UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Departamento de Computação

MIGUEL ANTONIO DE OLIVEIRA

Projeto e Implementação de um Ambiente de Capacitação em Computação em Nuvem com Virtualização

SÃO CARLOS-SP

2024

MIGUEL ANTONIO DE OLIVEIRA

Projeto e Implementação de um Ambiente de Capacitação em Computação em Nuvem com Virtualização

Trabalho de conclusão de curso apresentado ao Departamento de Computação para obtenção do título de Bacharel em Ciência da Computação.

Orientação Prof. Dr. Hélio Crestana Guardia

SÃO CARLOS-SP

2024

Dedico este trabalho aos meus colegas e professores, que deram significado à minha pesquisa, e à minha família, por sempre iluminar a trajetória da vida para mim.

"As máquinas podem ser definidas, classificadas e estudadas em sua evolução de acordo com qualquer critério que se deseje: força motriz, complexidade, utilização de princípios físicos etc. Mas se é obrigado, ao início, a escolher entre dois modos de pensar diferentes. O primeiro é o ponto de vista do engenheiro, que enxerga a tecnologia sobretudo em suas ligações internas e tende a definir a máquina em relação a si mesma, como um fato técnico. O outro é o enfoque social, que vê a tecnologia em suas conexões com a humanidade e define a máquina em relação com o trabalho humano, e como um artefato social."

(Harry Braverman)

Resumo

Plataformas de computação em nuvem baseadas em OpenStack são notoriamente complexas de se implantar, exigindo esforço de operadores aclimados com o processo e suas interações com o ambiente. Em particular, ambientes de capacitação de novos operadores sofrem com uma demanda frequente de novas implantações para uso em experimentação prática. Com isso em mente, é analisado o processo de implantação e técnicas de automação para aplicar nesse, visando simplificar o fluxo de trabalho e aumentar a eficiência de ambientes que exigem reimplantações sem interação humana, com foco na construção de nuvens OpenStack hospedadas sobre outras nuvens. São reunidas considerações sobre o procedimento e a aplicação dessas ferramentas, sugestões de uso futuro, e influências que essas tiveram no estado final do ambiente criado.

Palavras-chave: Computação em Nuvem; Automação; Juju; Nuvem; OpenStack; MAAS.

Abstract

OpenStack-based cloud computing platforms are notoriously complex to deploy, requiring the efforts of operators who are familiar with the process and its interactions with the surrounding environment. In particular, environments designed around the training of new operators suffer from high demand of new deployments for use in practical experimentation. With that in mind, the deployment process, as well as applicable automation techniques, are analyzed, aiming to simplify workflows and increase the efficiency of environments which require redeployment without human interaction, with a particular focus around OpenStack clouds hosted on other cloud platforms. Insights are gathered around the procedure and automation tool usages, future usage suggestions, and how these influenced the built environment's final state.

Keywords: Cloud Computing; Automation; Juju; Cloud; OpenStack; MAAS.

Lista de ilustrações

Figura 1 – Interface gráfica de um controlador MAAS. 6
Figura 2 – Largura de banda resultante por TCP. 12
Figura 3 – Largura de banda resultante por UDP. 12
Figura 4 – Taxa de perda de pacotes por UDP. 12
Figura 5 – Diagrama da topologia de rede do sistema. 13
Figura 6 – Diagrama da configuração de rede do nó controlador. 13
Figura 7 – Diagrama da organização física de rede do sistema. 14
Figura 8 – Fotografia do nó controlador com 8 discos instalados. 15
Figura 9 – Diagrama da organização da nuvem de substrato. 17
Figura 10 – Diagrama da organização da nuvem virtual. 20
Figura 11 – Visão geral de hipervisores da nuvem virtual. 23
Figura 12 – Unidades Juju da nuvem virtual. 23
Figura 13 – Unidades Juju da nuvem substrato. 23

Lista de quadros

Quadro 1 – Declaração de uma instância de OpenStack como um recurso. 7
Quadro 2 – Exemplo de um pacote Juju, com três máquinas e uma aplicação. . . . 18
Quadro 3 – Declaração de uma aplicação Juju como um recurso Terraform. 19

Sumário

1 INTRODUÇÃO 1
1.1 Objetivo 2
1.2 Trabalhos Relacionados 3
1.3 Visão Geral 4

2 FERRAMENTAS UTILIZADAS 5
2.1 Juju 5
2.2 MAAS 6
2.3 Terraform 7

3 DESENVOLVIMENTO 9
3.1 Instalação Física do OpenStack 9
3.1.1 Configuração de Rede 9
3.1.2 Configuração de Armazenamento 14
3.2 Implantação da Nuvem OpenStack de Substrato 16
3.3 Implementação do Ambiente Virtual 17
3.3.1 Automação por Pacotes 18
3.3.2 Declaração de Infraestrutura com o Terraform 19

4 RESULTADOS 21

5 CONCLUSÕES 25
5.1 Próximos Passos 25

REFERÊNCIAS 27

A CONTEÚDO DO ARQUIVO DE *BUNDLE* JUJU 29

B MÓDULOS TERRAFORM (PROVISIONAMENTO MANUAL) . . 35
B.1 Arquivo Raiz do Projeto 35
B.2 Arquivo do Módulo `known_host` 38
B.3 Arquivo do Módulo `await_ssh` 39

1 Introdução

A operação de serviços de computação em nuvem vem mostrando uma tendência de otimizar a eficiência dos seus processos. Com a marcha do custo de equipamento e eletricidade a valores consistentemente menores por unidade de recurso utilizado, os esforços de otimização se direcionam cada vez mais ao componente humano do empreendimento. O resultado é que o aumento de eficiência se torna uma tarefa de maximizar o domínio de cada operador sobre uma quantidade crescente de máquinas no desempenho do seu trabalho.

Na operação de ambientes de computação em nuvem modernos, uma das características de maior impacto nessa eficiência operacional é a agilidade na implantação e no gerenciamento de sistemas. Operadores em busca de agilidade investem recursos na criação de procedimentos capazes de alterar o estado de um sistema com escalabilidade e pouca intervenção manual. Ao mesmo tempo, a infraestrutura de um operador é relativamente dinâmica, com frequentes alterações de equipamento por consequência de falhas, atualizações, e mudança de escala.

Em nuvens baseadas em OpenStack, um dos aspectos mais custosos ao operador é o processo de implantação. Enquanto a implantação de plataformas baseadas em OpenStack desfruta de várias aproximações diferentes e alta personalização, ela também sofre por ter alta complexidade e ser sensível às características do ambiente físico.

Mesmo que a barreira imposta pela implantação pareça ser compensada pela baixa frequência com qual ela é aplicada, isso só é verdade em ambientes regulares de produção. Outros ambientes exigem implantação frequente de nuvens novas para execução de algumas tarefas. O exemplo de ambiente de interesse desse trabalho é a execução de exercícios práticos para capacitação em ambientes de computação em nuvem.

O conceito do ambiente de execução de exercícios práticos consiste em usar uma nuvem recém-criada e carregada com um cenário desejado para apresentar um exercício que explicita algum aspecto para aprendizado. Isso complementa a documentação e exemplos com uma experiência prática, e também pode expor falhas na própria documentação para futuras melhorias.

Em contraste com nuvens feitas para produção, as nuvens desse ambiente têm significativamente menos restrições práticas para sua realização. Elas podem ser simuladas sobre outra plataforma de computação ao invés de implantadas em equipamento físico, o desempenho de seus serviços não é crítico, e não é necessário simular aspectos de baixo nível do equipamento.

Essa combinação das demandas apresentadas e restrições brandas é ideal para prototipação e desenvolvimento de novas técnicas de automação. Uma vez desenvolvidas, essas podem ser estendidas e generalizadas para outros fluxos de trabalho, provendo conhecimento técnico para resolver problemas com aspectos similares.

Em particular, generalizar esses processos de automação para trabalhar em máquinas físicas possibilita a operação simultânea de várias nuvens em produção. Essas técnicas são de interesse para escalabilidade de serviços baseados em OpenStack, dadas as limitações atuais de escala a alguns milhares de máquinas. A ideia de usar múltiplas instâncias de OpenStack para escalar uma única nuvem é referida posteriormente como *múltiplas regiões*.

Além dos benefícios de escala, o uso de múltiplas regiões também oferece características de isolamento altamente desejáveis. Tal isolamento permite executar alterações estruturais a certas regiões sem afetar as outras. Isso, quando unido a recursos que podem ser movidos livremente entre regiões, torna possível trazer regiões inteiras sob manutenção para alterações de estado suscetíveis a erro, como atualizações dos serviços OpenStack, alterações na infraestrutura física, aplicação de configurações experimentais em grande escala, análise e correção de falhas graves, e até reimplantação da região.

Para explorar tanto os aspectos físicos quanto virtuais desse cenário, neste trabalho foi desenvolvida uma implantação manual de uma nuvem OpenStack e foram exploradas técnicas de automação desse processo. Foram aplicadas ferramentas de automação de infraestrutura, com sucesso na execução automática de parte das tarefas necessárias para implantação. Os arquivos de configuração resultantes desse trabalho são aplicáveis a nuvens existentes e, com algumas alterações, a infraestrutura física.

Além disso, este documento descreve como essas contribuições ao ambiente físico estão sendo usadas por outras equipes para pesquisa e desenvolvimento. Em particular, os arquivos de configuração foram agregados a um guia de implantação para simplificar o processo de implantação manual por membros de equipes de infraestrutura futuras.

1.1 Objetivo

Este trabalho busca explorar ferramentas e técnicas de automação e aplicá-las na criação de uma nuvem virtual para uso em capacitação prática com a infraestrutura OpenStack. Isso foi feito sob expectativa que os resultados obtidos possam ser aplicados na elaboração de exercícios práticos suplementares ao material desenvolvido pelo projeto de extensão “Pesquisa e desenvolvimento em tecnologias para data centers utilizando virtualização” em uma parceria entre DC/UFSCar e LuizaLabs.

Por maior flexibilidade, incluindo o caso de generalização apresentado acima, o foco

deste trabalho consistiu em investigar métodos de automação que possam ser montados em cima de várias plataformas de computação. Isso é desejável dado que a disponibilidade dessas plataformas varia dependendo dos recursos disponíveis às equipes responsáveis pela capacitação de operadores. Para isso, foi feita uma implementação do ambiente de execução sobre outra nuvem OpenStack, que serviu como *nuvem substrato* para os componentes do ambiente. Também foram escolhidas tecnologias modulares que possibilitam adaptar o processo de implantação do ambiente para tipos diferentes de nuvens substrato.

Independentemente das suas consequências como obra de pesquisa, este trabalho também dialoga com o projeto de extensão na sua construção de uma nuvem local ao departamento. Como um projeto de pesquisa e desenvolvimento focado em nuvens baseadas em OpenStack, a implantação de uma nuvem e documentação do processo é de interesse dos integrantes do projeto para uso em pesquisas e outros trabalhos. Com isso, também se vê desejável obter experiência sobre a implantação como processo, e a escrita de guias e documentos explicativos.

