Capítulo

5

Ambiente de Nuvem Computacional Privada para Teste e Desenvolvimento de Programas Paralelos

Anderson M. Maliszewski[1], Adriano Vogel[2], Dalvan Griebler[3], Claudio Schepke[4], Philippe O. A. Navaux[5]

Resumo

A computação de alto desempenho costuma utilizar agregados de computadores para a execução de aplicações paralelas. Alternativamente, a computação em nuvem oferece recursos computacionais distribuídos para processamento com um nível de abstração além do tradicional, dinâmico e sob-demanda. Este capítulo tem como objetivo introduzir conceitos básicos, apresentar noções básicas para implantar uma nuvem privada e demonstrar os benefícios para o desenvolvimento e teste de programas paralelos em nuvem.

5.1. Introdução

Com o aumento da complexidade e do número de problemas computacionais, assim como, do valor de aquisição de infraestruturas particulares, percebeu-se um aumento significativo na utilização de ambientes computacionais que proveem recursos de forma

[1]Grupo de Processamento Paralelo e Distribuído - GPPD, Instituto de Informática - INF, Universidade Federal do Rio Grande do Sul - UFRGS - Brasil, email: ammaliszewski@inf.ufrgs.br, ORCID: 0000-0001-5585-3471

[2]Escola Politécnica, Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS - Brasil e Laboratório de Pesquisas Avançadas para Computação em Nuvem - LARCC Faculdade Três de Maio - SETREM - Brasil, email: adriano.vogel@acad.pucrs.br, ORCID: 0000-0003-3299-2641

[3]Escola Politécnica, Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS - Brasil e Laboratório de Pesquisas Avançadas para Computação em Nuvem - LARCC Faculdade Três de Maio - SETREM - Brasil, email: dalvan.griebler@pucrs.br, ORCID: 0000-0002-4690-3964

[4]Laboratório de Estudos Avançados em Computação - LEA Universidade Federal do Pampa - UNIPAMPA - Campus Alegrete – Brasil, email: claudioschepke@unipampa.edu.br, ORCID: 0000-0003-4118-8831

[5]Grupo de Processamento Paralelo e Distribuído - GPPD, Instituto de Informática - INF, Universidade Federal do Rio Grande do Sul - UFRGS- Porto Alegre - RS - Brasil, email:navaux@inf.ufrgs.br, ORCID: 0000-0002-9957-5861

rápida, escalável e com pagamento pelo uso, como a computação em nuvem (MELL; GRANCE et al., 2011; BHOWMIK, 2017). A nuvem, por sua vez, vem sendo desenvolvida e disponibilizada de forma prática desde o inicio da década de 2010, levando a uma migração considerada agressiva de ambientes tradicionais para seu recinto. Entretanto, de acordo com previsões e pesquisas da Gartner[6], essa migração que já era considerada significativa antes da pandemia, tende a aumentar em aproximadamente 18% em 2021, chegando a impressionante marca de 304 bilhões de dólares investidos nesta tecnologia. Essa estatística leva em consideração principalmente a completa "validação" do ambiente de nuvem, uma vez que durante a crise do COVID-19, vários trabalhos tornaram-se remotos ou mesmo necessitaram maior flexibilidade, fazendo constante uso desta.

Consequentemente, essa crescente demanda por computação em nuvem no mercado e seus desafios no gerenciamento de recursos, beneficia a pesquisa e desenvolvimento da mesma. Apesar de que melhorias e novas solução são necessárias, ferramentas que criam as chamadas nuvens privadas (ambientes isolados para o gerenciamento de recursos dentro de instituições), como por exemplo, OpenNebula, OpenStack e CloudStack, já vem sendo estudadas e abordadas em diversos artigos científicos (VOGEL et al., 2016; ROVEDA et al., 2015; MALISZEWSKI et al., 2019). Desta forma, o objetivo deste capítulo é apresentar de forma útil e prática, a criação e implantação de uma nuvem computacional privada. Este tipo de nuvem pode ser uma alternativa para o gerenciamento de recursos computacionais, tornando-a um ambiente eficiente, flexível e de baixo custo para desenvolvimento ou teste de programas paralelos.

A estrutura do capítulo está dividida da seguinte forma. As primeiras seções explicam aspectos conceituais básicos. Na Seção 5.2 são discutidas as formas em que a computação pode ocorrer em ambientes paralelos e distribuídos. Na Seção 5.3 são apresentadas as formas de virtualização em hardware e em sistema operacional. Já na Seção 5.4 tem-se toda a formulação necessária para compreender como é feita a computação em nuvem. A Seção 5.5 mostra como é feita a implementação da nuvem privada com OpenNebula, configuração e realização de testes MPI. Por fim, a Seção 5.6, fecha o capítulo com uma conclusão.

5.2. Computação Paralela e Distribuída

Os computadores paralelos evoluíram de máquinas proprietárias e dedicadas para se tornaram as ferramentas diárias dos cientistas de diversas áreas de conhecimento, que precisam poder de processamento computacional para resolver seus problemas. A computação paralela e distribuída é imprescindível para que as aplicações computacionais possam usar os recursos de hardware disponíveis.

Apesar de terem diferentes significados e implicações, os termos computação paralela e computação distribuída são muitas vezes considerados como sinônimos (BUYYA; VECCHIOLA; SELVI, 2013). Porém, computação paralela corresponde a um modelo onde é possível separar as computações e processá-las individualmente em processadores usualmente homogêneos e se comunicando com memória compartilhada. Por outro lado, o termo computação distribuída é mais genérico, voltado para computações que po-

[6]<https://www.gartner.com/en/newsroom/press-releases/2020-11-17-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-grow-18-percent-in-2021>

dem ser executadas em múltiplos processadores, bem como em múltiplas máquinas, como execuções em *clusters*, onde a comunicação ocorre por trocas de mensagens.

O processamento paralelo ocorre quando múltiplas tarefas são executadas ao mesmo tempo em múltiplos processadores, onde diferentes técnicas, abordagens e modelos de exploração de paralelismo podem ser usados, como mestre-escravo e divisão e conquista. Ainda, chama-se programação paralela a ação de modelar e codificar uma determinada computação para executar em paralelo.

5.2.1. Modelos de Computação Paralela

Executar um programa de forma paralela demanda uma modelagem para explorar apropriadamente os recursos paralelos do hardware, criando um modelo computacional para execução paralela. Um modelo computacional pode ser visto como modelo conceitual, representando as operações e seus tipos disponíveis em um determinado programa, sem considerar sintaxes específicas e é usualmente pouco relacionado com a arquitetura do hardware (GROPP; LUSK; SKJELLUM, 2014). Portanto, um modelo computacional pode ser visto como uma estrutura de alto nível do programa. Porém, o desempenho e funcionalidade do programa paralelo é fortemente relacionado com a máquina a ser executado.

Os modelos computacionais podem ser classificados de diferentes formas e considerando diferentes aspectos, como a memória (compartilhada ou distribuída), a forma de comunicação, a demanda por comunicação, entre outros.

5.2.1.1. Memória Compartilhada

Um modelo computacional com controle simples do paralelismo e da comunicação é o de memória compartilhada. Nesse caso, cada processo ou *thread* tem acesso a toda memória da máquina que é representada como um único espaço compartilhado de endereçamento (GROPP; LUSK; SKJELLUM, 2014). É importante notar que as execuções paralelas demandam mecanismos de controle de acesso à memória, como *locks*, para coordenar o acesso à endereços modificados por múltiplos processos ou *threads*. Um exemplo do modelo de memória compartilhada são os programas executando em sistemas multicores, presentes em computadores pessoais, servidores e até em *smartphones*.

5.2.2. Memória Distribuída

O modelo de memória distribuída usa a troca de mensagens (*message-passing*) para comunicação entre processos que possuem memória local. Nesse caso, a rede de interconexão é usada para envio e recebimento de mensagens. Cada processo pode estar sendo executado em uma máquina diferente, caracterizando uma computação distribuída (GROPP; LUSK; SKJELLUM, 2014).

