporte de dados entre os data centers. Por fim, foi implementado um serviço de armazenamento na nuvem, provido pelo Nextcloud, para fazer uso da solução proposta de armazenamento seguro e altamente disponível.

Por fim, alguns aspectos da solução foram testados. Com um analisador de pacotes analisando o tráfego entre os data centers, mostrou-se que a túnel VPN foi implementado corretamente e os dados estão sendo transportados cifrados. Foi também mostrado que o GlusterFS implementa segurança de chaves para cifragem dos dados no volume NAS, adicionando um nível de segurança aos dados armazenados.

Palavras-chaves: engenharia de redes, software defined networks (SDN), data center, virtualização, armazenamento em nuvem, nuvem segura.

Abstract

In the last decade, there has been a growth in the size and complexity of data-processing specialized computer systems. Increasing complexity has led IT managers to rethink data center design. Traditional models linked application's computing and data to specific servers and storage systems. Such rigid provisioning is oftern translated to inaccurate scaling, making the data center costly and energy inefficient.

However, the landscape of inflexible data centers is beginning to change with the advancement of computing virtualization techniques, followed by advances in the study of software-defined networks. Decoupling the control plan from the hardware in a data center begins to make it possible the implementation of some software that makes the decisions that were previously up to the administrator. Therefore, the concept of Software-defined data center - SDDC was defined.

Thus, this work applies Software-defined data center (SDDC) concepts in a testing environment and deploys a private cloud, defining and providing Infrastructure as a Service (IaaS). With IaaS in place, a case study is proposed to solve a distributed and secure storage problem in private cloud environments.

The OpenStack platform was used for data center virtualization and IaaS availability. With this scenario installed in the distributed data centers, GlusterFS software was used to build an encrypted NAS volume and made it available on both data centers through the network. A VPN was deployed with Wireguard software to ensure secure data transport between data centers. Finally, a Cloud storage service, provided by Nextcloud, was implemented to make use of the proposed secure and replicated storage solution.

Finally, some aspects of the solution were tested. With a packet sniffer analysing data between data centers, it has been shown that the VPN tunnel has been implemented correctly and the data is being transported with encryption. It was also showned the GlusterFS implementation of secure volumes, with pre-shared key security for encryption of data on the NAS volume, adding a level of security to the stored data.

Key-words: network engineering, software defined networks (SDN), data center, virtualization, cloud storage, secure cloud.

Sumário

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	- *Application Programming Interface*: Interface de programação de aplicação
MMV	- *Virtual Monitor Machine*: Monitor de Maquina Virtual
VM	- *Virtual Machines*: Máquinas Virtuais
CPU	- *Central Processing Unit*: Unidade Central de Processamento
IoT	- *Internet of Things*: Internet das Coisas
ZIGBEE	- *Zonal Intercommunication Global-Standard. With long Battery life, Economical to deploy and Efficient*: Padrão de Intercomunicação por Zonas. Com longo ciclo de bateria, implantação econômica e eficiente.
MQTT	- *Message Queuing Telemetry Transport*: Transporte de Telemetria do Serviço de Enfileiramento de Mensagens
IaaS	- *Infrastructure as a Service*: Infraestrutura como Serviço
PaaS	- *Plataform as a Service*: Plataforma como Serviço
SaaS	- *Software as a Service*: *Software* como Serviço
STaaS	- *Storage as a Service*: Armazenamento como Serviço
DNS	- *Domain Name System*: Sistema de Domínios de Nome
NTP	- *Network Time Protocol*: Protocolo de Tempo de Rede
VCPU	- *Virtual Central Processing Unit*: Unidade Central de Processamento Virtual
URL	- *Uniform Resource Locator*: Buscador de Recursos Uniformes
DAS	- *Direct-attached storage*: Armazenamento Diretamente Acoplado
SAN	- *Storage Area Network*: Redes de Armazenamento
NAS	- *Network-attached Storage*: Armazenamento Acoplado por Rede
OSI	- *Open Systems Interconnection Model*: Modelo Aberto de Interconexões de Sistemas

CNSS - *Committee on National Security Systems*: Comitê de Sistemas de Segurança Nacional

CIA - *Confidenciality, Integrity and Availability*: Confidencialidade, Integridade e Disponibilidade

VPN - *Virtual Private Network*: Rede Virtual Privada

IP - *Internet Protocol*: Protocolo da Internet

TCP - *Transmission Control Protocol*: Protocolo de Controle de Transmissão

UDP - *User Datagram Protocol*: Protocolo de Datagrama de Usuário

TCP/IP - *Transmission Control Protocol and Internet Protocol*: Protocolo de Controle de Transmissão e Protocolo da Internet

IPSec - *Internet Protocol Security*: Protocolo de Segurança do Protocolo da Internet

L2TP - *Layer 2 Tunnelling Protocol*: Protocolo de Tunelamento de Camada 2

L2F - *Layer 2 Forwarding Protocol*: Protocolo de Encaminhamento de Camada 2

PPP - *Point-to-Point Protocol*: Protocolo Ponto-a-Ponto

PPTP - *Point-to-Point Tunneling Protocol*: Protocolo de Tunelamento Ponto-a-Ponto

GRE - *Generic Routing Encapsulation*: Encapsulamento de Roteamento Genérico

VHD - *Virtual Hard Disk*: Disco de Rígido Virtual

VMDK - *Virtual Machine Disk*: Disco de Máquina Virtual

QCOW2 - *Quick Emulator Copy On Write*: Emulador Rápido de Cópia em Escrita

RAW - *RAW*: Disco Bruto

VDI - *Virtual Desktop Infrastructure*: Infraestrutura de Desktop Virtual

ISO - *International Organization for Standardization*: Organização Internacional para Padronização

AMI	- *Amazon Machine Image*: Imagem de Máquina da Amazon
ARI	- *Amazon RawDisk Image*: Imagem de Disco Bruto da Amazon
AKI	- *Amazon Kernel Image*: Imagem do Kernel da Amazon
CLI	- *Command-line Interface*: Interface de linha de Comando
KVM	- *Kernel-based Virtual Machine*: Máquina Virtual baseada em Kernel
SQL	- *Structured Query Language*: Linguagem de Consulta Estruturada
VLAN	- *Virtual Local Area Network*: Área de Rede Local Virtual
DHCP	- *Dynamic Host Configuration Protocol*: Protocolo de Configuração Dinâmica de Host
VXLAN	- *Virtual Extensible Local Area*: Área Local Extensível Virtual
NAT	- *Network Address Translation*: Tradução do Endereço da Rede
IPv4	- *Internet Protocol version 4*: Protocolo da Internet versão 4
IPv6	- *Internet Protocol version 6*: Protocolo da Internet versão 6
RTT	- *Round Trip Time*: Tempo de Ida e Volta
RHEL	- *Red Hat Enterprise Linux*: Linux Empresarial da Red Hat
RAM	- *Random Access Memory*: Memória de Acesso Aleatório
SSH	- *Secure Socket Shell*: Socket Seguro em Shell
OVS	- *Open Virtual Switch*: Switch Virtual Aberto
QoS	- *Quality of Service*: Qualidade de serviço
CIDR	- *Classless Inter-Domain Routing*: Encaminhamento Inter-Domínio sem Classe
RSA	- *Rivest-Shamir-Adleman*: Rivest-Shamir-Adleman
ISCII	- *Indian Script Code for Information Interchange*: Código de *Script* Indiano para Intercâmbio de Informações
ppa	- *Personal Package Archives*: Arquivos de Pacotes Pessoais
OS	- *Operating System*: Sistema Operacional
xfs	- *Extended Filesytem*: Sistema de Arquivos Estendido

ASCII	- *American Standard Code for Information Interchange)*: Codificação americana padrão para intercâmbio de informação
I/O	- *Input or Ouput*: Entrada ou Saída
R/W	- *Read or Write*: Leitura ou Escrita

1 Introdução

O crescimento em tamanho e complexidade de data centers na última década levou administradores a repensarem o design de data centers. Modelos tradicionais vinculavam a computação e os dados de aplicações à servidores e sistemas de armazenamento específicos (SINGH; KORUPOLU; MOHAPATRA, 2008). Esse vínculo rígido faz com que administradores super provisionem recursos para lidar com picos de carga de trabalho ou falhas inesperadas. Esse tipo de configuração faz com que data centers se tornem dispendiosos e energeticamente ineficientes.

No entanto, a melhora em ferramentas de virtualização, redes definidas por software e pilhas de infraestrutura convergente tem possibilitado o que Richard Fichera (2012) chama de SSDDC - *Software-Defined Data Center* (Data Center Definido por Software) - uma abstração abrangente de data centers no modelo tradicional. Em um SDDC, recursos são dinamicamente descobertos, provisionados e configurados baseados nos requerimentos da carga de trabalho (FORCE, 2015).

Muitas empresas que provem serviços de data center têm investido grandes quantidades de recursos financeiros para desenvolver e implantar no mercado o conceito de data center definido por software, principalmente no que se refere aos três principais pilares ou camadas que permitem a ofertas de Infraestrutura como Serviço IaaS: a de virtualização de servidores no processamento, de uso de modelos lógicos e seguros de armazenamento ou *Software Defined Storage* (SDS), e de redes virtualizadas em ambientes definidos e gerenciados por software ou *Software Defined Network* (SDN). O SDDC possibilita que se controle todo o hardware pertencente a um Data Center (servidores, storages, etc.) através de um único software centralizado (HP, 2013), o que permite que se faça um gerenciamento de serviços muito mais eficaz, e proporciona um nível de escalabilidade que o modo tradicional não consegue atingir.

A virtualização de recursos e criação de nuvem pública ou privada, porém, não resolve todos os problemas. Usuários de nuvens públicas, principalmente em modelos de IaaS, ainda precisam se preocupar com aspectos importantes de segurança, alta disponibilidade, entre outros (RAMGOVIND; ELOFF; SMITH, 2010). Em especial, o armazenamento de dados sensíveis pode ser um fator determinante para clientes privados e públicos na adoção de modelos de nuvem. Na Figura 1.1, é possível observar o crescimento do interesse no uso dos modelos de nuvem nos anos de 2016 à 2018.

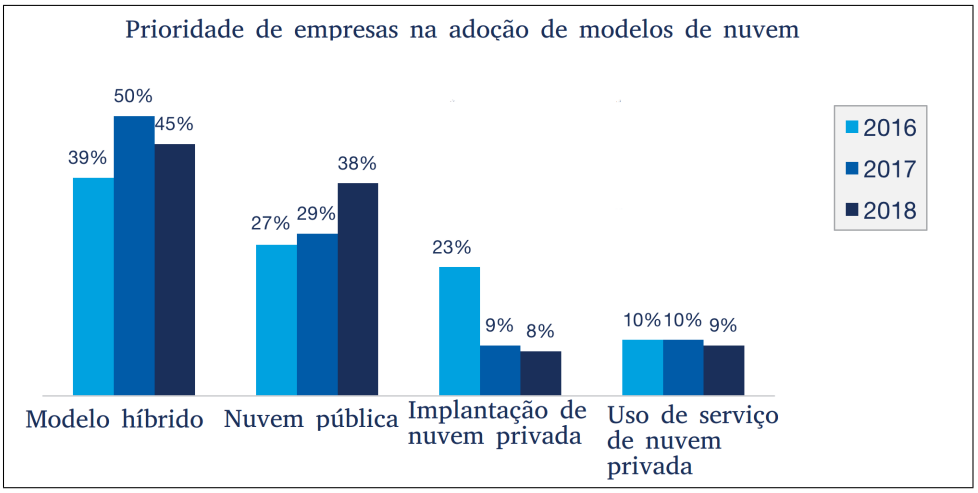


Figura 1.1 – Adoção de modelos de nuvem, nos anos 2016-2018, de acordo com Ani Miteva (2018).

1.1 Definição do problema

Em ambientes de data center centralizados ou geograficamente distribuídos, existem diversas medidas a serem tomadas de proteção aos dados armazenados e em trânsito. Essa é, normalmente, uma preocupação das equipes que gerenciam a infraestrutura física de um data center.

Em um ambiente de nuvem privada, porém, algumas preocupações diferentes surgem: É necessário garantir a segurança de dados armazenados e a segurança dos dados em transporte durante a replicação, mesmo em uma infraestrutura física que você não controla.

1.2 Objetivo Geral

O objetivo geral deste projeto é implementar um sistema que permita a virtualização e automação de recursos de data centers, viabilizando uma solução segura de armazenamento de dados em ambientes distribuídos de nuvem privada.

Objetivos específicos

Com a implementação de tais conceitos, efetivamente estará sendo implementando um data center seguro definido por *software*. Para alcançar esse objetivo, os seguintes objetivos específicos são propostos:

- Implementar uma prova de conceito de virtualização de data center, utilizando a plataforma OpenStack.

- Configurar uma infraestrutura lógica, com a plataforma OpenStack, quepossibilite a alocação de recursos com critérios de segurança para nuvens em modelo de Infraestrutura como Serviço (IaaS).
- Utilizando a infraestrutura lógica virtualizada pelo OpenStack, implementar uma solução de armazenamento lógico, geograficamente distribuído e seguro.
- Disponibilizar um serviço de *Cloud* (armazenamento de arquivos na nuvem) que utilize infraestrutura virtualizada, com alta disponibilidade de dados e com medidas de segurança do dado em transporte e armazenado.

1.3 Organização do trabalho

Os capítulos seguintes desse trabalho são organizados da seguinte forma: O capítulo 2 traz uma discussão acerca dos conceitos básicos que compões este trabalho; O capítulo 3 explica o problema abordado, mostra a topologia física e lógica da solução proposta e explica algumas ferramentas utilizadas na solução; O capítulo 4 define os níveis de serviço propostos pelo SSDDC e mostra como é feita a implementação segura de cada um; O capítulo 5 faz uma análise e discussão da segurança que essa solução propõe, além de alguns testes de impacto da segurança na performance do SDDC. Finalmente o capítulo 6 faz uma discussão final da solução e propõe pontos que ainda podem ser explorados em trabalhos futuros.

2 Fundamentação teórica

Para a compreensão das dimensões do problema abordado e dos conceitos utilizados na proposta de solução, é necessário garantir que alguns pontos fundamentais da teoria foram entendidos completamente. Esse capítulo, portanto, traz uma revisão dos fundamentos teóricos necessários.

2.1 Arquitetura de Data Centers

O grande avanço da internet nos anos 90 gerou uma demanda de infraestrutura de comunicação que operasse continuamente e permitisse que grandes empresas mantivessem constante sua participação online (KAYLIE GYARMATHY, 2019). Empresas de médio e grande porte começaram a investir em equipamentos e organizar instalações que pudessem prover resolver essa demanda de serviços online.

Dessa forma, data centers começaram a se formar, centralizando as operações de tecnologia da informação – TI. Conforme mencionado em Cisco (2014), data centers agregam ativos de rede, *firewalls*, sistemas de armazenamento, servidores de computação e controladores de entrega de aplicação. A arquitetura de data center (*layout* físico e lógico de recursos e equipamentos dentro das instalações) pode ser dividida em três componentes nucleares: servidores, armazenamento e redes.

Servidores

O Instituto Americano de Padrões de Tecnologia (*National Institute of Standarts and Technology* - NIST) define servidores como sistemas que fornecem serviços em resposta à requisições de clientes (GRANCE et al., 2002). Essa arquitetura é chamada de modelo cliente-servidor.

Servidores podem prover diversos tipos de serviços, como e-mail, páginas *Web*, autenticação, etc. Também podem estar agregados em grupos computacionais, ou *clusters*, ou servindo individualmente requisições de clientes (Bradley Mitchell, 2019). Servidores fazem uso da rede, seja ela local ou a internet, para receber e responder requisições de clientes. Além disso, é possível que servidores utilizem a rede para armazenar dados em *hardware* externo ao próprio servidor.

Armazenamento

Storage (armazenamento) de data center é um termo usado para definir ferramentas, tecnologias e processos utilizados para desenhar, implementar e gerenciar recursos computacionais especializados em armazenar dados digitais (Patrick Moorhead, 2018).

As especificidades de componentes computacionais especializados em armazenar dados, porém, têm passado por mudanças nos últimos anos. Como cita Leventhal (2008), os *Storages* tem migrado de grandes grupos de discos magnéticos, conhecidos como *Hard Disk Drives*, para matrizes de discos de estado sólido, mais conhecidos como *Solid state drives*.

Redes

O propósito principal de um data center é acomodar múltiplas aplicações com variadas características e cargas de trabalho (SDxCentral Staff, 2016). Uma distribuição de servidores está, a todo momento, preparada para responder requisições de clientes com o serviço prestado. Nesse processo, a rede tem um papel fundamental de infraestrutura base para que esse processo cliente-serviço aconteça. A rede fornece aos servidores, clientes, aplicações e *middleware* um mapa compartilhado para que a comunicação aconteça entre todos os ativos.

Data centers podem conter milhares de dispositivos servidores requisitando comutação de recursos que demandam banda de rede. A topologia clássica de rede em data center consiste em elementos de *switching* e *routing* em *tiers* (camadas), como mostrada na Figura 2.1, com equipamentos progressivamente mais especializados e caros, subindo na hierarquia de rede (HAMMADI; MHAMDI, 2014). Como é a rede que conecta todos os componentes do data center, é essencial que sua arquitetura seja bem pensada para suportar todo o tráfego gerado interna e externamente. De acordo com Al-Fares, Loukissas e Vahdat (2008), um dos maiores desafios de *clusters* [1] de larga escala é o afunilamento de banda de rede em comunicações entre nós de um mesmo *cluster*.

Data centers são, em resumo, uma composição entre servidores (de computação e armazenamento) e infraestrutura de redes. De acordo com Greenberg et al. (2008), o lado dos servidores, porém, já se encontram bem a frente no que se refere a comoditização: servidores empresariais superespecializados e caros foram substituídos por um grande número de servidores simples e não-especializados. Os avanços em computação distribuída e sistemas de gerenciamento de recursos permitiram que esses servidores não-especializados pudessem ser agregados de forma confiável e redundante. As infraestruturas de rede, agora, apresentam o próximo grande passo na comoditização de data centers.

[1] Clusters são agrupamentos de servidores interligados que funcionam em sincronia para execução performática de uma ou mais tarefas.

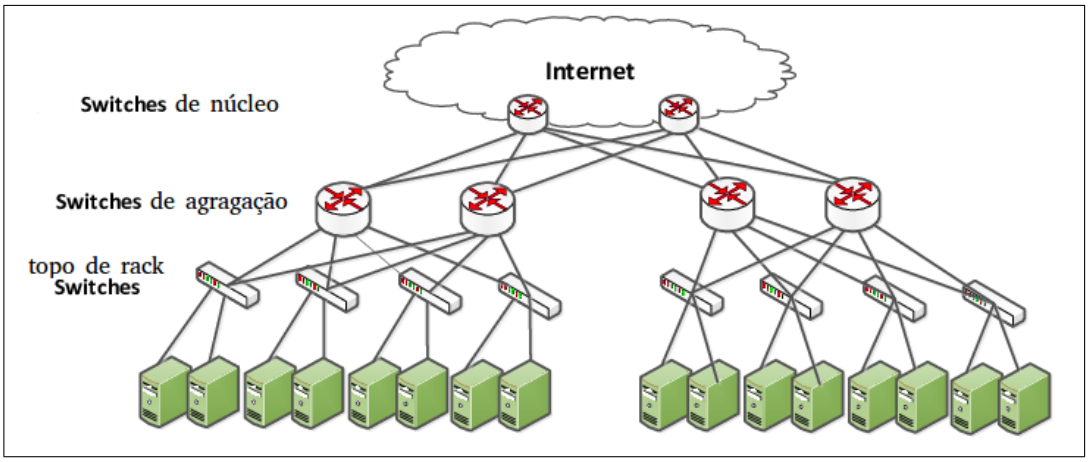


Figura 2.1 – Arquitetura de redes comumente encontrada em data centers

2.2 Virtualização e Hipervisores

De forma geral, um sistema de virtualização é uma metodologia para divisão de recursos computacionais em múltiplos ambientes, utilizado-se técnicas ou tecnologias tais como particionamento de *hardware*, compartilhamento de recursos, simulação de máquinas completas ou parciais, emulação e outros. Aplica-se a uma gama de recursos de infraestrutura de TI, tais como subsistemas de armazenamento, elementos de rede (roteadores, *switches* e *firewalls*) e servidores. (NEIVA, 2010)

As primeiras definições de virtualização datam do início da década de 60, seguidos pelos primeiros lançamentos comerciais com a empresa IBM. Detentora de equipamentos *mainframe* com alto poder de computação, a empresa percebeu a oportunidade de otimizar ainda mais seus já robustos *mainframes* que, para grande parte das implantações, era subutilizado (FISHER-OGDEN, 2006). Para tanto, técnicas de compartilhamento de recursos computacionais foram desenvolvidas, possibilitando que um sistema tratasse requisições de mais de um usuário simultaneamente.

De forma similar, a virtualização de armazenamento possibilita a criação de discos virtuais que podem ser, separadamente, utilizados por aplicações. Essa camada de abstração entre o disco de armazenamento físico e aplicações permite que os recursos de armazenamento sejam agrupados de uma forma global, entre diferente fornecedores e diferentes localizações (desde que se comuniquem pela rede local ou global). (SINGH; KORUPOLU; MOHAPATRA, 2008)

A virtualização, no entanto, precisou superar algumas barreiras antes de conseguir abstrair completamente o hardware e entregar ao usuário uma máquina virtual que se comportasse como um *hardware* físico. O maior desafio surge porque sistemas operacionais foram projetados para ter total controle sobre o *hardware* na qual rodam e, normalmente, não suportam o compartilhamento de recursos de *hardware* (HUMPHREYS; GRIESER,

2006). É então criado o hipervisor (ou Monitor de Máquinas Virtuais - MMV), cuja principal função é fazer com que os sistemas operacionais continuem a acreditar que possuem controle total do *hardware*.

Popek e Goldberg (1974) cita que hipervisores podem ser classificados em 2 grupos:

- Hipervisor nativo: Os hipervisores possuem o mais alto nível de privilégio nas instruções de *hardware*. Com esse privilégio, é entregue as máquinas virtuais VM - (*Virtual Machines*) dispositivos de CPU virtual (*Central Process Unit* - CPU), memória virtual e disco virtual. O sistema operacional instalado em uma máquina virtual passa a ser chamado de sistema operacional hóspede.
- Hipervisor hóspede: Nesse caso, os hipervisores são instalados em cima de um sistema operacional, que possui o mais alto nível de privilégio de instruções de *hardware*. Apesar do hipervisor precisar de requisitar instruções ao sistema operacional que o hospeda, sua função de abstração continua similar aos hipervisores nativos: As VMs hospedadas no hipervisor hospedeiro continuam ignorantes às camadas entre elas e o hardware.

A Figura 2.2 exemplifica a arquitetura dos dois tipos de hipervisores citados por Popek e Goldberg (1974).

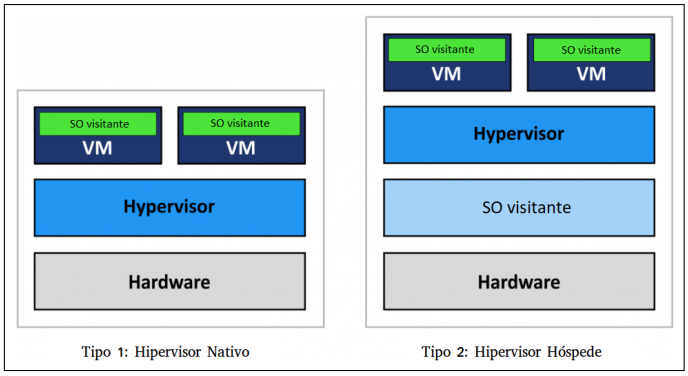


Figura 2.2 – Exemplo de arquiteturas de hipervisores tipo 1 e tipo 2. Fonte: <www.ibm.com>

2.3 *Cloud Computing*

Diversos autores definiram computação em nuvem e, ainda assim, há algumas divergências de conceitos. Foster et al. (2008) aborda o tema levando em consideração os aspectos mais comumente observados em nuvens públicas e privadas existentes:

> " *Paradigma de computação distribuída em larga escala, na qual um conglomerado de recursos abstratos, virtualizados e dinamicamente escaláveis de com-*

> *putação, armazenamento, plataforma e serviços, é entregue ao usuário sob demanda através da internet.* "(FOSTER et al., 2008)

É possível fazer algumas observações sobre essa definição. Primeiramente, computação em nuvem deve ser estruturada de forma distribuída. Isso é, o provedor da nuvem precisa entregar ao usuário, através da internet, aplicações processadas remotamente, plataformas de serviço e possibilidade de alocação de infraestrutura computacional, de armazenamento e de rede. Esses são alguns dos famosos modelos de serviço de nuvem. O tópico 2.3.1 trata do assunto com mais detalhes.

Ainda sobre a definição de Foster et al. (2008), é estabelecido a necessidade de um conglomerado de recursos na qual o usuário pode demandar poder computacional, de armazenamento e de rede. Como mencionado em Greenhalgh et al. (2009), *hardware* está, aos poucos, se tornando *commodity*, ou seja, as especificações de *hardware* estão perdendo importância frente aos serviços entregues. Sobretudo em um ambiente de nuvem, o usuário não deve se preocupar com equipamentos físicos. Através de uma interface simples, o provedor de serviços de nuvem deve possibilitar que o usuário requisite máquinas virtualizadas, com um limite de recurso definível e escalabilidade automática (SOUSA; MOREIRA; MACHADO, 2009).

Existe também uma definição comercial de computação na nuvem, criada em 2011 pelo Instituto de padronizações e tecnologias dos Estados Unidos (*National Institute of Standarts and Technology* - NIST) na intenção de estabelecer uma base conceitual na qual a computação em nuvem pudesse ser regulamentada, comercializada, fiscalizada, entre outros. Além de características essenciais, Mell, Grance et al. (2011) expõe modelos de serviço e modelos de implantação de computação em nuvem. Os tópicos 2.3.1 e 2.3.2 exploram com detalhes esses modelos.

2.3.1 Modelo de serviço de computação em nuvem

- Infraestrutura como serviço (*Infrastructure as a Service* - IaaS): Em nenhum modelo de serviço de computação em nuvem o usuário é responsável pelo *hardware* físico da infraestrutura da nuvem. Isso é uma das características de nuvem definidas por (FOSTER et al., 2008), (MELL; GRANCE et al., 2011) e muitos outros. Porém, dentro os modelos de serviço, o modelo de infraestrutura é o que permite um maior controle por parte do usuário, como observa-se na comparação da Figura 2.3. Com esse modelo de serviço, é garantido ao usuário o privilégio de provisionar recursos virtualizados de computação, armazenamento e de rede.

  Por conta do alto nível de controle da infraestrutura, esse modelo de serviço é mais adequado a administradores de TI, que serão responsáveis por implantar os mais diversos ambientes para o restante de sua organização. De acordo com Moreno-

Vozmediano, Montero e Llorente (2012), em breve os data centers se comportarão como nuvens IaaS e, nesse contexto, o administrador desses recursos virtuais providos pelo IaaS terá um papel fundamental.

- Plataforma como serviço (*Plataform as a Service* - PaaS): Nesse modelo de serviço, é fornecido ao usuário o poder de implantar na nuvem aplicações previamente disponibilizadas pelo provedor. O usuário, portanto, não controla a infraestrutura por trás das aplicações disponibilizadas, e sim as configurações da aplicação que servirá como plataforma para ele. A infraestrutura necessária para o desenvolvimento e/ou operacionalização das aplicações é também chamada de Ambiente de Aplicação (*Application Runtime Environment* - ARE).

  Além das AREs, o modelo PaaS pode oferecer ambientes de desenvolvimento (*Integrated Development Environment* - IDE), com suporte a bibliotecas e linguagens de programação para o desenvolvimento das aplicações. (BEIMBORN; MILETZKI; WENZEL, 2011).

- *Software* como serviço (*Software as a Service* - SaaS): No mais alto nível do diagrama da Figura 2.3 encontra-se o SaaS. Nesse modelo de serviço, é provido ao usuário a capacidade de utilização de *softwares* rodando em uma infraestrutura de nuvem. A utilização desse *software* pode ser feita através de uma interface *web* em um *browser*, através da interface de outra aplicação via chamadas API (*Application programming Interface*). O usuário não possui controle da infraestrutura por trás da aplicação, como acontece, em diferentes graus, nos modelos de serviço IaaS e PaaS.

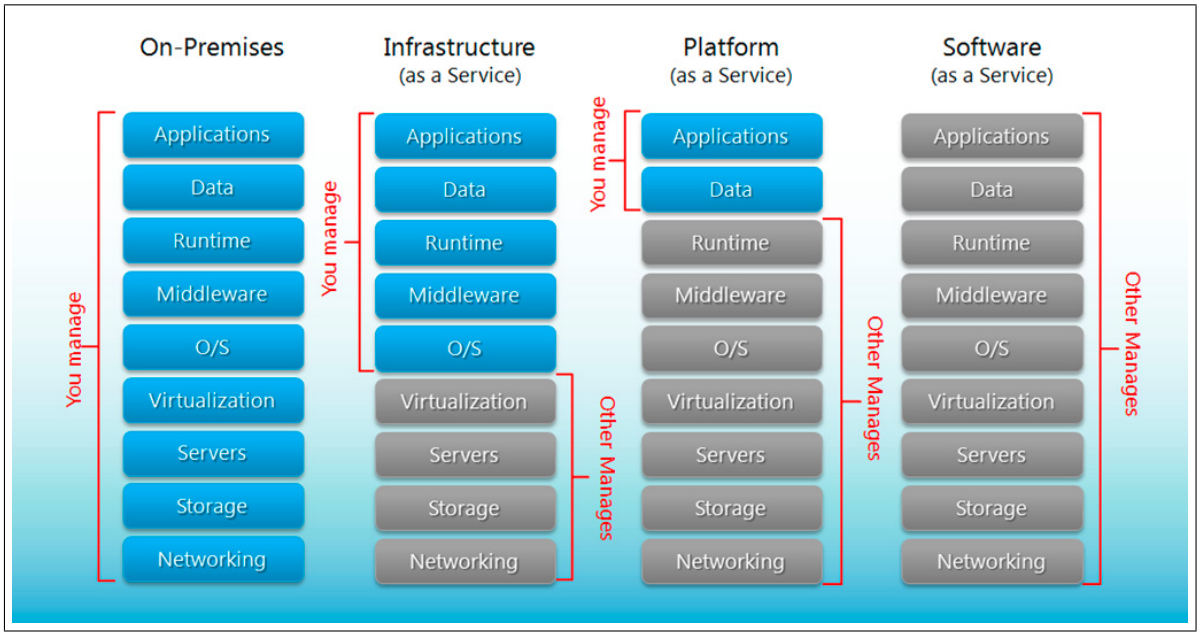


Figura 2.3 – Diagrama comparativo entre o modelo de data center *on-premise* e os modelos de serviço de nuvem.

Além dos modelos tradicionais de serviço, vários outros começaram a ser definidos e utilizados nas plataformas de nuvem pública e privada nos últimos anos. Robison et al. (2008) estabelece que, no futuro, todos os serviços digitais poderão ser entregues via internet, sem processamento local, em um modelo definido como XaaS - *Everything as a Service* (Tudo como serviço). Como exemplo, Duan et al. (2015) cita *Privacy as a Service* - PaaS (Privacidade como Serviço), *Database as a Service* - DaaS (Banco de dados como Serviço), *Desktop as a Service* - DaaS (Área de trabalho como Serviço), *Framework as a Service* - FaaS (Framework como Serviço), *Authentication as a Service* - AaaS (Autenticação como Serviço), *Data as a Service* (Dados como Serviço), entre outros.