1.2 Trabalhos Relacionados

Para a seleção de ferramentas de automação, foi feito o uso do estudo de ferramentas de automação em (MATTIOLI, 2023), outro integrante do projeto de extensão. A escolha das ferramentas MAAS e Juju foi feita devido a uma forte integração cruzada entre elas, enquanto a ferramenta Terraform foi escolhida devido a integrações com as outras duas, e uma forte presença de componentes de integração desenvolvidos pela comunidade.

O projeto OpenStack possui um subprojeto chamado DevStack (OPENSTACK, 2024a), desenvolvido para facilitar o desenvolvimento e testes da própria plataforma. A instalação típica é feita em uma máquina virtual na estação de trabalho do desenvolvedor e inclui o conjunto de serviços básicos de administração. Enquanto o projeto pode ser usado para estudar a arquitetura da nuvem, ele não trabalha os aspectos de implantação em múltiplas máquinas, dado que isso cai fora do escopo.

Dentre os outros métodos de implantação desenvolvidos pelo projeto OpenStack para produção, o projeto TripleO (OPENSTACK, 2024c) também exibe propriedades similares ao desenvolvimento feito neste trabalho. Em particular, nuvens implantadas com TripleO consistem de duas instâncias de nuvens OpenStack: uma sub-nuvem (*undercloud*) usa as ferramentas de controle para provisionar e gerenciar uma sobre-nuvem (*overcloud*), que oferece seus recursos aos usuários. Nesse caso, os nós da sobre-nuvem são máquinas físicas, gerenciadas pela ferramenta Ironic, que faz parte do projeto OpenStack.

Em (BERTOLINO et al., 2019), foi feita uma revisão sistemática de publicações que, entre outros tópicos, cobre testes onde uma nuvem é usada tanto como o objeto do teste quanto o ambiente onde o teste é realizado (*Testing of the Cloud in the Cloud*, seção

6.3). Os trabalhos explicitados focam nos aspectos de configuração de ambientes de testes em nuvens públicas já estabelecidas. Por exemplo, em (THIERY et al., 2014) é projetada uma linguagem de domínio específico (*domain-specific language*, DSL) para configuração de ambientes de testes e de produção. As estruturas, propósitos, e flexibilidade de provedor dessa linguagem não são dissimilares às propostas pela linguagem de configuração do projeto Terraform, detalhado na seção 2.3.

O projeto Open Cloud Testbed (ZINK et al., 2021) traz desenvolvimentos similares à nuvem física desenvolvida neste trabalho em maior escala para benefício público. Os recursos desenvolvidos pelo projeto disponibilizam controle de máquinas físicas e aceleradores FPGA sob demanda à comunidade acadêmica para pesquisa e desenvolvimento de novas tecnologias de computação em nuvem. O sistema também é federado ao projeto CloudLab (DUPLYAKIN et al., 2019), que possui objetivos parecidos.

1.3 Visão Geral

Considerando os conhecimentos e as ferramentas que servem de base para o desenvolvimento deste trabalho, as ferramentas de automação usadas durante o desenvolvimento: Juju, MAAS, e Terraform, são abordadas no capítulo 2, onde é descrita uma visão do seu comportamento, seus propósitos, e como esses interagem com o objetivo do trabalho.

O desenvolvimento dos dois estágios, incluindo detalhes, problemas encontrados, e resultados intermediários, é descrito no capítulo 3. A seção 3.1 desenvolve a construção física e configuração de baixo nível do sistema operacional do ambiente que hospedou a nuvem de substrato. A implantação da nuvem de substrato é coberta na seção 3.2, e as ferramentas de automação e sua aplicação na implantação da nuvem virtual são cobertas na seção 3.3.

O estado final do processo de implantação virtual é coberto no capítulo 4, assim como uma descrição das suas características operacionais, e comentários sobre influências feitas pelo substrato e ferramentas de automação nesse estado. Comentários são feitos sobre documentação gerada durante o processo.

As conclusões tomadas sobre o desenvolvimento e os resultados do processo de automação, a documentação gerada, e suas implicações para fora do trabalho, são apresentadas no capítulo 5. Este capítulo também traz consequências do trabalho que já ocorrem no escopo do projeto de capacitação da equipe de infraestrutura e do uso da nuvem de substrato por outras equipes para pesquisa, desenvolvimento e aprendizado.

2 Ferramentas Utilizadas

O desenvolvimento foi dividido em dois estágios. O primeiro estágio consistiu em uma análise exploratória dos componentes e do procedimento de implantação em um conjunto de máquinas físicas. Após isso, o segundo estágio foi composto pela aplicação de técnicas de automação propostas pelo resultado dessa análise, e uma investigação dos seus aspectos e trabalho futuro.

O estágio exploratório teve como propósito informar sobre os passos necessários para uma implantação, assim como seus requisitos e justificativas. Esse conhecimento serviu como fator determinante nas decisões tomadas sobre o planejamento do ambiente, usando o conhecimento da carga de trabalho comum de um serviço de nuvem em um processo de priorização de alvos. O estágio de aplicação trouxe esses resultados para a seleção e aplicação de ferramentas de automação para implantar o ambiente em uma nuvem OpenStack, usada como substrato.

2.1 Juju

Para gerenciar a implantação de um conjunto de serviços, é preciso uma ferramenta capaz de coordenar e abstrair esse procedimento. Essa abstração permite a manipulação dos componentes da nuvem como elementos em um arquivo de configuração ou uma interface de comando. Com isso em mente, os procedimentos de automação foram construídos sobre o sistema Juju [1], uma ferramenta de gerenciamento de ciclo de vida de aplicações de código aberto desenvolvida pela empresa Canonical Group Limited.

O modelo de implantações baseadas em Juju consiste em um conjunto de aplicações, que gerenciam o estado de algum componente de software. Essas aplicações podem ser distribuídas em uma ou mais unidades, cada unidade sendo alocada em uma máquina abstrata. Uma mesma máquina abstrata pode hospedar várias unidades de várias aplicações diferentes simultaneamente.

Cada máquina abstrata possui recursos computacionais e de armazenamento providos por um substrato. Substratos para várias nuvens públicas são suportados, como: Amazon Web Services, Google Cloud Platform, Microsoft Azure, assim como instâncias de nuvens privadas como OpenStack, Canonical MAAS, e em máquinas pré-provisionadas controladas por SSH.

Para a implantação própria do ambiente, foi usado o substrato OpenStack. Além disso, também foi usado o substrato manual quando o provisionamento de máquinas fosse

[1] <https://juju.is/>

gerenciado por outra ferramenta. Isso é necessário dado que o próprio modelo gerencia o provisionamento das suas próprias máquinas nos outros substratos.

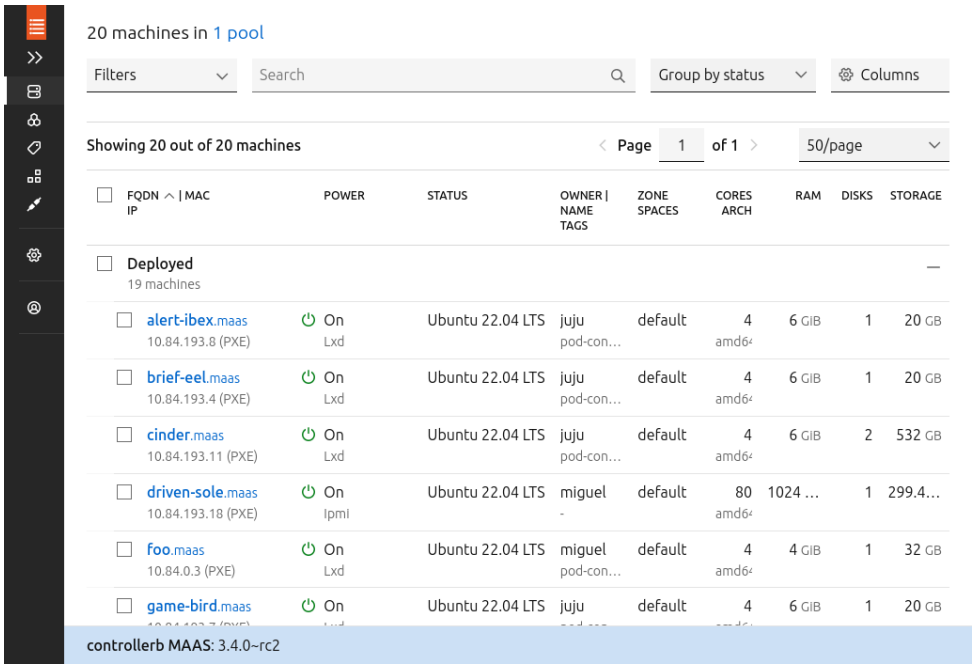
O foco da automação consiste em implantar o projeto *OpenStack-Charms* nos substratos mencionados. O projeto *OpenStack-Charms* é composto por um conjunto de pacotes que montam uma instalação de uma nuvem OpenStack, implantando e gerenciando os serviços necessários como aplicações Juju.

2.2 MAAS

Por compor uma ferramenta de abstração de software, o controlador Juju toma a decisão lógica de também trabalhar com equipamento físico abstrato. Porém, ele próprio não consegue abstrair tal equipamento. Para isso, outro serviço foi usado, o Canonical MAAS[2].

O MAAS consegue provisionar e gerenciar equipamento físico, e máquinas virtuais em cima desse, como um conjunto de máquinas abstratas, simplificando o gerenciamento da infraestrutura. É possível alternar o estado de energia, instalar e desinstalar sistemas operacionais, e configurar aspectos físicos de rede e armazenamento. Essa configuração é feita por uma interface gráfica web hospedada pelo nó controlador.

Figura 1 – Interface gráfica de um controlador MAAS.



Toda infraestrutura física inicia a partir de um conjunto de máquinas. Para chegar a uma nuvem, é preciso que essas máquinas estejam simultaneamente ligadas e ativas à espera por comandos a serem executados. O gerenciamento de energia pode ser feito pelo

[2] <https://maas.io>

protocolo IPMI (INTEL, 2013), e a inicialização de um sistema operacional pode ser feita pela rede por PXE (INTEL, 1999). Isso é feito pelo controlador MAAS.

No estágio exploratório, foi usado o substrato MAAS. A integração MAAS-Juju consegue interagir com esse inventário de máquinas e usar seus recursos para implantar aplicações definidas pelo usuário. Além disso, o substrato MAAS também é útil para a aplicação de técnicas baseadas em Juju para aplicação em produção com múltiplas regiões.