A computação distribuída é possível a partir da ligação de múltiplos sistemas e computadores independentes que através de abstrações são usados por usuários e gerenciados por programadores, como se fosse um único sistema. Tal abstração é possível graças a uma arquitetura bem definida e diversas interfaces entre as camadas de abstrações e entidades. A Figura 5.1 apresenta um exemplo de arquitetura de um sistema distribuído. No

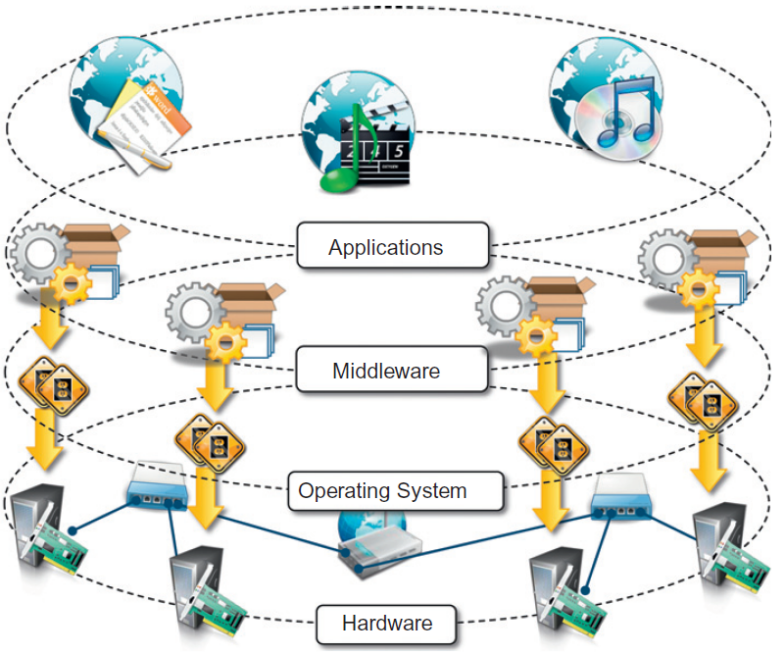


Figura 5.1. Visão geral de um sistema distribuído. Extraído de (BUYYA; VECCHIOLA; SELVI, 2013).

nível mais baixo tem-se o hardware, que é gerenciado pela camada superior: o sistema operacional. O hardware e sistemas operacionais se comunicam usando a infraestrutura de rede (VOGEL et al., 2017; MALISZEWSKI et al., 2019). Acima, o *middleware* é uma camada de abstração que controla os recursos subjacentes e oferece (*Application Programming Interface*) APIs para programar aplicações bem como mecanismos para gerenciar tais aplicações (BUYYA; VECCHIOLA; SELVI, 2013).

Na computação distribuída, uma interface amplamente aceita e utilizada para a programação é o MPI (SNIR et al., 1998), que pode ser definido como uma interface padrão com especificações e rotinas para implementar programas com execuções distribuídas, implementada nas linguagens C, C++ e Fortran. No Algoritmo 5.1 é mostrado um exemplo de código MPI, onde um vetor é computado em paralelo por `size` processos. Primeiramente, o processo identificado pelo `rank` 0 inicializa um vetor e os demais processos recebem esse vetor através de uma operação de *broadcast* (`MPI_Bcast`). Na sequência, cada processo realiza uma soma parcial de elementos de um trecho do vetor. Através de uma operação de redução, o processo de `rank` 0 recebe o resultado da computação parcial de cada processo e exibe o resultado final. Por fim, deve-se lembrar que toda a computação paralela de MPI deve estar limitada entre as chamadas `MPI_Init` e `MPI_Finalize`.

```
1 #include <mpi.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #define TAM 100
5
6 int main(int argc, char** argv) {
```

```
7   int myrank, size;
8   int i, local = 0, total;
9   int vet[TAM];
10
11  MPI_Init(&argc, &argv);
12  MPI_Comm_rank(MPI_COMM_WORLD, &myrank); //Quem sou?
13  MPI_Comm_size(MPI_COMM_WORLD, &size); //Quantos somos?
14
15  if (myrank == 0)
16    for(i = 0; i < TAM; i++)
17      vet[i] = 1;
18  MPI_Bcast(vet, TAM, MPI_INT, 0, MPI_COMM_WORLD);
19
20  for(i=(TAM/size)*myrank; i<(TAM/size)*(myrank+1); i++)
21    local += vet[i]; // Realiza as somas parciais
22
23  MPI_Reduce(&local,&total,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
24
25  if (myrank == 0)
26    printf ("Soma = %d\n", total);
27  MPI_Finalize();
28  return 0;
29 }
```

Listing 5.1. Um exemplo de computação paralela usando MPI.

5.3. Virtualização

A virtualização permite abstrair os recursos físicos como memória, processadores, armazenamento e rede, de tal forma que é possível criar ambientes virtuais e fornecer recursos usando máquinas virtuais (VMs). Portanto, a virtualização é um paradigma que revolucionou a forma como recursos computacionais são fornecidos para os usuários e é considerada a tecnologia principal para criação da computação em nuvem. Com a virtualização, é possível um melhor uso e gerenciamento dos recursos, instalando vários sistemas operacionais em diferentes máquinas virtuais no mesmo *hardware* (CHANDRASEKARAN, 2014), como representado na Figura 5.2.

Existem diferentes tipos e abordagens de virtualização. Neste documento, é relevante definir dois tipos: virtualização a nível de hardware usando virtualizadores representada pela solução (KVM) e virtualização de sistema operacional representada pelo LXC.

5.3.1. Virtualização de Hardware com KVM

A virtualização no nível de hardware é uma forma de virtualização que abstrai o hardware de computador, sendo possível executar nesse hardware múltiplos sistemas operacionais independentes e separados. Tais sistemas operacionais são executados dentro de VMs, hospedados pelo hardware físico do computador e gerenciados pelo virtualizador (*Hypervisor*). O virtualizador pode ser visto como uma camada que é um programa ou uma combinação de software e hardware que efetivamente abstrai o hardware (BUYYA; VECCHIOLA; SELVI, 2013).

KVM é uma solução de código aberto que cria um ambiente de virtualização com-

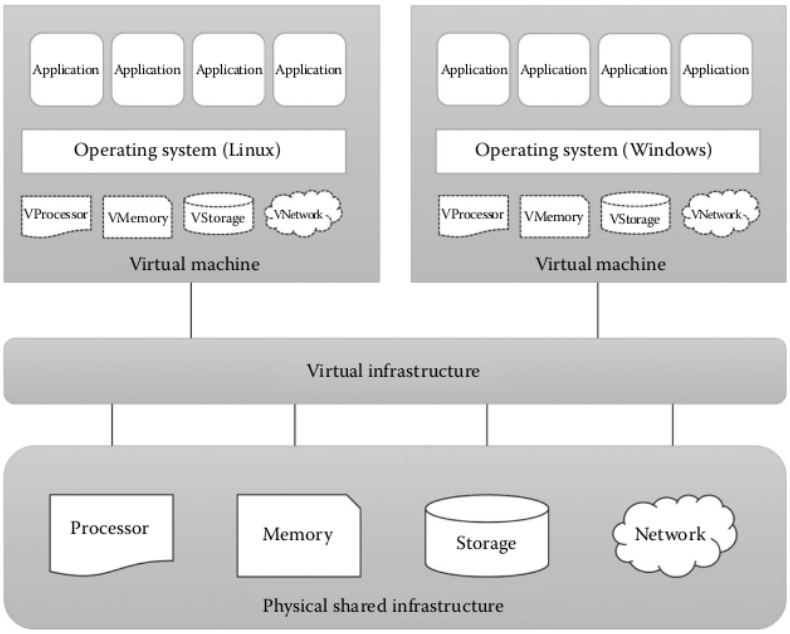


Figura 5.2. Visão geral de virtualização. Extraído de (CHANDRASEKARAN, 2014).

pleta de hardware. No KVM, cada máquina virtual criada é tratada com um processo regular do Linux e recebe um cotas de recursos e camadas de isolamento. Nesse cenário, cada VM tem seu próprio *kernel* separado.

5.3.2. Virtualização de Sistema Operacional LXC

Na virtualização no nível do sistema operacional, diferentemente da virtualização de hardware, não há gerenciador de máquina virtual ou virtualizador. Na virtualização de sistema operacional, o *kernel* do sistema operacional é uma das partes mais importantes pois este que permite várias instâncias isoladas seja executadas. Nesse cenário, o *kernel* do sistema operacional pode ser compartilhado e disponibiliza recursos do para as instâncias. Ainda, o *kernel* define e aplica cotas que limitam a quantidade de recursos que cada instância pode utilizar, como recursos de processador e memória.

O LXC é uma virtualização em nível de Sistema Operacional (SO), que abstrai os recursos computacionais por meio de Grupos de Controle (*cgroups*). No LXC, a criação e limitação do uso de recuros dos contêineres (LXC, 2019)é feita através de *namespaces*. Portanto, ps contêineres compartilham o mesmo *kernel* do sistema operacional nativo, onde seus processos e sistema de arquivos são acessíveis a partir do *host* hospedeiro possibilitando a execução de instruções nativas sem demandar mecanismos adicionais de interpretação. Nos contêineres, apesar do compartilhamento do *kernel* e de recursos, o sistema operacional fornece para as aplicações a ilusão de estarem executando em máquinas separadas.