Chama-se atenção para um modelo de serviço particularmente importante para esse projeto: *Storage as a Service* - StaaS (Armazenamento como Serviço). Nesse modelo de serviço de nuvem, o usuário consegue armazenar seus dados em discos remotos e acessá-los de qualquer lugar a qualquer hora. Também conhecidos como *Cloud Storage Systems* (Sistemas de Armazenamento em Nuvem) (WU et al., 2010), as StaaS devem atender requerimentos rigorosos para armazenar os dados do usuário, incluindo: alta disponibilidade de dados, confiabilidade e consistência.

2.3.2 Modelos de implantação de computação em nuvem

- Nuvem Privada: Nesse modelo de implantação, a infraestrutura da nuvem é provisionada para uso exclusivo de uma só organização, englobando todos seus funcionários e unidade de negócio. A organização pode decidir se a infraestrutura da nuvem existirá dentro ou fora de seus terrenos.

  Existem várias soluções de *hardware* no mercado que já prometem entregar soluções de nuvem privada. Dessa forma, uma organização pode escolher um modelo de negócios em que ainda conta com equipamentos de data center dentro de suas instalações, mas adiciona uma camada de abstração para seus usuários, entregando a eles os modelos de serviço de nuvem.

- Nuvem Comunitária: Possui uma estrutura de gerenciamento similar a uma nuvem privada, podendo existir dentro ou fora dos terrenos das organizações da comunidade, e podendo ser administrada pela comunidade ou por terceiros. A diferença, porém, está na existência de uma comunidade com interesses em comum para o uso dessa nuvem.

- Nuvem Pública: Nesse modelo de implantação, a infraestrutura de nuvem é aberta ao público geral, podendo ainda ser mantida por uma organização governamental, acadêmica ou por uma corporação. Esse é o mais popular dos modelos de implantação e é o principal foco de grandes empresas como Google, Amazon, Microsoft, entre outras.

- Nuvem Híbrida: Esse é um modelo de implantação de nuvem mista. Organizações (ou comunidades) podem decidir implementar, simultaneamente, um modelo de nuvem publica e privada. Esse modelo pode ser útil para organizações que possuem regras de conformidade que estabelecem a manutenção de dados e aplicações dentro das premissas da companhia (ou do país). Também é um modelo adotado temporariamente, quando organizações estão planejando uma migração completa para o modelo de nuvem pública.

2.4 *Software-Defined Network* - SDN

Para entender redes definidas por *software*, é preciso entender como funcionam as redes tradicionais. Equipamentos de rede tradicional possuem, em um único *hardware* (um *switch*, por exemplo), ambos os planos de controle e plano de dados. O plano de controle (*control plane*) é responsável pela tomada de decisões. Esse plano controla tabela de roteamento, métricas de protocolos de roteamento e demais decisões de encaminhamento (BERDE et al., 2014). O plano de dados (*data plane*), por sua vez, tem a função de encaminhar os pacotes de acordo com as decisões tomadas pelo plano de controle. Isso inclui responsabilidades de recebimento e reenvio de pacote no meio físico.

Redes definidas por *software* (*Software defined network* - SDN) emergiram como um novo paradigma no tráfego e gerenciamento de redes. Com a proposta de desacoplar o plano de controle da rede do plano de dados, as redes definidas por *software* centralizam o plano de controle de todos os dispositivos da rede em uma única controladora, que tomará as decisões à nível de plano de controle para os dispositivos de rede, que se tornam apenas agentes do plano de dados (KREUTZ et al., 2014).

Analogia da rede de transporte público

É possível fazer uma analogia, para fins de entendimento, entre uma rede de computadores típica com uma rede de transporte terrestres de uma cidade. Muitas vezes o caminho para ir de um ponto a outro da cidade envolve trocas de linhas em estações de transferência ou a utilização de linhas que dão a volta na cidade até chegar no destino, fazendo um traçado não ótimo. Redes de computadores funcionam de forma análoga. O caminho que os dados seguem em uma rede geralmente não é o mais eficaz.

Considere, porém, um novo sistema de transporte público. As estações de embarque e ruas de transporte permanecem, porém as linhas definidas de ônibus são extintas. A cada vez que passageiros embarcam no ônibus, um controlador de tráfego calcula uma nova rota, usando toda a malha rodoviária existente da cidade, que leve os passageiros para seus destinos de maneira mais eficiente. Além disso, esse controlador, que é responsável por todas as linhas de tráfego da cidade, leva em consideração o trânsito de cada rua da

cidade, o número de passageiros indo para cada destino e a importância da viagem de cada passageiro.

Isso é precisamente o que uma rede SDN faz. Através de um controlador de rede e uma rede virtualizada, é possível tomar decisões inteligentes de rotas baseadas no tipo de tráfego de dados, na importância do tráfego, nas características da aplicação, e diversas outras formas de tomadas de decisão. Em um ambiente com servidores virtualizados e um controlador de rede inteligente, é possível até realocar uma aplicação de máquina física, caso o controlador perceba que dessa forma a aplicação ganha em *throughput* e a rede inteira fica menos congestionada.

2.5 *Software-Defined Data Center* - SDDC

Em grandes data centers, especialmente àqueles com arquitetura distribuída e heterogeneidade de equipamentos, o controle e gerenciamento são sempre processos complexos DARABSEH et al. (2015). Sistemas definidos por software (*Software Defined Systems* - SDS) é um conceito recente que busca facilitar esses processos isolando os planos de dados e de controle, como explicado na sessão 2.4, aplicados a SDN. Através de chamadas de APIs e *softwares*, o administrador dos sistemas consegue controlar e gerenciar os recursos e dispositivos disponíveis.

O uso de *softwares* para controlar e gerenciar recursos não é, no entanto, um conceito novo. O diferencial de *software-defined* é a habilidade da camada de controle de controlar e gerenciar todos os recursos de infraestrutura abaixo, independente do fabricante ou arquitetura, isolando-se dos recursos de *hardware* no plano de dados. (MACVITTIE, 2014)

A Força Tarefa de Gerenciamento Distribuído (*Distributed Management Task Force* - DMTF) possui uma definição em construção para SDDC:

> *"Data center definido por software (SDDC): Uma abstração de computação lógica, de rede e de armazenamento, representada por software. Esses recursos são dinamicamente descobertos, provisionados e configurados baseados nos requerimentos da carga de trabalho. Portanto, SDDC possibilita a orquestração de cargas de trabalho baseado em políticas pré definidas, assim como medições e gerenciamento de recursos consumidos. "(FORCE, 2015)*

Do ponto de vista gerencial de TI, SDDC é um prospecto de como pode-se lidar com infraestrutura de maneira otimizada, servindo o propósito do negócio. A ideia principal é permitir que as aplicações definam seus requisitos de recurso (computação, rede e armazenamento) com recursos delimitados e de acordo com a conformidade estratégica do negócio (VOLK, 2012). Para garantir escalabilidade, flexibilidade e agilidade, é essencial

que tais definições de requisitos sejam executáveis através de um conjunto de chamadas API (*Application Programming Interface* - Aplicação de Interface Programável), que permitam que *softwares* de gerenciamento e virtualização possam provisionar, configurar, mover e excluir recursos relevantes à aplicação.

Essa permissividade e flexibilidade dada aos usuários e aplicações transforma o paradigma de infraestrutura centralizada (AZEEM; SHARMA, 2017), mudando as atividades da equipe de TI de reativas para proativas. Os *Software-defined Systems* se concentram em solucionar as demandas das aplicações que estão implantadas no ambiente, permitindo que os usuários tirem maior proveito da infraestrutura.

Explicitamente, o conceito de SDDC expõe algumas necessidades, entre elas: a necessidade de servidores físicos, ativos de rede e discos de armazenamento. Junto à estes, é necessário um hipervisor que abstraia a complexidade do *hardware* e entregue o plano de controle para o software que terá o controle centralizado de recursos *"hardware-independent"*, isto é, recursos não atrelados a *hardwares* específicos; e um *software* orquestrador, que está na topologia lógica do data center, capaz de descobrir novos ativos na topologia (independente de fabricante), gerenciar e mensurar os recursos consumidos e disponíveis, provisionar recursos de forma automática e baseado em regras definidas, além de outras funções. É importante notar a importância desse *software* orquestrador. A capacidade de disponibilizar uma interface para que as aplicações requisitem infraestrutura é, de fato, o diferencial entre um data center simplesmente virtualizado e um data center definido por *software*.

A Figura 2.4 apresenta os componentes nucleares nessa estrutura de SDDC explicada. Percebe-se que, como explicado na seção 2.1, mantém-se a importância dos componentes nucleares da arquitetura tradicional de data center, adicionando-se apenas uma plataforma de automação e gerenciamento à essa topologia.

Par que requisitos de aplicações e usuários rodem de maneira eficiente e dinâmica, o SDDC propõe uma abstração e centralização do gerenciamento de três principais recursos de infraestrutura: redes, armazenamento e computação.

Virtualização de redes

Como mencionado na seção 2.4, *Network virtualization* e *Software-Defined Networks* são tópicos muito populares ultimamente. Em particular, a ideia de virtualização de rede busca a separação do plano de controle da rede e do plano de dados (KREUTZ et al., 2014). Uma vez que haja esse desacoplamento, o SDDC pode assumir controle do plano de controle, centralizando as decisões de rede. As aplicações, por sua vez, passam a ter uma interface de chamadas API para negociar mudanças no plano de dados, conforme suas necessidades (VOLK, 2012).

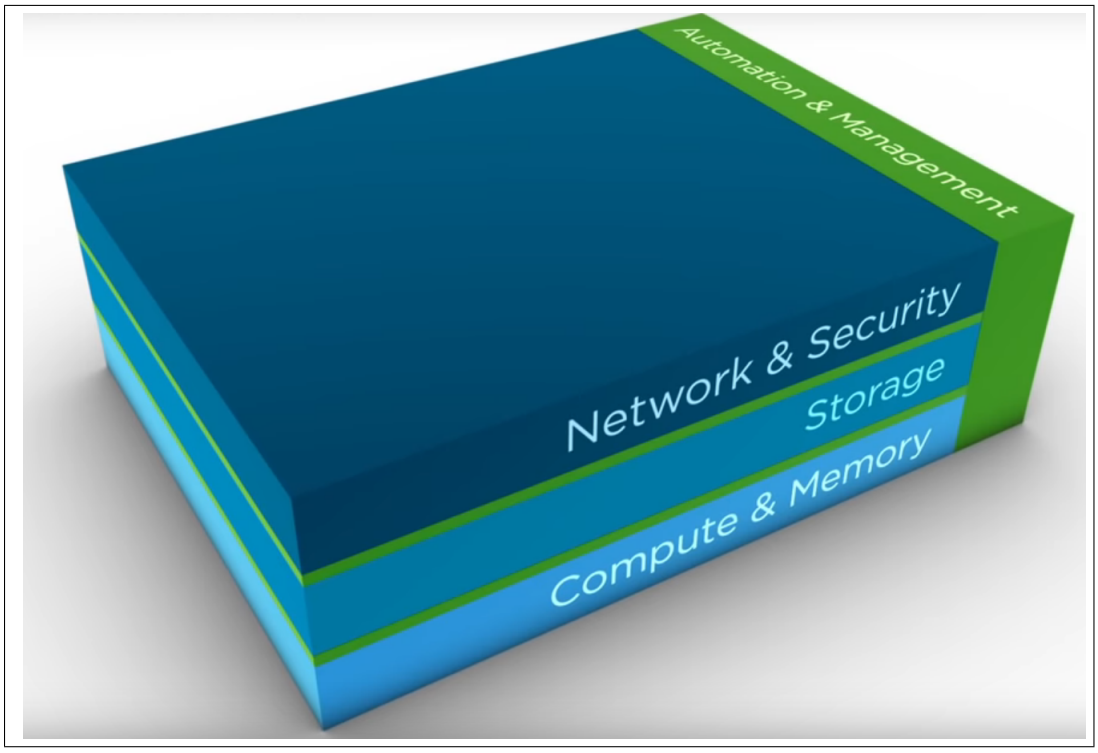


Figura 2.4 – Modelo lógico de um *Software-Defined Data Center*. Ilustração proposta pela fabricante VMWare. Software-Defined Data center – In Depth (Acessado online no dia 13 de outubro de 2019)

Virtualização de armazenamento

Sistemas de virtualização de armazenamento permitem usuários de gerenciar uma grande plataforma de armazenamento, ao invés de pequenas partições isoladas e independentes (KARPOFF; LAKE, 2009).

Os sistemas de virtualização de armazenamento abstraem as especificidades do hardware (incuindo LUNs e NAS) e agrupam os recursos em discos virtuais. Serviços típicos de hardware físico, como deduplicação, replicação, *snapshot* e *backup*, podem ser feitos com o disco virtual apresentado (Richard Fichera, 2012).

Virtualização de computação

Computação é o elemento mais importante de um SDDC, e também o mais maduro em relação a soluções já propostas (Richard Fichera, 2012). Como explicado na seção 2.2, a virtualização de computação é possibilitada pela criação do hipervisor nativo, que monta uma camada de interpretação entre o *hardware* e o sistema operacional (HUMPHREYS; GRIESER, 2006).

Particularmente, uma solução de SDDC também fará a implementação de um hipervisor que possa controlar o *hardware* (diretamente ou através do sistema operacional). Com o *hardware* abstraído e centralizado, o SDDC pode orquestrar suas demandas para

cada recurso computacional conforme a necessidade da aplicação ou do cliente.

Além dos três componentes principais, como mostra a Figura 2.4, há uma plataforma de automação e gerenciamento que controla todos esses componentes. Existem diversas opções de implementação para SDDC nesse quesito, como CloudStack, OpenNebula, OpenStack, entre outros. Uma comparação teórica entre OpenNebula e OpenStack é feita em Wen et al. (2012). Por conta da implementação em Python e de pequenas vantagens em implementação de segurança, a plataforma OpenStack foi escolhida nesse projeto. A seção 3.3.1 traz mais detalhes sobre a plataforma.

Segundo os resultados obtidos no artigo de Wen et al. (2012), há uma forte compatibilidade de serviços de outras nuvens, como a AWS, na solução de nuvem do OpenStack, podendo utilizar serviços como EC2 e S3. Além disso, há uma documentação mais especializada na construção de uma nuvem sem a necessidade de pessoas especializadas e os modelos de rede providos pela nuvem OpenStack serem maiores do que a solução OpenNebula. Esses e outros detalhes estudados de soluções de nuvem foram os motivos que influenciaram na escolha da solução do OpenStack para este trabalho.

2.6 Segurança

Em termos gerais, segurança é a *"qualidade ou o estado de segurança, de estar livre de perigo"*, de acordo com Houghton Mifflin (2019). Whitman (2012), faz referência a uma organização bem-sucedida em condimento de ter as seguintes camadas de segurança para proteger suas operações:

- **Segurança física**, para proteger itens físicos, objetos, ou áreas de acessos não autorizados ou de uso errado.
- **Segurança pessoal**, proteger o indivíduo ou grupo de pessoas autorizadas a acessar a organização e sua operação
- **Segurança Operacional**, protege os detalhes de uma operação particular ou uma série de atividades.
- **Segurança das Comunicações**, para proteger a mídia, a tecnologia e o conteúdo das comunicações
- **Segurança de Rede**, protege os componentes da rede, conexões e conteúdos.
- **Segurança da Informação**, proteger a confidencialidade, integridade e disponibilidade dos ativos de informações, seja em armazenamento, processamento ou transmissão. Isso é alcançado através da aplicação de políticas, educação, treinamento e conscientização e tecnologia.

O Comitê de Sistemas de Segurança Nacional (*Committee on National Security Systems* - CNSS), define segurança da informação como a proteção da informação e seus elementos críticos, incluindo os sistemas e o *hardware* que usam, armazenam e transmitem essas informações (Committee on National Security Systems, 2016). O modelo CNSS de segurança da informação evoluiu de um conceito desenvolvido pelo setor de segurança de computadores chamado CIA *triad (Confidenciality, Integrity and Availability - CIA)*. A *triad* é o padrão da indústria para segurança de computadores desde o desenvolvimento do *mainframe*.

A compreensão dos mecanismos que impõem a política de segurança, o conhecimento das premissas e da confiança relacionadas, que levam às ameaças e ao grau em que elas podem ser realizadas é requerido como análises da segurança de um sistema. Esse conhecimento permite projetas melhores mecanismos e políticas para neutralizar essas ameaças. A segurança em computadores baseia-se nos três pilares de segurança, confidencialidade, integridade e disponibilidade. As interpretações desses três aspectos variam, assim como os contextos que surgem.

Figura 2.5 – Conceito fundamental de segurança da informação baseado em confidencialidade, integridade e disponibilidade, também conhecida como CIA *triad*. Modificada. Fonte: (ABZETDIN ADAMOV, 2015)

- **Confidencialidade**: é a ocultação de informações ou recursos. A demanda de manter as informações em incógnita decorre do uso de computadores em campos sensíveis, como governo e indústria. Mecanismos de controle de acesso são suportados pela confidencialidade. Um mecanismo de controle de acesso para preservar a confi-

dencialidade é a criptografia, que mistura os dados para torná-los incompreensíveis. (BISHOP, 2005)

- **Integridade**: a integridade pode ser aplicada a um fluxo de mensagens, à uma única mensagem ou em campos selecionados de uma mensagem, assim, a abordagem mais útil e direta é a proteção total do fluxo. Um serviço de integridade orientado a conexão, que trata com um fluxo de mensagens, garante que as mensagem que sejam recebidas como enviadas, sem duplicação, inserção, modificação, reordenação ou repetição, ou seja, é garantido a integridade do dado em seu estado original de concepção. (STALLINGS, 2005)

- **Disponibilidade**: a disponibilidade é a garantia de que as informações estarão disponíveis para o consumidor de maneira oportuna e ininterrupta quando necessárias, independentemente da localização do usuário. Isso significa que a infraestrutura em nuvem, os controles de segurança e as redes que conectam os clientes e as infraestruturas em nuvem sempre devem estar funcionando corretamente. A disponibilidade é garantida por: tolerância a falhas, autenticação e segurança de rede. (AGARWAL; AGARWAL, 2011)

2.6.1 Segurança em Data centers

Seguindo para a linha de segurança em data centers, os níveis de virtualização, os acessos à serviços e dados sendo utilizados e salvos, são importantes fatores a serem considerados quando trata-se da segurança de dados dos usuários e da manutenção do funcionamento do sistema. Uma característica da computação em nuvem é que detalhes são invisíveis, ou seja, para o usuário que utiliza os recursos computacionais e de armazenamento do data center, é transparente para o mesmo o local onde estão sendo armazenados seus dados e a forma que o dado é armazenado. Para tranquilizar os clientes e incutir confiança no uso da computação em nuvem, são exatamente esses pontos de transparência que devem estar claros na hora da construção e projeção da utilização de serviços e dados em nuvem. O artigo de Okuhara, Shiozaki e Suzuki (2010) faz referência a problemas de segurança enfrentados por nuvens e como resolvê-los, provendo serviços mais estáveis e arquiteturas seguras de *Cloud computing*.

Componentes importantes da camada do provedor de serviços de uma nuvem como provisionamento de recursos, contabilidade, balanceador de carga, reserva de recursos, enfrentam alguns problemas de segurança relacionados à essa camada, como identidade, privacidade, transmissão de dados, integridade da nuvem. Relacionado a componentes da camada de virtualização como criação de máquinas virtuais, número de sistemas operacionais e seu monitoramento, também, sofrem problemas de segurança, como separação de clientes, regularidade na nuvem e gerenciamento de identidade e acesso. Peças da camada

de infraestrutura do data Center sofrem problemas de segurança na forma de repouso de dados (KRISHNA; REDDY, 2011). Assim esses problemas em várias camadas de data centers são enfrentados e discutidos para garantir um melhor controle entrega do que está sendo ofertado e utilizado pelos usuários de *Cloud computing*.

Métodos como controle unificado de tráfego, recursos isolados, dados distribuídos, encriptação de dados são utilizados para prover ao máximo a alta disponibilidade de dados, a segurança de acesso em gerenciamento, transporte e armazenamento de dados de forma seguro são implementados e utilizados para data centers (KALLAHALLA et al., 2004).

2.6.2 Encriptação

A criptografia é uma tecnologia bem estabelecida para proteger dados sensíveis. No entanto, uma vez criptografados, os dados não podem mais ser facilmente consultados, exceto as correspondências exatas. Os sistemas de banco de dados geralmente oferecem controles de acesso como meio de restringir o acesso a dados confidencias. Esse mecanismo protege a privacidade de informações confidenciais, desde que os dados sejam acessados usando as interfaces de sistema definidas. No entanto, o controle de acesso, embora importante e necessário, geralmente é insuficiente. Os ataques a sistemas de computadores mostraram que as informações podem ser comprometidas se um usuário não autorizado simplesmente obtiver acesso aos arquivos de dados brutos, ignorando completamente o mecanismo de controle de acesso (AGRAWAL et al., 2004).

Um dos método mais utilizados de proteção de informações armazenadas em um sistema de computador ou transmitido através da redes é o uso de criptografia de dados. A tecnologia de criptografia de dados é basicamente classificada em dois tipos de tecnologia: simétrica ou assimétrica. A criptografia simétrica usa a mesma chave para criptografar e descriptografar um conjunto de dados. A criptografia assimétrica usa duas chaves que compartilham um relacionamento. Assim, as informações criptografadas com uma chave podem ser descriptografadas apenas com a segunda chave. A criptografia simétrica é muito mais rápida que a criptografia assimétrica e, portanto, é mais adequada para criptografia de dados em massa. (FIELDER; ALITO, 2000).

Como a computação em nuvem está transformando a tecnologia da informação, cada vez mais mundos corporativos e acadêmicos investem nessa tecnologia. Essa tecnologia está resolvendo muitos problemas de computação convencional, incluindo manuseio de picos de carga, a instalação de atualizações de *software* e o uso de ciclos de computação, porém, a nova tecnologia também criou novos desafios, como segurança de dados, propriedade de dados e armazenamento. Questões como essas são motivos de estudos em artigos como Arora, Parashar e Transforming (2013), Somani, Lakhani e Mundra (2010), com a utilização de algoritmos de encriptação.

2.6.3 VPN

As redes privadas virtuais (VPNs) permitem a comunicação segura entre dois ou mais dispositivos em uma rede pública ou não confiável. Em um arranjo típico de VPN, um usuário final é associado a um dispositivo de terminal, como uma estação de trabalho, um computador pessoal ou um telefone celular, que executa o *software* cliente da VPN. O terminal estabelece uma conexão através de uma rede não confiável, como a internet pública, a um *gateway* ou outro nó da rede, que executa o *software* do servidor VPN e está associado a uma rede segura de uma empresa comercial ou outra entidade. O nó de extremidade e a rede negociam chaves de criptografia, criando essencialmente uma conexão de túnel criptografada através da rede não confiável. O nó de extremidade e a rede então comunicam informações criptografadas pela rede não confiável e as informações criptografadas são descriptografadas nos terminais.

Nesse arranjo, o usuário final pode obter informações com segurança de recursos de rede privada através do canal VPN, mesmo que uma ou mais redes das quais os dados devem passar até o seu destino, não tenham segurança. Os usuários típicos da VPN são trabalhadores corporativos que se comunicam ou teletrabalham (GLADSTONE; MCGREW, 2014). Os protocolos de transmissão de dados usados na internet, como TCP/IP, originalmente não foram projetados para fornecer segurança aos dados. Uma transmissão é considerada como segura quando um atacante não consegue usar os dados transmitidos, e só possível prover essa segurança através do encapsulamento, criptografia e autenticação de dados. Isso é alcançado adicionando protocolos seguros, como IPSec, L2TP ou PPTP aos protocolos existentes que são usados na internet, explicado de melhor forma em Barkie et al. (2015).

- O IPSec fornece várias técnicas padronizadas para estabelecer conexões VPN seguras. Os pacotes IPs basicamente não são seguros, uma vez que dados como endereço IP no cabeçalho, carga útil e outros campos, podem ser alterados ou interceptados durante a transmissão. O IPSec permite segurança da transmissão na camada IP, garante segurança dos dados por meio da autenticação de pacotes, verificando integridade e o encapsulamento de pacotes.

- O protocolo de encapsulamento da camada dois, L2TP, é baseado no protocolo de encaminhamento da camada dois L2F, descrito em Kolar e Valencia (1998). Permite o encapsulamento de um quadro completo da camada de enlace de dados, como por exemplo um quadro *Ethernet*, em um pacote UDP na camada de transporte. Encapsular apenas um quadro de camada dois em um pacote UDP não fornece autenticidade ou privacidade de dados, portanto, o L2TP geralmente é combinado com o IPSec. Desse modo, o transporte utilizando o IPSec é o aplicado para esse caso, pois encapsular o pacote L2TP em um novo pacote IP resultaria em sobrecarga

excessiva de protocolo.

- O protocolo de encapsulamento ponto a ponto da Microsoft é uma extensão do protocolo ponto a ponto PPP. O PPTP usa dois tipos de pacotes diferentes para estabelecer uma conexão VPN. Primeiro, os pacotes de encapsulamento de roteamento genérico GRE, transportam a carga da VPN adicionando cabeçalho GRE ao pacote original. O segundo tipo de pacote é a mensagem de controle PPTP, sendo um simples pacote TCP, utilizado na porta 1723, contendo informações de controle, como solicitações e respostas de conexão, parâmetros de conexão e mensagens de erro. Como nem as mensagens GRE nem PPTP fornecem autenticação ou criptografia, o PPTP deve ser combinado com métodos de segurança adicionais.

3 Proposta de arquitetura

Neste capítulo, será evidenciado o problema abordado e será feita uma descrição lógica e física da solução proposta pelo trabalho. Espera-se que, ao final do capítulo, seja possível compreender como as topologias lógicas e físicas interagem e de que maneira elas podem vir a resolver o problema abordado.

3.1 Problema abordado

Em ambientes de nuvem onde o usuário possui acesso a uma infraestrutura virtualizada (comumente entregue como um Infraestrutura como serviço - IaaS), aspectos relacionados ao armazenamento seguro são sempre preocupações importantes a serem consideradas.

Em uma IaaS, por se tratar de uma infraestrutura que o usuário não controlar o hardware, é preciso tomar providências para garantir a segurança dos dados armazenados e, em casos IaaS com data centers distribuídos, a segurança dos dados em transporte. Nesse trabalho, será então implantado alguns conceitos de SDDC e nuvem privada, de tal forma que seja possível um estudo de caso para solução do problema de armazenamento seguro em ambiente de nuvem distribuído.

3.2 Topologia física proposta

Para iniciar a construção de uma solução para o problema de armazenamento distribuído e seguro em ambiente de nuvem privada, é necessário primeiramente construir um cenário físico que permita o estudo do problema.

Foram usados, então, dois servidores dentro de um mesmo data center. Com esses servidores, é possível implementar um cenário de simulação de uma nuvem, como será visto na seção 3.3 de topologia lógica. Esses servidores estavam ligadas há um mesmo *switch* de camada 3, que realiza essa comunicação. A comunicação entre servidores pelo switch simula satisfatoriamente a comunicação entre dois data centers, uma vez que o problema se resumirá a roteamento.

Os servidores usados na implantação foram dois Intel Xeon(R), CPU X5560 de 2.80GHz, 16 núcleos, 32Gb de memória RAM, duas interfaces de rede de 1Gb/s e um disco HDD de 1Tb de armazenamento.

O *switch* usado para a comunicação entre os servidores foi um Dell Networking modelo N3000, com interfaces *Ethernet* de 1Gb/s. Duas conexões foram usadas para cada

servidor, permitindo a utilização de uma interface para gerenciamento e uma interface para disponibilização dos serviços do OpenStack. Todas as interfaces do *switch* estão configuradas na mesma VLAN 1, com MTU de 1500 bytes e endereços IP fixos. Esse *switch* também possui uma interface de 1Gb/s ligada ao *switch* core do data center, que permite a saída para a internet.

Nos servidores, é instalado o sistema operacional CentOS 7. É preciso um cuidado especial para as especificações de disco durante tal instalação. Em seguida, o OpenStack é também instalado em cada servidor, conforme Seção 4.1.

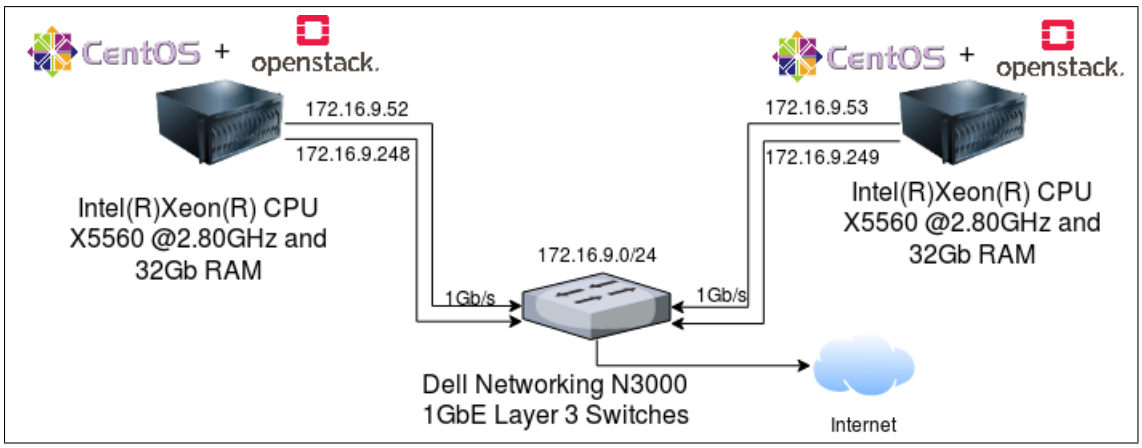


Figura 3.1 – Topologia física proposta. Na figura, observa-se o cenário físico onde a arquitetura aqui proposta foi montada.

3.3 Topologia lógica proposta

Tendo o problema de armazenamento seguro e distribuído em mente, propõe-se, inicialmente, a implantação de uma solução aberta de virtualização de recursos na infraestrutura física explicada na seção 3.2. A base da solução proposta é construída em cima da plataforma do Openstack, que permitirá a entrega de serviços IaaS e, de fato, o gerenciamento de uma nuvem privada. Na Figura 3.2, é possível encontrar uma representação lógica da solução completa aqui proposta.