2.3 Terraform

Modelos gerenciados por Juju sofrem com um menor alcance de personalização possível. Isso é devido às abstrações feitas, que permitem configurar aplicações usando campos de configuração que a aplicação própria disponibiliza. Porém, não é possível realizar ações arbitrárias à aplicação sem estender esse comportamento com algum outro componente. Uma dessas extensões suportadas pelo projeto Juju é o seu provedor de Terraform.

HashiCorp Terraform é um programa de infraestrutura como código (*Infrastructure as Code*, IaC) que permite a declaração do estado de uma infraestrutura usando interfaces de gerenciamento. Essa declaração é feita por uma linguagem de configuração baseada em *recursos*. Um recurso é um objeto físico ou lógico com um tempo de vida e atributos definidos pela configuração. Recursos como instâncias, pares de chave, modelos, e aplicações são declarados no arquivo de configuração e subsequentemente construídos e destruídos pelo operador Terraform, sob demanda, através de provedores feitos pela comunidade.

Quadro 1 – Declaração de uma instância de OpenStack como um recurso.

```
resource "openstack_compute_instance_v2" "instance" {
    name = "instance_1"
    image_id = "779050b9-f1cf-4e27-ac34-634c7b172fea"
    flavor_id = "c483caa1-61c1-4b74-b43d-72922d38ec05"
    key_pair = "keypair"

    network {
        uuid = "d416fb20-b785-4449-89c8-a22ecb559900"
    }
}
```

Os recursos coordenados pela ferramenta Juju também podem ser manipulados por Terraform através do provedor `terraform-provider-juju`[3]. Esse provedor possibilita

[3] <https://github.com/juju/terraform-provider-juju>

declarar modelos, máquinas, aplicações, e unidades como recursos Terraform, e aplicar ações de provisionamento sobre eles.

A declaração de máquinas para uso na implantação pode ser feita de duas maneiras. Primeiro, pode ser declarado um modelo usando OpenStack como substrato, onde as máquinas são provisionadas e gerenciadas pelo controlador Juju. Alternativamente, as máquinas podem ser declaradas no arquivo de configuração, usando o provedor `terraform-provider-openstack`[4], e instanciadas no modelo Juju usando o substrato manual. A segunda alternativa foi escolhida devido à integração pelo provedor OpenStack, mas ambas são de interesse.

[4] <https://github.com/terraform-provider-openstack/terraform-provider-openstack>

3 Desenvolvimento

3.1 Instalação Física do OpenStack

Como o componente exploratório da construção do ambiente, foi feita uma instalação de OpenStack em um conjunto de máquinas físicas. O objetivo desse processo é fornecer experiência necessária para fazer decisões apropriadas sobre a implementação do ambiente, e para desenvolver cenários de teste interessantes e dentro do escopo.

O processo de instalação ocorreu inicialmente em três máquinas Dell PowerEdge R900 com 24 núcleos lógicos de processamento e 130 GiB de memória. Essa instalação eventualmente foi refeita em três máquinas Dell PowerEdge R910: uma com 64 núcleos e 128 GiB, e duas com 80 núcleos e 1024 GiB.

Em todas as instalações, um nó foi designado como controlador, enquanto os outros dois foram designados como nós de computação. O nó controlador foi encarregado do gerenciamento dos outros componentes e do roteamento de pacotes entre a rede interna e externa. As tarefas de gerenciamento foram feitas através do nó controlador por SSH e, posteriormente, por uma rede virtual privada baseada em WireGuard.

3.1.1 Configuração de Rede

Durante os primeiros estágios da instalação, a configuração de rede se tornou objeto de experimentação. Um dos objetivos das experimentações com diferentes arranjos no uso das interfaces de rede era saber como explorar de maneira eficiente a existência de várias interfaces na divisão das cargas de múltiplos clientes virtualizados sobre os mesmos nós físicos. Enquanto essa instabilidade gerou resultados interessantes, estes não foram aplicados na configuração final, que se assentou devido a restrições de estabilidade impostas por outras tarefas de instalação.

Os nós foram conectados a um comutador. Além disso, o nó controlador foi conectado à rede externa. Um endereço de IP público foi atribuído ao nó controlador para gerenciamento externo. As conexões físicas entre os nós e o comutador variaram conforme experimentos foram conduzidos. A configuração no nível de software foi feita usando a interface de linha de comando do comutador, assim como a ferramenta de configuração declarativa Netplan[1].

A divisão da rede em redes locais virtuais (*virtual LANs*, VLANs), quando configurada corretamente, gera fortes garantias de isolamento entre pacotes. Isso reduz a

[1] <https://netplan.readthedocs.io/en/stable/>

superfície de ataque da rede e pode melhorar a escalabilidade, dependendo da carga de trabalho.

As ligações ao comutador podem ser isoladas com VLANs de dois modos principais: VLANs "nativas", "não rotuladas", ou "baseadas em porta" atribuem o tráfego em uma porta de rede a uma VLAN. Os pacotes que passam pela porta não possuem nenhuma estrutura especial. VLANs "encapsuladas", "rotuladas", ou "baseadas em rótulo" atribuem o tráfego de uma porta de rede a várias VLANs. Os pacotes que passam pela porta acompanham um rótulo 802.1Q que especifica a qual VLAN o pacote pertence. A interpretação correta do rótulo é responsabilidade do dispositivo conectado ao comutador.

No contexto de nuvens OpenStack, as VLANs podem ser usadas em dois aspectos: para isolar serviços do provedor, ou para isolar tráfego entre inquilinos. O isolamento no nível do provedor atribui um número fixo e pequeno de VLANs, separando tráfego segundo o seu propósito (por exemplo, administração, serviço de armazenamento, comunicação entre inquilinos, tráfego externo). Enquanto isso, o isolamento entre inquilinos consiste em alocar um número grande e variável de VLANs, geralmente uma para cada inquilino. O encapsulamento do tráfego entre instâncias com o ID atribuído é feito automaticamente por uma coordenação entre Neutron, OVN e Open vSwitch.

Um dos problemas com usar VLANs para isolamento entre inquilinos é o limite imposto pelo rótulo 802.1Q, restrito a um número de 12 bits. Com isso, não é possível alocar mais de 4096 redes independentes dentro desse padrão, e outras soluções de encapsulamento tendem a ser usadas.

Foi explorado o particionamento da rede no nível do provedor. Foram feitos testes de configurações de VLANs nativas, onde cada porta do comutador conversa em uma VLAN, quanto encapsulamento 802.1Q, com várias VLANs trafegando através de uma porta, declarando um campo de *tag* no pacote.

Enquanto o encapsulamento não apresentou perdas significativas de desempenho, a aplicação de encapsulamento introduziu certa complexidade à configuração. Como isso colide com a busca por simplicidade inicial na implementação, e devido a uma falta de suporte entre MAAS e uso de encapsulamento com interfaces de ponte, foi decidido usar VLANs nativas para a projeção inicial.

Como outra medida de simplificação, toda a rede local foi colocada em apenas uma VLAN. Essa abertura de isolamento, quando somada à falta de regras de firewall, deixa o sistema vulnerável a acesso por programas executando em instâncias com acesso à rede. Como o sistema não está exposto a executar instâncias de usuários externos, isso não é um problema imediato. Porém, é desejável estudar configurações com isolamento robusto para aplicação em ambientes de produção.

Existem formas de agregar duas ou mais ligações físicas entre dois pares em uma li-

gação lógica. Essa ligação lógica oferece melhor redundância e pode oferecer maior largura de banda, dependendo da carga de trabalho.

Uma questão interessante levantada durante a instalação foi: uma ligação agregada com 802.1Q poderia ter melhor desempenho que várias ligações não rotuladas? A hipótese é que sim, sob suposição de que a largura de banda agregada aumente linearmente com o número de ligações.

Foi testada a largura de banda de algumas configurações diferentes usando o utilitário `iperf`. Os testes foram feitos com TCP e UDP sobre IPv6 em uma agregação de três portas de 1 Gb/s cada, com MTU igual a 9000. Cada configuração foi testada 16 vezes e uma média simples foi tomada.

As Figuras 2 e 3 mostram que a agregação não consegue melhorar a largura de banda de um fluxo único. Ao mesmo tempo, é necessário um número de fluxos muito maior que o número de portas para se aproximar à soma de larguras de banda de cada porta individual, e essa média alcança cerca de 80% da capacidade total com 12 fluxos. Isso é devido a uma decisão da implementação de dispositivos de rede que visa mitigar a entrega de pacotes fora de ordem. Sobre a falta de balanceamento de carga de um único fluxo, Holmes ([2011?]) diz:

> Para a Dell e todos os outros fabricantes de comutadores Ethernet de camada 2, o cálculo da distribuição de tráfego por portas agregadas é baseado em características estáticas do endereço MAC do tráfego. Sem conhecimento da implementação da distribuição de tráfego agregado no comutador Ethernet PowerConnect 5316M (ou qualquer outro comutador Ethernet), um administrador poderia esperar que, conforme o tráfego supera a largura de banda de uma única porta, o tráfego deve transbordar, ou balancear sua carga, à próxima porta agregada. Essa não é a maneira como essa funcionalidade é implementada pela Dell ou qualquer outro fabricante de comutadores Ethernet padronizados. Esse tipo de balanceamento de carga não é possível para grupos de agregação de ligação (*link aggregation groups*, LAGs) baseados puramente em informação da camada 2 devido à natureza estática do cálculo, e do requisito de entregar pacotes da camada 3 em ordem. Tráfego para o mesmo par de endereços de fonte e destino vai sempre viajar na mesma ligação, mesmo se ela estiver superlotada. Não há balanceamento de tráfego.

Esse desempenho sub-ótimo é resultado do processo de decisão feito pelo comutador ao enviar um pacote. Se dois fluxos forem atribuídos a uma mesma porta, a largura de banda de 1 Gb/s precisa ser dividida entre ambos. Isso fica evidente ao observar o grande número de testes de dois fluxos na Figura 2 com largura de banda total de 1 Gb/s. Enquanto o TCP consegue lidar com esse efeito de saturação usando controle de congestionamento, os testes em UDP resultaram em uma alta taxa de perda de pacotes.

Figura 2 – Largura de banda resultante por TCP.

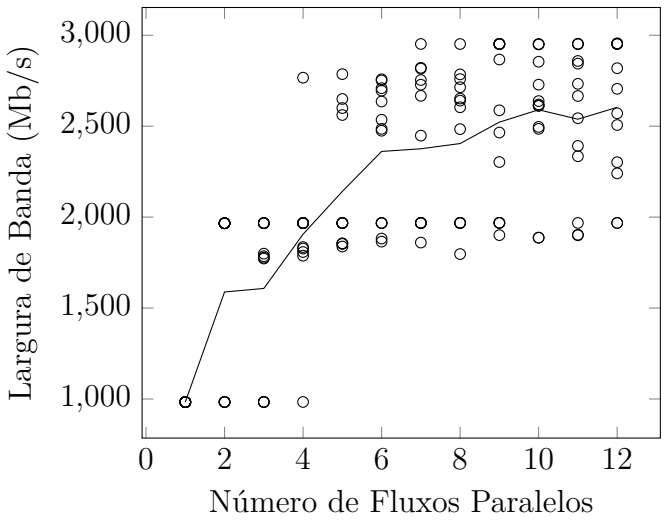


Figura 3 – Largura de banda resultante por UDP.