A Figura 5.3 mostra uma comparação entre LXC (virtualização no nível do SO) e KVM (virtualização de hardware). Como pode ser visto, o LXC requer menos camadas de

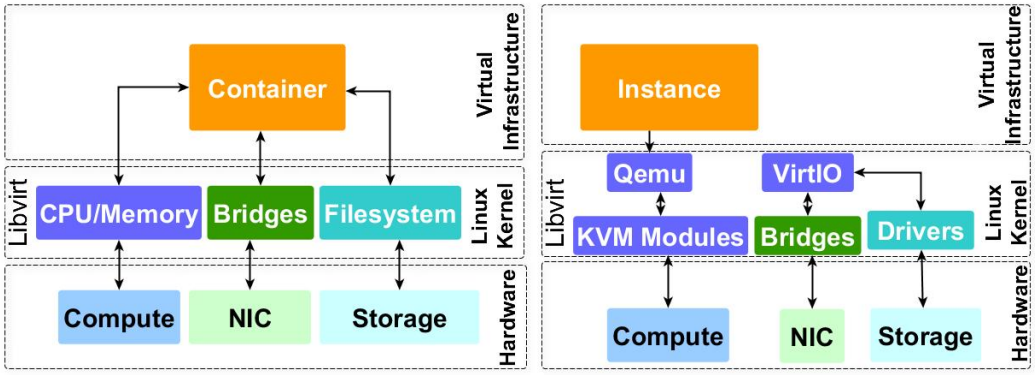


Figura 5.3. Visão geral do LXC (direita) e KVM (esquerda). Extraído de (VOGEL et al., 2017)

software, pois o KVM usa *drivers* VirtIO e bibliotecas para fornecer recursos e gerenciar as VMs.

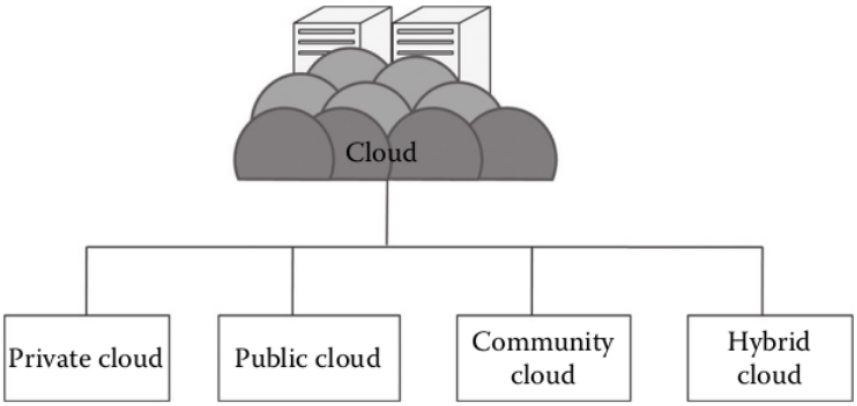
5.4. Computação em Nuvem

O paradigma de Computação em Nuvem surgiu para oferecer recursos computacionais na forma de serviços, facilitando o acesso e aumentando a disponibilidade de recursos computacionais (ROLOFF et al., 2012; VOGEL et al., 2016; GRIEBLER et al., 2018). Na infraestrutura, as aplicações e serviços são executados em ambientes virtualizados e oferecidos de forma simplificada usando conhecidos padrões e protocolos de rede. Conforme (BUYYA; VECCHIOLA; SELVI, 2013; CHANDRASEKARAN, 2014; BHOWMIK, 2017), as principais vantagens oferecidas pela Computação em Nuvem são: diminuição de custos de uso e investimento inicial, acesso amplo a recursos computacionais, potencial de escalabilidade e alta disponibilidade dos serviços. O provisionamento de serviços é dividido em IaaS (*Infrastructure as a Service*), PaaS (*Platform as a Service*), e SaaS (*Software as a Service*).

Computação em Nuvem é um conceito em consolidação de uma tecnologia que pode servir tanto para usuários finais, quanto para pesquisadores e/ou empresas, que através desta podem disponibilizar serviços com uma grande gama de opções e vantagens para seus clientes. Ela é constituída principalmente sobre os pilares de outras tecnologias (cluster, grid, redes), oferecendo serviços, que é basicamente definido pelo compartilhamento de recursos configuráveis. Sobre as tecnologias que compõem a nuvem, destaca-se a virtualização, que dentre vários benefícios, busca aproveitar e obter o máximo desempenho possível do hardware (BUYYA; VECCHIOLA; SELVI, 2013).

Dentre as características e vantagens que a Nuvem possui, pode-se destacar. A alta disponibilidade, na qual o usuário pode acessar sua infraestrutura de qualquer local apenas sendo necessária a conexão a internet. O provisionamento de recursos sob demanda, significa que a utilização dos recursos é disposta de acordo com o uso necessário do ambiente/aplicação, desta forma, evitando desperdício de utilização computacional que resulta em fatores positivos como a economia energética e compromisso ambiental. A rápida elasticidade, diferentemente de uma infraestrutura local, com a utilização da computação

Figura 5.4. Representação dos modelos de implantação de Nuvem. Figura extraída de (CHANDRASEKARAN, 2014)



em nuvem, pode-se aumentar ou diminuir a quantidade de recursos que se está utilizando. Para o usuário final, esta quantidade parece ser infinita e pode ser modificada em qualquer quantidade e em qualquer tempo.

Também é importante destacar o pagamento pelo uso. Apenas é necessário efetuar o pagamento em relação ao tempo de uso/quantidade de recursos que foram "locados" na nuvem, acarretando em economia financeira se comparada a uma infraestrutura convencional. Os serviços mensuráveis, qualquer serviço que é oferecido a partir da computação em nuvem é automaticamente controlado e otimizado no nível de abstração para sua demanda específica (ex. armazenamento, processamento, tráfego de rede, entre outros). Além disso, a utilização dos serviços pode ser monitorada e reportada, oferecendo transparência tanto para quem provisiona o serviço, quanto para quem o utiliza (VACCA, 2016; MELL; GRANCE et al., 2011; CHANDRASEKARAN, 2014).

A composição da nuvem é disposta em modelos de serviço que constituem uma pilha de três camadas, sendo elas o IaaS (*Infrastructure as a Service*) como base, PaaS (*Platform as a Service*) logo acima e SaaS (*Software as a Service*) no topo da pilha. Além disso, a nuvem inclui quatro modelos de implantação, descritos como Nuvem Privada, Nuvem Comunitária, Nuvem Pública e Nuvem Híbrida (BUYYA; VECCHIOLA; SELVI, 2013). Os modelos de implantação e os modelos de serviço serão descritos a seguir.

Visando uma melhor forma de disponibilização de serviços e objetivo final de sua utilização, a computação em nuvem divide-se em quatro principais modelos de implantação, cada qual com ênfase em uma determinada área. Na Figura 5.4 estão representados os 4 modelos de implantação, juntamente com suas características principais, a seguir descritos.

- **Nuvem Pública:** caracteriza-se pela utilização aberta ao público em geral. Neste modelo, é necessário que haja um provedor de nuvem. Os recursos oferecidos normalmente são gratuitos e passíveis de aquisição para realização de upgrades (ex. Maior capacidade de armazenamento). Esse é o modelo de nuvem mais conhecido entre os usuários. Destaca-se principalmente como provedores as empresas gigante da área de TI, que oferecem suas respectivas soluções em nuvem (ex. Amazon com

o AWS, Microsoft com o Azure, Google com o Google Drive).

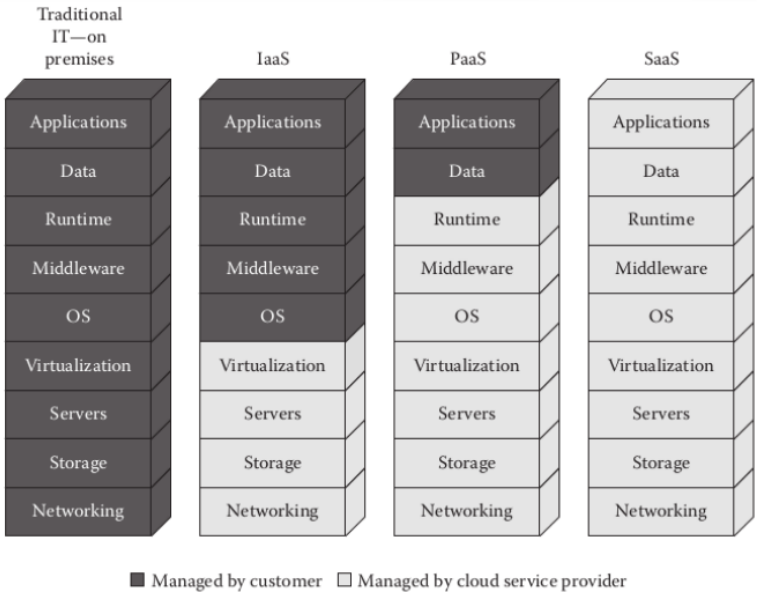
- **Nuvem Privada:** este modelo é criado/utilizado dentro de organizações empresariais, as quais gerenciam, criam e disponibilizam seus próprios recursos dentro da nuvem. Além disso, este modelo necessita de uma plataforma de criação de nuvem, sendo que a partir desta todos os recursos são gerenciados. Como exemplo temos o CloudStack, OpenNebula e o OpenStack, este último sendo a plataforma de cloud mais popular e utilizada.
- **Nuvem Híbrida:** este modelo tem como característica única a utilização de duas ou mais infraestruturas de nuvem distintas (ex. Nuvens privada, comunitária ou pública) que de forma isolada, continuam tendo a mesma designação, porém, utilizam recursos umas das outras (ex. Cloud bursting para balanceamento de carga entre nuvens).
- **Nuvem Comunitária:** é aquela em que os recursos são compartilhados entre várias organizações de uma área específica. Pode ser tanto gerenciada internamente, quanto terceirizada. Este é o modelo de nuvem menos conhecido e também o menos utilizado (MELL; GRANCE et al., 2011).