3.3.1 OpenStack

Openstack é uma plataforma que faz o controle de recursos computacionais, de armazenamento e de redes em um ou múltiplos data centers, controlados por meio de um painel com interface administrativa, que permite controle administrativo de recursos, e interface de usuário, que permite provisionamento de recursos em modelo IaaS, OpenStack's Documentation (2019c). OpenStack inclui a capacidade de fornecer redes sob demanda, endereços IP, *firewalls* e roteadores. Os recursos básicos para isso são integrados

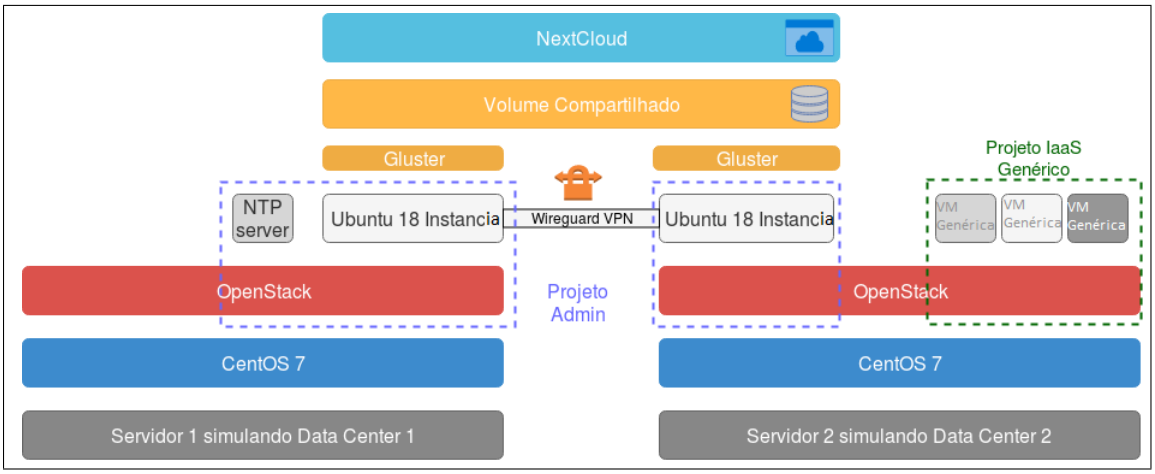


Figura 3.2 – Topologia lógica proposta. Na figura, observa-se a construção lógica de uma virtualização de data center, com a plataforma OpenStack. Além disso, a proposta lógica de uma solução de armazenamento distribuído e uma solução de STaaS.

e o OpenStack também pode ser integrado às ofertas de SDN orientadas à telecomunicações de fornecedores de equipamentos de rede, como Juniper, Cisco e Nokia. Para uma empresa, a capacidade de definir redes por meio de uma API permite automação de infraestrutura em ritmo acelerado e operações no estilo de nuvem. Da mesma forma, o OpenStack inclui uma estrutura para armazenamento definido por *software*, incluindo mecanismos de armazenamento em bloco, disco e objeto. (Canonical, 2019b). Lançado originalmente em julho de 2010 pela Rackspace e pela NASA como uma iniciativa de código aberto que combinava a plataforma Nebula da NASA e a plataforma Cloud Files da Rackspace. Várias empresas contribuem com o código do OpenStack, como IBM, Red Hat, Huawei, entre outas.

Capaz de agrupar recursos virtuais para criar e gerenciar nuvens públicas e privadas. As ferramentas que abrangem a plataforma OpenStack, chamadas de `projetos`, lidam com os serviços essenciais de *cloud computing*, rede, armazenamento, identidade e serviços de imagem. Na virtualização, recursos como armazenamento, CPU e RAM são obtidos de uma variedade de programas de fornecedores específicos e divididos por um hipervisor antes de serem distribuídos conforme a necessidade. Apesar do OpenStack utilizar recursos virtualizados, possuindo a capacidade de detectar, informar e automatizar processos em ambientes de fornecedores diferentes, ele não é considerado uma plataforma de gerenciamento de virtualização. As plataformas de gerenciamento de virtualização facilitam a manipulação de características e funções dos recursos virtuais, o OpenStack, na verdade, usa os recursos virtuais para executar uma combinação de ferramentas. Essas ferramentas criam um ambiente de *cloud* que atende a cinco critérios do *National Institute of Standards and Technology* para *cloud computing*: rede, recursos agrupados, interface de usuário, provisionamento de capacidade e alocação ou controle automático de recursos.

(National Institute of Standards and Technology, 2010)

Essa plataforma de nuvem provê uma solução de IaaS através de variados componentes, ou módulos, que se comunicam entre si e com o exterior através de chamadas de API, obtendo recursos virtuais e transformando-os em *pools* discretos usados para potencializar as ferramentas de *cloud computing* padrão, com as quais administradores e usuários interagem diretamente. Todo o código do projeto é aberto e sob a licença Apache 2 Sefraoui, Aissaoui e Eleuldj (2012). O OpenStack é essencialmente uma série de comandos conhecidos como *scripts*. Esses *scripts* são reunidos em pacotes chamados projetos que retransmitem tarefas que criam ambientes de *cloud*. Para criar esses ambientes, o OpenStack utiliza dois outros tipos de *software*:

- A virtualização que cria uma camada de recursos virtuais abstraídos do hardware.
- Um sitema operacional de base que executa os comandos dados pelos *scripts* do OpenStack.

Por ser composto de vários componentes que se integram, o Openstack permite uma modularidade adaptável a cada solução. A arquitetura do OpenStack é composta de diversos projetos de código livre. Há seis serviços básicos e estáveis que abrangem computação, rede, armazenamento, identidade e imagens, além de mais de uma dúzia de projetos opcionais em estágio de desenvolvimento. Esses seis serviços básicos constituem a infraestrutura que permite aos demais projetos ter acesso a painéis, orquestração, provisionamento de *bare metal*, sistema de mensageria, contêineres e governança. Os principais componentes do Openstack estão exemplificados na Figura 3.3 e explicados abaixo:

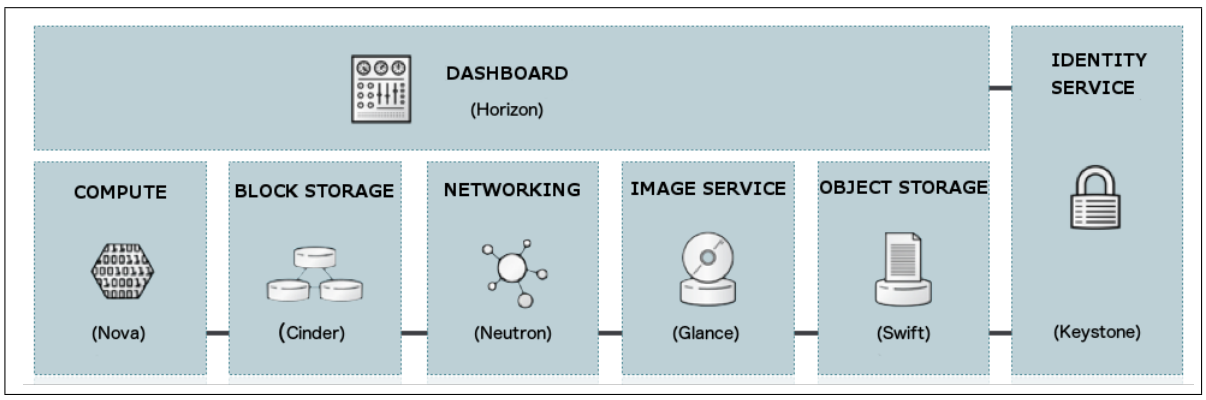


Figura 3.3 – Alguns dos componentes principais que formam a arquitetura lógica da pilha OpenStack. Fonte: OpenStack Documentation's image base (Acessado online no dia 26 de outubro de 2019)

- *Glance*, gerenciador de imagens. Esse componente do OpenStack é responsável pelo provisionamento de imagens para VMs. Nele, estão pré-instalados máquinas que são

usadas quando um cliente quer levantar uma instância. O módulo Glance suporta uma variedade de formatos de imagens para disco virtuais, além de suportar contêineres. OS formatos de discos suportados são: *RAW, VHD, VMDK, QCOW2, VDI, ISO, AMI, ARI e AKI.* (PEPPLE, 2011)

- *Neutron.* É o componente responsável pela rede da pilha OpenStack, incluindo infraestrutura de rede virtualizada. O Neutron possibilita a criação de topologias de rede virtual complexas, utilizando uma abstração de redes, subredes, roteadores e com o uso opcional de *firewall* e VPNs. Cada abstração tem sua funcionalidade e configuração similar ao componente físico (OpenStack's Neutron Documentation, 2019).

  O *Neutron* tem papel fundamental na criação de redes virtuais que servirão para conectar instância (VMs) de projetos. É um dos principais componentes que viabilizam a IaaS. O Neutron ajuda a garantir que cada um dos componente de uma implementação do OpenStack possa se comunicar rapidamente e eficiente.

- *Nova*, instância de computação. É o componente responsável pela criação, migração, pausa, redimensionamento e apagamento de instâncias (máquinas virtuais). O Nova funciona uma camada acima do hipervisor e tem como pré-requisitos os módulos Glance, Neutron, Keystone, e Placement. Nova é o principal mecanismo de computação do OpenStack, usado para implantar e gerenciar máquinas virtuais e instâncias para tarefas de computação. Nova suporta uma ampla variedade de hipervisores; KVM, XenServer®, VMware ESXi e Microsoft Hyper-V. Os contêineres Linux, como o LXC, também são suportados. (Canonical, 2019b)

- *Keystone*, serviço de identificação. É o componente responsável pela autenticação na pilha OpenStack. Em vários momentos, durante a utilização da nuvem, é preciso haver autenticação. Usuários são autenticados ao fazer login em seus projetos, administradores são autenticados antes de executar algum comando via Horizon ou linha de comando, e requisições API internas são autenticadas ao realizarem chamadas internas. Como todos os serviços são registrados no Keystone, esse componente serve também como catálogo de serviços, além de uma ferramenta de descoberta de novos serviços.

- *Cinder*, componente de armazenamento em *block storage*. Em especial, o *Cinder* será fundamental no gerenciamento de recursos de bloco de armazenamento persistente para serem usados com instâncias (VMs) nos projetos de IaaS. O Cinder usa blocos de dados de comprimento fixo anexados à execução de máquinas virtuais. O armazenamento em bloco é usado para adicionar armazenamento persistente a uma máquina virtual. (HAN; GILFIX, 2016)

- *Swift*, componente responsável por armazenamento em *object storage*. É um sistema de armazenamento de objetos e arquivos. O Swift funciona de maneira diferente do método tradicional de se referir a arquivos pelo local da unidade de disco. Em vez disso, é possível que os desenvolvedores se refiram a um identificador exclusivo que se refere ao arquivo de informações, permitindo que o OpenStack decida onde essas informações serão armazenadas. Um dos principais benefícios disso é a facilidade de uso criada para o dimensionamento. Removendo a necessidade de os desenvolvedores se preocuparem com a capacidade em um único sistema atrás do software. Outra vantagem adicional é que o sistema se encarrega de garantir o backup dos dados, em vez de contar com o desenvolvedor para fazer isso. (Canonical, 2019b)

- *Horizon Dashboard*. O *Dashboard* do OpenStack é um portal web para interagir com os componentes de serviço da pilha OpenStack. Através de uma interface gráfica, permite que administradores da nuvem e de projetos IaaS acessem, provisionem e automatizem recursos da nuvem privada. Os desenvolvedores podem acessar todos os componentes do OpenStack individualmente por meio de uma API (interface de programação de aplicativos), mas o painel fornece aos administradores de sistema uma visão do que está acontecendo na nuvem e para gerenciá-lo conforme necessário.

O design de uma nuvem OpenStack requer um entendimento dos requisitos e necessidades do usuário da nuvem para determinar a melhor configuração possível. Para maior compreensão do OpenStack, é mostrado na Figura 3.4, com as mínimas configurações requeridas, com a utilização de no mínimo dois nós (*hosts*) para criar uma básica *virtual machine* ou instância. Essa arquitetura foi projetado para fornecer uma prova de conceito mínima com o objetivo de aprender sobre o OpenStack, diferindo de uma implementação de produção. (OpenStack's Documentation, 2019b)

O nó controlador executa o serviço de identidade, serviço de imagem, serviço de posicionamento, partes de gerenciamento de computação, parte de gerenciamento de redes, vários agentes de rede e o painel. Inclui serviços de suporte também, como banco de dados SQL, fila de mensagens e NTP. Opcionalmente é possível que o nó de controle execute partes dos serviços de *block storage*, *object storage*, orquestração e telemetria. O nó de controle requer no mínimo duas interfaces de rede.

O nó de computação executa a parte de computação do hipervisor que opera as instâncias. Por padrão, o nó de computação usa o hipervisor KVM. O nó de computação também executa um agente de serviço de rede que conecta instâncias a redes virtuais e fornece serviços de *firewall* para instâncias por meio de grupos de segurança. Cada nó requer no mínimo duas interfaces de rede.

Os nós adicionais mostrados na Figura 3.4 fazem referência a utilização de nós de *block storage* e *object storage*. Para o nó de armazenamento em bloco, é contido os discos

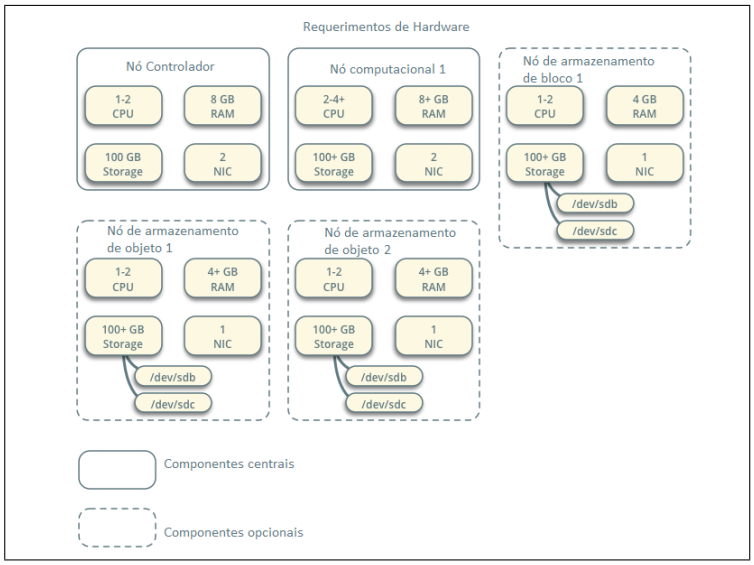


Figura 3.4 – Exemplo de arquitetura, com requisitos mínimos de *hardware* para a implementação de uma prova de conceito, com o propósito de melhor entendimento do OpenStack. Fonte: (OpenStack's Documentation, 2019b)

que o serviço de *block storage* e de *File System* compartilhado fornecem para as instâncias. Já o nó de armazenamento de objetos contém os discos que o serviço de *object storage* usa para armazenar contas, contêineres e objetos. Ambos os nós adicionais necessitam de no mínimo uma interface de rede.

Existem dois tipos de provisionamento de rede para a topologia da Figura 3.4. A primeira delas, denominada de *provider network*, implementa o serviço de rede do OpenStack da maneira mais simples possível, principalmente com serviços de camada 2 (*bridging/switching*) e VLAN para segmentação de redes. Essencialmente, esse primeiro provisionamento de rede, conecta redes virtuais a redes físicas e depende da infraestrutura da rede física para roteamento (camada 3). Além disso, um serviço de DHCP fornece informações de endereço IP para as instâncias. O segundo provisionamento de rede, tratado como *self-service network*, na qual aumenta a opção do provedor de rede com serviço de camada 3 (roteamento), que permitem redes de autoatendimento usando métodos de segmentação por sobreposição, como VXLAN. Fundamentalmente, essa opção de rede roteia redes virtuais para redes físicas usando NAT. Além disso, esta opção fornece a base para serviços avançados como LBaaS e FWaaS.

Partindo para a pilha de funcionalidades atribuídas ao administrador da nuvem do OpenStack, existe um conceito de Projetos, ou *Tenants* [1]. Através de projetos, o administrador da nuvem consegue isolar usuários e recursos de computação, rede e armazenamento, entregando separadamente uma infraestrutura totalmente independente e, se for preciso, isolada das demais.

[1] Os termos 'Projeto' e 'Tenants' se referem ao mesmo conceito e podem, nesse texto, ser usados permutavelmente.

O administrador da nuvem, porém, possui uma peculiaridade em suas permissões. Este possui controle, através do *Horizon*, das configurações de sistema do OpenStack e também do projeto Admin. Portanto, fazendo login com as credenciais de administrador, o usuário poderá escolher, nas abas laterais do *Horizon*, entre gerenciar as configurações de sistema ou gerenciar o projeto Admin. O gerenciamento de sistema e projeto Admin serão, daqui em diante, referenciados como *Configurações Admin* e *Projeto Admin*, respectivamente.

Configurações Admin

Um usuário com permissões de Admin conseguirá consultar e modificar configurações globais do sistema. Isto é, ele será capaz de fazer alterações que modificarão e impactarão todos os projetos e usuários, caso deseje.

É possível consultar, por exemplo, todo o conjunto de recursos agregados pelo hipervisor, por servidor. Essa visão é importante para a administração pois mostra o estado atual de uso dos recursos, além de exibir quais servidores estão sendo mais consumidos dentro do data center. A Figura 3.5 demonstra essa funcionalidade através da tela de *Hipervisor*.

Outra funcionalidade importante é a consulta de informações de uso por projeto IaaS. Cada projeto no OpenStack possui, como explicado na seção 4.2.2, uma quantidade limitada de recursos a serem utilizados. O OpenStack cuidará de provisionar, portanto, somente a quantidade de recursos limite para cada projeto. Dessa forma, é garantido que nenhum projeto utilizará mais recursos do que o lhe é devido.

Porém, pode ser interessante ao administrador da nuvem saber quanto de recurso está sendo, efetivamente, utilizado por cada projeto. Dados de uso como estes podem ser utilizados para alertar ao usuário quanto a iminente necessidade de mais recursos, alertar ao administrador de projetos que possuem recursos subutilizados e, à depender do modelo de negócio, ativar redimensionamentos automáticos de recursos[2].

A Figura 3.6 mostra a tela *Overview* com tais informações. É possível filtrar as informações por períodos de utilização, como exemplificado na imagem, viabilizando uma possível precificação e bilhetagem por utilização de recursos da nuvem.

Foram citados até agora ferramentas de administração passiva, onde o administrador consegue mensurar o uso dos recursos de seu data center. Além de ferramentas de monitoramento, existem também as opções de provisionamento, onde o administrador consegue, em um modelo de administração ativa, disponibilizar opções de *flavors* e *images* para todos os *tenants* cadastrados no OpenStack.

[2] Redimensionamento automático não é um recurso nativo da pilha OpenStack. A existência de APIs para controle administrativo, porém, viabiliza a criação de scripts que façam esse redimensionamento, cabendo ao administrador essa implantação.

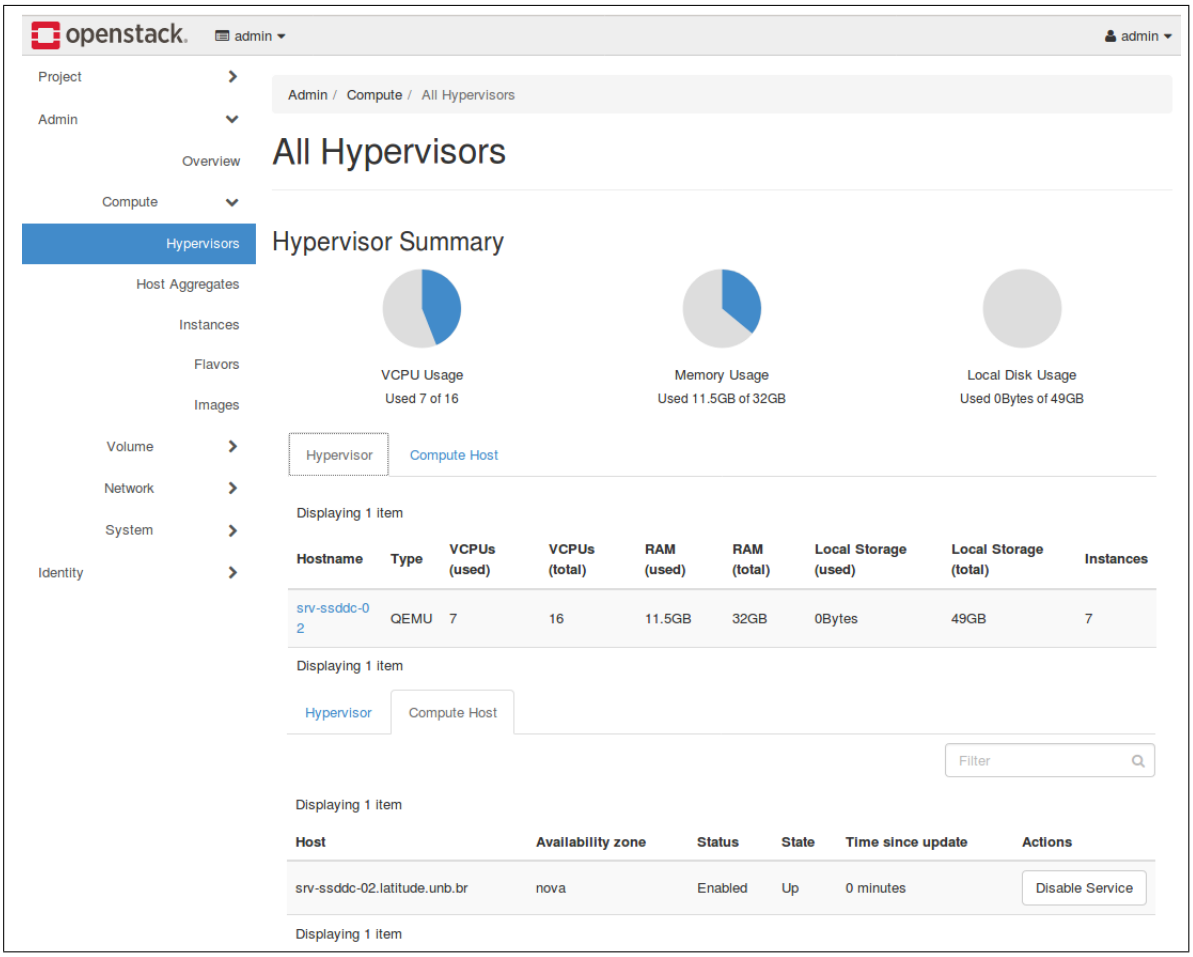


Figura 3.5 – Página de agregação de recursos globais. Administrador pode consultar nessa tela informações de RAM total e utilizadas, VCPU total e utilizado, instâncias criadas, além de muitas outras informações. Destaca-se a possibilidade de visualização de recursos por servidor, apesar de haver apenas um no exemplo.

Flavors definem, na pilha OpenStack, a capacidade de memória, computação e armazenamento para novas instâncias. De forma simples, um *flavor* é uma pré-configuração de *hardware* que define os parâmetros de uma instância virtual. (OpenStack Documentation, 2019)

Os *flavors* são importantes pois servirão de esqueleto para as instâncias criadas pelos usuários. O Administrador deve, portanto, criar *flavors* que julgue atender as necessidades dos usuários. Na Figura 3.7 é possível observar os *flavors* que são criados por padrão na instalação da pilha OpenStack. Destaca-se a importância dos *flavors* pois definem, para uma instância, o número de VCPUs, a RAM, o volume a ser ocupado pela instância, a memória SWAP, a razão escrita/leitura de disco, além de ser possível direcionar um *flavor* para um *tenant* em particular (por padrão, todos os *flavors* ficaram acessíveis a todos os *tenants*).

Outra ferramenta de administração ativa na pilha OpenStack é o provisionamento

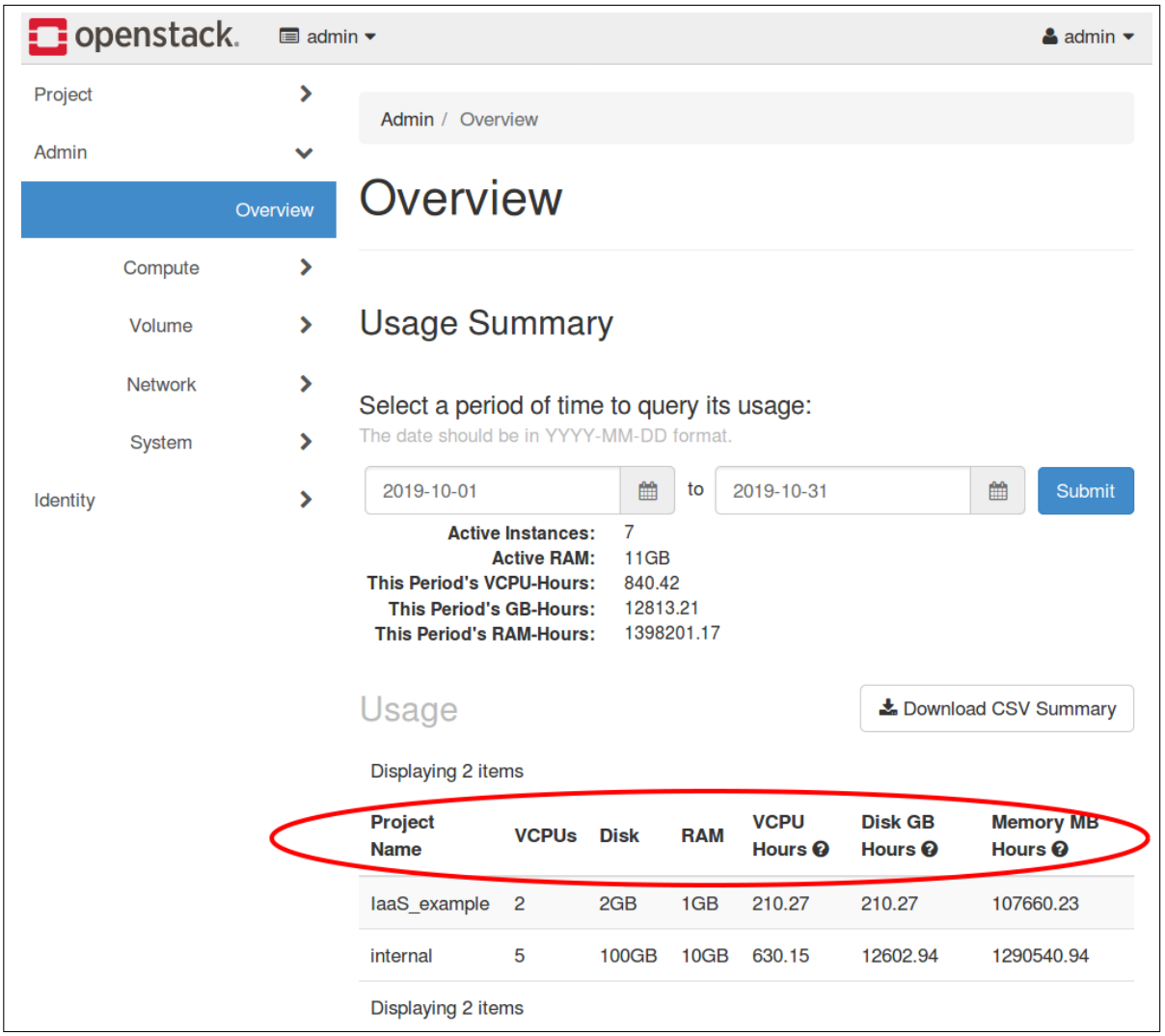


Figura 3.6 – Página de *Overview* com informações de consumo de recursos do data center por cada projeto. Exemplifica-se também o uso dos filtros por período para possível precificação e bilhetagem de IaaS. Destaca-se, também, a possibilidade de extração de dados para arquivo CSV.

de Imagens. Para montagem de uma instância, é necessário a utilização de um *flavor* anexado a uma Imagem de disco. Essa imagem servirá como um disco virtual, trazendo configurações para a instalação da instâncias, como o Sistema Operacional e requisitos mínimos.

Diferente dos *Flavors*, o cliente de IaaS pode, por conta própria, subir uma imagem de sua preferência e usar em seu projeto. O Administrador do data center, porém, pode prover algumas imagens como padrão para facilitar a vida do usuário. Imagens podem ser importadas a partir de vários formatos de arquivos, incluindo .ISO, .OVA, .QCOW2, .VDI, docker, etc.

Existe também a opção de importar Imagens com serviços pré instalados. A pilha OpenStack disponibiliza por padrão várias opções, como servidores web, softwares de

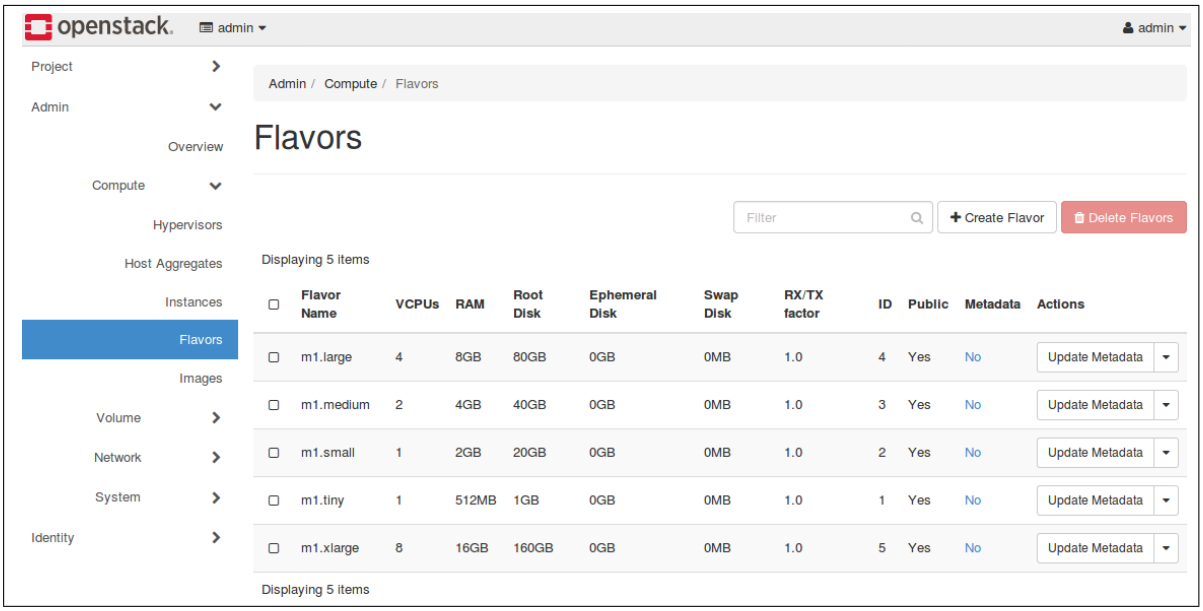


Figura 3.7 – Página de *Configurações Admin* com informações sobre *flavors* disponíveis por padrão na pilha OpenStack. Observa-se a importância dos *flavors*, pois constituem os parâmetros a serem usados por instâncias virtuais.

banco de dados, drivers VMware, etc. O administrador pode também personalizar um sistema operacional da forma que desejar e importar o disco virtual para uma imagem, disponibilizando assim para todos os *tenants* do OpenStack.

Por fim, também é função das *Configurações Admin* a criação e manutenção dos acessos ao OpenStack. Na aba de *Identity*, é possível criar projetos (como explicado em 4.2.1), usuários para acesso aos projetos e grupos de agregação de usuários.

Projeto Admin

O *Projeto Admin* possui as mesmas características e permissões, por padrão, que os outros projetos da pilha OpenStack. A necessidade de um projeto Admin surge por conta da dependência entre projetos e instâncias. Isto é, não é possível criar uma instância que não esteja atrelada a um projeto.

Portanto, o *Projeto Admin* pode ser usado para, por exemplo, criação de cenários de teste com instâncias e topologias de rede, ou hospedar serviços de uso comum entre todos os projetos. Para o segundo caso, o administrador cria uma instância com o serviço que deseja disponibilizar e coloca essa instância na rede *Backbone* de uso comum. Na seção 4.2.2 é demonstrado um uso para essa solução.

A Figura 3.8 demonstra a topologia lógica do OpenStack, incluindo a pilha sob a qual ele é implementado.

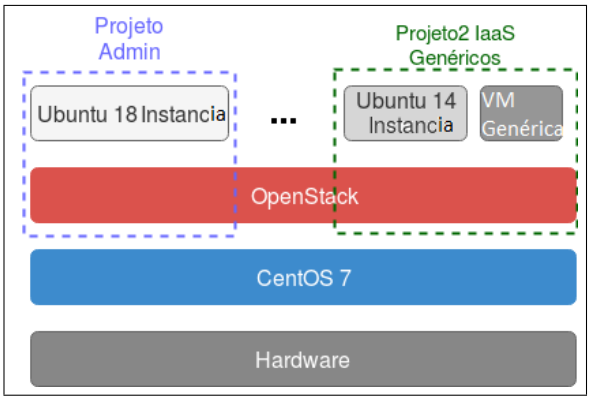


Figura 3.8 – Observa-se a topologia lógica do OpenStack e os *tenants* responsáveis por dividir, logicamente, os recursos virtualizados pelo OpenStack.