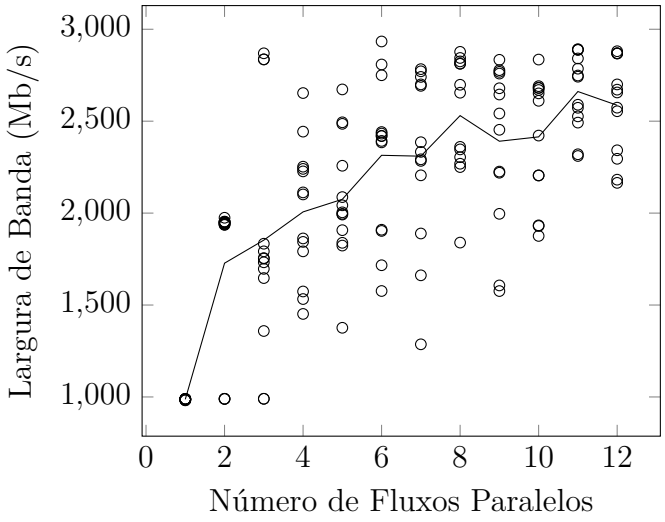
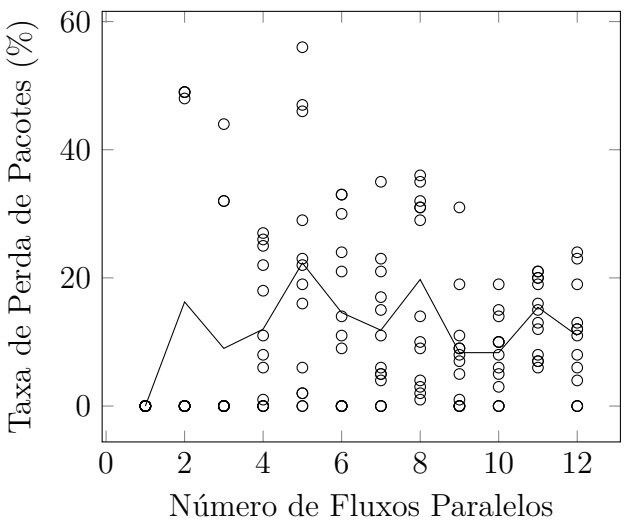


Figura 4 – Taxa de perda de pacotes por UDP.



Concluímos que a decisão de usar agregação pode melhorar o desempenho, mas isso não é sempre verdade. Com isso, é preciso levar as características como intensidade de tráfego da rede, número de fluxos esperado, e congestionamento em consideração antes de agregar duas ligações separadas com esperança de melhor desempenho.

A configuração final de rede usou o nó controlador como roteador entre as sub-redes relevantes do sistema. Foi criada uma sub-rede para comunicação entre nós de computação e máquinas virtuais dos serviços de controle, e outra para alocação de endereços IP para membros do projeto. O acesso à rede local por membros através da rede externa foi feito por WireGuard pelo nó controlador. A figura 5 mostra essa configuração e a figura 6 mostra a configuração interna no nó controlador, incluindo uma interface de ponte. Foi usada a técnica de *Network Address Translation* (NAT) para pacotes direcionados à rede externa roteados pelo nó controlador.

Figura 5 – Diagrama da topologia de rede do sistema.

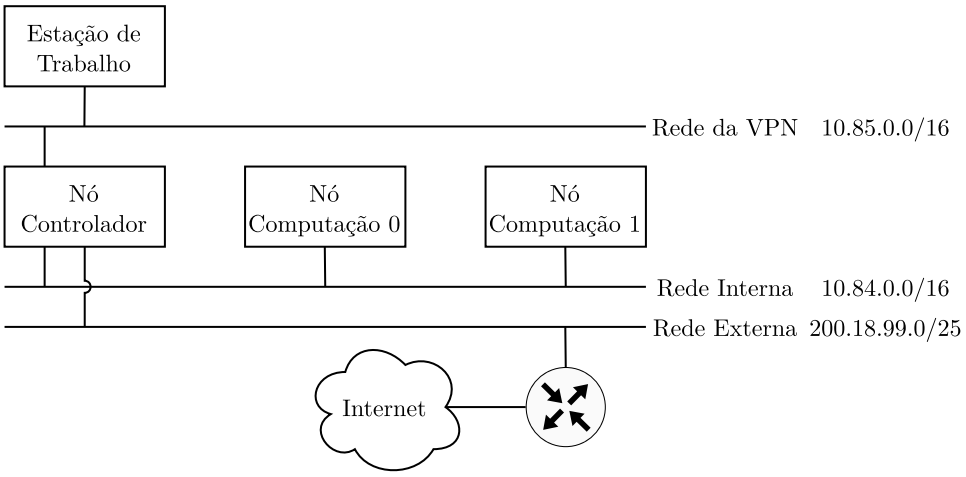
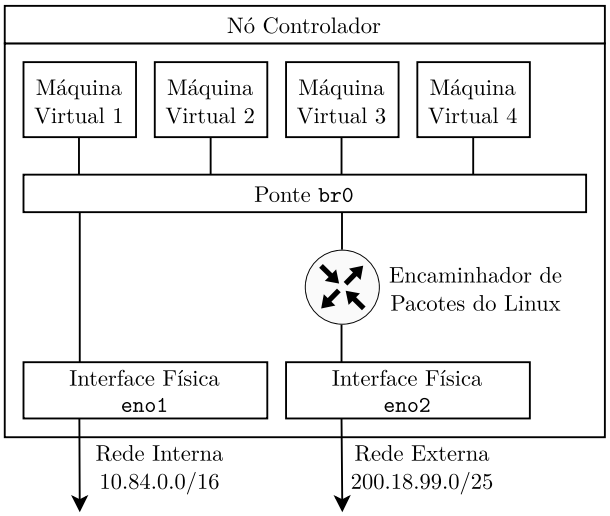
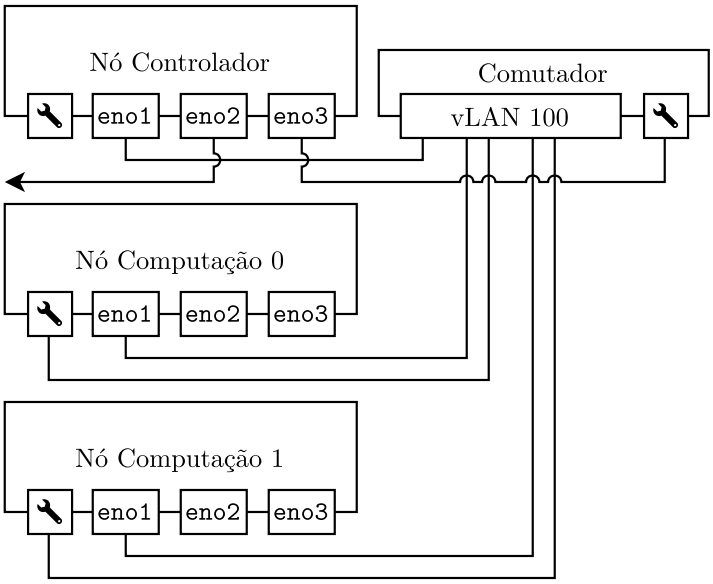


Figura 6 – Diagrama da configuração de rede do nó controlador.



A figura 7 mostra a organização física da rede. A interface `eno1` foi usada para a rede interna em todos os nós, na vLAN de número 100. Além disso, as portas de manutenção dos nós de computação também foram conectadas a ela. A figura também mostra uma conexão entre o controlador e a porta de manutenção do comutador, para configuração remota.

Figura 7 – Diagrama da organização física de rede do sistema.



3.1.2 Configuração de Armazenamento

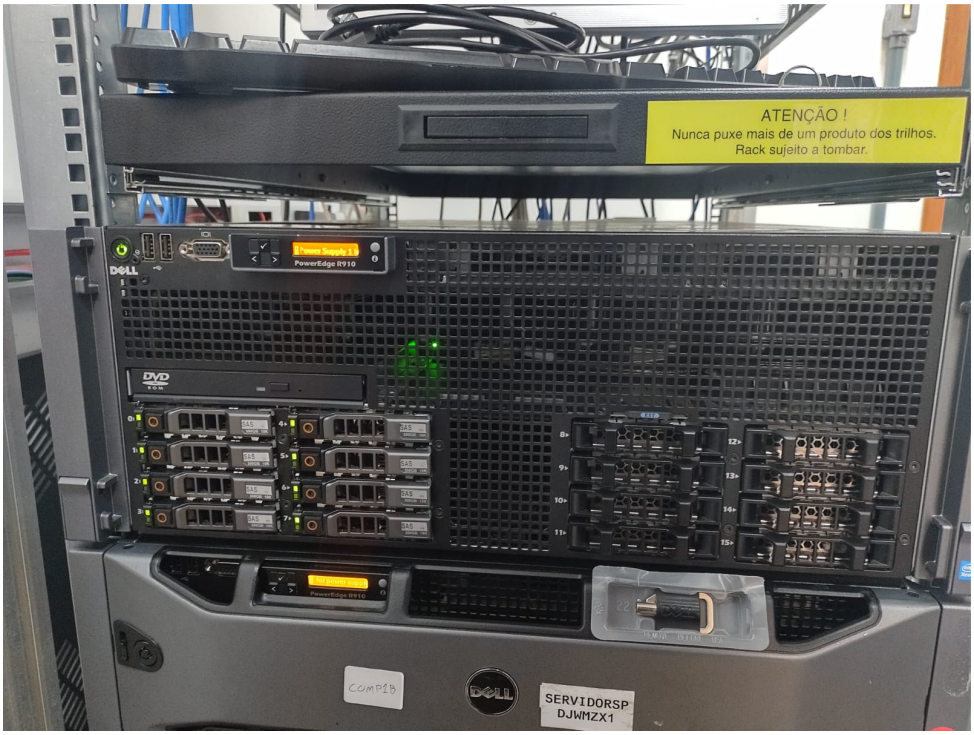
Foram disponibilizados 12 discos com capacidade de 300 GB para a infraestrutura do ambiente físico. Todas as máquinas são equipadas com controladoras para RAID físico, incluindo suporte a cache no modo *write-back* com auxílio de uma bateria.