Através de três modelos de serviço, a nuvem busca atender a todo o tipo de demanda requisitada pelo usuário e/ou empresa. Na Figura 5.5 é ilustrado a representação em camadas de como os tipos de serviço se situam e qual é o campo de trabalho de cada uma. Na primeira camada, tem-se o IaaS (Infrastructure as a Service). Esta permite que os usuários realizem modificações de baixo nível, em comparação com as demais camadas. Nela são efetuados o provisionamento de recursos (ex. Armazenamento e processamento) a disposição do usuário. Além disso, este é capaz de implantar e executar qualquer software que inclui desde sistemas operacionais (SO) até aplicações propriamente ditas (BUYYA; VECCHIOLA; SELVI, 2013; CHANDRASEKARAN, 2014). Entretanto, mesmo esta sendo o camada de menor nível, o usuário que contrata um ambiente de nuvem IaaS não pode efetuar modificações ou controles “abaixo” de sua infraestrutura (ex. Modificações físicas no hardware). Esta camada é a que fornece infraestrutura para as demais (ex. instâncias), além de gerenciar questões como a virtualização.

Logo a seguir, temos o PaaS (Platform as a Service) que está diretamente associado a utilização e desenvolvimento de software através de uma infraestrutura já criada. O usuário pode configurar e realizar modificações ao nível de software, porém, não controla questões relacionadas a infraestrutura (ex. Armazenamento, rede ou servidores). Portanto, neste modelo de serviço é provisionado ao usuário a capacidade de realizar a implantação e criação de programas, além do compartilhamento deste para com os demais usuários (VACCA, 2016).

No topo da pilha de camadas temos o SaaS (Software as a Service), que como o próprio nome sugere, provisiona ao usuário o uso de aplicações executadas diretamente na nuvem. Além disso, as aplicações ficam disponíveis, em relação ao acesso, de várias maneiras (ex. Thin Client, navegador de internet ou mesmo uma interface da aplicação). Esta é a camada de maior nível, ou seja, mais distante do hardware. Com isso, o usuário fica limitado a realizar modificações apenas de acesso a aplicação por exemplo, excluindo qualquer modificação na infraestrutura (MELL; GRANCE et al., 2011).

Figura 5.5. Representação das camadas e serviços na nuvem. Figura Extraída de (VACCA, 2016).



5.4.1. Ferramentas para Nuvem Privada

Existem diversas ferramentas de código aberto para gerenciamento de infraestrutura de nuvem e criação de ambientes de nuvem pública, privada e comunitária. A virtualização que é uma camada adicionado no *hardware* é controlada e gerenciado por tais ferramentas. Logo a seguir, são apresentadas as principais ferramentas para implantação de ambientes de nuvem privada.

5.4.1.1. OpenStack

OpenStack[7] é uma ferramenta *open source* muito usada para criação de ambientes de nuvens privadas e públicas e para IaaS como um todo. O projeto OpenStack foi lançado pelas respeitadas Rackspace e NASA, e posteriormente diversas gigantes do mundo da tecnologia uniram-se ao projeto.

O OpenStack pode ser visto com uma combinação de diversos componentes e serviços independentes que funcionam de forma interoperável (VOGEL et al., 2016), o que aumenta a flexibilidade de implantações de nuvem. A escolha de qual componente usar é customizável de acordo com a demanda de cada arquiteto de nuvem (CHOWDHURY, 2017), sendo apenas necessária adoção de alguns componentes imprescindíveis do *core* da ferramenta.

Alguns componentes se destacam por serem imprescindíveis para uma ambiente de nuvem. Por exemplo, o *Keystone* que controla a autorização e autenticação de todo o ambiente de nuvem (componentes, serviços, usuários, etc) e precisa ser configurado e

[7]<https://www.openstack.org/>

conectado a cada componentes configurado no ambiente de nuvem. Ainda, o *Glance* é um componente que provê um repositório de imagens para a criação de instâncias na nuvem e componente *Neutron* oferece interconectividade. Diversos outros componentes podem ser adicionados a nuvem de forma customizável e a comunicação e sincronização entre os componentes ocorre por trocas de mensagens, sendo o *RabbitMQ* o mensageiro mais popular.

5.4.1.2. OpenNebula

OpenNebula[8] é uma ferramenta que surgiu na academia em um projeto que buscava propor novas soluções para gerenciamento de recursos virtuais em *data centers*. Após isso se tornou uma ferramenta para provisionamento de serviços de nuvem. O OpenNebula tem uma característica de buscar uma implantação simplificada, intuitiva, eficiente, e customizável para usuários e desenvolvedores (MILOJIČIĆ; LLORENTE; MONTERO, 2011).

A arquitetura do OpenNebula também buscar ser modular, iniciando com os *drivers* básicos que implementar controle sob a virtualização e no núcleo da ferramenta são implementadas as rotinas mais importantes como o controle de serviços, usuários, e escalonamento.

5.4.1.3. CloudStack

CloudStack[9] foi inicialmente desenvolvido pela empresa estadunidense de software *Cloud.com* que mais tarde foi adquirida pela *Citrix Systems*. Porém, a *Citrix Systems* doou o CloudStack à *Apache Software Foundation*. Dessa forma, o CloudStack passou a se chamar Apache CloudStack, que se tornou uma ferramenta estável principalmente para implantação de ambientes de nuvem privada. O Apache CloudStack tem APIs próprias e também suporta APIs compatíveis com a AWS (*Amazon Web Services*) que permitem integrar a nuvem privada com nuvens públicas tornando-se, portanto, uma nuvem híbrida (BHOWMIK, 2017; VOGEL et al., 2016).

O Apache CloudStack é focado para alta disponibilidade e flexibilidade e a sua instalação tem 2 componentes importantes: o gerente e o agente de nuvem. O gerente controla a infraestrutura virtual e é instalado no servidor principal, enquanto o agente é instalado nos demais nodos formandos clusters de disponibilidade de recursos.

5.4.2. Gerenciamento de Infraestrutura e Recursos

Em ambientes de nuvem é possível otimizar o gerenciamento de recursos através da flexibilidade da camada de virtualização e do controle oferecido por ferramentas de IaaS (BHOWMIK, 2017; VOGEL et al., 2016). Da perspectiva de gerenciamento da infraestrutura, exemplos de funcionalidades relevantes são:

- Cotas de uso de recursos: permite ao administrador do ambiente definir a quantidade que cada usuário pode alocar e utilizar. Por exemplo, no IaaS a flexibilidade

[8]<https://opennebula.io/>
[9]<https://cloudstack.apache.org/>

é maior pois é possível um usuário alocar uma determinada quantidade de recursos como processadores e memória ao invés de receber alguma máquina física. Tal cota terá um impacto na disponibilidade de recursos para as aplicações executadas no ambiente de nuvem, o que demanda um controle do uso re recursos de processamento (VOGEL; GRIEBLER; FERNANDES, 2021).

- Nodos dedicados para determinados usuários: para usuários que demandam um maior controle sobre o ambiente é possível criar máquinas dedicadas para usuários. Por exemplo, um usuário com requisitos rígidos de segurança.

- Otimizar a alocação de instâncias nos nodos físicos: a flexibilidade da virtualização também permite otimizar quais instâncias são executadas em quais nodos físicos, podendo oferecer ganhos de desempenho ou maior eficiência energética. Por exemplo, sob menor demanda as VMs podem ser movidas para alguns nodos específicos enquanto outros nodos podem ser desligados, potencialmente reduzindo o consumo energético. Em outros casos, VMs com maior demanda de desempenho podem ser alocadas de forma dedicada em nodos adequados.

Considerando as evidentes vantagens oferecidas por ambientes de nuvem, a tendência é que cada vez mais aplicações e cargas de trabalho sejam migradas para a nuvem. Dessa forma, existem diversas demandas de melhorias para o processo de migração, pois costuma ser mais complexo gerenciar tais ambientes robustos, o que demanda automações e abstrações adicionais para se ter produtividade. Uma potencial solução é o uso de contêineres que facilitam a execução de aplicações e o gerenciamento do ambiente. Quanto a programabilidade, se acredita que os *frameworks* de programação serão cada vez mais integrados e flexíveis para ambientes e aplicações da nuvem.