3.3.2 WireGuard

WireGuard é uma ferramenta gratuita e *open-source* que utiliza do protocolo de mesmo nome para implementação de *virtual private networks* (túnel seguro de rede). O protocolo WireGuard de túnel seguro opera em camada 3 do modelo OSI e é implementado como uma interface virtual do *kernel* do Linux. O protocolo WireGuard foi desenvolvido com a intenção de substituir as soluções de IPsec e OpenVPN, buscando ser mais seguro, performático e de fácil uso Donenfeld (2018). Mesmo sendo extremamente simples e moderno, a VPN criada pelo WireGuard utiliza criptografia de ponta. Tem a pretensão de ser tão fácil de configurar e implantar quanto SSH, a conexão é feita simplesmente pela troca de chaves públicas, exatamente como a troca de chaves do SSH. Seu uso é comparável com populares soluções de túnel VPN como IPSec e OpenVPN e visa substituir na maioria dos casos por ser mais seguro, performático e fácil de utilizar.

A interface virtual de túnel baseia-se em um princípio fundamental proposto de túneis seguros, uma associação entre o par chave pública e o endereço IP de origem do túnel (DONENFELD, 2018). O WireGuard usa uma única troca de chaves de ida e volta, com base no NoiseIK, e lida com toda a criação de sessões de forma transparente para o usuário. Conforme Trevor Perrin (2018), NoiseIK é um *framework* parar criação de protocolos *crypto*. Os protocolos de NoiseIK suportam autenticação mútua e opcional, ocultação de identidade, sigilo de encaminhamento, criptografia zero de ida e volta e outros recursos avançados.

O WireGuard elimina o uso de separações em camadas que os demais protocolos de túneis VPN. Em vez da complexidade do IPSec e das camadas xfrm, o WireGuard simplesmente fornece uma interface vitual, `wg0`, por exemplo, que pode ser administrada usando os utilitários padrões de configurações de interface no Linux, `ip(8)` e `ifconfig(8)`. Depois de configurar a interface com uma chave privada e as várias chaves públicas de pares com quem se comunicará com segurança, o túnel estará funcionando. No WireGuard,

troca de chaves, conexões, desconexões, reconexões, descobertas e desenvolvimentos importantes acontecem por de trás de forma transparente e confiável, e o administrador não precisa se preocupar com esses detalhes. Em outras palavras, a partir da perspectiva de administração, a interface do WireGuard parece estar sem estado. Regras de *firewall* podem ser configuradas usando a infraestrutura comum para interfaces de *firewall*, com a garantia de que pacotes provenientes de uma interface WireGuard serão autenticados e criptografados.

Suporta transporte para ambos os protocolos, IPv4 e IPv6, e consegue encapsular versão 4 em versão 6, assim como, versão 6 em versão 4. O WireGuard reúne esses princípios, concentrando-se na simplicidade e em uma base de código auditável, enquanto ainda é extremamente alta velocidade e adequado para uma pequena quantidade de ambientes. Combinando a troca de chaves e a criptografia de transporte da camada 3 em um mecanismo e usando uma interface virtual em vez de uma camada de transformação, o WireGuard realmente quebra os princípios tradicionais de camadas, em busca de uma solução sólida de engenharia que seja mais prática e mais segura. Ao longo do caminho, emprega várias soluções criptográficas e de sistemas inovadoras para atingir seus objetivos.

O princípio fundamental de uma VPN segura é uma associação entra pares e os endereços IP que cada um pode usar como IPs de origem. No WireGuard, os *peers* são identificados estritamente por suas chaves públicas, um `Curve25519 point` de 32 bytes. Definindo que existe um mapeamento de associação simples entre chaves públicas e um conjunto de endereços IP permitidos. A interface do WireGuard em si possui uma chave privada e uma porta UDP na qual ela escuta, seguida de uma lista de *peers*. Cada *peer* é identificado por sua chave pública. Cada um possui uma lista de IPs de origem permitidos. Quando um pacote de saída está sendo transmitido em uma interface `wg0`, esta tabela é consultada para determinara qual chave pública usar para criptografia.

Para começar a enviar pacotes encapsulados criptografados, um *handshake* de troca de chave 1 RTT deve ocorrer primeiro. O iniciador envia uma mensagem ao destinatário e o destinatário envia uma mensagem de volta ao iniciador. Após esse *handshake*, o emissor pode encaminhar mensagens criptografadas usando um par compartilhado de chaves simétricas, uma para enviar e outra para receber, para o respondente e após a primeira mensagem criptografas do emissor para o destinatário, o destinatário pode começar a enviar mensagens criptografadas para o emissor, conforme a Figura 3.9. Restrições de pedidos de exigência de confirmação são abordados conforme descrito para o KEA + C Lauter e Mityagin (2006), além de permitir que a mensagem de *handshake* seja processada de forma assíncrona para transportar mensagens de dados (DONENFELD, 2018). Essas mensagens usam o padrão "IK"do Noise Trevor Perrin (2018), além de uma nova construção de *cookies* para mitigar ataques de *denial of service*. O resultado líquido do protocolo é um sistema de segurança bastante robusto, que atende aos requisitos de segurança

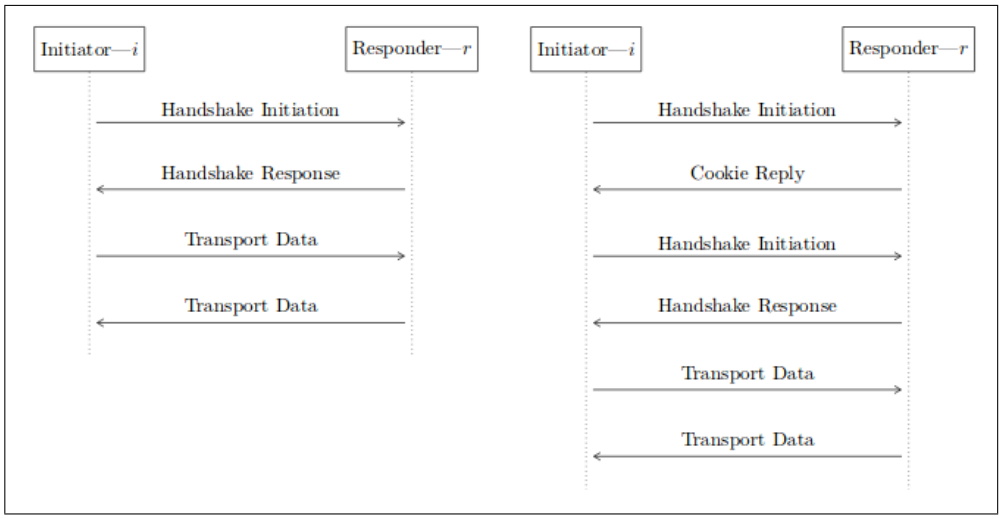


Figura 3.9 – Do lado esquerdo é exemplificado um *handshake* que ocorre na maioria dos casos, no qual é completado com apenas 1 RTT, seguido do transporte dos dados. Do lado direito demonstra se um *peer* está sobrecarregado, realizando a adição de uma mensagem de resposta de *cookie* ao *handshake*, evitando assim ataques de *denial of service*. Fonte: Donenfeld (2018)

.

de troca de chaves autenticada (AKE) Lauter e Mityagin (2006), evita a representação de comprometimento de chave, evita ataques de repetição, fornece sigilo direto perfeito, oferece ocultação de chaves públicas estáticas, e tem resistência a ataques de *denial of service*.

Túnel VPN entre instâncias

Uma das preocupações que precisam ser abordadas pela solução de implementação proposta nesse projeto é a segurança dos dados em trânsito (além de outras preocupações, também abordadas). Em particular, é proposto nesse projeto uma solução segura de armazenamento distribuído (detalhada na seção 4.2.4). Nesse contexto, a ferramenta WireGuard serve precisamente o propósito na qual foi construída: garantir privacidade em comunicações pela rede (redes internas ou *Internet*).

Com isso em mente, um túnel VPN é configurado entre duas instâncias, uma em cada servidor, no projeto *Admin*. Procura-se, com essa implantação, garantir que toda a comunicação entre as duas instâncias seja segura e privada. De fato, o usuário de uma IaaS como a proposta sequer precisa saber se a comunicação física entre os dois data centers que servem suas IaaS é segura.

Como o WireGuard estará sendo implantado nas instâncias virtuais à nível de kernel (DONENFELD, 2018), os pacotes saem dessa instância pela interface virtual criada pelo túnel do WireGuard já cifrados, não ficando disponíveis para leitura nem pelo OpenStack nem pelo sistema operacional da máquina física onde está instalado. A Figura 3.10

mostra a topologia lógica da solução proposta com o WireGuard.

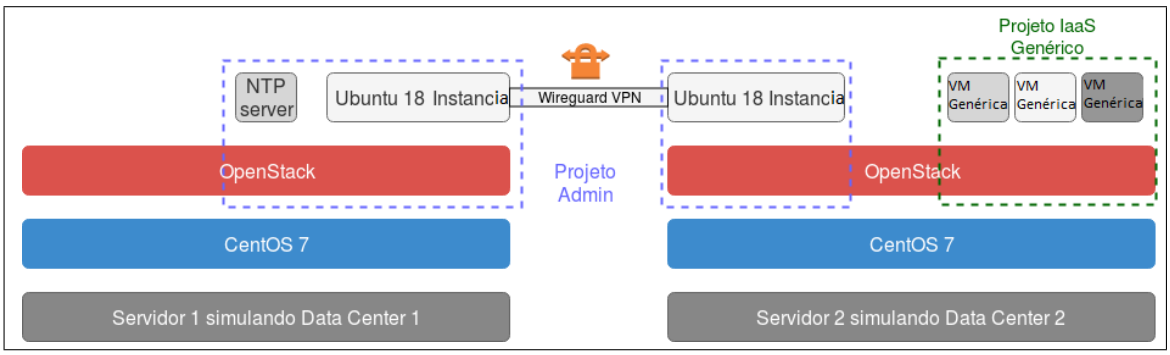


Figura 3.10 – Topologia lógica da solução com WireGuard. Observa-se que um túnel VPN foi implantado entre duas instâncias de data centers diferentes.

3.3.3 GlusterFS

Em uma organização clássica de data center, umas das preocupações é em como lidar com armazenamento (e backup dos dados armazenados). Existem diversas opções de configuração para armazenamento em data center, como explorado em 2.1. Em uma infraestrutura virtualizada, como é o caso da solução IaaS aqui proposta, conceitos que definem topologias físicas perdem o sentido. Em particular, não há necessidade da construção de redes SAN dentro de um projeto IaaS. Essa preocupação à nível físico ficará com o administrador do hardware onde a pilha OpenStack está instalada.

Os usuários responsáveis por um *projeto* ainda possuem, no entanto, algumas decisões a serem tomadas em respeito a armazenamento. É possível subir instâncias de computação associando uma Imagem (que carrega um OS e requisitos mínimos) a um *Flavor* (que trás uma pré-configuração de recursos de hardware). Apesar da abstração de virtualização presente nessa instância, ela funciona com um modelo DAS de armazenamento, que pode não ser o ideal para instâncias que precisem de sincronismo de dados entre instâncias, ou para um sistema que precise disponibilizar dados armazenados com alta disponibilidade.

Para resolver esse problema, é proposto uma solução de armazenamento distribuído com a utilização do GlusterFS. O GlusterFS é usado para armazenamento redundante e escalável de dados por meio de *clusters*[3] de servidores (ou instâncias). É utilizado uma arquitetura distribuída sem uma figura centralizadora, provendo escalabilidade e redundância. (DEVELOPERS, 2006). O Gluster usa sua lógica de software para garantir a replicação e distribuição dos dados por todo o *cluster*, independente do hardware que esteja por baixo (ou, em particular, independente de qual instância esteja instalado).

[3] Clusters são agrupamentos de servidores interligados que funcionam em sincronia para execução performática de uma ou mais tarefas.

O GlusterFS, na prática, funciona como um serviço em cima de instâncias de computação. Cada instância rodando o serviço do GlusterFS (*GlusterFS node*) fará o trabalho de interpretar o volume anexado àquela instância e disponibilizar ao agrupamento compartilhado de recursos de armazenamento do *cluster*. Dessa forma, disponibilizado na rede, existirá um volume de blocos (*Block Storage*) no qual, qualquer instância pode montar no seu *filesystem* e começar a escrever dados. O procedimento completo de instalação e operação do GlusterFS está descrito no Anexo B, assim como os tipos de volume que o GlusterFS disponibiliza para diferentes usos.

Uma solução como a proposta aqui, com o GlusterFS, trás para a infraestrutura de IaaS diversas opções e muita autonomia. Abaixo são listados alguns exemplos de cenários que podem se beneficiar de uma solução com o GlusteFS.

- Um ambiente de servidor Web, onde há duas ou mais instâncias servindo uma mesma página Web. É possível, nesse cenário, criar um volume em modo réplica, compartilhado e montado por todas as instâncias que servem a página Web. Nesse volume compartilhado poderá ficar, então, o banco de dados em comum dessas instâncias, assim como os arquivos de configuração dos servidores e os arquivos de conteúdo do site. Usando um balanceador para as requisições Web, as instâncias poderão servir o site em alta disponibilidade, sempre lendo e escrevendo do mesmo volume, evitando assincronismo.

- Uma instância rodando uma aplicação de coleta de imagens, por exemplo, poderia depositar todos os dados em um volume distribuído. Outra instância parte do *cluster*, por sua vez, poderia fácil e continuamente fazer uma análise e categorização dessas imagens do volume distribuído, não precisando requisitar da instância de coleta de imagens por novas imagens de tempos em tempos.

- Um simples compartilhamento manual de arquivos entre usuários do mesmo projeto. O GlusterFS, nesse caso, entrega uma solução prática que pode substituir um servidor de FTP rodando localmente para troca de arquivos.

Estes são alguns casos de exemplo onde o GlusterFS poderia ser útil em uma IaaS. Muitos outros problemas podem ser resolvidos com essa solução, cabendo apenas ao usuário decidir como fará o uso do GlusterFS. A Figura 3.11 mostra a topologia lógica da solução proposta com um volume criado com o GlusterFS.

3.3.4 NextCloud

Sistemas de armazenamento em nuvem permitem que usuários acessem seus dados de qualquer lugar, sendo possível fazer *upload* ou *download* de seus arquivos, além de compartilha-los com qualquer pessoa, tudo isso através da internet.

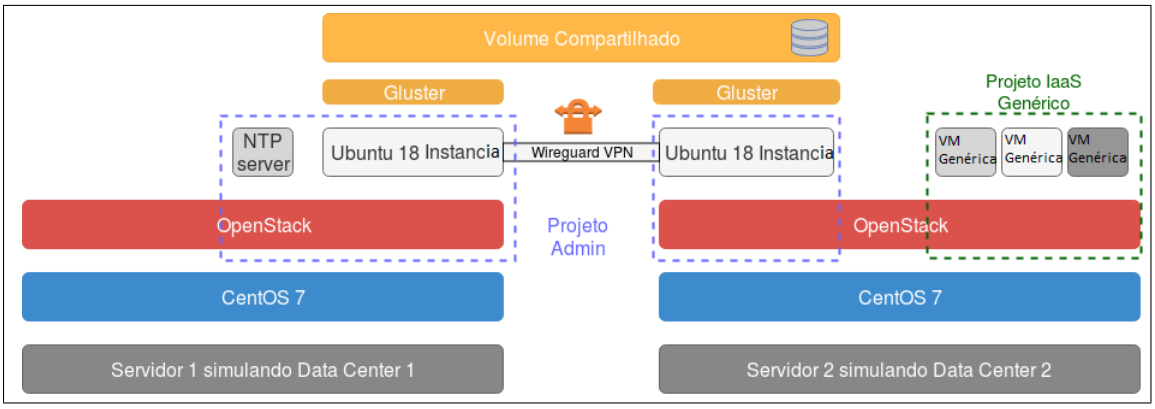


Figura 3.11 – Topologia lógica da solução proposta com um volume criado com o GlusterFS. Observa-se que este volume está acessível às instâncias dos projetos Admin, além de estar transitando por um túnel VPN.

Serviços de armazenamento de dados na nuvem (como Google Drive e Dropbox) vem ganhando muita popularidade nos últimos anos, e companhias vem disputando fatia desse mercado de armazenamento na nuvem. Muito pouco, porém, é sabido sobre a infraestrutura que os provedores de nuvem arquitetam suas soluções, como menciona Drago et al. (2012).

Existem, no entanto, soluções de código aberto para StaaS (*Storage as a Service* - armazenamento como serviço). Essas soluções permitem que um usuário utilize sua própria infraestrutura para prover um StaaS. É o caso de soluções como OwnCloud, Nextcloud, Seafile e Pydio. A Figura 3.12 compara o número de buscas a cada um dos termos das soluções mencionadas. É possível notar que a solução mais recente das quarto, o Nextcloud, agora lidera as pesquisas, mostrando uma possível liderança entre soluções de StaaS abertas. Por esse motivo, foi decidido seguir a implementação lógica desse trabalho com a plataforma Nexcloud.

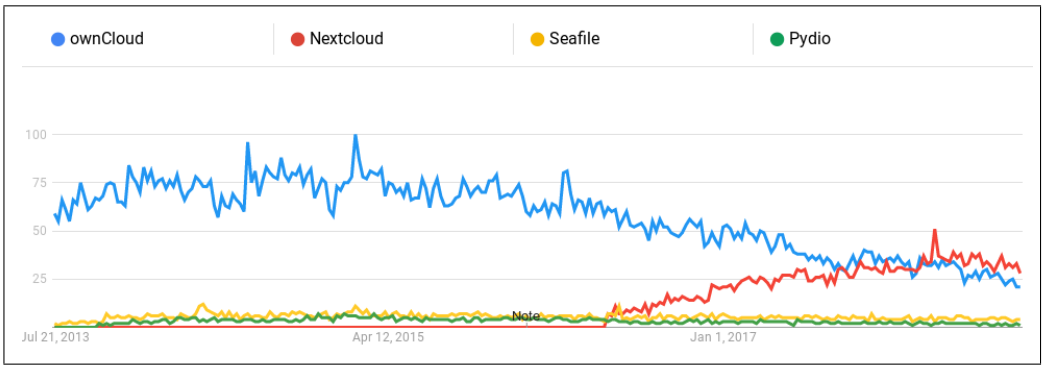


Figura 3.12 – Comparação do número de buscas a cada uma das soluções abertas de StaaS. É possível notar que a solução mais recente das quarto, o Nextcloud, agora lidera as pesquisas, mostrando uma possível liderança entre soluções de StaaS abertas.

Nextcloud é uma plataforma cliente-servidor para criação e utilização de serviço de hospedagem de arquivos. A aplicação *Nextcloud* do lado do cliente é muito similar ao *Dropbox*, onde usuários cadastrados podem fazer *upload* de seus arquivos pessoais para a nuvem, através de um aplicativo móvel ou de um *web browser*. Do ponto de vista do servidor, é utilizado a infraestrutura existente no ambiente em que o *Nextcloud* é instalado para o armazenamento que a ferramenta provê, sendo possível disponibilizar uma página de acesso ao serviço localmente ou na internet. *Nextcloud* é uma plataforma gratuita e de código aberto.

Uma vez que o *Nextcloud* utiliza-se da infraestrutura existente, é possível executar sua instalação na própria infraestrutura de nuvem proposta por esse trabalho. De fato, usando o volume compartilhado montado pelo Gluster e explicado na seção 3.3.3, é possível hospedar o serviço do *Nextcloud* em infraestrutura distribuída, podendo efetivamente criar um serviço com alta disponibilidade e seguro.

A seção 4.2.4 propõe e implementa uma solução com a estrutura aqui discutida. Além disso, a Figura 3.13 exemplifica a topologia lógica proposta. É importante reparar que a Figura 3.10 e a Figura 3.2 são iguais. Não é, entretanto, por acaso: significa que, com a implementação do NextCloud, está completa a estrutura proposta por esse trabalho. Ademais, as figuras desse capítulo 3 foram progressivamente incluindo mais componentes lógicos. Essa estrutura encadeada de abstração levará, no capítulo 4, há uma proposta de serviços de SSDDC utilizando-se das tecnologias e conceitos apresentados até aqui.

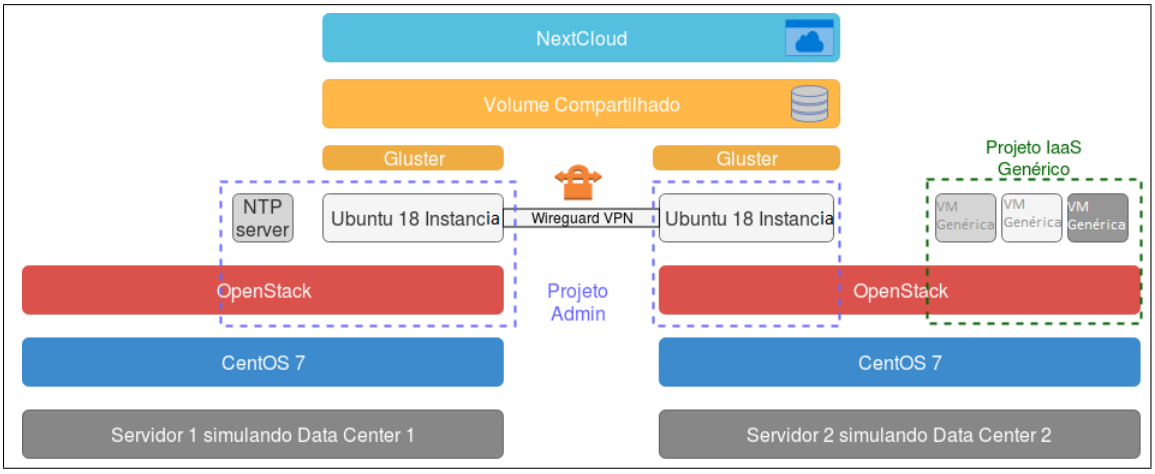


Figura 3.13 – Topologia lógica proposta. Na Figura, observa-se, em particular, a localização lógica do serviço Nextcloud, fazendo uso da infraestrutura criada pelo GlusterFS para provimento de alta disponibilidade. A seção 4.2 tratará de serviços SSDDC possibilitados por essa estrutura lógica.

4 Implementação

O capítulo 3 buscou explicar o problema abordado e as topologias lógicas e físicas da solução proposta por esse trabalho. Já neste capítulo 4, busca-se demonstrar toda a implementação necessária para alcançar os objetivos propostos.

Ao final deste capítulo, espera-se que seja possível entender como pode ser executada a implementação das tecnologias envolvidas na solução, assim como o papel de cada etapa na construção da solução segura de armazenamento distribuído em ambiente de nuvem privada.

4.1 Implementação do Openstack

A implementação do OpenStack é estritamente dependente do sistema operacional de base e dos componentes de *hardware* existentes, assim, onde ele será implementado tem grande significância para o seu correto funcionamento e provisão dos recursos esperados pelo administrador da *cloud* ou usuário dos recursos. Para a instalação do OpenStack é necessário distribuições Linux baseadas em Red Hat Enterprise Linux (RHEL), no mínimo na versão 7 e com arquitetura de processador igual a x86_64, sendo a única suportada. Assim a distribuição CentOS na sua versão 7 foi à utilizada como sistema operacional base para o OpenStack. Os requisitos mínimos de *hardware*, como processador com extensão de virtualização de *hardware*, interface de rede, uma máquina com pelo menos 16GB de memória RAM, foram todos atendidos, conforme é descrito na seção 3.2, explicando quais recursos o projeto teve a disposição.

Com a instalação do sistema operacional CentOS 7 realizada, configurações de rede foram executadas, sendo de extrema importância para que as instâncias criadas no OpenStack tenham acesso a rede externa, ou seja, uma rede que será criada no OpenStack chamada de rede **external** (*backbone*) é uma abstração da rede física na qual o servidor que o OpenStack está sendo instalado está ligado, sendo abordada sua criação adiante nesse capítulo. Uso de IP estático nas interfaces utilizadas no servidor, conforme a distinção de cada uma das interfaces na seção 3.2. As configurações de rede aplicadas no CLI do CentOS estão descritas no anexo A deste documento. A instalação do OpenStack pode ser realizada com vários instaladores, dos quais irão diferenciar em recursos disponíveis ou funcionalidades presentes. A utilizada foi o PackStack, uma ferramenta de instalação do OpenStack intencionada para demonstração e prova de conceito de desenvolvimento. PackStack usa SSH para conectar com cada nó e chama uma execução *puppet* (especificamente uma aplicação *puppet*) em cada um dos nós para instalar e configurar o OpenStack.

Na instalação do OpenStack, segundo o anexo A, uma passo importante a ser mencionado é o uso do comando `packstack -allinone` para a criação da rede externa (*backbone*) de forma abstraída no OpenStack. É repassado juntamente ao comando argumentos que definirão como funcionará a comunicação entre as instâncias dentro do OpenStack com a Internet. É feito um mapeamento de interfaces para que ocorra essa comunicação e transporte de dados, descendo toda a pilha de virtualização presente no contexto. Defini-se um nome lógico para o segmento físico externo de camada 2 como "extnet", esse segmento de camada 2 que será responsável por conectar a interface física presente no servidor, com a interface virtual criada "br-ex", para que toda a pilha de infraestrutura do OpenStack à utilize. Essa comunicação entre as duas interfaces, virtual "br-ex"e a física de serviço, "em1"presente, é realizada através de uma *bridge*. Uma *bridge* é um dispositivo de rede que cria uma agregação de rede de múltiplas comunicações de rede ou segmentos de rede, atua como um filtro transmitindo apenas quadros de mensagens recebidos em um lado da *bridge*, tendo o endereço de destino de um dispositivo localizado no outro lado da *bridge* (KOCH et al., 1988). O serviço do OpenStack por fazer essa comunicação em *bridge* é o Neutron, cuja sua função é mencionada na seção 3.3.1. O recurso que o Neutron utiliza para isso é o `OVS`, projetado para permitir automação de rede massiva por meio de extensão programática, isolamento de VLAN, filtro de tráfego, propriedades de QoS e monitoramento de fluxos, segundo (Linux Foundation Collaborative Projects, 2016).

As configurações das interfaces mencionadas anteriormente para o correto funcionamento da *bridge* e com os parâmetros a serem repassado é exemplificado no anexo A. Com todos os procedimentos de instalação do OpenStack executados, os passos seguintes utilizados dentro da CLI do OpenStack, seguindo o conteúdo presente no anexo A, referem-se a criação dos componentes de rede que serão visíveis e utilizáveis nos projetos e pelos usuários da infraestrutura da nuvem.

Um procedimento importante na instalação do OpenStack, é a criação da rede `external`, seu processo de criação é novamente citado nesse capítulo pela importância que a rede, na qual, abstrai a rede externa existente, tem nos projetos do OpenStack. Com essa rede, as instâncias criadas tem conectividade com o mundo externo, sendo possível instalar *softwares* e pacotes. A criação da rede `external` necessariamente é executada no CLI do OpenStack, utilizando o componente Neutron. Os parâmetros passados no comando da criação fazem referência ao nome lógico do segmento físico externo de camada 2, "extnet", criado anteriormente e que o roteamento dos dados encaminhados para a rede `external`, será tratado pela própria rede externa, podendo ser visto no anexo A. O OpenStack difere em questões de conceitos à utilização e criação de uma rede. É definido por ele que uma rede corresponde ao tipo de usabilidade, ou seja, se é uma rede interna ou externa. Na rede não é configurado o CIDR e o *gateway*, a sub-rede que é responsável por ter esses requisitos. Com a criação da rede externa, criou-se a sua sub-rede. Para a criação da

sub-rede foi necessário utilizar um *range* de IPs fora do alcance do DHCP da rede externa (RDO Project, 2019), esses IPs serão importantes para a utilização de *floating* IPs para as instâncias. A explicação dos *floating* IPs será abordada mais adiante neste capítulo. Foi criado a sub-rede passando como parâmetros o IP do *gateway* da rede externa existente, uma faixa de dez IPs da rede externa e a CIDR da rede (*backbone*).

Uma vez finalizado todo o processo de instalação e implantação do OpenStack com as configurações de conectividade com o mundo externo, é preciso iniciar as demais implementações e configurações que envolvem o projeto. Essas configurações poderiam ser executados por linha de comando, chamadas de API ou pela interface gráfica. Para maior facilidade de entendimento e exemplificação, os próximos passos foram executados na interface gráfica *Horizon*, disponível no IP da interface de serviço do servidor onde foi instalado o OpenStack (*controller node*), porta 80.

Entrando, por meio de um *web browser*, no IP do *Horizon*, é requisitado do usuário um login e senha de autenticação. Nesse momento da implantação, só existe o usuário de administração, criado durante a instalação da pilha OpenStack. As credenciais para o usuário administrativo estão dentro do arquivo `keystonerc_admin` na pasta de instalação do *controller node*.

A partir do *login* com as credencias utilizadas, é acessível o projeto *Admin*. O acesso ao projeto *Admin* é visto como um administrador da *cloud*, possuindo gerência em projetos, usuários e regras. No projeto *Admin* é possível criar novos projetos, criar novos usuários e atribuir usuários à esses projetos com regras de gerenciamento que cada usuário terá em seus respectivos projetos. Usuários podem ser classificados de acordo com projetos em membros, administradores, leitores entre outras regras, sendo possível a criação de novas. A rede *external* criada anteriormente é visível e usável em todos os projetos devido a sua criação ter sido no projeto *Admin*. Assim como à usabilidade da rede para os demais projetos, imagens, *flavors*, grupos de segurança, roteadores, volumes, entre outros serviços disponíveis para serem criados através do Horizon também são disponíveis para uso em outros projetos quando sua criação é realizada no projeto *Admin*. Tomando como base esses conhecimentos aprendidos durante o projeto, foi criado uma imagem no projeto *Admin* referente ao sistema operacional Linux, mais especificamente uma distribuição Ubuntu, na sua versão 18. Imagem utilizada com formato de disco ISO com 64 MB de tamanho.

Para a criação de instâncias foi necessário selecionar uma imagem disponível, um *flavor* que define os requisitos de *hardware* e *software* que a instância irá obter, um volume e o tamanho do mesmo, em qual rede a instância será conectada, o grupo de segurança que delimitará quais protocolos poderão entrar ou sair da rede, e uma chave para o acesso via SSH com a instância. Dentro de um projeto criado com alguma funcionalidade, para a segmentação de funcionalidade e dados, o usuário do projeto pode criar redes internas e

conectá-las a rede *external*. Através dessa opção de gerenciamento do tráfego é necessário criar um roteador que conectará as duas redes. As configurações presentes no OpenStack envolvem somente a configurações de rotas estáticas ou interfaces. Cada projeto apresentado nas subseções de 4.2.2 à 4.3. Para a conectividade no sentido do mundo exterior com as instâncias é necessário a utilização dos *floatings* IPs. Os IPs privadas das instâncias são utilizados para uma comunicação entre instâncias, *floating* IPs são usados para a comunicação fora da *cloud*, incluindo a Internet (OpenStack's Documentation, 2019a).

4.2 Serviços SSDDC

Uma vez que toda a implantação do OpenStack é concluída, o administrador desse data center (que agora, por definição, passou a ser um administrador de nuvem privada), já controla a totalidade dos recursos do data center. É possível, portanto, a adição de serviços a serem consumidos pelos usuários da nuvem.