Inicialmente foram alocados 4 discos por máquina, usando RAID-5, resultando em cerca de 900 GB de capacidade de armazenamento por máquina. Essa configuração foi eventualmente alterada para 8 discos ao nó controlador, em RAID-5, e 2 discos para cada nó de computação, em RAID-1. A configuração nova foi escolhida para melhorar o desempenho de disco no controlador, devido a falhas de implantação causadas por um pico de demanda. Essa nova configuração também disponibiliza mais capacidade aos serviços de armazenamento e ao banco de dados da nuvem.

Os discos foram particionados em um volume de inicialização, e um volume físico do LVM. Em cima desse volume físico, foi criado um volume lógico com o sistema de arquivos Btrfs, montado na raiz do sistema.

A escolha de LVM e Btrfs foi feita pela sua flexibilidade em suportar uma possível alocação de discos adicionais ao controlador sem interromper o serviço da nuvem às

Figura 8 – Fotografia do nó controlador com 8 discos instalados.



pesquisas de outros usuários. Essa disrupção ocorreria dado que não existe redundância de controladores no sistema.

Caso mais discos fossem alocados ao controlador, essa combinação conseguiria absorver a capacidade extra ao sistema de arquivos existente. Para isso, basta adicionar os novos discos ao grupo de volumes, e crescer o tamanho do sistema de arquivos para corresponder à capacidade total.

Além desse caso, o sistema Btrfs também suporta diminuição da capacidade de armazenamento *online*. Com isso, é possível desalocar discos do controlador sem interromper o sistema. Além disso, é possível provisionar um volume lógico "fino" (ou *thin provisioning*), que ocupam apenas o espaço utilizado no volume, independentemente do tamanho declarado. Para isso, é necessário suporte do sistema de arquivos ao descarte de blocos pela instrução TRIM. Esse suporte existe no Btrfs.

Infelizmente, a organização de discos feita pela controladora RAID cria complicações com esse modelo flexível. A adição de um disco a um grupo requer a reconstrução do grupo, e subsequente perda dos dados deste. Ao mesmo tempo, não é possível usar os recursos de RAID de software do LVM, pois é desejável haver redundância na partição de inicialização, e o firmware é incapaz de interpretar o conteúdo de discos construídos com esses recursos.

Durante a realocação de discos dos nós de computação para o controlador, outra questão levantada é se o uso de discos nessas máquinas é realmente necessário. Como

esses nós existem para execução de máquinas virtuais, é plausível uma instalação completamente efêmera para esse propósito. Nesse caso, o cargo de armazenamento de arquivos dos inquilinos seria movido completamente ao serviço de armazenamento de blocos, e nenhum dado persistente seria hospedado no próprio nó.

Essa proposta, se aplicada com sucesso, possui alguns benefícios operacionais. Primeiro, haveria ganho de capacidade geral, previamente perdido devido a mais discos dedicados a prover redundância. Além disso, se elimina qualquer falha de disco em nós de computação, visto que esses discos não existem.

A maior barreiras para implementação é o suporte pelos outros componentes do sistema a uma organização efêmera. Versões mais recentes do projeto MAAS já conseguem implantar sistemas completamente em memória primária, mas não existe suporte pelos modelos de controladores Juju a máquinas que podem desaparecer. Talvez uma solução híbrida seja possível, montando a raiz do sistema pela rede, mas isso requer mais investigação. Finalmente, Dependendo da arquitetura do sistema, do número de discos por nó, e da especialização de tarefas de cada nó, esses ganhos podem não se aplicar.

3.2 Implantação da Nuvem OpenStack de Substrato

A implantação da nuvem de Substrato é o objetivo principal do estágio exploratório. Esta seção explicita o uso das ferramentas MAAS e Juju na instalação do sistema operacional, gerenciamento de máquinas virtuais, e instalação de serviços. O material de referência para isso é o guia de implantação do projeto *OpenStack-Charms* (OPENSTACK, 2024b), com algumas modificações feitas para encaixar no ambiente.

A estrutura final da implantação consiste em dois nós de computação para execução de máquinas virtuais de inquilinos, e onze máquinas virtuais hospedadas no nó controlador para os serviços de controle da nuvem.

O MAAS foi o serviço usado para provisionar os nós de computação, e para gerenciar serviços implantados por máquinas virtuais no nó controlador. Ele foi usado para gerenciar o estado de energia e instalar um sistema operacional. Para isso, os nós foram configurados para receber uma concessão de endereço de IP feita pelo MAAS por DHCP.

Foi instalado o sistema operacional Ubuntu 22.04 LTS nas três máquinas. A instalação no controlador foi manual por um terminal físico, enquanto a instalação nos nós de computação foi gerenciada pela interface web MAAS.

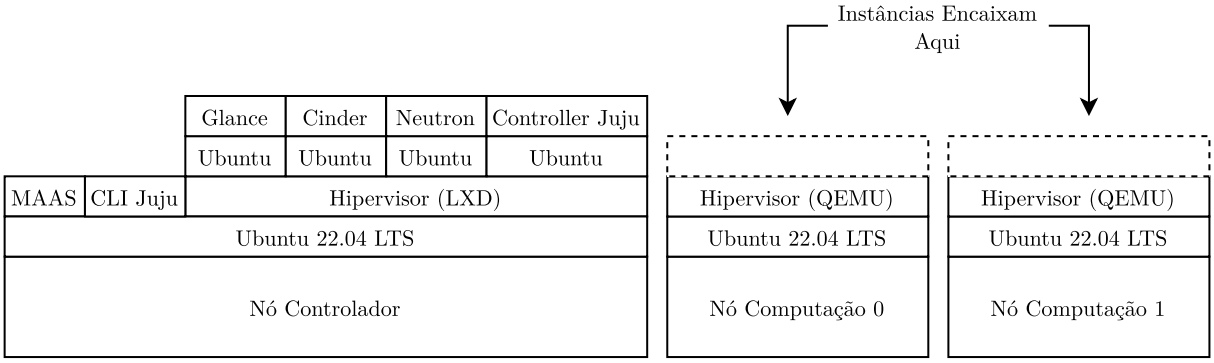
A arquitetura de gerenciamento do sistema MAAS é escalável por ser distribuída. Uma região é controlada por um controlador de região que gerencia vários controladores de *rack*. Cada controlador de *rack* controla um conjunto de máquinas físicas e reporta o estado delas ao controlador de região. Como a infraestrutura presente é de pequena

escala, foram atribuídos ambos os cargos ao nó controlador.

Por uma tecnicalidade na arquitetura de gerenciamento do sistema MAAS, uma máquina definida como controladora não pode gerenciar a si mesma. Isso implicaria na perda de $\frac{1}{3}$ da capacidade computacional do grupo de 3 máquinas. Porém, o MAAS também suporta registrar máquinas virtuais sob um hipervisor ao seu modelo de gerenciamento. Por esse motivo, os serviços da nuvem OpenStack foram implantados sobre máquinas virtuais instanciadas sobre o nó controlador, evitando o desperdício de recursos. Em ambientes de produção, esse problema é de menor tamanho, visto que uma única máquina pode ter seus recursos saturados gerenciando milhares de outras máquinas, e a perda aparente de recursos é uma menor fração do todo.

Com isso em mente, foram usadas máquinas virtuais do controlador para hospedar os serviços de controle da nuvem OpenStack, e máquinas físicas para operar hipervisores do serviço de computação. O uso de máquinas virtuais para segmentar serviços possui benefícios de isolamento, mas também apresenta custos adicionais. Esses custos não foram estudados dado que desempenho não era o foco principal do ambiente, tornando isso é um tópico para trabalho futuro.

Figura 9 – Diagrama da organização da nuvem de substrato.



3.3 Implementação do Ambiente Virtual

O ambiente virtual implementado se baseou nos procedimentos da implementação física para sua realização. A tarefa principal da implementação é encontrar formas automáticas de seguir os passos do guia de implantação do projeto *OpenStack-Charms*, adaptando o processo para o novo ambiente. O alvo dessa etapa é maximizar o número de tarefas do guia que podem ser executadas automaticamente em um substrato OpenStack, e estudar alternativas possíveis para as tarefas restantes.

Para facilitar a automação, foram escolhidas algumas ferramentas que trabalham com o controlador Juju para atingir o estado final do modelo da nuvem OpenStack. Essas

ferramentas, assim como suas características, problemas encontrados, e adaptações feitas, são apresentadas abaixo.

3.3.1 Automação por Pacotes

O controlador Juju possui suporte nativo à implantação de "pacotes" (*bundles*). Esse recurso habilita um usuário a definir um conjunto de aplicações, unidades, máquinas, e integrações entre aplicações, e implantar isso em um modelo. Dada a requisição, o controlador provisiona um conjunto de máquinas que cumpre os requisitos especificados, e implanta as aplicações em cima dessas.

Quadro 2 – Exemplo de um pacote Juju, com três máquinas e uma aplicação.

```
series: jammy
machines:
  '0':
    constraints: mem=6G cores=4 root-disk=20G
  '1':
    constraints: mem=6G cores=4 root-disk=20G
  '2':
    constraints: mem=6G cores=4 root-disk=20G
applications:
  mysql-innodb-cluster:
    charm: ch:mysql-innodb-cluster
    channel: 8.0/stable
    num_units: 3
    to:
    - '0'
    - '1'
    - '2'
```

Como o controlador Juju também suporta usar OpenStack como substrato, é possível implantar um pacote sobre uma nuvem OpenStack. Para isso, foi projetado um pacote replicando o resultado da implantação física, e este foi implantado sobre a nuvem de substrato. Essa aproximação efetivamente criou uma nuvem OpenStack sobre outra nuvem OpenStack.

Um problema com essa aproximação é a necessidade de algumas adaptações para a automação. Em particular, o armazenamento de certificados pelo serviço HashiCorp Vault, como instruído no guia, requer inicialização manual de chaves de criptografia. Para resolver isso, seria necessário um usuário simulado para criar as credenciais necessárias. Esse procedimento também é importante para eventuais adaptações para produção, dado que automatizar isso de maneira segura significaria armazenar essas credenciais em outra instância do serviço Vault, ou delegar o armazenamento de certificados a essa instância.

Hipoteticamente, essa aproximação poderia ser adaptada à implantação diretamente em máquinas físicas, trocando o substrato por uma nuvem baseada em MAAS. Porém, não seria possível aplicar essa técnica ao modelo e ambiente físico presente, devido ao número de máquinas físicas necessárias. A análise da viabilidade dessa aproximação em ambientes de produção e de múltiplas regiões requer trabalho futuro.

3.3.2 Declaração de Infraestrutura com o Terraform

Outra aproximação consiste em declarar a infraestrutura usando a integração entre Terraform, Juju e OpenStack. Essa aproximação é menos modular quanto ao *backend* suportado, dado que é preciso adaptar pelo menos parte do código de configuração aos provedores diferentes de cada substrato, mas também possui benefícios quanto à configurabilidade do sistema em um nível maior.