5.5. Ambiente de Desenvolvimento e Execução

Para este curso, escolheu-se utilizar a ferramenta de nuvem privada OpenNebula. Essa escolha baseou-se em fatores como uma maior facilidade para a implantação e configuração, o desenvolvimento da ferramenta ser de código aberto, além de já estar sendo utilizada em produção para realização de pesquisas no LARCC (Laboratório de Pesquisas Avançadas para Computação em Nuvem). A seguir serão descritos todos os procedimentos para instalar, configurar e gerenciar a nuvem privada OpenNebula[10].

5.5.1. Implantação do OpenNebula

A ferramenta OpenNebula fornece um ambiente intuitivo e simples, mas ao mesmo tempo oferece várias funcionalidades e soluções flexíveis para a implantação de nuvens e ambientes de virtualização privados ou híbridos. Primeiramente, serão descritos os processos de definição da arquitetura e projeto da nuvem a ser implantada.

[10]Os itens a seguir estão descritos igualmente como na documentação oficial da ferramenta, localizada em <https://docs.opennebula.io/5.12/index.html>

5.5.1.1. Arquitetura da Nuvem

Para criar um ambiente confiável é necessário criar um plano de implantação. Neste projeto, devem ser alinhados funcionalidades esperadas e também quais componentes de hardware serão adicionados e abstraídos para a nuvem. Isso engloba:

- A infraestrutura - *hardware* (rede, *storages* e servidores).
- O dimensionamento de recursos da nuvem, baseando-se principalmente em características como número de usuários e cargas de trabalho que serão utilizadas.
- Fluxo de provisionamento, o qual inclui a forma de como os usuários serão isolados, utilização da nuvem e qual será a forma de seu acesso.

É necessário criar um plano que inclua ferramentas, desempenho, escalabilidade e características de alta disponibilidade. Desta forma, a arquitetura de nuvem é definida em três componentes, sendo eles, o armazenamento, rede e a virtualização. O OpenNebula presume que o ambiente ao qual será instalado utiliza a arquitetura clássica de um *cluster beowulf*, com um *Frontend* e demais *hosts* que serão utilizados para hospedar as VMs[11]. Além disso, também é necessário de uma rede física para que os nodos se comuniquem e sejam adicionados no *frontend*. Os componentes previamente descritos da arquitetura, tem funções específicas definidas pelo OpenNebula, sendo elas:

- *Frontend*: Executar os serviços do OpenNebula.
- *Hosts*: Provisionam os recursos necessários para as VMs. (Necessário que esteja habilitado o suporte à virtualização.)
- *Storage*: Responsável por armazenar as imagens de *templates* e discos das VMS.
- Redes Físicas: Usadas para realizar a comunicação entre o *storage*, *frontend*, e *hosts* com as VMs.

Na Figura 5.6 temos a representação da arquitetura descrita e a interconexão entre os componentes. Somado a isso, uma ampla gama de recursos pode ser incorporada juntamente ao OpenNebula.

5.5.1.2. Dimensionando a Nuvem

O dimensionamento de recursos da infraestrutura de nuvem é um dos pontos chave para garantir o bom funcionando do sistema. Existem requisitos mínimos disponibilizados pelo OpenNebula para o *frontend*, nodos KVM e nodos LXC. A seguir eles serão listados na Tabela 5.1.

[11]Embora a utilização do OpenNebula possa ser realizada em apenas um *host*, onde serão instalados ambas soluções de *frontend* e nodo, essa forma de implantação terá várias limitações, incluindo desempenho e escalonamento, por exemplo. Assim, recomenda-se a utilização de um *cluster*

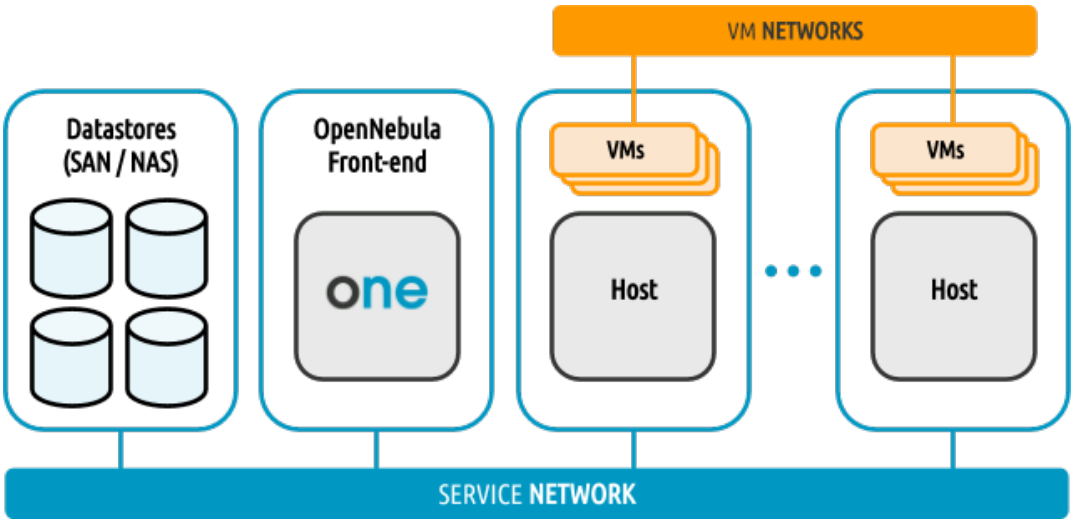


Figura 5.6. Representação da arquitetura da nuvem. Figura extraída de (OPENNEBULA, 2021)

.

	Frontend	Nodos KVM	Nodos LXD
Recursos	Mínimo	Mínimo	Mínimo
Memória	8 GB	1 GB para cada CPU core	>Nodos KVM
CPU	2 CPU (4 cores)	Varia de acordo com a quantidade de *hosts* e utilização de *overcommitment*.	>Nodos KVM
Disco	200 GB		
Rede	2 NICs		

Tabela 5.1. Requisitos Mínimos de Hardware, incluindo o *frontend*, nodos que utilizam o virtualizador KVM e LXD.

Os requisitos de *hardware* dos nodos KVM variam de acordo a utilização de premissas como *overcommitment* de CPU. Caso esta opção não esteja sendo utilizada, definise que cada núcleo de CPU delegado para uma VM deve existir fisicamente e 1 GB de memória RAM disponível para cada núcleo de CPU. Por outro lado, com a utilização de *overcommitment* de CPU, o dimensionamento de CPU pode ser realizado com antecedência, usando os atributos de CPU e vCPU. Por outro lado, os nodos LXD não definem precisamente uma quantia mínima de recursos necessários, uma vez que este não emula/-virtualiza o hardware e seu *overhead* de virtualização é menor se comparado ao KVM. Portanto, os recursos mínimos para os nodos LXD são menores que os requisitos mínimos para nodos KVM.

Para dimensionar o armazenamento no OpenNebula, é necessário primeiro entender como o armazenamento é organizado. Primeiramente, ele é dividido em forma de três *datastores*: o de imagem, sistema e de arquivos. O primeiro é onde a ferramenta guarda todas as imagens que podem ser utilizadas para criar VMs (imagens de sistemas operacionais). As imagens são movidas, clonadas, de/para o sistema de *datastores* quando as VMs são implantadas, desligadas, adicionados discos ou criados *snapshots*. Desta forma, é necessário que o armazenamento criado tenha pelo menos o tamanho para guardar o número de imagens a serem utilizadas. O *datastore* de sistemas é representado por onde as VMs em execução armazenam seus dados. Dependendo da tecnologia utilizada no armazenamento, essas imagens podem ser copias completas da imagem original `QCOW` ou simplesmente os links do sistema de arquivos. Estimar o tamanho mínimo das imagens é mais complexo, uma vez que podem ser utilizadas discos voláteise varia conforme o sistema operacional utilizado nas VMs. Por fim, o *datastore* de arquivos é usado para armazenar arquivos comuns, como documentos ou anotações.

Em relação a rede, é necessário que esta seja criada com cuidado a fim de garantir a confiabilidade da infraestrutura de nuvem. Recomenda-se 2 interfaces físicas de rede (NICs) no *frontend*, ou 3 dependendo da forma de armazenamento utilizada. Nos *hosts* são recomendadas 4 interfaces (IP privado, IP público, serviços e armazenamento). No entanto, para ambientes com necessidades menores, um número menor de interfaces pode ser adotada.