Nesse momento, serão definidos níveis de serviço implementados como parte da solução proposta. Os níveis representam, progressivamente, a abstração feita em relação ao modelo tradicional de data center. Não há, porém, qualquer metodologia científica que baseie a escolha de níveis para cada tipo de serviço. As escolhas foram feitas de forma a auxiliar o leitor no entendimento da topologia implementada.

4.2.1 Nível 1: IaaS - Solução de Infraestrutura como serviço

Com a pilha do OpenStack implantada, o administrador da nuvem pode provisionar recursos de rede, computação e armazenamento aos usuários. Esses recursos são aplicados a um projeto e esse projeto é administrado pelo próprio usuário, ou grupo de usuários participantes do projeto.

Como exemplo de implementação de IaaS (nível 1), foi criado um projeto com nome de *IaaS_example*. Na criação do projeto, por parte do administrador, é possível definir as cotas de infraestrutura, isto é, definir a quantidade de recursos a serem utilizados, como por exemplo:

- Número máximo de instâncias a serem criadas por um projeto.
- VCPU máximo dedicado ao projeto.
- Memória RAM máxima dedicada ao projeto.
- Volume total de armazenamento dedicado ao projeto.
- Limite de redes e subredes que o projeto poderá criar.
- Número de IP's da rede *Backbone* dedicados ao projeto.

Existem muitas outras configurações a serem definidas, como mostra a Figura 4.1. É importante, durante a alocação de recursos aos projetos, que o administrador atente-se aos recursos totais de sua infraestrutura física.

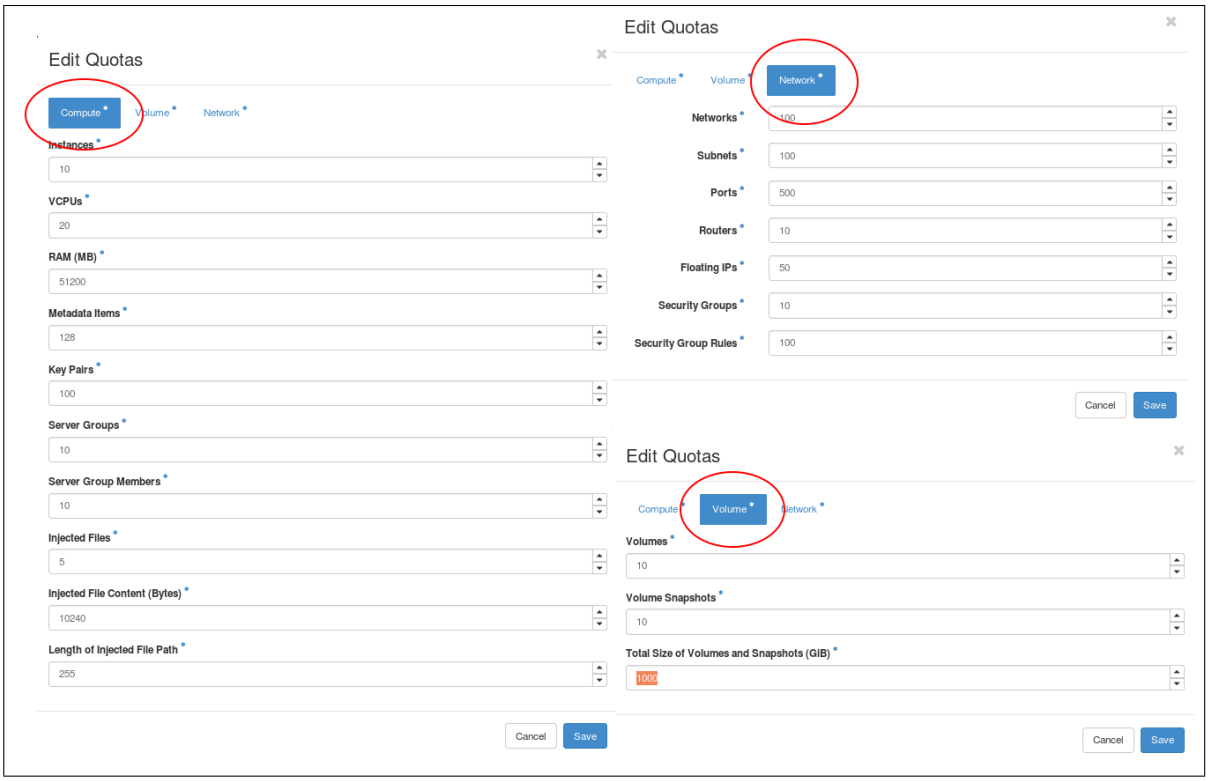


Figura 4.1 – Opções de cotas definidas pelo administrador da nuvem ao projeto de IaaS de exemplo.

Durante a criação de projetos, é importante também a criação e adição de usuários para o projeto. Um projeto pode ter mais de um usuário, e cada usuário terá um nível de acesso à IaaS, a ser decidido pelo administrador. Também é possível definir grupos de usuários, que herdarão as permissões do grupo.

Após a criação do projeto e definição das cotas de infraestrutura, o projeto já está pronto para ser utilizado. O usuário, criado durante a criação do projeto, pode acessar sua IaaS através de chamadas de API ou do *Dashboard Horizon*. Para acesso ao *Horizon*, o usuário necessitará de autenticação. Caso deseje utilizar as chamadas API, poderá encontrar, na aba *API Access* do *Horizon*, todos os *Services Endpoints* (URLs para chamadas de API) para cada serviço de sua IaaS.

Fazendo login através do *Dashboard Horizon*, o usuário poderá ver, na primeira página, as cotas definidas ao seu projeto, assim como a utilização atual, como mostra a Figura 4.2

Uma vez que o usuário tenha acesso a sua infraestrutura, é possível começar a usá-la para qualquer que seja o motivo. Para exemplificar as funcionalidades da ferramenta, foi criado um pequeno ambiente de servidor Web no projeto *Iaas_exemple*.

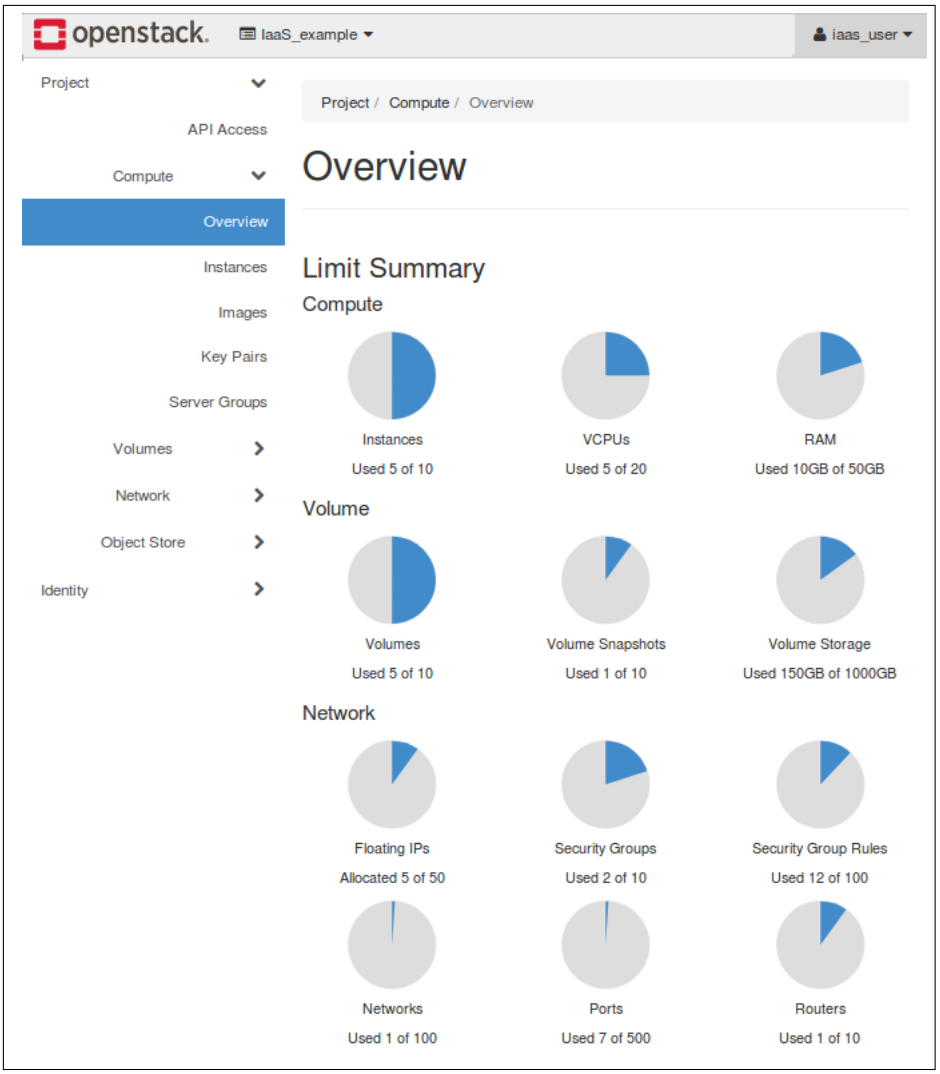


Figura 4.2 – Página de *Overview* das cotas do projeto *IaaS_example*. Nessa página, é possível ver a alocação de vários recursos de computação, rede e armazenamento, assim como seus respectivos usos.

Iaas_example: Ambiente *Web Server*

O primeiro passo para a utilização da Infraestrutura é a preparação da rede. Todo projeto por padrão possui acesso a rede *Backbone*, como explicado em 4.1. Porém, não é possível adicionar interfaces de instâncias diretamente à rede de *Backbone* por padrão. É preciso, então, adicionar um roteador virtual, aqui chamado de *Border_Router*, que terá uma interface na rede *Backbone* (172.16.9.0/24) e servirá de gateway para as redes e instâncias internas do projeto *Iaas_exemple*.

Em seguida, duas redes internas foram criadas para exemplo: *Production Network* (192.168.1.0/24) e *Homol Network* (192.168.2.0/24). Foi criado, também, uma interface no roteador *Border_Router* em cada uma dessas redes, sendo essas interfaces o *defaut gateway* de cada rede.

Em seguida, duas instâncias (Ubuntu 16.04 de 512 MB de RAM e 1 VCPU) foram

criadas, uma em cada rede interna. Para criação de instâncias, como já explicado em 4.1, o usuário conta as opções de template disponibilizados pelo administrador da nuvem.

O objetivo dessa implementação é simular um ambiente com uma rede de produção, com uma instância de servidor web e banco de dados, e outra rede de homologação com as mesmas características. A Figura 4.3 mostra a aba de *Network Topology* disponível no *Dashboard Horizon*, que ilustra a topologia criada. Ambas as redes se comunicam com a internet através da rede *Backbone*. Mas, caso necessário, essas redes poderiam ser totalmente isoladas, sendo acessíveis apenas internamente pelas instâncias à elas ligadas.

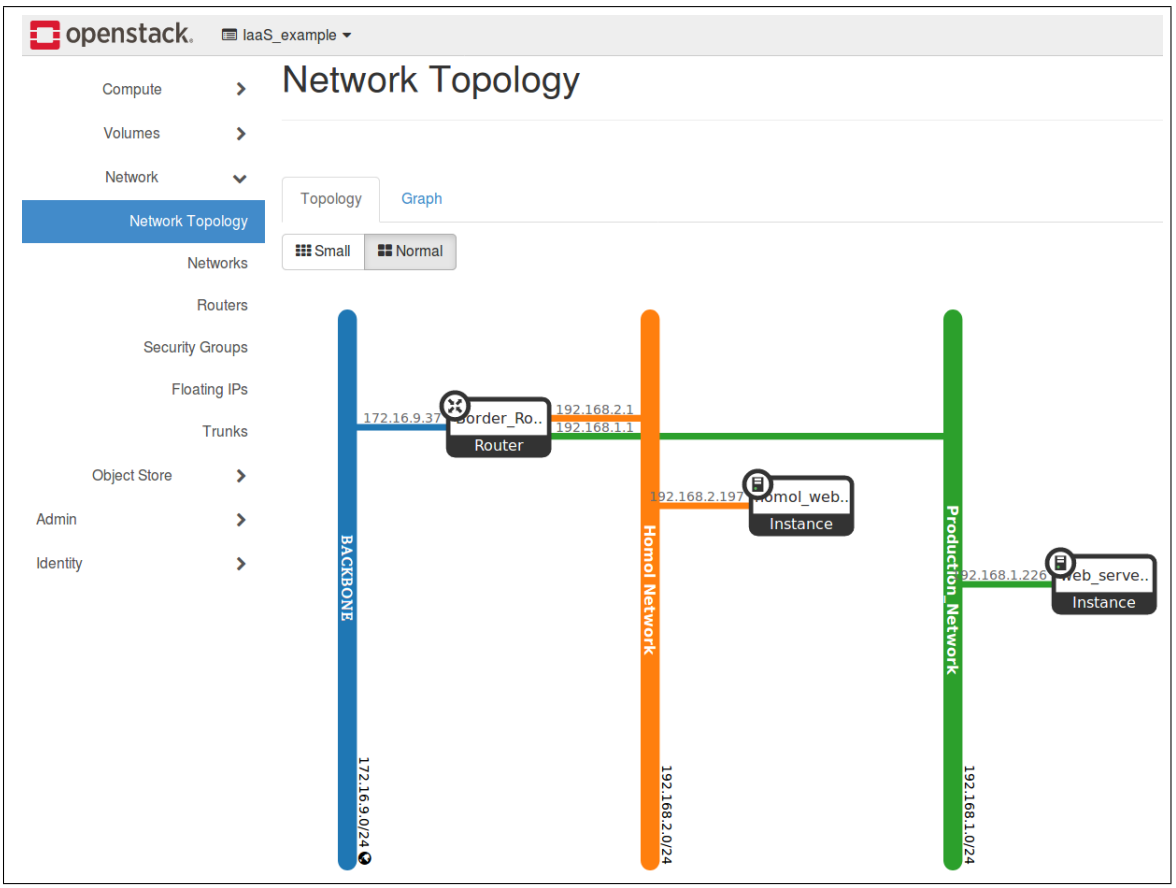


Figura 4.3 – Visualização da topologia de rede do projeto *IaaS_example*. Demonstração de um ambiente de webserver, com rede de Produção e Homologação.

Esse exemplo demonstra, de forma simples, o potencial de implementação de uma infraestrutura IaaS como essa. Através de uma interface gráfica simples, usuários da nuvem (desenvolvedores, DevOps, SysAdmins, etc) podem criar e editar ambientes inteiros de produção, homologação e testes, subir e editar instâncias com vários sistemas operacionais à escolha, definir políticas de segurança e QoS para cada rede interna, além de muitas outras funcionalidades, sem jamais precisar se preocupar com o Hardware onde a infraestrutura está rodando. Além disso, todo o provisionamento pode ser feito através de chamadas de API, possibilitando a automação de processos, efetivamente reduzindo o tempário com permissões de Admin conseguirá consultar e modificar configurações glo-

bais do sistema. Isto é, ele será capaz de fazer alterações que modificarão e impactarão todos os projetos e usuários, caso deseje. É possível consultar, por exemplo, todo o conjunto de recurso necessário para levantamento de infraestrutura de TI e acabando com os duradouros confrontos entre as equipes de infraestrutura e desenvolvimento.

4.2.2 Nível 2: Serviços IaaS compartilhados

O administrador da nuvem pode também prover serviços a serem consumidos por todos os projetos e usuários da nuvem. Essa decisão pode ser baseada, por exemplo, em uma análise dos serviços mais utilizados pelos usuários. Alguns exemplos de serviço seriam: Servidor de DNS, servidor de NTP, servidor de AAA, serviço de abertura de tickets para comunicação dos clientes da nuvem e *helpdesk*, entre outros. Tais serviços são aqui classificados como nível 2, pois são executados em cima de instâncias providas pela IaaS (nível 1) e entregues ao usuário como serviço adicional à infraestrutura de nuvem.

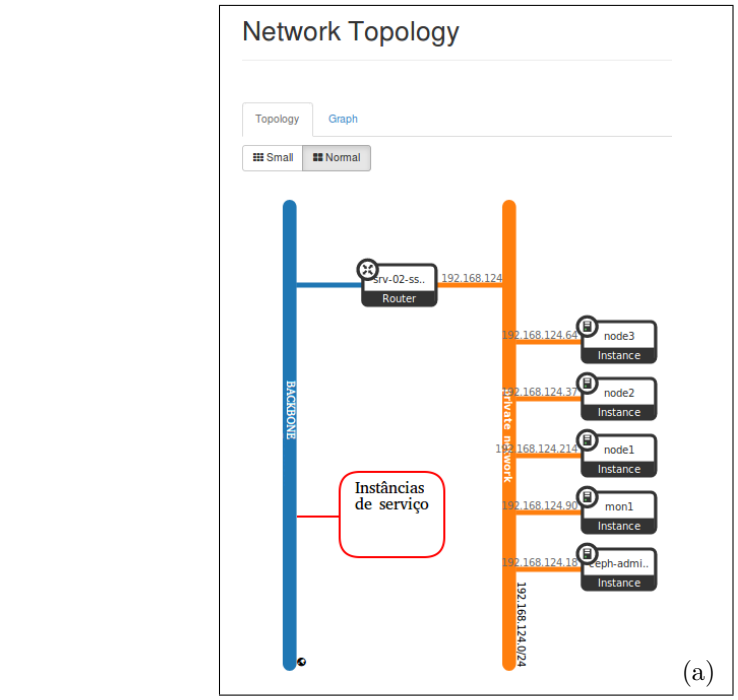
Como exemplo de implementação de serviço nível 2, um servidor de NTP foi instalado em uma instância dentro do projeto *admin*. Por se tratar de um serviço que deve ser compartilhada por todos os projetos, essa instância é colocada na rede *Backbone*, como explicado em 4.1.

Uma vez concluído a preparação da instância e a configuração do servidor NTP, outros projetos já podem utilizar esse serviço caso desejem. A instância possui um IP na rede *Backbone* (172.16.9.0/24 acessível à todos os projetos) e o serviço é disponibilizado na porta UDP 123. A Figura 4.4 mostra a topologia lógica de rede de um projeto de exemplo, e uma representação das instâncias de serviço disponibilizadas pelo Administrador. Além disso, é possível observar também uma instância, dentro de um projeto, sincronizada com o servidor NTP de uma instância em outro projeto (Projeto Admin).

4.2.3 Nível 3: Solução de armazenamento seguro e redundante com Gluster

Os usuários da IaaS provida pelo OpenStack podem precisar tomar decisões a respeito de armazenamento para seus projetos, como mencionado em 3.3.3. Para adereçar essa preocupação, propõe-se como serviço de Nível 3 do SSDDC uma solução de armazenamento distribuído com a utilização do Gluster. Pelas vantagens oferecidas pelo Gluster como sua alta escalabilidade em relação a adição de novos volumes de armazenamento ou a retirada dos mesmo, é alcançável por milhares de usuários no ponto de vista de usabilidade, provê replicação, quotas entre usuários em relação a volumes, geo-replicação, criação de *snapshots* e faz com que o *hardware* utilizado se torne um commoditie.

A solução de armazenamento seguro na implementação proposta a seguir, vem com a utilização do WireGuard como meio de transporte seguro, ou seja, o Gluster irá funcionar em cima da rede criada pelo WireGuard, assim, todo dado transportado entre



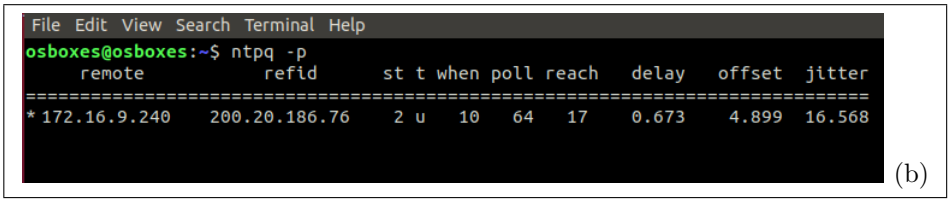


Figura 4.4 – Na Figura 4.4 (a) é possível ver a topologia de rede de um projeto de exemplo. Esse projeto possui acesso à rede *Backbone* e, consequentemente, à todos os serviços de nível 2 que o administrador disponibilizar, incluindo o servidor NTP. Esses serviços, apesar de disponíveis, não ficam visíveis na tela de *Network Topology* pois esta apenas trás a topologia criada por este projeto. Na Figura 4.4 (b) é possível observar o terminal de uma das instância, dentro de um projeto, sincronizada com o servidor NTP de uma instância em outro projeto (Projeto Admin).

os nós do Gluster será de forma segura. A outra especificação de segurança explicitada no projeto envolve o armazenamento dos dados de forma segura, como abordado em 2.6.2. O Gluster provê uma solução de armazenamento dos dados de forma criptografada, assim garantindo segurança no quesito dado armazenado.

Implementação

Como prova de conceito, foi montado um cenário com três *Gluster nodes* utilizando os servidores do OpenStack, ligados através de um *switch ethernet*, como explicado na subseção 3.2. Cada servidor possui o sistema operacional CentOS 7 e a pilha do OpenStack já instalada conforme subseção 4.1. A ideia é simular uma situação de compartilhamento de *storage* entre data centers. Com essa solução, é possível garantir alta disponibilidade

de dados armazenados, pois mesmo que algo aconteça com um data center, existe uma réplica dos dados em outro.

Três instâncias foram provisionadas para a prova de conceito, uma em um *controler node* e duas outras instância no outro *controller node*, utilizando o OpenStack. É utilizado uma imagem do Ubuntu 18.04 *server* para todas as instâncias e configurado grupos de segurança, *keypair* e volume com um tamanho definido para a instalação do OS nas instâncias conforme A. Além do da disponibilidade de crias instâncias de computação, o OpenStack, também disponibiliza instâncias de armazenamento, virtualizando os recursos do *hardware* usado. Como forma de separar os recursos armazenados no disco utilizado somente para o Gluster e os pacotes e bibliotecas utilizadas pelo OS é criado um volume exclusivo para o GlusterFS para as instâncias.

Através do CLI do OpenStack, com o usuário destinado ao projeto utilizado como prova de conceito, foram criados volumes em partições diferentes da partição que o OS é salvo para disponibilizar o disco novo inteiramente ao Gluster. Volumes de 10 GB foram adicionados as instâncias. A forma como é adicionado esses volumes está demonstrado no anexo A. É recomendável formatar a partição adicionado antes de sua utilização pelo GlusterFS, sendo definido o formato igual a `xfs`.

Na instalação do GlusterFS, conforme o Anexo B, para que as *bricks* sejam criadas entre as três instâncias é necessário utilizar os IPs atribuídos a cada instância utilizado pela rede que o WireGuard cria. Foi definido que a rede a ser utilizada pelo WireGuard tem CIDR igual a 10.9.0.0/24. Antes de ser executado o *probe* entre as instâncias é aplicado em todas as instâncias as regras de **firewall** que possibilitam o funcionamento do GlusterFS. Assim, com os IPs providos pelas interfaces que o WireGuard cria, é realizado o *probe* entre as instâncias dos dois *controller nodes*.

Com o *cluster* estabelecido utilizando o IP da rede do WireGuard, criou-se um volume em modo Réplica. Como explicado no Anexo B, este modo trabalhará sincronizando os dados de forma que tudo seja replicado entre os *controller nodes*. A topologia lógica da solução ficou conforme a Figura 4.5. A opção de volume do tipo `replicated` impõem o uso de três instâncias por causa do `split_brain`, segundo Gluster Docs (2019b). *Split brain* é uma situação em que duas ou mais cópias replicadas de um arquivo se tornam divergente. Quando um arquivo está em *split brain*, há uma inconsistência nos dados ou nos metadados do arquivo entre as *bricks* de uma réplica e não possui informações suficientes para escolher autoritariamente uma cópia como intocada e curar as cópias ruins, apesar de todas as *bricks* serem e estarem *online*.

Partindo para o ponto de vista de armazenamento seguro, o Gluster possui uma opção em relação aos volumes já criados que envolve a criptografia dos dados utilizando chaves. Depois do volume criado, é possível passar parâmetros como uso dos dados do volume de forma encriptada e o local e tamanho da chave utilizada para encriptar os

dados, conforme anexo B.

Com o volume criado, foi possível utilizá-lo como *filesystem* distribuído por rede e seguro em transporte e armazenado. O GlusterFS assim produziu um NAS entre todos os usuários da rede criada utilizando o WireGuard.

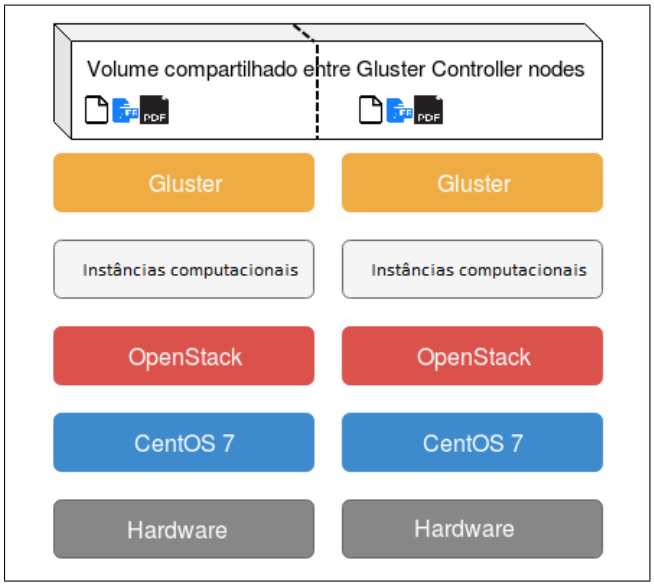


Figura 4.5 – Topologia lógica da solução nível 3 de armazenamento redundante com o Gluster. Destaca-se o volume compartilhado disponibilizado pelo Gluster em modo Réplica. Todo dado gravado em um data center é sincronizado com o outro, provendo alta disponibilidade de dados.

Essa mesma solução de nível 3 aqui proposta poderia ser replicada em um cenário entre *tenants* em um mesmo data center. Pode haver, por exemplo, uma colaboração entre dois laboratórios que utilizam a IaaS do mesmo centro de pesquisa. Nesse cenário, cada projeto poderia colaborar com uma instância para a criação de um volume compartilhado, facilitando assim o compartilhamento ou a alta disponibilidade dos dados.

4.2.4 Nível 4: StaaS - Uma solução de armazenamento como serviço segura e redundante

Propõe-se como serviço de nível 4 uma solução de armazenamento de arquivos na nuvem privada, disponibilizada à usuários independente de *tentants*, com alta disponibilidade, segurança e de fácil utilização. Para isso, é utilizado a plataforma *Nextcloud*.

Para prova de conceito do serviço de nível 4 do SSDDC, é utilizado o *Projeto Admin* para a base da implementação. Como explicado na seção 4.1, esse projeto de IaaS abriga os serviços complementares que os administradores da nuvem desejam disponibilizar aos demais usuários. Esse projeto será usado em ambos os servidores com a pilha OpenStack instalada, conforme explicado mais adiante.

Além da abstração de *hardware*, provida pelo serviço de nível 1 (subseção 4.2.1), é utilizado também a solução nível 3 de armazenamento seguro e redundante com Gluster. Como demonstrado na subseção 3.3.3, é possível estruturar, em cima da abstração de hardware provida pela IaaS, uma solução de armazenamento redundante e com alta disponibilidade de dados.

Portanto, novamente foi montado um cenário com dois servidores diferentes, ligados através de um switch ethernet, simulando uma situação de nuvem privada com data centers distribuídos. Em cada servidor, foi instalado o CentOS 7 e a pilha do OpenStack, conforme o Anexo A. Além disso, novamente o Gluster foi utilizado para criação de um volume compartilhado, em modo réplica, conforme Anexo B.

Com a estrutura de volume compartilhado (provida pelo Gluster) configurada, uma nova instância foi colocada na rede. Essa instância é responsável pela hospedagem do serviço Nextcloud, pela montagem do volume compartilhado em seu *filesystem* (que será o ponto de escrita dos arquivos do Nextcloud) e por responder às requisições de usuários que desejem utilizar do seu serviço. É importante, portanto, atribuir um IP flutuante (como explicado na seção 4.1) da rede *Backbone* à essa instância.

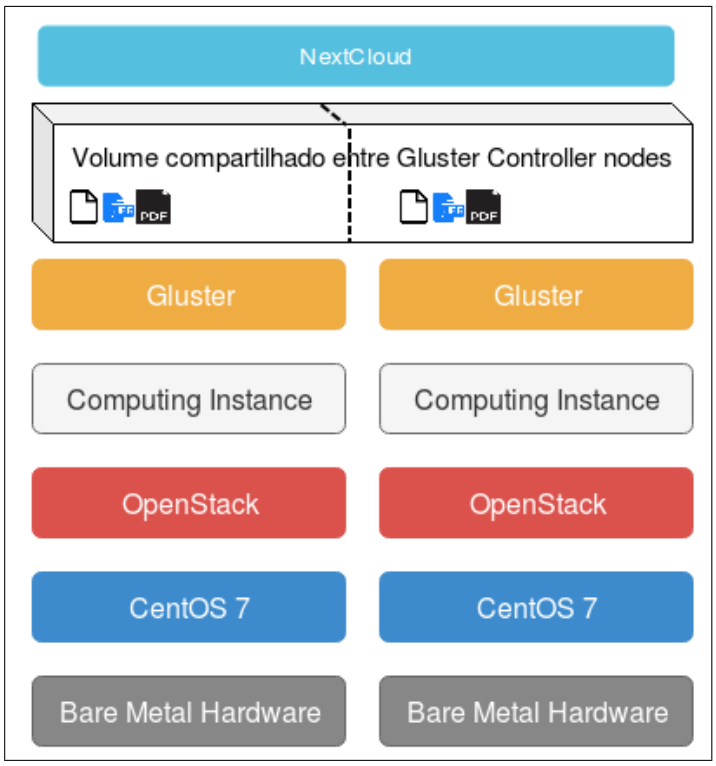


Figura 4.6 – Topologia lógica da solução nível 4 de STaaS redundante com o Gluster. Destaca-se a utilização de uma infraestrutura montada para alta disponibilidade, já explicada na seção 3.3.3. O Nextcloud encontra-se, portanto, rodando em cima dessa infraestrutura, garantindo *Storage as a Service* com alta disponibilidade de dados.

A instalação do Nexcloud está descrita no Anexo D. Além disso, é possível observar

a topologia lógica da solução de nível 4 de STaaS na Figura 4.6. Ao final do passo-a-passo de instalação do Nextcloud, já é possível acessar a página Web disponibilizada pelo serviço (no ip configurado durante a instalação). O primeiro acesso a página web do nextcloud resultará em uma página HTML de configuração inicial, para que o administrador do serviço cadastre o banco de dados a ser usado e o diretório onde o Nextcloud salvará os arquivos, como é possível ver na Figura 4.7.

Esse passo, portanto, é de extrema importância: é desejado implementar um serviço de StaaS seguro e altamente disponível. Para isso, o Nextcloud será apontado para o caminho de diretório onde o volume Gluster foi montado. Dessa forma, todo o conteúdo que o Nextcloud guardar será armazenado em um volume encriptado e replicado, com a segurança de transporte provida pelo WireGuard, entre dois data centers. A Figura 4.7 mostra a tela onde é possível apontar um *Folder Location* (localização do diretório) que será utilizado pelo serviço.