Foi usado o provedor OpenStack para gerenciar instâncias, pares de chave e endereços de IP para provisionar um conjunto inicial de instâncias. Em cima dessas máquinas, o provedor Juju foi usado para criar um modelo e o conjunto de aplicações que implementam os serviços de OpenStack. A partir disso, é possível alterar o comportamento fino dos serviços usando *scripts* de propósito específico.

Quadro 3 – Declaração de uma aplicação Juju como um recurso Terraform.

```
resource "juju_application" "mysql_innodb_cluster" {
  model = "openstack"
  units = 3
  placement = "0,1,2"

  charm {
    name = "mysql-innodb-cluster"
    channel = "8.0/stable"
  }
}
```

O provedor Juju precisa ter acesso a um controlador existente para gerar um plano de ação. Não é possível criar um controlador pela configuração. Isso é intencional[2]. Por causa disso, um controlador foi manualmente instalado em uma instância provisionada para usar na geração do plano.

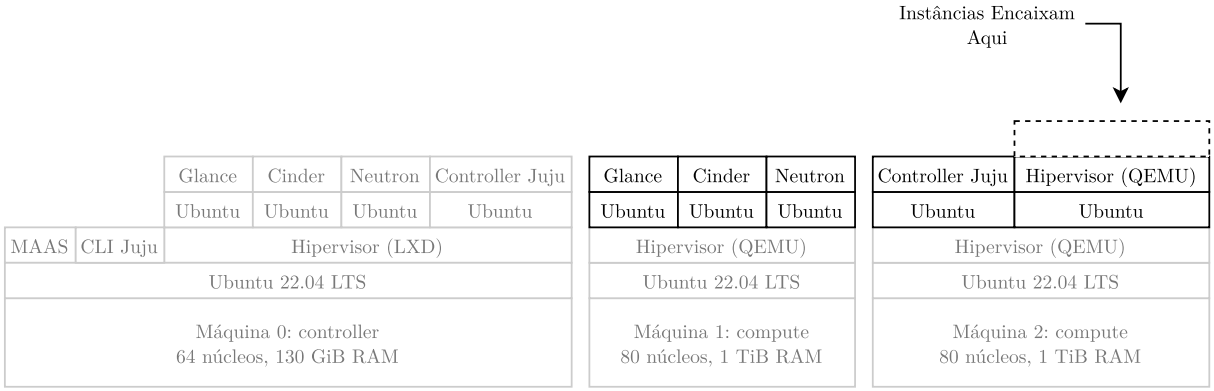
O provisionamento manual foi usado. As máquinas foram declaradas no arquivo de configuração e o endereço IP foi fornecido ao provedor Juju. Um par de chaves SSH foi gerado para acesso, e também foi modificado o arquivo em `~/known_hosts` dinamicamente

[2] <https://github.com/juju/terraform-provider-juju/issues/117>

durante a aplicação da configuração. Isso foi necessário dado que o provedor Juju atualmente não suporta arquivos personalizados de `known_hosts`. Suporte a personalização do caminho desse arquivo foi eventualmente adicionado em uma contribuição[3] feita por membros do projeto de extensão como parte de um *hackathon* promovido pela LuizaLabs em novembro de 2023, mas a alteração foi agendada para ter efeito apenas na próxima versão principal do projeto Juju.

Os principais erros do provisionamento manual resultaram de falhas na lógica do provedor Juju. Por exemplo: uma interrupção na aplicação resultando em uma mesma máquina sendo provisionada duas vezes, inconsistências de atributos que, enquanto logicamente corretas, são consideradas inaceitáveis pelo Terraform[4], e modelos permanecerem mesmo após um pedido de destruição. A frequência de ocorrência desses erros é exacerbada por instabilidades na nuvem substrato.

Figura 10 – Diagrama da organização da nuvem virtual.



[3] <https://github.com/juju/juju/pull/16662>
[4] <https://github.com/juju/terraform-provider-juju/issues/344>

4 Resultados

A nuvem substrato foi implantada várias vezes durante o desenvolvimento das atividades de extensão às quais este trabalho está associado. Esse processo gerou, além do ambiente, recursos de documentação de implantação desse. Os principais documentos concretos do processo são um guia de implantação em desenvolvimento, um guia que descreve acesso à infraestrutura a usuários externos, e um conjunto de *scripts* e arquivos de configuração que facilitam a implantação manual de nuvens temporárias para experimentação no nível físico.

Durante o processo foram encontrados erros diversos de ambiente e configuração. A resolução desses erros foi integrada aos guias de implantação e modificações nos modelos usados. Por exemplo, alguns conflitos entre serviços diferentes no modelo inicial apresentado pelo guia de implantação do projeto *OpenStack-Charms* resultaram na criação de mais máquinas virtuais para isolar esses serviços. Esse isolamento foi feito como uma medida de simplicidade e não considerou o motivo pelo qual o conflito ocorreu. A tarefa de minimizar o número de máquinas virtuais necessárias para um ambiente operacional, assim como a quantidade de recursos alocada ao sistema na sua totalidade, foi deixada para trabalho futuro.

Tanto a implantação pelo método de pacotes quanto pelos provedores Terraform geraram resultados similares, como era esperado.

A implantação do ambiente através do pacote Juju ocorreu de maneira similar à implantação sobre o ambiente físico, desconsiderando o conjunto de erros e falhas de operação particulares ao OpenStack. A inicialização do serviço Vault foi feita manualmente, dado que o arquivo de configuração do pacote não é expressivo o suficiente para isso.

O conteúdo do arquivo YAML que representa o pacote está disponível no apêndice A. O modelo gerado foi inspirado no modelo final feito pelo guia de implantação do projeto *OpenStack-Charms*, com as modificações de isolamento mencionadas acima, e algumas omissões de serviços. Os serviços omitidos foram: `cinder`, por precisar de uma configuração manual de disco no provisionamento, `ceph-osd`, `ceph-mon` e `ceph-radosgw`, por não serem usados neste trabalho, `nova-compute` e `ovn-chassis`, por precisarem de uma configuração manual de rede no provisionamento e por não serem serviços hospedados em máquinas virtuais, dado que foram usadas máquinas físicas para o serviço de computação.

Dado esse arquivo e um modelo vazio hospedado em algum substrato Juju, é possível implantar o conjunto de serviços usando o comando `juju deploy ./bundle.yaml`.

Após isso, é necessário inicializar o serviço Vault manualmente[1], implantar o serviço de computação Nova e, opcionalmente, armazenamento por Cinder. Isso é o suficiente para criar uma nuvem OpenStack básica operacional. O painel de controle é hospedado pela aplicação `openstack-dashboard` por HTTPS e a senha de administração é obtida pelo comando `juju run keystone/leader get-admin-password`.

A implantação baseada em Terraform foi relativamente mais instável. Erros e instabilidades nesta são, em geral, produto de falhas na integração de modelos entre Juju e Terraform, como apontado na subseção 3.3.2. Por causa disso, os melhores modelos implantados com sucesso consistiram de três máquinas hospedando uma instância da aplicação `mysql-innodb-cluster`.

Como consequência dessa instabilidade e de problemas relacionados, foram escritos módulos auxiliares para mitigar os problemas mais evidentes no processo de implantação. O código-fonte da configuração apresentado no apêndice B foi dividido entre esses módulos auxiliares e a configuração principal. O módulo `known_host` presente na seção B.2 gerencia a criação e destruição de linhas no arquivo `~/.ssh/known_hosts` do usuário sob demanda, como mencionado na subseção 3.3.2. Enquanto isso, o módulo `await_ssh` na seção B.3 foi criado para esperar acesso por SSH a instâncias recém-criadas.

[1] Para isso, siga a seção do guia de implantação do serviço Vault em <https://opendev.org/openstack/charm-vault/src/branch/stable/1.8/src/README.md#post-deployment-tasks>.

Figura 11 – Visão geral de hipervisores da nuvem virtual.

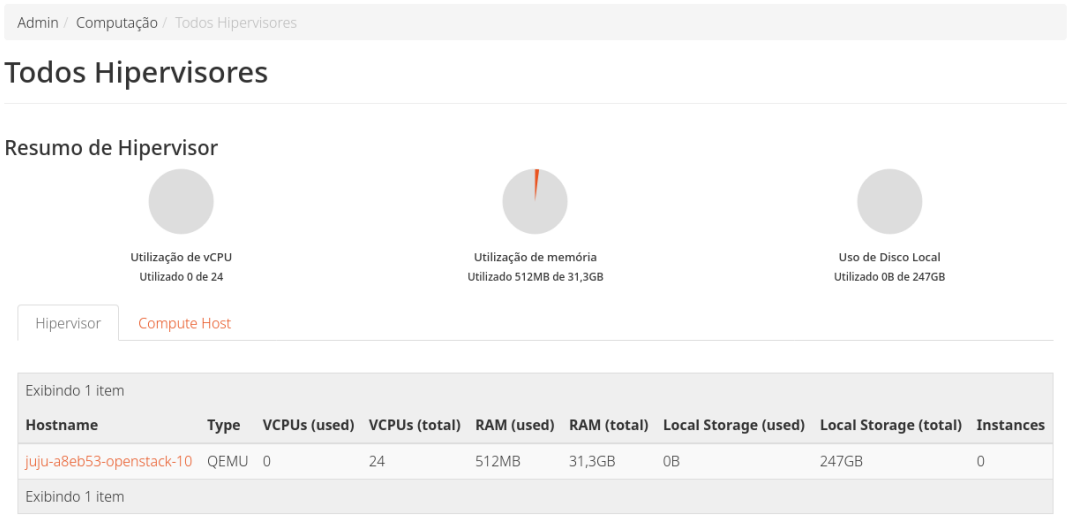


Figura 12 – Unidades Juju da nuvem virtual.