5.5.1.3. Frontend

A máquina física que hospeda a instalação do OpenNebula (ONE) é chamada de *frontend*. Ela necessita de conexão de rede com todos os *hosts* e o *storage* de *datastores*. Os serviços básicos do OpenNebula incluem o *daemon* de gerenciamento (`oned`), escalonador (`mm_sched`), servidor de interface web (`sunstone-server`). Outros serviços podem ser instalados e habilitados em instalações regulares do OpenNebula. Além disso, o banco de dados padrão utilizado é o *sqllite*. Caso o ambiente seja de produção pode-se considerar a utilização do *MySQL* que é uma solução mais robusta.

5.5.1.4. Monitoramento

O monitoramento reúne informações relacionadas aos *hosts* e máquinas virtuais, como por exemplo, status do *host*, indicadores de desempenho, status das VMs, quantidade de recursos consumidos. Tais informações são coletadas através da execução de rotinas de coleta de dados disponibilizadas pelo ONE. No processo para a coleta de dados, cada *host* envia os dados monitorados para o *frontend* e este os processa em um modulo dedicado. Este modulo é altamente escalável e é limitado pelo desempenho do servidor executando o *oned* e o servidor de banco de dados.

5.5.1.5. Hosts Virtualizados

Os *hosts* são as máquinas físicas que hospedam as VMs em um ambiente de nuvem usando o OpenNebula. O subsistema de virtualização é responsável por intermediar os virtualizares instalados nos *hosts* e as ações necessárias em cada passo do ciclo de vida de uma VM. Nativamente, o OpenNebula suporta três virtualizadores de código aberto, o KVM, LXD e Firecracker. Idealmente, as configurações dos *hosts* devem ser homogêneas em termos de softwares instalados, usuário administrador *oneadmin*, acessibilidade ao armazenamento e conectividade de rede. Por outro lado, é natural que existam diferentes tipos de *hosts* (com configurações e propósitos diferentes), desta forma, estes podem ser agrupados em aglomerados computacionais. Por exemplos, *cluster* KVM, *cluster* LXD.

5.5.1.6. Armazenamento

A ferramenta OpenNebula utiliza *datastores* para armazenar imagens de disco das VMs. Um *datastore* pode ser um tipo de servidor de armazenamento, por exemplo, um servidores NAS (*Network Attached Storage*) ou SAN (*Storage Area Network*). No geral, os *datastores* precisam estar accessíveis através do *frontend* usando qualquer tecnologia (NAS, SAN ou armazenamento local). Quando uma VM é implantada, sua imagem é copiada para o *host*. Baseado na tecnologia de armazenamento utilizada, isso pode significar uma cópia real, um link simbólico ou a configuração de um volume LVM (*Logical Volume Manager*. O OpenNebula é disponibilizado com 3 classes de *datastores*, descritos anteriormente (de imagens, sistemas e arquivos).

Os tipos de *datastores* de imagem podem ser diferentes, dependendo da tecnologia de armazenamento utilizada. São eles: *Filesystem*, que armazena as imagens em forma de arquivo, divididos em três tipos (SSH, compartilhados e QCOW), LVM e Ceph. Se implantado de forma usual (sem customizações em relação ao armazenamento), o *datastore* de imagens fica localizado no *frontend* usando o tipo *filesystem*, e no momento em que as VMs são implantadas, elas são copiadas para os *hosts* que hospedam os discos da VM criada, usando SSH.

5.5.1.7. Rede

O ONE possui um subsistema de rede que é facilmente customizável e adaptável para integrar os requerimentos do sistema ao *cluster* existente. Recomenda-se ao menos duas redes físicas diferentes, sendo elas:

- Rede de Serviços: é utilizada pelo *frontend* para acessar os *hosts* para gerenciá-los, monitorar os virtualizadores e mover imagens. É recomendável que esta seja uma rede separada da rede das instâncias.
- Rede de Instância: Provisiona a conectividade de rede para as VMs entre diferentes *hosts*.

Quando uma VM é criada, o OpenNebula conecta suas interfaces de rede até o virtualizador com as definições estabelecidas nas redes virtuais. Isso irá permitir que a VM tenha acesso a diferentes redes, sejam elas públicas ou privadas. O gerenciador de nuvens privada suporta quatro modos de redes para criação da conexões virtuais, sendo elas:

- *Bridge*: a máquina virtual é diretamente anexada a uma *bridge* Linux existente no virtualizador.
- VLAN: as redes virtuais são implementadas através das TAGs 802.1Q.
- VXLAN: as redes virtuais implementam VLANs usando o protocolo VXLAN.
- Open vSwitch: Similar ao modo VLAN mas usando Openvswitch ao invés de *bridges* Linux. Também pode ser utilizado com VXLAN.

5.5.1.8. Autenticação

A autenticação na interface de linha de comando do OpenNebula pode ser realizada de quatro formas diferentes. Usuário/Senha criados durante a instalação da ferramenta, chaves SSH, certificados x509 (sendo possível utilizar tanto para acessar a CLI como o Sunstone) e o LDAP, permitindo a utilização da autenticação por meio de um gerenciador de contas centralizado externo ao ONE.

A autenticação do usuário nas suas VMs geralmente é feita através de chaves privadas. Esse recursos deve ser configurado pelo administrador do sistema durante a criação do usuário, no qual ele adiciona a chave pública deste. Além disso, nas *templates* compartilhadas com os usuários, deve estar selecionado a opção de contextualização SSH. Desta forma, quando o usuário criar uma VM, o OpenNebula automaticamente irá inserir a chave publica e este poderá acessar suas VM com segurança.

5.5.1.9. Gerenciamento de recursos

Uma das estratégias mais concisas dentro do OpenNebula em relação a recursos computacionais entregues aos usuários/grupos é a criação de cotas. O sistema de cotas rastreia as informações de utilização de recursos por parte dos usuários/grupos e permite que os administradores limitem o uso destes. Além disso, elas podem ser modificadas constantemente de acordo com a necessidade.

Esse sistema de cotas pode ser feito para usuários, criando a limitação de cotas individuais, ou para grupos, onde a limitação atua como média de uso por todos os usuários do grupo. Finalmente, os recursos que podem ser limitados vão desde *datastores* (quantidade de recursos alocados para cada usuário/grupo em cada *datastore*), computação (incluindo memória, CPU), rede (número de IPs) e imagens (número de imagens). Por padrão o número de recursos provisionado aos usuários é infinito, causando a impressão de que existem recursos ilimitados.

5.5.1.10. Gerenciamento de *templates*

Umas das vantagens que se destacam na utilização de ambientes como nuvens privadas, são as chamadas *templates*. Estes discos pré-configurados facilitam a implantação com configurações homogênea em escala e também o tempo de configuração necessário para novas VMs é significativamente reduzido. No OpenNebula, primeiramente deve-se instanciar ou criar uma VM. Depois disso, a VM pode ser configurada conforme as necessidades de utilização, e por fim, podem ser criadas as *templates* baseadas no disco da VM. Uma vez criadas, elas podem ser instanciadas em escala, possibilitando a criação de um ambiente pré-configurado.

Existem atributos que podem ser configurados pelos administradores em relação as escolhas de recursos das *templates*. Por exemplo, a criação de uma *template* para usuários finais. O dono da *template* pode configurar para que atributos da capacidade de hardware (CPU, memória e discos) sejam usados em determinadas quantidades e se podem ou não ser modificados. Outro ponto importante é a possibilidade de inserir a funcionalidade de sugerir o usuário a inserir certos atributos, ou seja, tornar os atributos pré-estabelecidos como dinâmicos. Desta forma, é necessário modificá-los todas as vezes que a *template* é instanciada. Este cenário torna-se interessante para ambientes com poucos recursos, ou mesmo para o correto dimensionamento das VMs, sem alocar *hardware* físico de forma demasiada.

Além disso, o gerenciamento das *templates* torna o ambiente altamente dinâmico e flexível (premissas clássicas da computação em nuvem), através de suas rotinas, como por exemplo criar, deletar, clonar, atualizar e compartilhar entre usuários. Outra técnica que quando combinada com o uso de *templates* torna o utilização da nuvem ainda melhor é a criação de *snapshots* de discos. Com isso, durante a utilização da VM, o usuário pode realizar um *snapshot*, sendo salvo o atual estado do disco e memória da mesma, possibilitando retornar no futuro a esse estado salvo. Também é possível exportar esse disco e utilizá-lo para novas *templates*.

5.5.2. Instalação

A instalação da ferramenta OpenNebula se inicia por sua parte central, o *frontend*. Nesta máquina será instalada o servidor do software que irá gerenciar todos os recursos computacionais e provisioná-los. A primeira etapa da instalação parte da escolha entre as edições "*Enterprise*"e "*Community*", sendo necessário na primeira destas, a aquisição de uma licença para utilização. (Utilizamos neste minicurso a versão sem assinatura). Uma vez realizada a escolha entre as edições, é necessário adicionar o repositório da ferramenta e atualizar a lista de pacotes disponíveis. Após, é necessário realizar a instalação propriamente dita, utilizando pacotes disponibilizados pelo repositório previamente adicionado. Por fim, é possível iniciar o OpenNebula, sendo disponibilizado o acesso a sua interface gráfica.