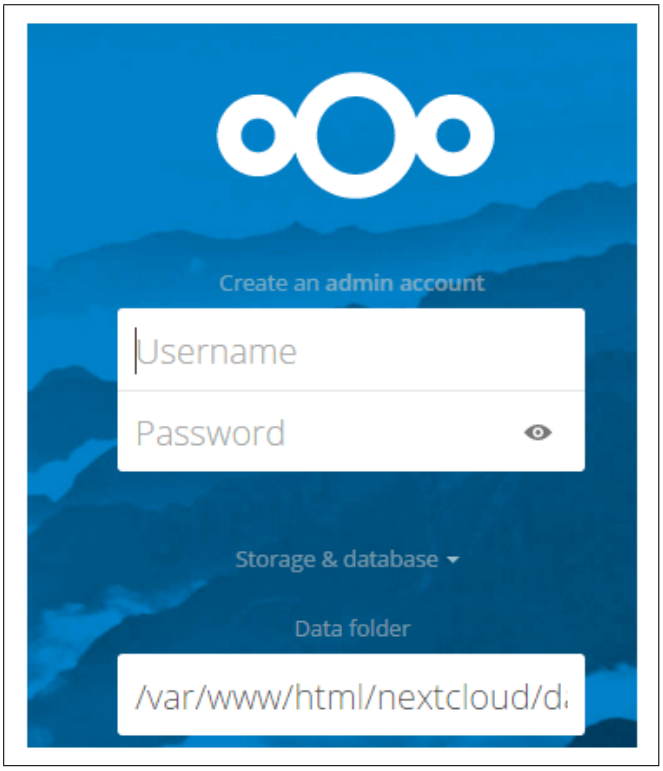


Figura 4.7 – Página de configuração de diretório, disponibilizada no primeiro acesso após instalação do serviço Nextcloud

Após essa configuração inicial, qualquer usuário da rede já pode acessar a página inicial do Nextcloud através do IP flutuante onde o serviço foi instalado. O acesso à pagina web, por parte do usuário, será feito através de um *login* e senha, que pode ser criada na própria página web. As informações de usuário e senha serão salvas no banco de dados criado durante a instalação para o serviço Nextcloud (também é possível fazer autenticação no nextcloud via LDAP e outros serviços de autenticação, desde que previamente

configurados).

Na Figura 4.8 é mostrado o portal web de *upload* de arquivos, na visão do usuário. Como é possível observar, o cliente pode visualizar seus arquivos, deletá-los, verificar sua cota de espaço utilizado, entre outras coisas.

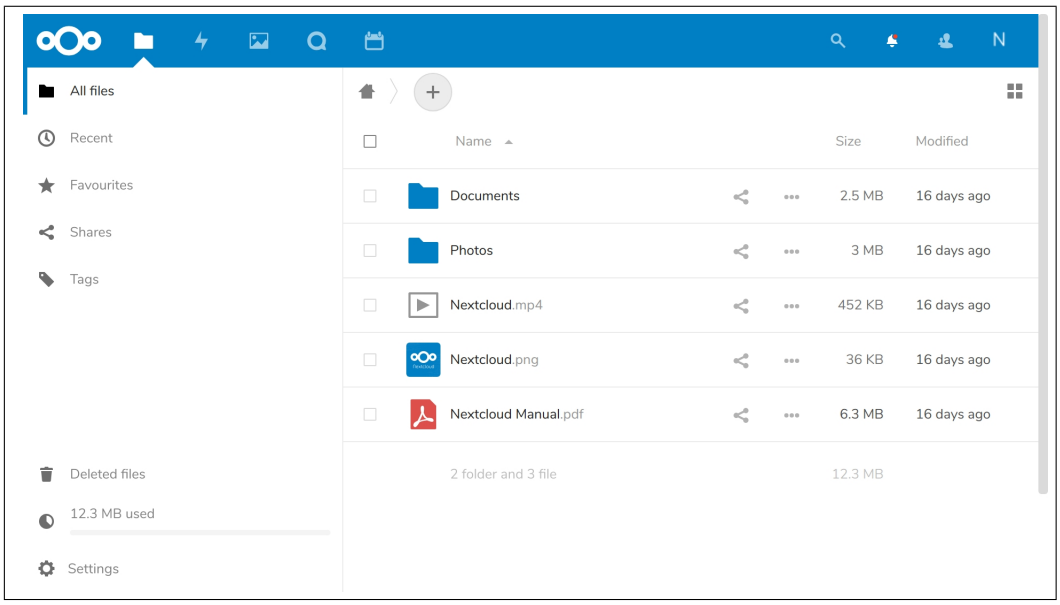


Figura 4.8 – pagina de *login* e *upload* do nextcloud

Com isso, é finalizado a implementação do serviço de nível 4 proposto - Solução de armazenamento como serviço segura e redundante. Ademais, a seção 4.3 traz uma discussão a respeito das etapas de implementação de segurança da solução. Além disso, testes de segurança e performance de leitura e escrita são feitos no capítulo 5.

4.3 Segurança

Os níveis de serviço implementados por esse trabalho tratam, em várias camadas, de soluções de segurança no armazenamento de dados em ambiente virtualizado e distribuído.

Por se tratar de um assunto importante e muito sensível, vide Lei Geral de Proteção de Dados Pessoais (Constituição da República Federativa do Brasil, 2018), decidiu-se realizar uma sumarização da implementação geral de segurança da solução proposta.

Na Figura 4.9, é possível observar as camadas de segurança abordadas nesta seção.

- Segurança no transporte de dados: Como a solução proposta sugere uma replicação de dados entre data centers geograficamente distribuídos, foi preciso um cuidado especial com a confidencialidade de tais dados atravessando redes públicas, como a internet. Para isso, foi criado um túnel VPN, com o *software* WireGuard, entre todas

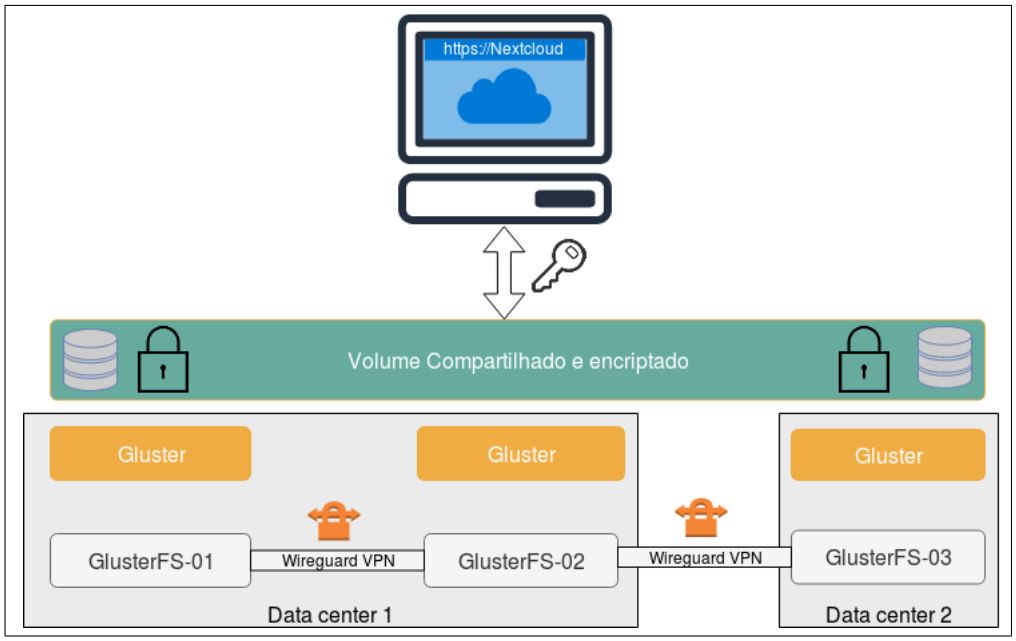


Figura 4.9 – Modelo lógico de segurança proposta para o armazenamento em ambiente virtualizado.

as instâncias GlusterFS que servem o `gluster-volume`. Esse túnel VPN garante que apenas o destinatário possa interpretar os bits de maneira inteligível, resguardando assim o conteúdo da mensagem.

- Segurança no armazenamento: o conceito fundamental de segurança que envolve a confidencialidade de dados, está altamente relacionado na forma como o dado é armazenado. Armazenamento na forma criptografada é a garantia que o dado estará ilegível para um usuário que não tem as permissões necessárias, sendo considerado um possível atacante. Assim a cifragem do dados é garantida pela criptografia que o GlusterFS oferece em relação aos seus dados dos seus volumes, sendo somente possível utilizar o volume criado com a chave atribuída na criptografia.

- Segurança na conexão cliente-servidor: Uma vez garantida a segurança dos dados em transporte (por meio de tunel VPN com Wireguard) e em armazenamento (por meio de volumes cifrados com o GlusterFS), restou apenas a garantia de uma conexão segura entre o usuário e o serviço do Nextcloud. Como o Nextcloud é implementado através de uma interface Web, utilizou-se de uma camada adicional de segurança SSL/TLS em cima do protocolo HTTP. Essa camada permite que os dados sejam transmitidos, por meio de uma conexão criptografada, até o volume compartilhado.

5 Testes e Resultados

Esse capítulo é dedicado à realização de testes de segurança da infraestrutura proposta no trabalho e apresentação dos resultados encontrados.

5.1 Segurança dos dados em transporte

Segurança de dados em transporte significa, como mencionado por (ALLCOCK et al., 2001), garantir que só o destinatário da mensagem possa interpretar os bits de maneira correta. Em particular, há duas preocupações principais para a garantia de segurança na replicação de dados entre localidades geograficamente distribuídas: O garantimento de uma transferência segura, confiável e eficiente; e a habilidade de manter o registro e estado das múltiplas cópias do dado.

5.1.1 Transporte seguro

Para garantir segurança no transporte de dados, é possível fazer uso de um túnel VPN. Como explicado na seção 2.6.3, VPNs permitem a comunicação segura entre dispositivos através de uma rede não confiável, como a internet. Portanto, foi implementado uma VPN, como explicado na seção 4.2.3, com o software Wireguard, entre os *peers*[1] que servem o volume compartilhado.

Para testar a segurança do transporte de dados entre as instâncias de armazenamento do Gluster, foi utilizado o NextCloud, conforme topologia descrita na seção 3.3.4, além do *software* Tcpdump, que é um analisador de pacotes.

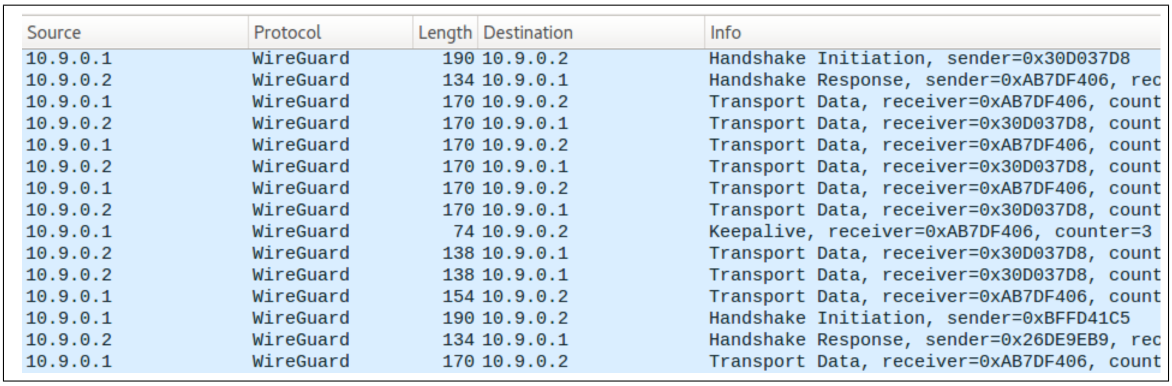
Primeiramente, o Tcpdump foi instalado em uma das instâncias de armazenamento do Gluster que servem o volume compartilhado, onde o Nextcloud escreve seus arquivos. Em seguida, foi iniciado uma captura de pacotes na interface de saída da instância com o comando abaixo. O parâmetro `-n` desabilita a conversão de nome (DNS), o parâmetro `-X` tenta interpretar o campo de dados do pacote em hexadecimal and ASCII, o parâmetro `-i` permite a especificação de uma interface para captura (no caso, a interface eth0) e o parâmetro `-w` direciona a saída do programa para um arquivo (no caso, wireguard_sniff).

```
#tcpdump -n -X -i eth0 -w wireguard_sniff
```

[1] *peers*, para o Gluster, são nós que servem o volume compartilhado.

Com o analisador de pacotes habilitado, foi feito um *upload* de arquivo para o Nextcloud. O serviço do Nextcloud, então, escreve esse arquivo no volume compartilhado. Por configuração, esse arquivo é então replicado entre todas as instâncias. O analisador de pacotes, portanto, poderá fazer a captura desse fluxo.

Finalizando a captura do Tcpdump, é possível observar o conteúdo capturado e analisar os pacotes interceptados. A Figura 5.1 mostra o fluxo de pacotes recebidos durante a escrita dos dados. Já na Figura 5.2, um pacote de *Handshake initiation* é explorado e percebe-se que é negociado uma criptografia estática.



Source	Protocol	Length	Destination	Info
10.9.0.1	WireGuard	190	10.9.0.2	Handshake Initiation, sender=0x30D037D8
10.9.0.2	WireGuard	134	10.9.0.1	Handshake Response, sender=0xAB7DF406, rec
10.9.0.1	WireGuard	170	10.9.0.2	Transport Data, receiver=0xAB7DF406, count
10.9.0.2	WireGuard	170	10.9.0.1	Transport Data, receiver=0x30D037D8, count
10.9.0.1	WireGuard	170	10.9.0.2	Transport Data, receiver=0xAB7DF406, count
10.9.0.2	WireGuard	170	10.9.0.1	Transport Data, receiver=0x30D037D8, count
10.9.0.1	WireGuard	170	10.9.0.2	Transport Data, receiver=0xAB7DF406, count
10.9.0.2	WireGuard	170	10.9.0.1	Transport Data, receiver=0x30D037D8, count
10.9.0.1	WireGuard	74	10.9.0.2	Keepalive, receiver=0xAB7DF406, counter=3
10.9.0.2	WireGuard	138	10.9.0.1	Transport Data, receiver=0x30D037D8, count
10.9.0.2	WireGuard	138	10.9.0.1	Transport Data, receiver=0x30D037D8, count
10.9.0.1	WireGuard	154	10.9.0.2	Transport Data, receiver=0xAB7DF406, count
10.9.0.1	WireGuard	190	10.9.0.2	Handshake Initiation, sender=0xBFFD41C5
10.9.0.2	WireGuard	134	10.9.0.1	Handshake Response, sender=0x26DE9EB9, rec
10.9.0.1	WireGuard	170	10.9.0.2	Transport Data, receiver=0xAB7DF406, count

Figura 5.1 – Fluxo de pacotes recebidos durante a escrita dos dados. Pacotes interceptados em interface com VPN.

5.1.2 Sincronismo

Em relação a habilidade de manter o registro e estado das múltiplas cópias do dado: Como explicado na seção 3.3.3, e na implementação de serviço nível 3, seção 4.2.3, o Gluster é responsável pela criação de volumes compartilhados na rede, por meio de *clusters*, e da manutenção de sincronismo de dados entre os servidores do *cluster*. Além disso, de acordo com a documentação oficial do Gluster (DEVELOPERS, 2006), o serviço Gluster possui um *daemon* rodando de 10 em 10 minutos em plano de fundo que diagnostica erros de conexão e desencadeia processos de reconexão e re-sincronismo automaticamente.

Para testar o serviço de re-sincronismo do Gluster, foi usando a topologia de Nível 3, descrita na seção 4.2.3. Começando com os *peers* sincronizados e volume iniciado em modo réplica (como mostra a Figura 5.3). Uma outra instância na rede (aqui chamada de nó Cliente) monta o volume em seu filesystem e escreve, nesse volume, um arquivo de teste (`teste_sincronismo.txt`) como mostra a Figura 5.4.

Em seguida é verificado nos servidores que o arquivo foi distribuído entre todos os nós. Após a distribuição, um dos *peers* é retirado do *cluster*, (desabilitando a interface de rede desse nó) como mostra a Figura 5.5. Verifica-se, então, que o cliente ainda consegue acessar o arquivo montado. Além disso, o cliente escreve mais um arquivo de teste (`teste_sincronismo_2.txt`) no volume compartilhado. A Figura 5.6 mostra o primeiro

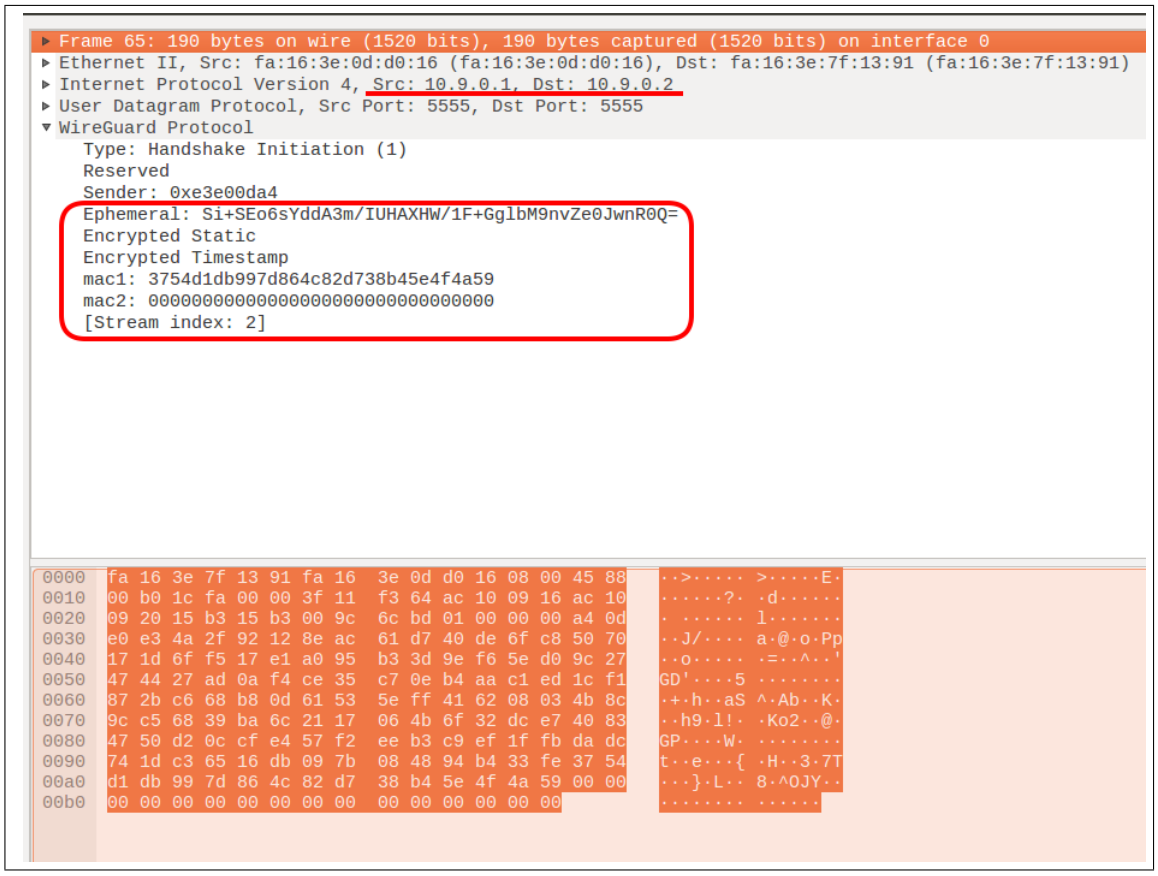


Figura 5.2 – Pacote de Handshake analisado pelo Tcpdump em uma interface onde havia sido configurado um túnel VPN.

```
Volume Name: vol1
Type: Replicate
Volume ID: 053c3176-2e65-4c99-9656-26135728c9e3
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 3 = 3
Transport-type: tcp
Bricks:
Brick1: glusterfs01:/gfsvolume/gv0
Brick2: glusterfs02:/gfsvolume/gv0
Brick3: glusterfs03:/gfsvolume/gv0
```

Figura 5.3 – Evidência de sincronia entre *peers* que servem o Gluster volume

```
total 8
drwxr-xr-x 3 root root 4096 Dec  8 20:28 .
drwxr-xr-x 3 root root 4096 Dec  8 20:00 ..
-rw-r--r-- 1 root root    0 Dec  8 20:28 teste_sincronismo.txt
```

Figura 5.4 – Evidência de escrita do primeiro arquivo de teste pelo cliente Gluster

arquivo de teste ainda disponível para o cliente, assim como o segundo arquivo de teste escrito.

Em seguida, a interface de rede do nó retirado do *cluster* é reativada e observa-se a

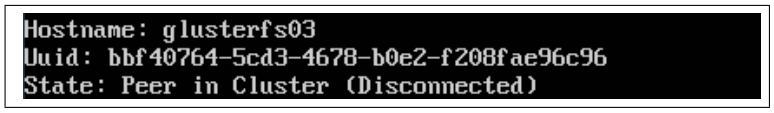


Figura 5.5 – Evidência de desligamento de um dos nós servidores do Gluster Volume.

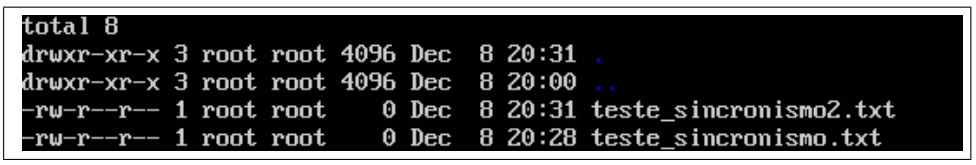


Figura 5.6 – Visão do cliente Gluster. Evidência de que o primeiro arquivo de testes ainda está disponível em seu filesystem. Além disso, segundo arquivo de teste é escrito no volume.

reconexão desse nó com o volume compartilhado (Figura 5.7). Finalmente, listando os arquivos que esse servidor possui, percebe-se que o segundo arquivo de teste foi sincronizado com sucesso, conforme Figura 5.8.

Figura 5.7 – Evidência de reconexão do *peer* desconectado anteriormente.

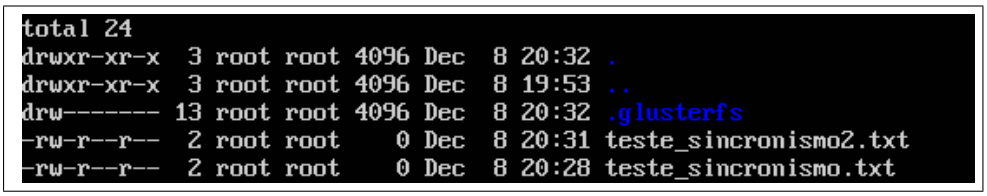


Figura 5.8 – Na visão do servidor do volume Gluster. Evidência de que a reconexão ao cluster disparou processo de re-sincronização de dados.

Com isso, comprova-se o sincronismo entre *peers* em um *cluster*. A habilidade de um cliente de interagir com o volume, mesmo quando um *peer* está offline, e o re-sincronismo desse *peer* após uma falha de conexão, é um aspecto fundamental do armazenamento do Gluster para garantia de alta-disponibilidade oferecida à IaaS.

5.2 Segurança dos dados em repouso

A segurança no repouso é um fator decisivo em opções de fornecedores de nuvens. Usando uma estratégia de segurança por criptografia, é garantido que os dados só possam ser acessados por papéis e serviços autorizados, com acesso auditivo à chave de criptografia. Possibilitando um processo de coleta de dados legíveis e os transformando em uma saída que revela pouca ou nenhuma informação a respeito do conteúdo original.

Criptografia de dados no Volume do GlusterFS

A garantia do repouso de dados de forma segura é realizada pela opção de volume cifrado no GlusterFS. A criptografia utilizada pelo GlusterFS envolve a utilização de uma chave criada pelo *software* OpenSSL, sendo gerada randomicamente composta por valores em hexadecimal. A chave criada é utilizada pelo cliente como única forma de montagem do volume criado pelo GlusterFS.

Como forma de teste da confidencialidade dos dados existentes e em uso do volume criado no GlusterFS, foi realizado um teste de escrita e leitura de dado no volume da topologia de Nível 3, abordada na seção 4.2.3. O teste consistiu em criar um arquivo com um conteúdo legível no nó cliente[2] do GlusterFS, (servidor que foi montado o volume utilizando a chave de cifragem do volume), e realizada a tentativa de leitura do arquivo anteriormente criado em um dos nós[3] que servem de armazenamento para o volume do GlusterFS.

Com toda a topologia do GlusterFS criada e o volume montado no cliente, foi criado um arquivo de teste (`marx.txt`) contendo a seguinte frase: "*se a classe operaria tudo produz a ela tudo pertence*". A Figura 5.9 ilustra a criação do arquivo e seu conteúdo presente.

```
root@glusterfs-client:/mnt/volume_montado# cat marx.txt
se a classe operaria tudo produz a ela tudo pertence
root@glusterfs-client:/mnt/volume_montado#
```

Figura 5.9 – Criado um arquivo e com sua leitura de forma legível pelo cliente do GlusterFS, ou seja, nó que utilizou a chave de cifragem do volume para a montagem do *filesystem* distribuído.

Em seguida, executou-se uma tentativa de leitura do arquivo criado pelo cliente, em um dos nós utilizados como *storage* para o volume do GlusterFS. No servidor do Gluster foi utilizado o comando `cat` que mostra o conteúdo na terminal sem a necessidade de entrar em modo de edição do arquivo. O resultado obtido é um conteúdo não possível de ser compreensível ou legível. A Figura 5.10, evidência a tentativa de leitura do arquivo `marx.txt`.

```
root@glusterfs02:/gfsvolume/gv0# cat marx.txt
♦6♦♦
    ♦g♦O♦qI♦Af0♦♦fs♦9♦xZM♦X6j♦♦4N♦g♦♦{->q♦♦♦♦♦3D
```

Figura 5.10 – Tentativa de leitura do arquivo `marx.txt` no qual possuía um conteúdo legível na visão do cliente e na visão do servidor o conteúdo do arquivo encontra-se irreconhecível.

[2] a implementação possui `glusterfs-client` como *hostname*.
[3] é utilizado o nó com *hostname* igual à `glusterfs02`.

Dessa maneira, evidenciou-se a criptografia dos dados armazenados no volume criado no GlusterFS, sendo não possível a decifração da informação contida nos arquivos presentes no volume.

5.2.1 Performance de escrita e leitura

Como forma de verificar a velocidade de escrita e leitura de dados contendo vários tamanhos, realizou-se uma bateria de testes utilizando a ferramenta IOzone. IOzone é uma ferramenta de *benchmarking* útil para analisar desempenho de sistemas arquivos em diferentes plataformas, como Linux, HP-UX, Solaris (William D. Norcott, 2016). Ele utiliza a entrada e saída de sistema de arquivos como sua geração de carga principal, apresentando aos sistemas sob teste uma grande variedade de solicitações de arquivos de entrada e saída, executando de arquivos de tamanho pequeno à arquivos de tamanho grande, com tamanhos variados de gravação, *reclen*[4]. Permite que o administrador modifique os parâmetros de testes do sistema de forma que mais se ajuste aos testes e resultados desejados.

O uso do IOzone como ferramente de testes é de grande abordagem por artigos e revistas como sendo um teste válido de *filesystem*. Inúmero artigos de grande visibilidade e referência utilizam essa ferramenta para os seus testes, alguns exemplos de artigos são; Wu e Reddy (2011), Lee et al. (2015) e NORCOTT (2003).

Para os testes, foram realizadas escritas e leituras de arquivos de tamanho iguais a 100 Megabytes, 500 Megabytes utilizando o túnel VPN e também sem a utilização do túnel, entre os nós do GlusterFS e também no cliente. Essa comparação foi realizada para ver como o túnel afetaria a experiência do usuário Nextcloud em relação a utilização dos arquivos. A escrita e leitura desses arquivos foi feita em tamanhos de *chunks* de 16384 Megabytes, sendo o valor máximo a ser utilizado como parâmetro de *reclen* no IOzone. Esse valor utilizado é a metade do tamanho máximo de *striped-block-size* do GlusterFS, sendo o tamanho em bytes da unidade que será lida ou gravada no volume do GlusterFS. Foram feitas no total 20 escritas e leituras de cada arquivo de tamanho definido como forma de se obter uma média de valores de velocidade de leitura e escrita.

Tabela 1 – Resultados de testes com o WireGuard. MB/s

	100 Megabytes	500 Megabytes
Leitura	56,345	61,815
Escrita	45,510	46,115
Leitura Randômica	45,007	64,530
Escrita Randômica	47,411	45,469

[4] *reclen* ou comprimento de registro, é o tamanho dos *chunks* nos quais o IOzone divide um arquivo antes de executar uma operação de I/O na (R/W) no disco

Como forma de comprovação de *overhead* de leitura e escrita contendo cabeçalhos do WireGuard foram feitos os mesmo testes sem a utilização do túnel. Utilizando a mesma ferramenta de *benchmarking*, IOzone, 20 arquivos foram escritos e lidos para a obtenção do valores médios. Também foram obtidos os valores de leitura e escrita randomicamente, como forma de acesso em áreas diferentes dos arquivos, simulando a escrita e leitura verdadeiramente de um arquivo, em ambas as tabelas 1 e 2.

Tabela 2 – Resultados de testes sem o WireGuard. MB/s

	100 Megabytes	500 Megabytes
Leitura	85,428	86,313
Escrita	110,677	61,6875
Leitura Randômica	84,647	80,8125
Escrita Randômica	110,268	107,979

Conforme os valores obtidos das duas tabelas apresentadas 1 e 2, que explicitam valores dos resultados, fica clara a diferença de velocidade tanto de escrita, quanto leitura de arquivos quando é utilizado o túnel VPN e quando não é utilizado. Mostrando que a utilização do túnel apesar de garantir a segurança no transporte, a sua utilização acaba gerando atrasos nas operações de escrita e leitura, prejudicando na experiência do usuário ao utilizar o Nextcloud.

6 Conclusão

Nesse projeto foi desenvolvido uma solução de armazenamento de dados com segurança e confiabilidade, em ambiente virtualizado e geograficamente distribuído. A solução conta com várias etapas de implementação, a começar por virtualização: Foi utilizado a plataforma OpenStack para orquestração de hardware. Com a plataforma OpenStack, foi possível abstrair as configurações físicas de hardware e trabalhar com um *cluster* de recursos. Além disso, o OpenStack permite a separação lógica de recursos, fornecendo assim, na prática, uma solução de *Infrastructure as a Service - IaaS*.

Utilizando-se da separação lógica de recursos de infraestrutura, montou-se em dois servidores diferentes (ambos com o OpenStack instalado, afim de simular dois data center geograficamente dispersos e ligados através da internet) um ambiente IaaS para gerenciamento da solução de armazenamento de dados com segurança e confiabilidade.

Em cada um desses ambientes IaaS, foi criado uma instância virtual com recursos computacionais e de armazenamento. Foi então instalado o *Wireguard*. Com esse software, foi possível configurar um túnel VPN entre as duas instâncias. Como todo tráfego entre essas duas instâncias passa pelo túnel VPN, foi garantido a segurança de dados no transporte.

Em seguida, foi instalado em cada uma dessas instâncias o *software* Gluster. Com ele, foi possível criar uma sincronia entre os recursos de armazenamento das duas instâncias. Na prática, o Gluster criou um volume criptografado na rede (similar a um *storage NAS*), que pode ser montado e usado por instâncias dentro do projeto IaaS para armazenar dados de forma segura e com alta disponibilidade, uma vez que todos os dados são replicados, de forma segura, nas duas instâncias que servem o volume.

Por fim, para utilização prática dessa topologia de armazenamento seguro montada, foi instalado o Nextcloud em uma instância do projeto IaaS. O Nextcloud fornece serviço de armazenamento simples de dados na nuvem para usuários, através de uma página web. O Nextcloud foi então configurado para armazenar dados internamente no volume disponibilizado pelo Gluster.