Unit	Workload	Agent	Machine	Public address	Ports	Message
glance/0*	active	idle	3	192.168.0.34	9292/tcp	Unit is ready
glance-mysql-router/0*	active	idle		192.168.0.34		Unit is ready
keystone/0*	active	idle	4	192.168.0.210	5000/tcp	Unit is ready
keystone-mysql-router/0*	active	idle		192.168.0.210		Unit is ready
mysql-innodb-cluster/0	active	idle	0	192.168.0.21		Unit is ready: Mode: R/O, Cluster is ONLINE
mysql-innodb-cluster/1	active	idle	1	192.168.0.12		Unit is ready: Mode: R/O, Cluster is ONLINE
mysql-innodb-cluster/2*	active	idle	2	192.168.0.98		Unit is ready: Mode: R/W, Cluster is ONLINE
neutron-api/0*	active	idle	5	192.168.0.120	9696/tcp	Unit is ready
neutron-api-plugin-ovn/0*	active	idle		192.168.0.120		Unit is ready
neutron-mysql-router/0*	active	idle		192.168.0.120		Unit is ready
nova-cloud-controller/0*	active	idle	7	192.168.0.148	8774-8775/tcp	Unit is ready
nova-mysql-router/0*	active	idle		192.168.0.148		Unit is ready
nova-compute/0*	active	idle	10	192.168.0.213		Unit is ready
ovn-chassis/0*	error	idle		192.168.0.213		hook failed: "ovsdb-relation-joined"
openstack-dashboard/0*	active	idle	8	192.168.0.156	80,443/tcp	Unit is ready
dashboard-mysql-router/0*	active	idle		192.168.0.156		Unit is ready
ovn-central/0	active	idle	0	192.168.0.21	6641-6642/tcp	Unit is ready
ovn-central/1	active	idle	1	192.168.0.12	6641-6642/tcp	Unit is ready (northd: active)
ovn-central/2*	active	idle	2	192.168.0.98	6641-6642/tcp	Unit is ready (leader: ovnnb_db, ovnsb_db)
placement/0*	active	idle	6	192.168.0.172	8778/tcp	Unit is ready
placement-mysql-router/0*	active	idle		192.168.0.172		Unit is ready
rabbitmq-server/0*	active	idle	2	192.168.0.98	5672,15672/tcp	Unit is ready
vault/0*	active	idle	9	192.168.0.53	8200/tcp	Unit is ready (active: true, mlock: enabled)
vault-mysql-router/0*	active	idle		192.168.0.53		Unit is ready

Figura 13 – Unidades Juju da nuvem substrato.

Unit	Workload	Agent	Machine	Public address	Ports	Message
glance/0*	active	idle	3	10.42.0.11	9292/tcp	Unit is ready
glance-mysql-router/0*	active	idle		10.42.0.11		Unit is ready
keystone/0*	active	idle	4	10.42.0.12	5000/tcp	Unit is ready
keystone-mysql-router/0*	active	idle		10.42.0.12		Unit is ready
mysql-innodb-cluster/0*	active	idle	0	10.42.0.6		Unit is ready: Mode: R/W, Cluster is ONLINE
mysql-innodb-cluster/1	active	idle	1	10.42.0.7		Unit is ready: Mode: R/O, Cluster is ONLINE
mysql-innodb-cluster/2	active	idle	2	10.42.0.9		Unit is ready: Mode: R/O, Cluster is ONLINE
neutron-api/0*	active	idle	5	10.42.0.15	9696/tcp	Unit is ready
neutron-api-plugin-ovn/0*	active	idle		10.42.0.15		Unit is ready
neutron-mysql-router/0*	active	idle		10.42.0.15		Unit is ready
nova-cloud-controller/0*	active	idle	7	10.42.0.16	8774-8775/tcp	Unit is ready
nova-mysql-router/0*	active	idle		10.42.0.16		Unit is ready
nova-compute/0*	active	idle	10	10.42.0.3		Unit is ready
ovn-chassis/0*	active	idle		10.42.0.3		Unit is ready
nova-compute/1	active	idle	11	10.42.0.2		Unit is ready
ovn-chassis/1	active	idle		10.42.0.2		Unit is ready
openstack-dashboard/0*	active	idle	8	10.42.0.8	80,443/tcp	Unit is ready
dashboard-mysql-router/0*	active	idle		10.42.0.8		Unit is ready
ovn-central/0*	active	idle	0	10.42.0.6	6641-6642/tcp	Unit is ready (leader: ovnnb_db, ovnsb_db)
ovn-central/1	active	idle	1	10.42.0.7	6641-6642/tcp	Unit is ready
ovn-central/2	active	idle	2	10.42.0.9	6641-6642/tcp	Unit is ready (northd: active)
placement/0*	active	idle	6	10.42.0.13	8778/tcp	Unit is ready
placement-mysql-router/0*	active	idle		10.42.0.13		Unit is ready
rabbitmq-server/0*	active	idle	2	10.42.0.9	5672,15672/tcp	Unit is ready
vault/0*	active	idle	9	10.42.0.14	8200/tcp	Unit is ready (active: true, mlock: enabled)
vault-mysql-router/0*	active	idle		10.42.0.14		Unit is ready

5 Conclusões

A documentação e a experiência agregadas pelo processo exploratório já estão sendo usadas para capacitação da nova equipe de infraestrutura no momento de escrita deste trabalho. O processo de implantação manual em sua totalidade progrediu tanto qualitativamente quanto em aspectos de agilidade.

A nuvem que resultou do processo exploratório, usada posteriormente como substrato, também está sendo usada para pesquisa e desenvolvimento de tópicos relacionados que não são suficientemente destrutivos para requerer uma nuvem dedicada a eles. Ao mesmo tempo, já existem planos para uso da documentação em implantações manuais para outras tarefas de pesquisa.

As implantações automáticas, enquanto menos complexas que desejado inicialmente, também geraram técnicas aplicáveis ao processo manual. Em particular, a implantação usando o pacote de modelo consegue aplicar as alterações necessárias a partir de um estado base com apenas um comando e em um tempo razoável. Além disso, essa simplificação de processo também é refletida na documentação, o que permite ganhos na acessibilidade do processo de implantação para um número maior de usuários interessados. O guia de implantação OpenStack em desenvolvimento pela equipe de infraestrutura do projeto de extensão usa o arquivo de pacote presente no apêndice A para facilitar a implantação manual.

Para a integração de algum método de implantação de ambiente com o resto de um sistema para capacitação, ainda é necessário trabalhar em personalizar o ambiente. Enquanto o uso de pacotes com o controlador Juju não é configurável por si só, pode ser possível compor esse processo em um sistema maior, ou incluir passos de configuração subsequentes. Também é possível adaptar o método baseado em Terraform para essa tarefa, usando o suporte a declarações de sistema para personalização. Para isso, porém, seria necessário mais trabalho para resolver as instabilidades encontradas, e, possivelmente, desenvolver outro provedor Juju com garantias de estabilidade mais próximas às desejadas.

5.1 Próximos Passos

O modelo atual de isolamento entre as máquinas que executam serviços é particularmente ineficiente. Isso é em parte devido à decisão de favorecer simplicidade de implementação sobre eficiência de recursos nos ambientes estudados. Atualmente, os serviços de controle de uma nuvem executam em 10 máquinas virtuais, tomando cerca de

60 GB de memória no total, incluindo cache do sistema de arquivos. Parte desse uso decorre de redundâncias inerentes da virtualização usada, hospedando cópias idênticas do kernel e de objetos compartilhados.

Para melhorar essa eficiência de recursos, seria interessante buscar otimizar o uso do cache nesse ambiente, diminuir o isolamento entre serviços minimizando o número de máquinas virtuais, e investigar a possibilidade de usar contêineres para hospedar esses serviços, incluindo sua capacidade de integração com LXD e MAAS. Além disso, um estudo sobre o uso de memória dos serviços associados, tanto em situações normais como em picos de uso, pode levar a táticas mais refinadas de alocação de recursos a esses serviços.

O uso das técnicas aplicadas pode ser portado à automação do ambiente físico em certa capacidade. Isso é devido ao suporte da implantação por MAAS tanto como substrato Juju quanto como provedor Terraform. Conhecer a viabilidade dessa integração é essencial para a aplicabilidade dessas técnicas em ambientes de produção e implantação de múltiplas regiões. Para isso, também será necessário investigar técnicas desenvolvidas para múltiplas regiões e suas interações com as técnicas de automação.

Uma aproximação de automação que ainda não foi explorada consiste em usar o Provedor Juju e Terraform de maneira híbrida. Ao invés de informar ao modelo Juju uma máquina construída independentemente através do provedor OpenStack, é possível provisionar uma instância diretamente pelo controlador Juju. Com isso, seria possível construir um modelo com estabilidade e suporte similar ao uso de pacotes, mas com uma configuração declarativa e extensível. Cabe a uma investigação futura avaliar se essa técnica é viável e superior a integrações alternativas entre Juju e outra ferramenta.

No aspecto físico, na subseção 3.1.2 foram apresentadas ideias sobre a viabilidade de uma arquitetura de nós de computação sem disco ou com armazenamento remoto do sistema de arquivos raiz. Uma análise mais aprofundada sobre esses aspectos, incluindo aproximações apresentadas na literatura e interoperabilidade com os processos de gerenciamento presente, pode ser proveitosa.

Referências

BERTOLINO, A. et al. A systematic review on cloud testing. *ACM Computing Surveys*, Association for Computing Machinery, New York, NY, USA, v. 52, n. 5, sep 2019. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3331447>. Citado na página 3.

DUPLYAKIN, D. et al. The design and operation of {CloudLab}. In: *2019 USENIX annual technical conference (USENIX ATC 19)*. [S.l.: s.n.], 2019. p. 1–14. Citado na página 4.

HOLMES, B. A. *Link Aggregation on the Dell PowerEdge 1855 Server Ethernet Switch*. [S.l.], [2011?]. Disponível em: <https://web.archive.org/web/20120313024309/http://support.dell.com/support/edocs/network/LAG1855/LAGConsiderationv0.5.pdf>. Acesso em: 19 jan. 2024. Citado na página 11.

INTEL. *Preboot Execution Environment (PXE) Specification Version 2.1*. Santa Clara, California, Estados Unidos, 1999. Disponível em: <https://web.archive.org/web/20131102003141/http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf>. Acesso em: 18 jan. 2024. Citado na página 7.

INTEL. *Intelligent Platform Management Interface Specification Second Generation v2.0*. Santa Clara, California, Estados Unidos, 2013. Disponível em: <https://www.intel.com/content/www/us/en/products/docs/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html>. Acesso em: 18 jan. 2024. Citado na página 7.

MATTIOLI, M. T. *Estudo de ferramentas de automação para computação em nuvem*. Trabalho de Conclusão de Curso (Bacharelado em Ciência de Computação) — Universidade Federal de São Carlos, 2023. Disponível em: <https://repositorio.ufscar.br/handle/ufscar/18507>. Acesso em: 18 jan. 2024. Citado na página 3.

OPENSTACK. *DevStack Documentation*. 2024. Disponível em: <https://docs.openstack.org/devstack/latest/>. Acesso em: 8 fev. 2024. Citado na página 3.

OPENSTACK. *OpenStack Charms Deployment Guide*. 2024. Disponível em: <https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/latest/>. Acesso em: 8 fev. 2024. Citado na página 16.

OPENSTACK. *TripleO Documentation*. 2024. Disponível em: <https://docs.openstack.org/developer/tripleo-docs/>. Acesso em: 8 fev. 2024. Citado na página 3.

THIERY, A. et al. A dsl for deployment and testing in the cloud. In: IEEE. *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*. [S.l.], 2014. p. 376–382. Citado na página 4.

ZINK, M. et al. The open cloud testbed (oct): A platform for research into new cloud technologies. In: IEEE. *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*. [S.l.], 2021. p. 140–147. Citado na página 4.