A próxima etapa a ser realizada é a instalação e configuração do nodo, que será utilizado como hospedeiro das VMs criadas. Primeiro deve-se escolher quais tipos de virtualização serão utilizados. As tecnologias mais comuns dentro do ambiente OpenNebula são o KVM e LXD. A instalação de ambos é semelhante e parte do mesmo ponto que o *frontend*, onde são adicionados os respectivos repositórios da ferramenta e atualizados a lista de pacotes disponível. A seguir são instalados os pacotes da ferramenta. A próxima etapa é a configuração SSH. Nela, o serviço SSH deve ser configurado para realizar conexões entre todos os *hosts* que compõe o sistema, sejam eles *frontend* ou nodos. Após isso, é necessário configurar a rede e como ela será utilizada. Como descrito anteriormente, a rede pode ser utilizada de várias formas. Aqui, usaremos a rede em forma de *bridge*, criando uma *bridge* Linux da interface de rede dos *hosts* que possui acesso a Internet. Por fim, é necessário registrar o nodo no OpenNebula, para que este possa utilizar os recursos computacionais do *host*. Esta última etapa pode ser realizada pela interface gráfica do ONE, conhecida como *Sunstone*, ou via linha de comando. É importante ressaltar que também é necessário configurar o arquivo "*host*". Este é utilizado pelo sistema operacional para relacionar *hostnames* e endereços IP. O processo de instalação e configuração do OpenNebula, tanto do *frontend* como dos *hosts* nodos está descrito no Github[12].

Uma vez instalados ambos *frontend* e nodo(s), deve-se verificar se os *hosts* encontram-se em status "ON". Após, pode-se realizar o download de imagens KVM/LXD pré-configuradas pela loja da ferramenta. Quando finalizado, as VMs podem ser instanciadas e customizadas de acordo com as preferências. Entretanto, as VMs não possuem uma rede configurada ainda, sendo necessário criar uma rede virtual no OpenNebula.

5.5.3. Executando programas MPI no OpenNebula

Após a configuração e instalação do OpenNebula, o usuário pode logar na interface Web deste, chamada de *Sunstone*. O painel de entrada está dividido nas Figuras 5.7 e 5.8, visualizado logo após o login do usuário, onde são mostrados as VMs, os recursos disponíveis e a porcentagem de utilização de cotas disponibilizadas para o usuário.

Como pode ser visto, o usuário possui 13 VMs criadas no total, 7 em execução e 6 desligadas. Além disso, na parte das cotas, pode-se visualizar a quantidade de recursos utilizada de um total "infinito", que na verdade apenas não esta delimitado pelo

[12]<https://github.com/larcc-group/opennebula-erad2021>

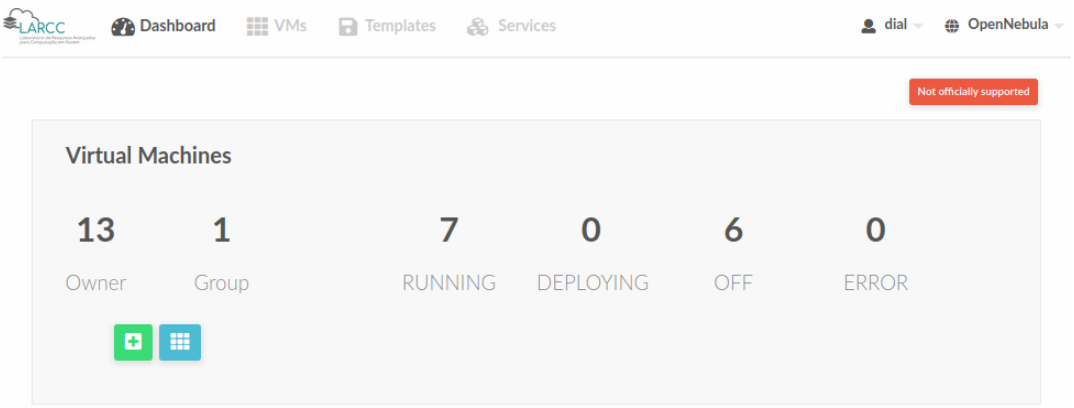


Figura 5.7. Painel de entrada dos usuários OpenNebula.

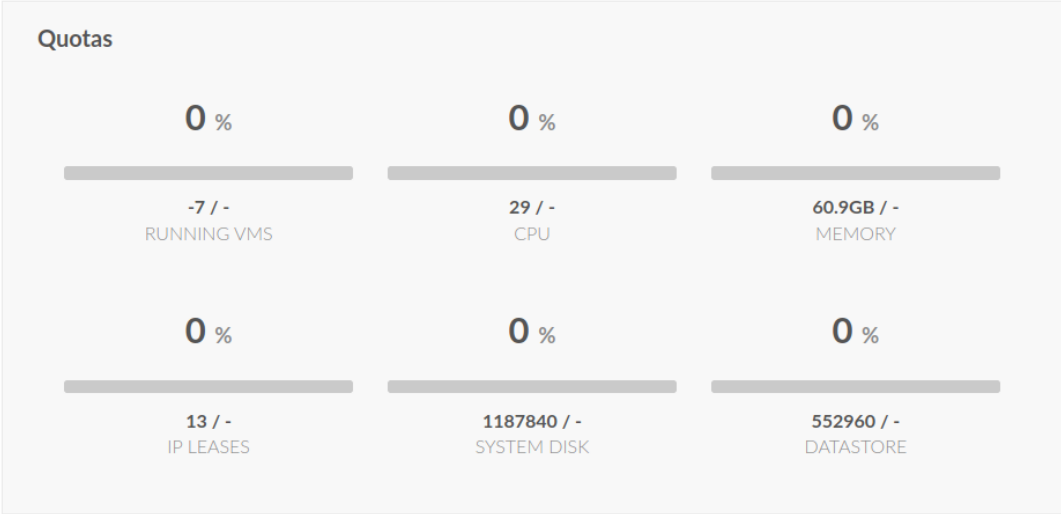


Figura 5.8. Cotas delimitadas pelo administrador da nuvem.

administrador. Agora realizaremos a criação de duas instâncias para demonstrar uma experimentação usando *benchmarks* paralelos que utilizam MPI. O conjunto de *benchmark* utilizado é o conhecido *NAS Parallel Benchmarks* (BAILEY et al., 1991). Para esta experimentação utilizaremos instâncias criadas pelo OpenNebula com o virtualizador KVM, sendo a primeira totalmente configurada e a segunda sendo criada em forma de *template* com base na primeira instância.

O processo para criação de uma VM pode ser feita pelo Sunstone ou por linha de comando. Utilizaremos a interface Web como forma mais intuitiva e simples. Basta clicar no sinal de "+" em verde, visto na Figura 5.7 e selecionar uma *template* que foi disponibilizada pelo administrador durante o processo de instalação (veja a Seção 5.5.2). Feita a criação da primeira VM, podemos fazer o acesso SSH até esta usando o IP disponibilizado. Após, faremos a atualização do sistema e instalação de alguns pacotes.

```
sudo apt update -y sudo apt upgrade -y
sudo apt install make gfortran openmpi-bin libopenmpi-dev \
g++ openssh-server -y
```

Preferencialmente, edita-se o nome do *hostname*, o que facilitará o acesso entre as instâncias. Isso é feito usando os seguintes comandos.

```
vim /etc/hostname
```

Dentro do arquivo *hostname* digitamos o nome `inst1`, e reiniciamos a instância. Após a reinicialização, o sistema já terá o nome que foi definido, e será necessário modificar o arquivo *hosts*, incluindo o IP e nome do *host*.

```
vim /etc/hosts
```

Para realizar a execução de programas paralelos é indicado que se utilize um usuário comum, sendo assim, realizamos a criação do mesmo a seguir.

```
sudo adduser erad
su erad
cd
```

Os campos que serão solicitados pós utilização do primeiro comando acima serão ignorados, apenas prosseguindo usando a tecla *Enter* e por fim `Y` para confirmar a criação do usuário. O segundo comando realizara login e o terceiro irá para a *home* no *user* erad. Agora, é necessário configurar o SSH sem senha entre as instâncias, realizando a criação da chave RSA através do comando:

```
ssh-keygen -t rsa
```

Os campos que serão solicitados pós utilização do comando não necessitam de modificações, apenas prosseguir usando a tecla *Enter* para cria a chave. Como criaremos

uma *template* pronta do ambiente já configurado, podemos liberar o acesso SSH da instância para com ela mesma, inserindo a chave pública no documento de chaves autorizadas com o seguinte comando:

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Apenas no primeiro acesso será necessário aceitar a conexão, a partir da próxima vez, ela será feita de forma automática e sem senha. Agora vamos realizar o download do *benchmark* NAS, logo após a de-compressão, sendo feita da seguinte maneira:

```
cd $HOME
wget https://www.nas.nasa.gov/assets/npb/NPB3.4.1.tar.gz
tar -zxv NPB3.4.1.tar.gz
cd NPB3.4.1/NPB3.4-MPI/
```

Os *benchmarks* serão compilados usando um arquivo de configuração do compilador e outro da suíte, ambos localizados no diretório *config*. Por primeiro realizaremos a copia dos arquivo de configuração.

```
cp config/make.def.template config/make.def
cp config/suite.def.template config/suite.def
```

Após realizaremos uma alterações no arquivo de compilação, sendo a troca do compilador Fortran para `mpifort`. Para realizar essas alteração utilize o comando abaixo:

```
sed -i "s/MPIFC = mpif90/MPIFC = mpifort/g" config/make.def
```

O arquivo da suíte descreve os *benchmarks* e tamanhos de execução (definidos em letras). Por padrão todos os programas estão listados para serem compilados com a classe S, mas iremos modificá-los para uma classe com maior entrada de dados. Para realizar isso, execute os comandos a seguir que realizam a inserção das aplicações junto com a classe A no arquivo `suite.def`.

```
for bench in bt cg ep ft is lu mg sp; do
    for size in A; do
        echo "$bench    $size" >> config/suite.def;
    done;
done
```

Agora realizamos a compilação dos programas, utilizando o comando:

```
make suite
```

Por fim, podemos testar rapidamente um dos programas compilados:

```
mpiexec -np 1 bin/bt.S.x
```

Ao final da execução, que deve acontecer em poucos segundos, será exibido um relatório onde está localizado o tempo de execução da aplicação em questão. Agora podemos criar uma *template* da instância. Para criar, primeiro precisamos nos logar na interface web do OpenNebula, clicar na representação da VM e desligá-la. Após ir no ícone de salvar em verde e aguardar o processo terminar. A última parte deste processo está ilustrada na Figura 5.9. Na próxima tela será necessário nomear a *template* e concluir esta etapa.

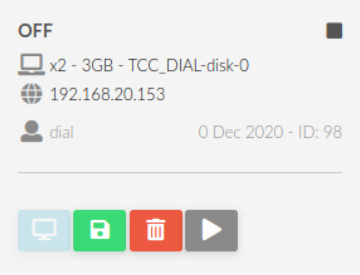


Figura 5.9. Criando *templates* do VMs.

Por fim, podemos religar a instância normalmente, clicando no ícone "*Play*" em cinza na Figura 5.9 e criar uma nova VM pré-configurada usando a *template*. Para realizar a criação da VM, deve-se utilizar o processo já descrito, clicar no ícone "+" da Figura 5.7 e selecionar o nome da *template* salva. Com a nova VM criada, precisamos apenas modificar o nome desta no arquivo *hostname* para inst2, e adicionamos o IP e *hostname* no arquivo *hosts*. É necessário reiniciar a VM.

```
vim /etc/hostname
vim /etc/hosts
```

Ainda, na primeira VM configurada, também é necessário adicionar o IP e *hostname* no arquivo *hosts* da VM criada a partir da *template*.

```
vim /etc/hosts
```

Agora, é necessário realizar o primeiro acesso entre as VMs e após esse acesso, esse processo será realizado sem senha. Assim, através da utilização da *template* para criar a segunda VM (poderiam ser N VMs), criou-se um ambiente computacional totalmente funcional em menos de 5 minutos. Agora podemos executar uma aplicação usando o seguinte comando:

```
mpiexec -np 4 --host inst1,inst1,inst2,inst2 \
NPB3.4.1/NPB3.4-MPI/bin/ft.A.x
```

Desta forma, estamos executando a aplicação FT classe A com 4 processos localizados 2 em cada instância. Ainda poderíamos fazer essa delimitação do numero de processos usando um arquivo:

```
echo "inst1 slots=2" > hostsmpi
echo "ints2 slots=2" >> hostsmpi
```

Assim, podemos modificar o comando de execução para:

```
mpiexec -np 4 --machinefile hostsmpi \
NPB3.4.1/NPB3.4-MPI/bin/ft.A.x
```

5.6. Conclusão

Este minicurso abordou aspectos relacionados a utilização de uma nuvem privada de forma útil e prática. Este ambiente torna-se uma alternativa para o gerenciamento e disponibilização de recursos computacionais de forma rápida e flexível, por meio de ferramentas como as *templates*, que por sua vez, podem criar aglomerados computacionais em poucos minutos. Foram elucidados os principais conceitos metodológicos que regem um ambiente de nuvem criado com a ferramenta OpenNebula, assim como realização da parte prática de instalação, configuração e teste de aplicações paralelas na nuvem. Espera-se que através dos conceitos e passos mostrados neste documento, os participantes possam realizar a implantação da nuvem privada e aplicação dos conceitos abordados.

Agradecimentos

Este trabalho foi realizado com o apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001, o Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), e dos projetos “GREEN-CLOUD: Computação em Cloud com Computação Sustentável"(Nº 16/2551-0000 488-9), da FAPERGS e CNPq Brasil, programa PRONEX 12/2014 e o projeto SPARCLOUD (Nº 17/2551-0000871-5) do Edital Universal MCTIC/CNPq 28/2018. Por fim, os autores agradecem ao Laboratório de Pesquisa Avançada em Computação em Nuvem (LARCC / SETREM, Brasil) por fornecer recursos de computação em nuvem, que contribuíram para a realização deste minicurso. URL: <https://larcc.setrem.com.br>

Referências

BAILEY, D. H. et al. The NAS Parallel Benchmarks; Summary and Preliminary Results. In: *ACM/IEEE Conference on Supercomputing (SC)*. [S.l.: s.n.], 1991.

BHOWMIK, S. *Cloud Computing*. [S.l.]: Cambridge University Press, 2017. ISBN 9781316638101.

BUYYA, R.; VECCHIOLA, C.; SELVI, S. T. *Mastering Cloud Computing: Foundations and Applications Programming*. [S.l.]: Newnes, 2013.

CHANDRASEKARAN, K. *Essentials of Cloud Computing*. [S.l.]: Taylor & Francis, 2014. ISBN 9781482205435.

CHOWDHURY, O. K. C. D. *Mastering OpenStack*. [S.l.]: Packt Publishing Ltd, 2017.

GRIEBLER, D. et al. Performance of Data Mining, Media, and Financial Applications under Private Cloud Conditions. In: *IEEE Symposium on Computers and Communications (ISCC)*. Natal, Brazil: IEEE, 2018.

GROPP, W. D.; LUSK, E.; SKJELLUM, A. *Using MPI: portable parallel programming with the message-passing interface*. [S.l.]: MIT press, 2014.

LXC. *Linux Containers (LXC)*. 2019. Último acesso em dezembro de 2020. Disponível em: <http://linuxcontainers.org/>.

MALISZEWSKI, A. M. et al. Minimizing Communication Overheads in Container-based Clouds for HPC Applications. In: IEEE. *IEEE Symposium on Computers and Communications (ISCC)*. Barcelona, Spain, 2019.

MELL, P.; GRANCE, T. et al. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology (NIST)*, Gaithersburg, United States, 2011.

MILOJIČIĆ, D.; LLORENTE, I.; MONTERO, R. Opennebula: A cloud management tool. *IEEE Internet Computing*, IEEE, 2011.

OPENNEBULA. *OpenNebula 5.12 Documentation*. 2021. Available on: <https://docs.opennebula.io/5.12/index.html>. Access date: 20 March.

ROLOFF, E. et al. High Performance Computing in the Cloud: Deployment, Performance and Cost Efficiency. In: *International Conference on Cloud Computing Technology and Science Proceedings (CloudCom)*. [S.l.: s.n.], 2012.

ROVEDA, D. et al. Analisando a Camada de Gerenciamento das Ferramentas CloudStack e OpenStack para Nuvens Privadas. In: *Escola Regional de Redes de Computadores (ERRC)*. Passo Fundo, Brazil: [s.n.], 2015.

SNIR, M. et al. *MPI-the Complete Reference: the MPI core*. [S.l.]: MIT press, 1998.

VACCA, J. R. *Cloud Computing Security: Foundations and Challenges*. [S.l.]: CRC Press, 2016.

VOGEL, A.; GRIEBLER, D.; FERNANDES, L. G. Providing High-level Self-adaptive Abstractions for Stream Parallelism on Multicores. *Softw Pract Exp*, 2021.

VOGEL, A. et al. Private IaaS Clouds: A Comparative Analysis of OpenNebula, CloudStack and OpenStack. In: *Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. Heraklion Crete, Greece: IEEE, 2016.

VOGEL, A. et al. An Intra-Cloud Networking Performance Evaluation on CloudStack Environment. In: *Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. St. Petersburg, Russia: IEEE, 2017.