Dessa forma, com o Nextcloud Configurado para armazenar os dados no volume criado pelo Gluster, montou-se uma estrutura, disponível a qualquer usuário da IaaS, de armazenamento na nuvem com segurança, réplica de dados e fácil de usar. Uma Cloud segura e altamente disponível.

6.1 Trabalho Futuros

Para uma proposta mais elaborada de um SDDC, é essencial a implementação de formas de automatização do provisionamento dos recursos pela *cloud*, além da utilização mais completa dos serviços ofertados pelo OpenStack, como a virtualização de redes para construção de novos ambientes e testes.

Entende-se que seja necessário, também, uma bateria maior de testes e estresse da rede, de modo a comprovar a segurança de um ambiente de produção, onde possíveis ataques podem acontecer e um grande fluxo de dados são transmitidos pelas interfaces dos servidores através das requisições de serviços e dados dos usuários da nuvem.

Além disso, com uma infraestrutura como serviço, pode-se começar a pensar em mais aplicações que são facilitadas por essa topologia. Uma aplicação seria, como proposto por Desai, Sheth e Anantharam (2015), um *Gateway as a Service* IoT. Nesse contexto, um SSDDC poderia ser utilizado para implantar simultaneamente infraestruturas com servidores para tratamento de dados IoT, *middleware*, painel de exibição, etc; e *Gateways* de protocolos como MQTT, IP, ZigBee, implantados em dispositivos em redes locais (como residências, universidades, etc) podendo fazer a comunicação com resto da infraestrutura IaaS montada para essa rede IoT.

Referências

ABZETDIN ADAMOV. *CIA triad – Fundamental concept of Information Security*. 2015. <https://aadamov.files.wordpress.com/2015/04/securitymodel.png>. [Acessado online no dia 2 de dezembro de 2019]. Citado 2 vezes nas páginas ix e 16.

AGARWAL, A.; AGARWAL, A. The security risks associated with cloud computing. *International Journal of Computer Applications in Engineering Sciences*, Citeseer, v. 1, p. 257–259, 2011. Citado na página 17.

AGRAWAL, R. et al. Order preserving encryption for numeric data. In: ACM. *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. [S.l.], 2004. p. 563–574. Citado na página 18.

AL-FARES, M.; LOUKISSAS, A.; VAHDAT, A. A scalable, commodity data center network architecture. In: ACM. *ACM SIGCOMM Computer Communication Review*. [S.l.], 2008. v. 38, n. 4, p. 63–74. Citado na página 5.

ALLCOCK, B. et al. Secure, efficient data transport and replica management for high-performance data-intensive computing. In: IEEE. *2001 Eighteenth IEEE Symposium on Mass Storage Systems and Technologies*. [S.l.], 2001. p. 13–13. Citado na página 54.

Ani Miteva. *8 cloud adoption statistics that reveal the future of the cloud*. 2018. <https://www.cloudworldwideservices.com/en/cloud-adoption-statistics-cloud-future/>. [Acessado online no dia 12 de novembro de 2019]. Citado 2 vezes nas páginas ix e 2.

ARORA, R.; PARASHAR, A.; TRANSFORMING, C. C. I. Secure user data in cloud computing using encryption algorithms. *International journal of engineering research and applications*, v. 3, n. 4, p. 1922–1926, 2013. Citado na página 18.

AZEEM, S. A.; SHARMA, S. K. Study of converged infrastructure & hyper converge infrastructre as future of data centre. *International Journal of Advanced Research in Computer Science*, International Journal of Advanced Research in Computer Science, v. 8, n. 5, 2017. Citado na página 13.

BARKIE, E. J. et al. *Authentication in virtual private networks*. [S.l.]: Google Patents, 2015. US Patent 9,094,400. Citado na página 19.

BEIMBORN, D.; MILETZKI, T.; WENZEL, D.-W. I. S. Platform as a service (paas). 2011. Citado na página 9.

BERDE, P. et al. Onos: towards an open, distributed sdn os. In: ACM. *Proceedings of the third workshop on Hot topics in software defined networking*. [S.l.], 2014. p. 1–6. Citado na página 11.

BISHOP, M. *Introduction to Computer Security*. 75 Arlington Street, Suite 300, Boston, MA 02116: Pearson Education, 2005. ISBN 0-321-24744-2. Citado na página 17.

Bradley Mitchell. *The internet wouldn't exist without servers*. 2019. <https://www.lifewire.com/servers-in-computer-networking-817380>. [Acessado online no dia 09 de dezembro de 2019]. Citado na página 4.

Canonical. *Download Ubuntu Server*. 2019. <http://archive.ubuntu.com/ubuntu/dists/bionic/main/installer-amd64/current/images/netboot/mini.iso>. [Acessado online no dia 07 de dezembro de 2019]. Citado na página 76.

Canonical. *What is OpenStack?* 2019. <https://ubuntu.com/openstack/what-is-openstack>. [Acessado online no dia 05 de dezembro de 2019]. Citado 3 vezes nas páginas 23, 25 e 26.

Cisco. *What is a Data Center?* 2014. <https://www.cisco.com/c/en/us/solutions/data-center-virtualization/what-is-a-data-center.html>. [Acessado online no dia 04 de dezembro de 2019]. Citado na página 4.

Committee on National Security Systems. *CNSS Responsibilities*. 2016. <http://www.cnss.gov/CNSS/about/about.cfm>. [Acessado online no dia 3 de dezembro de 2019]. Citado na página 16.

Constituição da República Federativa do Brasil. *LEI Nº 13.709, DE 14 DE AGOSTO DE 2018*. 2018. <http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/L13709.htm>. [Acessado online no dia 08 de dezembro de 2019]. Citado na página 52.

DARABSEH, A. et al. Sddc: A software defined datacenter experimental framework. In: *2015 3rd International Conference on Future Internet of Things and Cloud*. [S.l.: s.n.], 2015. p. 189–194. Citado na página 12.

DESAI, P.; SHETH, A.; ANANTHARAM, P. Semantic gateway as a service architecture for iot interoperability. In: IEEE. *2015 IEEE International Conference on Mobile Services*. [S.l.], 2015. p. 313–319. Citado na página 62.

DEVELOPERS, G. *Gluster File System 3.3. 0 Administration Guide*. 2006. Citado 2 vezes nas páginas 35 e 55.

DONENFELD, J. A. Wireguard: next generation kernel network tunnel (2018). *URL https://www. wireguard. com/papers/wireguard. pdf. version 416d63b*, p. 06–30, 2018. Citado 4 vezes nas páginas x, 32, 33 e 34.

DRAGO, I. et al. Inside dropbox: understanding personal cloud storage services. In: ACM. *Proceedings of the 2012 Internet Measurement Conference*. [S.l.], 2012. p. 481–494. Citado na página 37.

DUAN, Y. et al. Various "aas" of everything as a service. In: IEEE. *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. [S.l.], 2015. p. 1–6. Citado na página 10.

FIELDER, G. L.; ALITO, P. N. *File encryption method and system*. [S.l.]: Google Patents, 2000. US Patent 6,049,612. Citado na página 18.

FISHER-OGDEN, J. Hardware support for efficient virtualization. *University of California, San Diego, Tech. Rep*, v. 12, 2006. Citado na página 6.

FORCE, I. D. D. M. T. Software defined datacenter (sddc) definition. 2015. Citado 2 vezes nas páginas 1 e 12.

FOSTER, I. et al. Cloud computing and grid computing 360-degree compared. *arXiv preprint arXiv:0901.0131*, 2008. Citado 2 vezes nas páginas 7 e 8.

GLADSTONE, P. J. S.; MCGREW, D. A. *Security for remote access VPN*. [S.l.]: Google Patents, 2014. US Patent 8,806,609. Citado na página 19.

Gluster Docs. *Managing GlusterFS Volumes*. 2019. <https://docs.gluster.org/en/latest/Administrator%20Guide/Managing%20Volumes/>. [Acessado online no dia 07 de dezembro de 2019]. Citado na página 82.

Gluster Docs. *Split brain and the ways to deal with it*. 2019. <https://docs.gluster.org/en/latest/Administrator%20Guide/Split%20brain%20and%20ways%20to%20deal%20with%20it/>. [Acessado online no dia 07 de dezembro de 2019]. Citado na página 48.

GRANCE, T. et al. *Security Guide for Interconnecting Information Technology Systems: Recommendations of the National Institute of Standards and Technology*. [S.l.], 2002. Citado na página 4.

GREENBERG, A. et al. Towards a next generation data center architecture: scalability and commoditization. In: ACM. *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*. [S.l.], 2008. p. 57–62. Citado na página 5.

GREENHALGH, A. et al. Flow processing and the rise of commodity network hardware. *ACM SIGCOMM Computer Communication Review*, ACM, v. 39, n. 2, p. 20–26, 2009. Citado na página 8.

HAMMADI, A.; MHAMDI, L. A survey on architectures and energy efficiency in data center networks. *Computer Communications*, Elsevier, v. 40, p. 1–21, 2014. Citado na página 5.

HAN, S. C. S.; GILFIX, D. *OpenStack Storage For Dummies*. 1. ed. [S.l.]: John Wiley & Sons, Inc., 2016. ISBN 978-1-119-29253-1. Citado na página 25.

Houghton Mifflin. *Definition of Security*. 2019. <https://www.dictionary.com/browse/security?s=ts>. [Acessado online no dia 3 de dezembro de 2019]. Citado na página 15.

HP. *What is a Data Center?* 2013. <https://www.hpe.com/br/pt/what-is/software-defined-data-center.html>. [Acessado online no dia 12 de outubro de 2019]. Citado na página 1.

HUMPHREYS, J.; GRIESER, T. Mainstreaming server virtualization: The intel approach. *White paper. Sponsored by Intel*, 2006. Citado 2 vezes nas páginas 7 e 14.

KALLAHALLA, M. et al. Softudc: A software-based data center for utility computing. *Computer*, IEEE, v. 37, n. 11, p. 38–46, 2004. Citado na página 18.

KARPOFF, W.; LAKE, B. *Storage virtualization system and methods*. [S.l.]: Google Patents, 2009. US Patent 7,577,817. Citado na página 14.

KAYLIE GYARMATHY. *How to Define a Data Center*. 2019. <https://www.vxchnge.com/blog/data-center-definition>. [Acessado online no dia 27 de novembro de 2019]. Citado na página 4.

KOCH, S. R. et al. *Local area network bridge*. [S.l.]: Google Patents, 1988. US Patent 4,737,953. Citado na página 40.

KOLAR, M. L. T.; VALENCIA, A. *Cisco Layer Two Forwarding (Protocol) "L2F"*. [S.l.], 1998. 1-29 p. Disponível em: <https://tools.ietf.org/html/rfc2341>. Citado na página 19.

KOROMICHA. *Install and Setup GlusterFS on Ubuntu 18.04*. 2019. <https://kifarunix.com/install-and-setup-glusterfs-on-ubuntu-18-04/>. [Acessado online no dia 07 de dezembro de 2019]. Citado na página 78.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. *arXiv preprint arXiv:1406.0440*, 2014. Citado 2 vezes nas páginas 11 e 13.

KRISHNA, V.; REDDY, L. Security architecture of cloud computing. *International Journal of Engineering Science and Technology (IJEST)*, v. 3, n. 9, p. 7149–7155, 2011. Citado na página 18.

LAUTER, K.; MITYAGIN, A. Security analysis of kea authenticated key exchange protocol. In: SPRINGER. *International Workshop on Public Key Cryptography*. [S.l.], 2006. p. 378–394. Citado 2 vezes nas páginas 33 e 34.

LEE, C. et al. F2fs: A new file system for flash storage. In: *13th {USENIX} Conference on File and Storage Technologies ({FAST} 15)*. [S.l.: s.n.], 2015. p. 273–286. Citado na página 59.

LEVENTHAL, A. Flash storage memory. *Communications of the ACM*, ACM, v. 51, n. 7, p. 47–51, 2008. Citado na página 5.

Linux Foundation Collaborative Projects. *Production Quality, Multilayer Open Virtual Switch*. 2016. <https://www.openvswitch.org/>. [Acessado online no dia 07 de dezembro de 2019]. Citado na página 40.

MACVITTIE, L. *Stop Conflating Software-Defined with Software-Deployed*. 2014. <https://devcentral.f5.com/s/articles/stop-conflating-software-defined-with-software-deployed>. [Acessado online no dia 17 de outubro de 2019]. Citado na página 12.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National ..., 2011. Citado na página 8.

MORENO-VOZMEDIANO, R.; MONTERO, R. S.; LLORENTE, I. M. Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures. *Computer*, IEEE, v. 45, n. 12, p. 65–72, 2012. Citado na página 9.

National Institute of Standards and Technology. *NIST Cloud Computing Program - NCCP*. 2010. <https://www.nist.gov/programs-projects/nist-cloud-computing-program-nccp>. [Acessado online no dia 05 de dezembro de 2019]. Citado na página 24.

NEIVA, A. S. Estudo comparativo de desempenho em ambiente tradicional e virtualizado aplicado a banco de dados em platafrma x86. 2010. Citado na página 6.

NORCOTT, W. Iozone filesystem benchmark. *http://www. iozone. org/*, 2003. Citado na página 59.

OKUHARA, M.; SHIOZAKI, T.; SUZUKI, T. Security architecture for cloud computing. *Fujitsu Sci. Tech. J*, v. 46, n. 4, p. 397–402, 2010. Citado na página 17.

OpenStack Documentation. *OpenStack's Documentation - Flavors*. 2019. <https://docs.openstack.org/nova/latest/user/flavors.html>. [Acessado online no dia 2 de novembro de 2019]. Citado na página 29.

OpenStack Documentation's image base. *OpenStack's Documentation*. Acessado online no dia 26 de outubro de 2019. <https://docs.openstack.org/security-guide/_images/marketecture-diagram.png>. [Acessado online no dia 26 de outubro de 2019]. Citado 2 vezes nas páginas ix e 24.

OpenStack's Documentation. *OpenStack's Documentation - Manage IP addresses*. 2019. <https://docs.openstack.org/ocata/user-guide/cli-manage-ip-addresses.html>. [Acessado online no dia 07 de dezembro de 2019]. Citado na página 42.

OpenStack's Documentation. *OpenStack's Documentation - Overview*. 2019. <https://docs.openstack.org/install-guide/overview.html>. [Acessado online no dia 06 de dezembro de 2019]. Citado 3 vezes nas páginas ix, 26 e 27.

OpenStack's Documentation. *OpenStack's Documentation - Preface*. 2019. <https://docs.openstack.org/install-guide/preface.html>. [Acessado online no dia 13 de outubro de 2019]. Citado na página 22.

OpenStack's Neutron Documentation. *OpenStack's Documentation - Neutron*. 2019. <https://docs.openstack.org/neutron/latest/install/concepts.html>. [Acessado online no dia 13 de outubro de 2019]. Citado na página 25.

Patrick Moorhead. *Data Center Storage Matters More Than Ever in 2018*. 2018. <https://www.forbes.com/sites/patrickmoorhead/2018/01/28/datacenter-storage-matters-more-than-ever-in-2018/#58e5c5a468e7>. [Acessado online no dia 09 de dezembro de 2019]. Citado na página 5.

PEPPLE, K. *Deploying openstack*. [S.l.]: "O'Reilly Media, Inc.", 2011. Citado na página 25.

POPEK, G. J.; GOLDBERG, R. P. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, ACM, v. 17, n. 7, p. 412–421, 1974. Citado na página 7.

Pradeep Kumar. *How to Setup GlusterFS Storage on CentOS 7 / RHEL 7*. 2016. <https://www.linuxtechi.com/setup-glusterfs-storage-on-centos-7-rhel-7/>. [Acessado online no dia 07 de dezembro de 2019]. Citado na página 78.

RAMGOVIND, S.; ELOFF, M. M.; SMITH, E. The management of security in cloud computing. In: IEEE. *2010 Information Security for South Africa*. [S.l.], 2010. p. 1–7. Citado na página 1.

RDO Project. *Packstack: Create a proof of concept cloud*. 2019. <https://www.rdoproject.org/install/packstack/>. [Acessado online no dia 30 de novembro de 2019]. Citado 2 vezes nas páginas 41 e 71.

Richard Fichera. *he Software-Defi ned Data Center Is The Future Of Infrastructure Architecture*. 2012. <https://www.vmware.com/files/include/microsite/sddc/the_software-defined_datacenter.pdf>. [Acessado online no dia 07 de dezembro de 2019]. Citado 2 vezes nas páginas 1 e 14.

ROBISON, S. et al. The next wave: Everything as a service. *Executive Viewpoint: hp.com*, 2008. Citado na página 10.

SDxCentral Staff. *Data Center Networking Explained*. 2016. <https://www.sdxcentral.com/data-center/definitions/data-center-networking-explained/>. [Acessado online no dia 04 de dezembro de 2019]. Citado na página 5.

SEFRAOUI, O.; AISSAOUI, M.; ELEULDJ, M. Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, Published by Foundation of Computer Science, v. 55, n. 3, p. 38–42, 2012. Citado na página 24.

SINGH, A.; KORUPOLU, M.; MOHAPATRA, D. Server-storage virtualization: integration and load balancing in data centers. In: IEEE PRESS. *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. [S.l.], 2008. p. 53. Citado 2 vezes nas páginas 1 e 6.

Software-Defined Data center – In Depth. *Software-Defined Data center – In Depth*. Acessado online no dia 13 de outubro de 2019. <www.vmware.com/software-defined-datacenter>. [Acessado online no dia 13 de outubro de 2019]. Citado 2 vezes nas páginas ix e 14.

SOMANI, U.; LAKHANI, K.; MUNDRA, M. Implementing digital signature with rsa encryption algorithm to enhance the data security of cloud in cloud computing. In: IEEE. *2010 First International Conference On Parallel, Distributed and Grid Computing (PDGC 2010)*. [S.l.], 2010. p. 211–216. Citado na página 18.

SOUSA, F. R.; MOREIRA, L. O.; MACHADO, J. C. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. *II Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI)*, p. 150–175, 2009. Citado na página 8.

STALLINGS, W. *Cryptography and Network Security Principles and Practices*. 4. ed. [S.l.]: Prentice Hall, 2005. ISBN 0-13-187319-9. Citado na página 17.

Trevor Perrin. *Noise Protocol Framework*. 2018. <http://www.noiseprotocol.org/>. [Acessado online no dia 06 de dezembro de 2019]. Citado 2 vezes nas páginas 32 e 33.

VOLK, T. *The essential guide to VMware NSX SDN technology*: Sddc (software-defined data center). enterprisemanagement, 2012. Acessado em 05/12/2019. Disponível em: <http://blogs.enterprisemanagement.com/torstenvolk/2012/08/16/softwaredefined-datacenter-part-1-4-basics/>. Citado 2 vezes nas páginas 12 e 13.

WEN, X. et al. Comparison of open-source cloud management platforms: Openstack and opennebula. In: IEEE. *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*. [S.l.], 2012. p. 2457–2461. Citado na página 15.

WHITMAN, H. J. M. M. E. *Principles of Information Security*. 4. ed. [S.l.]: Course Technology, 2012. ISBN 1-111-13821-4. Citado na página 15.

William D. Norcott. *IOzone Filesystem Benchmark*. 2016. <http://www.iozone.org/docs/IOzone_msword_98.pdf>. [Acessado online no dia 11 de dezembro de 2019]. Citado na página 59.

WU, J. et al. Cloud storage as the infrastructure of cloud computing. In: IEEE. *2010 International Conference on Intelligent Computing and Cognitive Informatics*. [S.l.], 2010. p. 380–383. Citado na página 10.

WU, X.; REDDY, A. Scmfs: a file system for storage class memory. In: ACM. *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. [S.l.], 2011. p. 39. Citado na página 59.

Anexos

ANEXO A – Instalação OpenStack

A instalação do OpenStack pode ser realizada de diversas maneiras, através do tripleO, devstack, packstack, entre outros métodos. O método utilizado segue a utilização do PackStack, segundo o tutorial de instalação do (RDO Project, 2019). Como pré-requisitos de instalação, há a necessidade de popular o arquivo `/etc/environment` com a configuração de fuso horário para localizações não-inglês.

```
LANG=en_US.utf-8
LC_ALL=en_US.utf-8
```

Os pré-requisitos necessários antes da instalação contemplam requisitos de *software, hardware* e *network*. Para o *software*, a versão *Red Hat Enterprise Linux* (RHEL) 7 é a mínima recomendada, ou uma versão equivalente da distribuição Linux baseada em RHEL como CentOS, Linux Científico, e somente a arquitetura x86_64 é a suportada. Já em relação ao *hardware* é necessário uma máquina com pelo menos 16GB RAM, processador com extensão de virtualização de hardware, e pelo menos um adaptador de rede. Finalizando os pré-requisitos, as configurações de *network* devem ser realizadas devidamente, assim, um IP estático deve ser aplicado a interface de rede, a desabilitação do *firewalld* e do *Network Manager*, responsável por gerenciar as configurações de rede facilmente e automaticante.

```
#sudo systemctl disable firewalld
#sudo systemctl stop firewalld
#sudo systemctl disable NetworkManager
#sudo systemctl stop NetworkManager
#sudo systemctl enable network
#sudo systemctl start network
#echo SELINUX=disable > /etc/selinux/config
#echo SELINUXTYPE=targeted >> /etc/selinux/config
```

No CentOS, o repositório `Extras` provê o RPM (*Red Hat Package Manager*) é um arquivo que habilita rapidamente e facilmente a instalação de *softwares* no Red Hat/CentOS Linux. O RPM habilita o repositório do OpenStack. O repositório `Extras` é habilitado por padrão no CentOS7, portanto, é simples instalar o RPM para configurar o repositório do OpenStack. A versão do OpenStack utilizada na instalação é a `stein`.

```
#sudo yum install -y centos-release-openstack-stein
```

Inicialmente o repositório do OpenStack utilizando a versão `stein` encontra-se desabilitado, para habilitá-lo é necessário instalar o pacote *yum-utils*. O gerenciador de configurações do `yum` é o encarregado de habilitar o repositório do OpenStack com a versão `stein`.

```
#sudo yum install yum-utils
#sudo yum-config-manager --enable openstack-stein
```

Após o procedimento é recomendável atualizar os repositórios para começar a instalação do instalador do Packstack.

```
#sudo yum update -y
#sudo yum install -y openstack-packstack
```

O método seguinte é utilizado para que qualquer máquina que tenha acesso a rede possa acessar as instâncias criadas no OpenStack, através do IP flutuante das mesmas. É um método usado para o packstack com uma rede externa existente.

```
#packstack --allinone --provision-demo=n \
-os-neutron-ovs-bridge-mappgins=extnet:br-ex \
 --os-neutron-ovs-bridge-interfaces=br-ex:em1 \
--os-neutron-ml2-type-drivers=vxlan,flat
```

O comando executado leva em consideração a criação de uma ponte (bridge) entre a interface virtual que será criada pelo packstack (br-ex) e a interface física existente no servidor, para isso é utilizado do recurso de ovs (Open Virtual Switch), mapeando as duas interfaces. O termo –provision-demo=n é utilizado para que não seja criado nenhuma topologia demonstrativa inicialmente no projeto. Esse comando define um nome lógico para o segmente de camada 2 e o físico, externo como "extnet".

Após a conclusão do processo efetuado anteriormente, é necessário alterar as interfaces de rede que serão usadas. Para a interface virtual "br-ex"criada, o seu arquivo de configuração ficará desse formato:

```
#sudo vi /etc/sysconfig/network-scripts/ifcfg-br-ex
```

```
DEVICE=br-ex
DEVICETYPE=ovs
TYPE=OVSBridge
BOOTPROTO=static
IPADDR=172.16.9.252 # IP from a network interface adaptor
NETMASK=255.255.255.0  # your netmask
GATEWAY=172.16.9.1  # your gateway
DNS1=8.8.8.8     # your nameserver
ONBOOT=yes
```

E editando o arquivo de configuração da interface de rede física:

```
#sudo vi /etc/sysconfig/network-scripts/ifcfg-em1
```

```
DEVICE=em1
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
ONBOOT=yes
```

Realiazada as configurações das interfaces, é necessário reiniciar o serviço de rede do servidor, para isso utiliza-se:

```
#service network restart
```

Com o OpenStack instalado e acessível por interface *web*, através do IP utilizado de forma estática nas configurações de rede, serão realizadas as criações de redes, dispositivos de rede, grupos de segurança que são utilizados para a criações de instâncias, imagens que serão atribuídas as instâncias que ditarão qual OS (*Operational System*) a instância terá, e chaves de segurança que ditarão o acesso via rede às instâncias, através da rede externa.

O próximo passo envolve a criação de uma rede no OpenStack, essa rede chamada de *external_network*, simulará a rede externa existente, na qual o servidor está diretamente conectado. O serviço do OpenStack responsável pela criação dessa e de todas as outras redes e dispositivos de rede é o Neutron. A criação da *external_network*, atráves do administrador do OpenStack, possibilita todos os projetos criados no OpenStack se conectarem nessa rede e terem suas instâncias com conectividade com o mundo externo.

```
#. keystonerc_admin
#neutron net-create external_network --provider:network_type flat \
--provider:physical_network extnet --router:external
```

O comando anterior define que a criação da rede dentro do OpenStack, denominada como *external_network*, utilizará o segmento de camada 2 definido anteriormente em `-os-neutron-ovs-bridge-mappings`, no comando `packstack -allinone`. A rede que foi criada é do tipo *flat*, definida como uma rede plana, de abordagem de *design* de rede de computadores que visa reduzir custos, manutenção e administração, pois, os dispositivos são conectadas à único comutador.

Com a criação da rede *external_network*, é necessário criar a sua sub-rede, ela que definirá qual o CIDR (*Classless Inter-Domain Routing*) a rede utilizará, se será necessário utilizar o servidor de DHCP e qual o alcance de IPs da rede externa que será provido para as instâncias como IPs flutuantes, ou seja, a criação dessa sub-rede define virtualmente a rede física existente, delimitando uma faixa de IPs da rede física para serem utilizados nas instâncias, e essa sub-rede será a rede de acesso a *Internet*, na qual as instâncias utilizaram para a comunicação com o mundo.

```
#neutron subnet-create --name public_subnet --enable_dhcp=False \
--allocation-pool=start=172.16.9.20,end=172.16.9.30 \
--gateway=172.16.9.1 external_network 172.16.9.0/24
```

Essas configurações realizadas, na perspectiva do administrador do OpenStack é o necessário para que os usuários em outros projetos possam conectar suas redes privadas no mundo externo. Partindo da perspectiva que nenhum usuário do OpenStack foi criado, é necessário criar um usuário com privilégios de administrador de um projeto.

```
#openstack project create --enable <project_name>
#openstack user create --project <user_name> \
```

```
--pasword <password> --email  <email> --enable <project_name>
#openstack role add --user <user_name> --project <project_name> admin
```

Para realizar configurações no projeto criado anteriormente é necessário alterar o arquivo `keystonerc_admin` gerado pelo comando "packstack –allinone"e executar o arquivo. São utilizados os seguintes campos e comandos para isso:

```
#export OS_USERNAME=<user_name>
#export OS_TENANT_NAME=<project_name>
#export OS_PASSWORD=<password>
```

Executando o arquivo `keystonerc_admin` com as novas configurações de usuário e projeto.

```
#. keystonerc_admin
```

Dentro do projeto, utilizando o usuário atribuído foram criados uma rede interna, um roteador virtual. Roteador responsável pelo roteamento da rede interna com a rede externa. Criação do roteador e atrelando o mesmo a rede externa previamente criada:

```
#neutron router-create router1
#neutron router-gateway-set router1 external_network
```

Após a criação do roteador, foi criado a rede interna, nessa rede que as instâncias utilizadas pelo usuário ficaram conectadas. E em seguida conectado a rede interna ao roteador.

```
#neutron net-create private_network
#neutron subnet-create --name private_subnet \
private_network 192.168.123.0/24
# neutron router-interface-add router1 private_subnet
```

Finalizado essa parte de configurações executadas através da linha de comando do OpenStack foi utilizado o *Dashboard* do OpenStack, Horizon, para as restantes das configurações para a criação das instâncias que serão utilizadas na topologia do projeto. No Horizon primeiramente, logado como usuário internal, no projeto internal, foi inserido uma imagem do Ubuntu 18.04 de formato ISO, imagem que pode ser obtida em Canonical (2019a). Realizado o passo anteriormente da criação de uma imagem para as instâncias, é necessário criar um grupo de segurança no qual tem a utilidade de um *firewall* virtual para servidores (instâncias) e outros recursos na rede, especificando as regras de acesso a rede. O grupo de segurança (*security group*) cobre as seguintes regras:

Tabela 3 – Regras de *firewall* utilizadas no grupo de segurança do OpenStack

Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix
Egress	IPv4	Any	Any	0.0.0.0/0
Egress	IPv6	Any	Any	::/0
Ingress	IPv4	ICMP	Any	0.0.0.0/0
Ingress	IPv4	TCP	1 - 65535	0.0.0.0/0
Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0
Ingress	IPv4	TCP	53 (DNS)	0.0.0.0/0
Ingress	IPv4	TCP	80 (HTTP)	0.0.0.0/0
Ingress	IPv4	UDP	1 - 65535	0.0.0.0/0

E o último passo antes da criação de uma instância é um par de chaves *keypair*, no qual é uma chave pública do par de chaves do OpenSSH, usado para acesso aos servidores (instâncias) criados. Utiliza criptografia RSA para a sua criação.

Criação de uma instância

Nomear a instância, definir se será utilizado uma imagem, um *snapshot* de uma instância já presente no projeto, um volume no qual esse volume pode ser do tipo ISCII ou um volume já com uma imagem de um sistema operacional contido nele e um *snapshot* de um volume também já existente no projeto. Selecionar um flavor que definirá quanto de RAM será aloca, número de VCPUs e o tamanho do armazenamento mínimo da instância, a rede na qual a instância pertencerá, um grupo de segurança criado anteriormente e o par de chaves criado anteriormente também. Para o projeto foi criado duas instâncias, uma em cada servidor com uma imagem do Ubuntu 18.04 *server*, um *flavor* `m1.small` que define 2Gb de RAM, 1 VCPU e o tamanho do disco de 20Gb, e selecionado que essa instância será conectada a rede interna. Na instalação do Ubuntu é necessário escolher quais as configurações de teclado que serão aplicadas, o hostname na instância que servirá para sua visualização pelos outros *hosts* na rede, criar um usuário e uma senha para esse usuário, definir um *mirror* no qual será feitas as buscas por pacotes de instalação do sistema, escolher o local de instalação no disco, definindo uma partição para isso, definindo aplicações a serem instaladas juntamente com o sistema operacional como OpenSSH, e se atualizações de segurança devem ser aplicadas ou não.

Criação de um volume adicional à uma instância

Para a adição de volumes adicionais as instâncias dos projetos, de forma a escalar os recursos de armazenamento que são providos as instâncias é necessário utilizar o CLI do OpenStack, com o usuário e projeto pertencentes à instância específica na qual deseja-se adicionar mais espaço de armazenamento. De forma a diferenciar a partição do sistem operacional, é criado uma partição nova. Os passos para a criação desse volume podem ser visualizados abaixo.

```
#. keystonerc_admin
#[.keystonerc_admin]openstack volume list
#[.keystonerc_admin]openstack volume create \
--size 10 "my-new-volume(1,2)"
#[.keystonerc_admin]openstack server add volume "nome-da-instância" \
"my-new-volume(1,2)" --device /dev/vdb
```

Utilizando no lugar `"nome-da-instância"` o nome da instância daquele projeto que deseja-se acrescentar um volume. O tamanho do volume adicionado irá depender dos recursos do *hardware* ainda disponíveis para serem utilizados.

ANEXO B – Instalação Gluster

Para a instalação do GlusterFS foi utilizado essas duas referências Pradeep Kumar (2016) e KOROMICHA (2019). Primeiramente é necessário alterar o arquivo /etc/hosts com o IP dos *hosts* utilizadas para a instalação do GlusterFS e o *hostname* delas utilizando o comando com o "sudo"para a edição do arquivo.

```
172.16.9.22 glusterfs-client
172.16.9.23 glusterfs01
172.16.9.36 glusterfs02
172.16.9.32 glusterfs03
```

Os próximos passos será executado somente nas três máquinas que servirão de servidor para o GlusterFS.

Adição do repositório do GlusterFS

Instalação do pacote (software-properties-common). O Glusterfs-3 é o disponível por padrão nos repositórios do Ubuntu 18.04, porém será utilizado a versão 5 do GlusterFS, mais recente versão, necessitando adicionar o repositório "ppa".

```
#sudo apt update
#sudo apt upgrade
#sudo apt install software-properties-common -y
#sudo add-apt-repository ppa:gluster/glusterfs-5
#sudo apt update
```

Instalando o GlusterFS nós três servidores

```
#sudo apt install glusterfs-server
```

O "daemon"do GlusterFS não é configurado por padrão em sua instalação, assim é necessário habilitá-lo com o "boot"do sistema.

```
#sudo systemctl enable glusterd
#sudo systemctl start glusterd
```

Para o correto funcionamento do GlusterFS é necessário configurar a comunicação entre os servidores, ou seja, as regras de *firewall* dos servidores devem ser aplicadas para a formação do *cluster* de armazenamento. As regras utilizadas referem-se as portas utilizadas pelo GlusterFS.

```
#sudo firewall-cmd --zone=public --add-port=24007-24008/tcp --permanent
#sudo firewall-cmd --zone=public --add-port=24009/tcp --permanent
#sudo firewall-cmd --zone=public --add-service=nfs \
--add-service=samba --add-service=samba-client --permanent
#sudo firewall-cmd --zone=public --add-port=111/tcp \
--add-port=139/tcp --add-port=445/tcp --add-port=965/tcp \
--add-port=2049/tcp --add-port=38465-38469/tcp \
--add-port=631/tcp --add-port=111/udp \
--add-port=963/udp --add-port=49152-49251/tcp --permanent
#sudo firewall-cmd --reload
```

Com as regras de *firewall* configuradas é possível criar os *peer* entre os servidores. O comando utilizado do GlusterFS é mostrado abaixo.

```
#sudo gluster peer probe glusterfs01
#sudo gluster peer probe glusterfs02
#sudo gluster peer probe glusterfs03
```

A partir da criação dos *peers* com os servidores, é realizado a criação do volume. O GlusterFS disponibiliza três tipos essências de volumes que podem ser criados, podendo ser agrupados criando funcionalidades diferentes. Os tipos de volumes base são:

- *Distributed*: distribuem os arquivos entre as *bricks* do volume. É utilizável para escalar o armazenamento, sendo redundância um ponto não importante para esse tipo de volume.

- *Replicated*: replicam arquivos entre *bricks* no volume. Utilizado em ambientes onde alta disponibilidade e alta confiabilidade são críticas.
- *Dispersed*: os volumes dispersos são baseados em códigos de apagamento, fornecendo proteção com espaço eficiente contra falhas de disco ou servidor. Ele armazena um fragmento codificado do arquivo original em cada bloco, de forma que apenas um subconjunto dos fragmentos seja necessário para recuperar o arquivo original. O número de *bricks* que podem estar faltando sem perder o acesso aos dados é configurado pelo administrador no momento da criação do volume.

As outras opções de volume é unindo de duas em duas opções base mencionadas acima. Para a criação do volume é necessário criar em cada servidor um diretório onde será utilizado a capacidade de armazenamento do disco utilizado e onde será salvo os arquivos que o cliente do Gluster "montará"e utilizará para guardar arquivos. O caminho e nome do diretório em todos os servidores deve ser o mesmo por padrão de instalação e evitar erros de criação.

```
#sudo gluster volume create <volume_name> \
glusterfs01:/<path-directory>/ \
glusterfs02:/<path-directory>/ \
glusterfs03:/<path-directory>/
```

Lembrando que todos esse passos anteriores a criação do comando de criação do volume no GlusterFS é necessário ser feito em ambos os servidores. Para a "montagem"automática do volume criado é necessário editar o arquivo `/etc/fstab` de todos servidores. Um exemplo de como o arquivo pode ser editado é abordado abaixo.

```
#sudo vi /etc/fstab
/dev/vdb    /<path-directory>/  xfs   defaults  0   0
```

Após a criação do volume é necessário ativá-lo para que o servidor cliente possa "montalo"e utilizá-lo como um *filesystem* distribuído por rede. A ativação do volume é vista abaixo.

```
#sudo gluster volume start <volume_name>
```

Já com o volume iniciado é possível utilizá-lo pelo cliente, na visão do cliente do GlusterFS é necessário instalar o glusterfs-client, adicionando o repositório do mesmo.

```
#sudo apt-get install software-properties-common
#sudo add-apt-repository ppa:gluster/glusterfs-5
#sudo apt install glusterfs-client
```

As regras de *firewall* utilizados pelos servidores de armazenamento também são válidas e devem ser usadas para que seja possível "montar"o volume criado pelos servidores, possibilitando assim, a comunicação entre os servidores através das portas e protocolo utilizado pelo GlusterFS.

```
#sudo firewall-cmd --zone=public --add-port=24007-24008/tcp --permanent
#sudo firewall-cmd --zone=public --add-port=24009/tcp --permanent
#sudo firewall-cmd --zone=public --add-service=nfs \
--add-service=samba --add-service=samba-client --permanent
#sudo firewall-cmd --zone=public --add-port=111/tcp \
--add-port=139/tcp --add-port=445/tcp --add-port=965/tcp \
--add-port=2049/tcp --add-port=38465-38469/tcp \
--add-port=631/tcp --add-port=111/udp \
--add-port=963/udp --add-port=49152-49251/tcp --permanent
#sudo firewall-cmd --reload
```

Com os pré-requisitos executados é necessário criar um diretório no qual será "montado"o volume compartilhado. Logo depois da "montagem"do volume também é necessário que seja acrescentado uma configuração de "montagem"automática do volume pelo *reboot* do sistema operacional, sem que aja necessidade de sempre ter que "montar"o volume repassando o comando de "montagem".

```
#sudo mkdir /<path-directory-client>/
#sudo mount -t glusterfs glusterfs01:/<volume_name> \
/<path-directory-client>/
```

E editando o arquivo `/etc/fstab`:

```
glusterfs01:/<volume_name>/<path-directory-client>/  glusterfs \
defaults,_netdev  0  0
```

Com todos os procedimentos realizados de forma correta o volume distribuído por rede é capaz de ser utilizado pelo cliente do GlusterFS.

Criação de um volume criptografado no GlusterFS

O GlusterFS possibilita a configuração de parâmetros que influenciarão no comportamento do volume. É possível alterar o protocolo utilizado para a criação dos volumes, estabelecer quotas que definem quais IPs que podem "montar"o volume, definir tamanhos de blocos de armazenamento, entre outros parâmetros, conforme Gluster Docs (2019a).

Um parâmetro importante a ser utilizado é o `encryption`, ele define se o armazenamento dos dados no volume será de forma encriptada ou não. A utilização desse parâmetro requer a criação de uma chave mestre, essa chave é útil para a criptografia do volume e para a descriptografia do volume na visão do cliente. Parâmetros que definem o tamanho da chave utilizada, o tamanho do bloco de armazenamento também devem ser utilizados como requisito da criptografia do volume.

Um fato importante a ser considerado para o funcionamento da criptografia é que o volume já tem de estar iniciado pelo administrador do GlusterFS e "montado"no cliente. Assim os parâmetros podem ser inseridos no volume sem o problema de montagem no lado do cliente. A criação da chave envolve a utilização do programa OpenSSL. A chave criada é gerada de forma randômica e de formato hexadecimal.

```
#openssl rand -hex 32 > mk.key
```

Habilitando a criptografia ao volume:

```
# gluster volume set <vol_name> encryption on
```

Como a criptografia é feita pelo serviço do xlator, módulo pertencente ao GlusteFS, é necessário desabilitar a performance de escrita, leitura e abertura dos dados, pois é incompatível com o xlator.

```
# gluster volume set <vol_name> performance.quick-read off
# gluster volume set <vol_name> performance.write-behind off
# gluster volume set <vol_name> performance.open-behind off
```

Configurar a localização da chave para o volume, definir parâmetros como tamanho da chave utilizada e tamanho do bloco de encriptação. Os padrões *defaults* do tamanho da chave e do tamanho do bloco são 256 bits e 4096 bytes.

```
#gluster volume set <vol_name> encryption.master-key \
<master_key_location>
#gluster volume set <vol_name> encryption.data-key-size \
<data_key_size>
#gluster volume set <vol_name> encryption.block-size \
<block_size>
```

Com as etapas de configurações concluídas é necessário que a chave utilizada fica em mãos do cliente para poder utilizar o volume. O caminho no qual a chave deve ser armazenada no cliente consiste no mesmo caminho do lado do servidor. Assim o volume estará disponível para uso e de forma segura.

```
Volume Name: vol1
Type: Replicate
Volume ID: 11b357e4-7301-47df-8734-43f25bcb0203
Status: Created
Snapshot Count: 0
Number of Bricks: 1 x 3 = 3
Transport-type: tcp
Bricks:
Brick1: glusterfs01:/gfsvolume/gv0
Brick2: glusterfs02:/gfsvolume/gv0
Brick3: glusterfs03:/gfsvolume/gv0
Options Reconfigured:
encryption.block-size: 4096
```

```
encryption.data-key-size: 256
encryption.master-key: /root/mk.key
performance.open-behind: off
performance.write-behind:off
performance.quick-read:off
features.encryption: on
transport.address-family: inet
nfs.disable: on
performance.client-io-threads: off
```

Para garantir que o volume é mesmo criptografado é necessário verificar o gráfico de tradução na máquina cliente. O gráfico é criado quando é "montado". O caminho padrão de "montagem"é em `/usr/local/var/log/glusterfs/<mount-point>.log`. O dado contido nesse arquivo será mais ou menos dessa forma:

```
13: volume vol1-crypt
14:     type encryption/crypt
15:     option master-key /root/mk.key
16: end-volume
```

ANEXO C – Instalação WireGuard

Na instalação do WireGuard, é iniciado pela instalação do *software* e depois a criação dos pares de chave criptografadas para cada servidor. Depois, um pequeno arquivo de configuração será criado para definir as informações de conexão dos pares. Com a iniciação da interface criada pelo WireGuard, já será possível enviar mensagens seguras entre os servidores. O projeto WireGuard fornece um "ppa"com pacotes atualizados para sistemas Ubuntu. Em cada servidor é executado as seguintes ações:

Adicionando o repositório "ppa"do WireGuard ao sistema

```
#sudo add-apt-repository ppa:wireguard/wireguard
```

É necessário pressionar o botão de `ENTER`, quando solicitado, para adicionar a nova fonte de pacote a configuração "apt"dos servidores. Após a adição do "ppa", é fundamental a atualização do índice do pacote local para obtenção de informações sobre os novos pacotes disponíveis, sendo possível em seguida instalar o WireGuard.

```
#sudo apt-get update
#sudo apt-get install wireguard-dkms wireguard-tools
```

Criação das chaves pública e privada

Cada participante de uma VPN WireGuard se autentica em seus pares usando criptografia de chaves públicas. As conexões entre novos pares podem ser estabelecidas trocando chaves públicas e executando uma configuração mínima. Para gerar uma chave privada e gravá-la diretamente em um arquivo de configuração do WireGuard, é fundamental digitar o seguinte em cada servidor:

```
#(umask 077 && printf "[Interface]\nPrivateKey = " | sudo tee \
/etc/wireguard/wg0.conf > /dev/null)
#wg genkey | sudo tee -a /etc/wireguard/wg0.conf | wg pubkey | \
sudo tee /etc/wireguard/publickey
```

O primeiro comando grava o conteúdo inicial de um arquivo de configuração em `/etc/wireguard/wg0.conf`. O valor `umask` em um `sub-shell`, para que possamos criar o arquivo com permissões restritas sem afetar o ambiente regular. O segundo comando gera uma chave privada usando o comando `wg` do WireGuard e a grava diretamente em um arquivo de configuração restrito. Também canalizamos a chave de volta ao comando `wg pubkey` para derivar a chave pública associada, que gravamos em um arquivo chamado `/etc/wireguard/publickey` para fácil referência. As trocas das chaves públicas criadas em cada servidor será de grande importância para a criação do túnel VPN do WireGuard, sendo possível a comunicação entre os servidores.

Criando o arquivo de configuração inicial

É necessário a criação do arquivo de configuração da interface `wg`. Dentro do arquivo de configuração é possível verificar a chave privada gerada na seção `[Interface]`. Essa seção contêm as configurações da parte local da conexão. É requerido definir o IP que a VPN utilizará e a porta que escutará as conexões do *peers*. Em todos os servidores que utilizarão a conexão com o WireGuard é necessário repassar a mesma porta e um IP pertencente ao CIDR definido previamente.

```
#sudo nano /etc/wireguard/wg0.conf
```

O caminho do arquivo de configuração da interface é `/etc/wireguard/wg0.conf`, sendo necessário ser configurado em todos os nós.

```
[Interface]
PrivateKey = generated_private_key
ListenPort = 5555
SaveConfig = true
Address = 10.9.0.1/24
```

Em todos os nós é necessário configurar os dados dos outros *peers* que participarão da conexão VPN. Na seção `[Peer]` os dados necessários que devem ser incluídos são: chave pública no outro nó, o IP da VPN designado para cada par e o IP público utilizado por aquele nó. A configuração do arquivo de forma mais visual e entendível é exemplificado logo abaixo.

```
[Interface]
...
[Peer]
PublicKey = public_key_of_first_server
AllowedIPs = 10.9.0.1/32
Endpoint = public_IP_of_first_server:5555

[Peer]
PublicKey = public_key_of_second_server
AllowedIPs = 10.9.0.2/32
Endpoint = public_IP_of_second_server:5555

[Peer]
PublicKey = public_key_of_third_server
AllowedIPs = 10.9.0.3/32
Endpoint = public_IP_of_third_server:5555

[Peer]
PublicKey = public_key_of_fourth_server
AllowedIPs = 10.9.0.4/32
Endpoint = public_IP_of_fourth_server:5555
```

Em seguida o arquivo pode ser fechado e salvo. Em todos os servidores é necessário aplicar regras de *firewall*, habilitando a porta definida anteriormente no arquivo de configuração da interface do WireGuard.

```
#sudo ufw allow 5555
```

Para finalizar o processo de instalação, somente é necessário iniciar o serviço do WireGuard, `wg-quick` usando a interface `wg0`. Com a inicialização da interface de rede nos servidores é possível confirmar seu funcionamento com as demais interfaces de rede existentes. Com todas as etapas concluídas a conexão ponto a ponto estará disponível.

```
#sudo systemctl start wg-quick@wg0
```

```
#ip addr show wg0
```

Para testar o túnel VPN é so "pingar"os IPs atribuídos aos outros *peers* das interfaces do WireGuard. Para iniciar o túnel com o *boot* dos servidores é necessário habilitar o serviço em cada servidor.

```
#ping -c 3 10.9.0.2

PING 10.9.0.2 (10.9.0.2) 56(84) bytes of data.
64 bytes from 10.9.0.2: icmp_seq=1 ttl=64 time=0.635 ms
64 bytes from 10.9.0.2: icmp_seq=2 ttl=64 time=0.615 ms
64 bytes from 10.9.0.2: icmp_seq=3 ttl=64 time=0.841 ms

--- 10.9.0.2 ping statistics ---
3 packets transmitted, 3 received, 0\% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.615/0.697/0.841/0.102 ms
```

```
#sudo systemctl enable wg-quick@wg0
```

ANEXO D – Instalação Nextcloud

O Nextcloud, por ser um serviço que interage com os usuários através da web, tem como pré-requisito um servidor web rodando localmente. Nginx foi a escolha de servidor web para esse fim. O Nginx é instalado e iniciado no linux com os comandos:

```
sudo apt install nginx
sudo systemctl enable nginx
sudo systemctl start nginx
```

Em seguida, concede-se permissões especiais ao usuário do Nginx (www-data) no diretório a ser usado para hospedar os arquivos do web server:

```
sudo chown www-data:www-data /usr/share/nginx/html -R
```

Além de um servidor web, o Nextcloud também necessita de um banco de dados. Foi utilizado a opção *Open source* MariaDB, com a seguinte instalação e ativação:

```
sudo apt install mariadb-server mariadb-client
sudo systemctl start mariadb
sudo systemctl enable mariadb
```

Após a instalação, executa-se o script seguro de instalação com o comando:

```
sudo mysql_secure_installation
PRESSIONE ENTER para remover usuário anônimo
PRESSIONE ENTER para desabilitar login de root remoto
PRESSIONE ENTER para remover tabela de dados de teste
PRESSIONE ENTER para efetivar opções anteriores.
```

O PHP e algumas extensões também são necessárias para instalação do Nextcloud. Com os comandos a seguir, o PHP, extensões e dependências serão instalados e iniciados:

```
sudo apt install php7.2 php7.2-fpm php7.2-mysql php-common php7.2-cli \
    php7.2-common php7.2-json php7.2-opcache php7.2-readline \
    php7.2-mbstring php7.2-xml php7.2-gd php7.2-curl
sudo systemctl start php7.2-fpm
sudo systemctl enable php7.2-fpm
```

Por boas práticas, não é adequado rodar código PHP no bloco padrão de servidor do Nginx. Para isso, é removido o link simbólico do diretório *sites-enabled* e criado um novo arquivo de bloco para o servidor com os comandos:

```
sudo rm /etc/nginx/sites-enabled/default
sudo nano /etc/nginx/conf.d/default.conf
```

Dentro desse aquivo, estarão listados as funções que o Nginx fará no sistema. Em particular, é preciso que o Nginx escute a porta IPV4 80 e IPV6 80 da instância e processe todas as requisições. Para isso, o arquivo *default.conf* deverá conter o seguinte código:

```
server {
  listen 80;
  listen [::]:80;
  server_name _;
  root /usr/share/nginx/html/;
  index index.php index.html index.htm index.nginx-debian.html;

  location / {
    try_files $uri $uri/ /index.php;
  }

  location ~ \.php$ {
    fastcgi_pass unix:/run/php/php7.2-fpm.sock;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
```

```
    include fastcgi_params;
    include snippets/fastcgi-php.conf;
  }


 # A long browser cache lifetime can speedup repeat visits to ur page
  location ~*
  \.(jpg|jpeg|gif|png|webp|svg|woff|woff2|ttf|css|js|ico|xml)$
  {
       access_log        off;
       log_not_found     off;
       expires           360d;
  }

  # disable access to hidden files
  location ~ /\.ht {
      access_log off;
      log_not_found off;
      deny all;
  }
}
```

Com o arquivo salvo, reinicia-se o Nginx para que as ações sejam aplicadas:

```
sudo systemctl reload nginx
```

Com o Nginx, PHP e banco de dados instalados, já possui-se todos os componentes necessários para levantar um serviço baseado em Web. O próximo passo é baixar o Nextcloud. Neste projeto, decidiu-se seguir com a versão 13.0.2. O arquivo com o Nextcloud é baixado com o comando *wget* e descompactado no diretório do web server Nginx. Em seguida, é dado permissão especial nesse diretório ao usuário no Nginx. Os comandos a serem executados são:

```
sudo apt install unzip
```

```
sudo unzip nextcloud-13.0.2.zip -d /usr/share/nginx/
sudo chown www-data:www-data /usr/share/nginx/nextcloud/ -R
```

Antes da ativação do Nextcloud, é necessário a configuração de um banco de dados para esse serviço. É criado, então, um banco de dados com nome de `nextcloud`, um usuário e uma senha para o banco. Essas serão as credenciais do Nextcloud para executar ações no banco de dados. Por esse motivo, é preciso garantir os privilégios necessários do banco de dados recém criado ao novo usuário. Os comandos a serem executados são:

```
sudo mariadb
create database nextcloud;
create user nextclouduser@localhost identified by 'nextcloudpassword';
grant all privileges on nextcloud.* to nextclouduser@localhost /
identified by 'nextcloudpassword';
flush privileges;
exit;
```

Com o banco de dados criado, o serviço do Nextcloud já está pronto para ser executado. Porém, é necessário ainda criar, nos arquivos de configuração do Nginx, um arquivo de configuração para que o web server saiba que aquele serviço será o responsável por requisições recebidas na porta 80, além de outras configurações. O arquivo deve ser criado em `/etc/nginx/conf.d/nextcloud.conf`. Na segunda linha, deve ser colocado o IP da instância onde o serviço do Nextcloud está sendo instalado, uma vez que será nesse IP que o servidor responderá às requisições web. O arquivo completo deve ter o seguinte formato:

```
server {
    listen 80;
    server_name nextcloud.your-domain.com;

    # Add headers to serve security related headers
    add_header X-Content-Type-Options nosniff;
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Robots-Tag none;
```

```
    add_header X-Download-Options noopen;
    add_header X-Permitted-Cross-Domain-Policies none;
    add_header Referrer-Policy no-referrer;

    #I found this header is needed on Ubuntu, but not on Arch Linux.
    add_header X-Frame-Options "SAMEORIGIN";

    # Path to the root of your installation
    root /usr/share/nginx/nextcloud/;

    access_log /var/log/nginx/nextcloud.access;
    error_log /var/log/nginx/nextcloud.error;

    location = /robots.txt {
        allow all;
        log_not_found off;
        access_log off;
    }

    # The following 2 rules are only needed for
    #the user_webfinger app.
    # Uncomment it if you're planning to use this app.
    #rewrite ^/.well-known/host-meta
    #/public.php?service=host-meta last;
    #rewrite ^/.well-known/host-meta.json
    #/public.php?service=host-meta-json
    # last;

    location = /.well-known/carddav {
        return 301 $scheme://$host/remote.php/dav;
    }
    location = /.well-known/caldav {
       return 301 $scheme://$host/remote.php/dav;
    }

    location ~ /.well-known/acme-challenge {
      allow all;
```

```
    }

    # set max upload size
    client_max_body_size 512M;
    fastcgi_buffers 64 4K;

    # Disable gzip to avoid the removal of the ETag header
    gzip off;

    # Uncomment if your server is build with the ngx_pagespeed module
    # This module is currently not supported.
    #pagespeed off;

    error_page 403 /core/templates/403.php;
    error_page 404 /core/templates/404.php;

    location / {
       rewrite ^ /index.php$uri;
    }

    location ~ ^/(?:build|tests|config|lib|3rdparty|templates|data)/ {
       deny all;
    }
    location ~ ^/(?:\.|autotest|occ|issue|indie|db_|console) {
       deny all;
     }

    location ~ ^/(?:index|remote|public|cron|core/ajax/update|status|/
    ocs/v[12]|updater/.+|ocs-provider/.+|core/templates//
    40[34])\.php(?:$|/)
    {
       include fastcgi_params;
       fastcgi_split_path_info ^(.+\.php)(/.*)$;
       fastcgi_param SCRIPT_FILENAME $document_root$ \
       fastcgi_script_name;
       fastcgi_param PATH_INFO $fastcgi_path_info;
       #Avoid sending the security headers twice
```

```
        fastcgi_param modHeadersAvailable true;
        fastcgi_param front_controller_active true;
        fastcgi_pass unix:/run/php/php7.2-fpm.sock;
        fastcgi_intercept_errors on;
        fastcgi_request_buffering off;
    }

    location ~ ^/(?:updater|ocs-provider)(?:$|/) {
        try_files $uri/ =404;
        index index.php;
    }

    # Adding the cache control header for js and css files
    # Make sure it is BELOW the PHP block
    location ~* \.(?:css|js)$ {
        try_files $uri /index.php$uri$is_args$args;
        add_header Cache-Control "public, max-age=7200";
        # Add headers to serve security related headers
        #(It is intended to
        # have those duplicated to the ones above)
        add_header X-Content-Type-Options nosniff;
        add_header X-XSS-Protection "1; mode=block";
        add_header X-Robots-Tag none;
        add_header X-Download-Options noopen;
        add_header X-Permitted-Cross-Domain-Policies none;
        add_header Referrer-Policy no-referrer;
        # Optional: Don't log access to assets
        access_log off;
    }

    location ~* \.(?:svg|gif|png|html|ttf|woff|ico|jpg|jpeg)$ {
        try_files $uri /index.php$uri$is_args$args;
        # Optional: Don't log access to other assets
        access_log off;
    }
}
```

Após isso, as configurações realizadas são salvas e o serviço do Nginx é reiniciado para que as ações sejam aplicadas:

```
sudo nginx -t
sudo systemctl reload nginx
```

Criação do certificado *self-signed SSL/TLS*

Com todos os passos anteriores executados, o Nextcloud é acessível através do IP atribuído no campo `server_name` no *browser*, porém, a conexão entre um usuário e a aplicação Nextcloud não é realizada de forma segura, assim para garantir uma troca de dados criptografados é necessário configurar o protocolo HTTPS, no qual envolve a implementação do HTTP sobre uma camada adicional de segurança que usa o protocolo SSL/TLS.

Quando não é utilizado um nome de domínio, a forma de criar uma conexão segura é através de um certificado autoassinado, *self-signed certificate*, no qual não é assinado por uma autoridade de certificação (CA). Para a geração do certificado, é utilizado o OpenSSL, no qual cria 2 arquivos, consistindo em uma chave privada e uma chave pública. Antes da geração das chaves, é criado um arquivo de certificado de configuração que especificará os bits do certificado. Os passos para a criação do arquivo de certificado de configuração juntamente com seu conteúdo e em seguida, a criação do certificado usando o OpenSSL são definidos a seguir.

```
sudo nano localhost.conf
```

```
[req]
default_bits       = 2048
default_keyfile    = localhost.key
distinguished_name = req_distinguished_name
req_extensions     = req_ext
x509_extensions    = v3_ca

[req_distinguished_name]
countryName                 = Country Name (2 letter code)
```

```
countryName_default         = US
stateOrProvinceName         = State or Province Name (full name)
stateOrProvinceName_default = New York
localityName                = Locality Name (eg, city)
localityName_default        = Rochester
organizationName            = Organization Name (eg, company)
organizationName_default    = localhost
organizationalUnitName      = organizationalunit
organizationalUnitName_default = Development
commonName                  = Common Name (e.g. YOUR name)
commonName_default          = localhost
commonName_max              = 64

[req_ext]
subjectAltName = @alt_names

[v3_ca]
subjectAltName = @alt_names

[alt_names]
DNS.1   = localhost
DNS.2   = 127.0.0.1
```

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
-keyout localhost.key -out localhost.crt -config localhost.conf
```

Com a criação, é necessário copiar o par de chaves geradas para o diretório de certificados no Ubuntu, esse diretório será utilizado para a configuração do *web server* Nginx, mais especificamente, no arquivo de configuração do Nextcloud. Após a cópia das chaves e a alteração no arquivo de configuração do Nginx, somente é necessário salvar as configurações realizadas. O Nextcloud irá responder de imediato na porta 443 utilizado-a para acesso no *browser*. Para o Nextcloud somente responder requisições na porta 443, deve ser inserido um comentário na linha correspondente ao uso da porta 80 no arquivo de configuração do Nextcloud no Nginx.

Cópia da chave pública para o diretório `/etc/ssk/certs` e cópia da chave privada para o diretório `/etc/ssl/private`.

```
sudo cp localhost.crt /etc/ssl/certs/localhost.crt
sudo cp localhost.key /etc/ssl/private/localhost.key
```

```
sudo nano /etc/nginx/conf.d/nextcloud.conf
```

```
server {
    #listen 80;
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    server_name localhost;
    server_name nextcloud.your-domain.com;

    ssl_certificate /etc/ssl/certs/localhost.crt;
    ssl_certificate_key /etc/ssl/private/localhost.key;

    ssl_protocols TLSv1.2 TLSv1.1 TLSv1;

    # Add headers to serve security related headers
    add_header X-Content-Type-Options nosniff;
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Robots-Tag none;
    add_header X-Download-Options noopen;
    add_header X-Permitted-Cross-Domain-Policies none;
    add_header Referrer-Policy no-referrer;

    #I found this header is needed on Ubuntu, but not on Arch Linux.
    add_header X-Frame-Options "SAMEORIGIN";

    # Path to the root of your installation
    root /usr/share/nginx/nextcloud/;
```

```
access_log /var/log/nginx/nextcloud.access;
error_log /var/log/nginx/nextcloud.error;

location = /robots.txt {
    allow all;
    log_not_found off;
    access_log off;
}

# The following 2 rules are only needed for
#the user_webfinger app.
# Uncomment it if you're planning to use this app.
#rewrite ^/.well-known/host-meta
#/public.php?service=host-meta last;
#rewrite ^/.well-known/host-meta.json
#/public.php?service=host-meta-json
# last;

location = /.well-known/carddav {
    return 301 $scheme://$host/remote.php/dav;
}
location = /.well-known/caldav {
   return 301 $scheme://$host/remote.php/dav;
}

location ~ /.well-known/acme-challenge {
  allow all;
}

# set max upload size
client_max_body_size 512M;
fastcgi_buffers 64 4K;

# Disable gzip to avoid the removal of the ETag header
gzip off;
```

```
    # Uncomment if your server is build with the ngx_pagespeed module
    # This module is currently not supported.
    #pagespeed off;

    error_page 403 /core/templates/403.php;
    error_page 404 /core/templates/404.php;

    location / {
       rewrite ^ /index.php$uri;
    }

    location ~ ^/(?:build|tests|config|lib|3rdparty|templates|data)/ {
       deny all;
    }
    location ~ ^/(?:\.|autotest|occ|issue|indie|db_|console) {
       deny all;
     }

    location ~ ^/(?:index|remote|public|cron|core/ajax/update|status|/
    ocs/v[12]|updater/.+|ocs-provider/.+|core/templates//
    40[34])\.php(?:$|/)
    {
       include fastcgi_params;
       fastcgi_split_path_info ^(.+\.php)(/.*)$;
       fastcgi_param SCRIPT_FILENAME $document_root$ \
       fastcgi_script_name;
       fastcgi_param PATH_INFO $fastcgi_path_info;
       #Avoid sending the security headers twice
       fastcgi_param modHeadersAvailable true;
       fastcgi_param front_controller_active true;
       fastcgi_pass unix:/run/php/php7.2-fpm.sock;
       fastcgi_intercept_errors on;
       fastcgi_request_buffering off;
    }

    location ~ ^/(?:updater|ocs-provider)(?:$|/) {
       try_files $uri/ =404;
```

```
        index index.php;
    }

    # Adding the cache control header for js and css files
    # Make sure it is BELOW the PHP block
    location ~* \.(?:css|js)$ {
        try_files $uri /index.php$uri$is_args$args;
        add_header Cache-Control "public, max-age=7200";
        # Add headers to serve security related headers
        #(It is intended to
        # have those duplicated to the ones above)
        add_header X-Content-Type-Options nosniff;
        add_header X-XSS-Protection "1; mode=block";
        add_header X-Robots-Tag none;
        add_header X-Download-Options noopen;
        add_header X-Permitted-Cross-Domain-Policies none;
        add_header Referrer-Policy no-referrer;
        # Optional: Don't log access to assets
        access_log off;
   }

   location ~* \.(?:svg|gif|png|html|ttf|woff|ico|jpg|jpeg)$ {
        try_files $uri /index.php$uri$is_args$args;
        # Optional: Don't log access to other assets
        access_log off;
   }
}
```

Recarregar as alterações no arquivo de configuração do Nextcloud no Nginx.

```
sudo service nginx reload
```