A Conteúdo do arquivo de *bundle* Juju

```
series: jammy

machines:
  '0': # mysql 1, ovn-central 1
    constraints: mem=6G cores=4 root-disk=20G
  '1': # mysql 2, ovn-central 2
    constraints: mem=6G cores=4 root-disk=20G
  '2': # mysql 3, ovn-central 3
    constraints: mem=6G cores=4 root-disk=20G
  '3': # glance
    constraints: mem=6G cores=4 root-disk=20G
  '4': # keystone
    constraints: mem=6G cores=4 root-disk=20G
  '5': # neutron-api
    constraints: mem=6G cores=4 root-disk=20G
  '6': # placement
    constraints: mem=6G cores=4 root-disk=20G
  '7': # nova-cloud-controller
    constraints: mem=6G cores=4 root-disk=20G
  '8': # dashboard
    constraints: mem=6G cores=4 root-disk=20G
  '9': # vault
    constraints: mem=6G cores=4 root-disk=20G

relations:
- - nova-cloud-controller:identity-service
  - keystone:identity-service
- - glance:identity-service
  - keystone:identity-service
- - neutron-api:identity-service
  - keystone:identity-service
- - neutron-api:amqp
  - rabbitmq-server:amqp
- - glance:amqp
  - rabbitmq-server:amqp
```

```
36 - - nova-cloud-controller:image-service
37   - glance:image-service
38 - - nova-cloud-controller:amqp
39   - rabbitmq-server:amqp
40 - - openstack-dashboard:identity-service
41   - keystone:identity-service
42 - - nova-cloud-controller:neutron-api
43   - neutron-api:neutron-api
44 - - placement:identity-service
45   - keystone:identity-service
46 - - placement:placement
47   - nova-cloud-controller:placement
48 - - keystone:shared-db
49   - keystone-mysql-router:shared-db
50 - - glance:shared-db
51   - glance-mysql-router:shared-db
52 - - nova-cloud-controller:shared-db
53   - nova-mysql-router:shared-db
54 - - neutron-api:shared-db
55   - neutron-mysql-router:shared-db
56 - - openstack-dashboard:shared-db
57   - dashboard-mysql-router:shared-db
58 - - placement:shared-db
59   - placement-mysql-router:shared-db
60 - - vault:shared-db
61   - vault-mysql-router:shared-db
62 - - keystone-mysql-router:db-router
63   - mysql-innodb-cluster:db-router
64 - - nova-mysql-router:db-router
65   - mysql-innodb-cluster:db-router
66 - - glance-mysql-router:db-router
67   - mysql-innodb-cluster:db-router
68 - - neutron-mysql-router:db-router
69   - mysql-innodb-cluster:db-router
70 - - dashboard-mysql-router:db-router
71   - mysql-innodb-cluster:db-router
72 - - placement-mysql-router:db-router
73   - mysql-innodb-cluster:db-router
74 - - vault-mysql-router:db-router
```

```
75    - mysql-innodb-cluster:db-router
76  - - neutron-api-plugin-ovn:neutron-plugin
77    - neutron-api:neutron-plugin-api-subordinate
78  - - ovn-central:certificates
79    - vault:certificates
80  - - ovn-central:ovsdb-cms
81    - neutron-api-plugin-ovn:ovsdb-cms
82  - - neutron-api:certificates
83    - vault:certificates
84  - - vault:certificates
85    - neutron-api-plugin-ovn:certificates
86  - - vault:certificates
87    - glance:certificates
88  - - vault:certificates
89    - keystone:certificates
90  - - vault:certificates
91    - nova-cloud-controller:certificates
92  - - vault:certificates
93    - openstack-dashboard:certificates
94  - - vault:certificates
95    - placement:certificates
96  - - vault:certificates
97    - mysql-innodb-cluster:certificates
98
99  applications:
100   glance-mysql-router:
101     charm: ch:mysql-router
102     channel: 8.0/stable
103   glance:
104     charm: ch:glance
105     channel: 2023.1/stable
106     num_units: 1
107     to:
108     - '3'
109   keystone-mysql-router:
110     charm: ch:mysql-router
111     channel: 8.0/stable
112   keystone:
113     charm: ch:keystone
```

```
114     channel: 2023.1/stable
115     num_units: 1
116     to:
117     - '4'
118   neutron-mysql-router:
119     charm: ch:mysql-router
120     channel: 8.0/stable
121   neutron-api-plugin-ovn:
122     charm: ch:neutron-api-plugin-ovn
123     channel: 2023.1/stable
124   neutron-api:
125     charm: ch:neutron-api
126     channel: 2023.1/stable
127     num_units: 1
128     options:
129       neutron-security-groups: true
130       flat-network-providers: physnet1
131     to:
132     - '5'
133   placement-mysql-router:
134     charm: ch:mysql-router
135     channel: 8.0/stable
136   placement:
137     charm: ch:placement
138     channel: 2023.1/stable
139     num_units: 1
140     to:
141     - '6'
142   nova-mysql-router:
143     charm: ch:mysql-router
144     channel: 8.0/stable
145   nova-cloud-controller:
146     charm: ch:nova-cloud-controller
147     channel: 2023.1/stable
148     num_units: 1
149     options:
150       network-manager: Neutron
151     to:
152     - '7'
```

```
  dashboard-mysql-router:
    charm: ch:mysql-router
    channel: 8.0/stable
  openstack-dashboard:
    charm: ch:openstack-dashboard
    channel: 2023.1/stable
    num_units: 1
    to:
    - '8'
  rabbitmq-server:
    charm: ch:rabbitmq-server
    channel: 3.9/stable
    num_units: 1
    to:
    - '2'
  mysql-innodb-cluster:
    charm: ch:mysql-innodb-cluster
    channel: 8.0/stable
    num_units: 3
    to:
    - '0'
    - '1'
    - '2'
  ovn-central:
    charm: ch:ovn-central
    channel: 23.03/stable
    num_units: 3
    to:
    - '0'
    - '1'
    - '2'
  vault-mysql-router:
    charm: ch:mysql-router
    channel: 8.0/stable
  vault:
    charm: ch:vault
    channel: 1.8/stable
    num_units: 1
    to:
```

```
192         - '9'
```

B Módulos Terraform (provisionamento manual)

B.1 Arquivo Raiz do Projeto

```
terraform {
  required_providers {
    openstack = {
      source = "terraform-provider-openstack/openstack"
      version = "~> 1.53.0"
    }

    juju = {
      source = "registry.terraform.io/juju/juju"
      version = "0.11.0"
    }

    random = {
      source = "hashicorp/random"
      version = "3.5.1"
    }
  }
}

provider "openstack" {
  auth_url = "https://10.84.193.6:5000/v3"
  user_domain_name = "cloud_on_cloud"
  user_name = "user"
  password = "password"
  region = "RegionOne"
  cacert_file = "./cacert.pem"
}

provider "juju" { }

variable "instance_count" {
```

```
32   type = number
33 }
34
35 resource "openstack_compute_keypair_v2" "kp" {
36   name = "kp"
37   public_key = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCChOLON/zrxY/kjAsb
     ↪ zyBe4Vcym2Y2eFLxzPxtQ4l6qsOGK8OCNsNhtaHsyv/kdDkS67vex2xEiaSuin1xLn
     ↪ vHEiX8VPDPOfF209ti9xMnL6BGd7aV4ZtoRktBLo2XvMB2dzMRahwDCHJ/5VD2Sdvy
     ↪ y2TDJvgft5JznkBNNTYzM+QX6oGFK51183tf/Rh0uU1robcme2H97ImeIQuyn4hHG7
     ↪ RqIMxRxVv0ZRFmqiqH8AwDbu3VgEqG834oXEr49L+3TCbyjpEjtv/JLslxKNIvY8I9
     ↪ TpNb7Yo9RYhKXOLCgyIwhicU8aD+eSgu1xrx6DRfUTS0r1qis3kgLG/i+L5vMK1DVc
     ↪ dq6nbelS8f0cBc7mwp6ahAp/Ly/Jo6uHncZsq2P1lf8Xd+DeeiixDRITCxHwj8/gNg
     ↪ 5Lf0Dgt4jsmytJYARnJ6n38kQE2wLP2zUJpKFqZNu+VDT5dx8VD1KnsOPY5z61iGwO
     ↪ Ny5uMFJcJl9sazBOj7/8P4l10vW6Bu1c8=
     ↪ miguel@archlinux"
38 }
39
40 resource "tls_private_key" "instance_host_key" {
41   count = var.instance_count
42   algorithm = "ED25519"
43 }
44
45 resource "openstack_networking_floatingip_v2" "fip" {
46   count = var.instance_count
47   pool = "ext_net"
48 }
49
50 resource "openstack_compute_instance_v2" "instance" {
51   count = var.instance_count
52   name = "instance_${count.index}"
53   image_id = "44238452-81a8-4b80-9604-78424b78cbea"
54   flavor_id = "c483caa1-61c1-4b74-b43d-72922d38ec05"
55   key_pair = openstack_compute_keypair_v2.kp.name
56
57   user_data = templatefile("./instance.yaml", {
58     rsa_host_private = replace(tls_private_key.instance_host_key[count.i
       ↪ ndex].private_key_openssh, "\n",
       ↪ "\n\n")
59     rsa_client_public = file("./id_rsa.pub")
```

```
60    })
61
62    network {
63      uuid = "d416fb20-b785-4449-89c8-a22ecb559900"
64    }
65  }
66
67  resource "openstack_compute_floatingip_associate_v2" "ip_assoc" {
68    count = var.instance_count
69    floating_ip =
      ↪  openstack_networking_floatingip_v2.fip[count.index].address
70    instance_id = openstack_compute_instance_v2.instance[count.index].id
71  }
72
73  module "instance_await_ssh" {
74    count = var.instance_count
75    source = "./await_ssh"
76    host = openstack_compute_floatingip_associate_v2.ip_assoc[count.index]
      ↪  .floating_ip
77    identity = "./id_rsa"
78  }
79
80  module "instance_known_host" {
81    count = var.instance_count
82    source = "./known_host"
83    host = openstack_compute_floatingip_associate_v2.ip_assoc[count.index]
      ↪  .floating_ip
84    key = tls_private_key.instance_host_key[count.index].public_key_openssh
85  }
86
87  resource "juju_model" "model" {
88    name = "mymodel"
89  }
90
91  resource "juju_machine" "juju_machine" {
92    count = var.instance_count
93    depends_on = [ module.instance_await_ssh, module.instance_known_host ]
94    model = juju_model.model.name
```

```
95    ssh_address = "ubuntu@${openstack_compute_floatingip_associate_v2.ip_a
      ↪  ssoc[count.index].floating_ip}"
96    private_key_file = "./id_rsa"
97    public_key_file = "./id_rsa.pub"
98    name = "juju_machine_${replace(openstack_compute_floatingip_associate_
      ↪  v2.ip_assoc[count.index].floating_ip, ".",
      ↪  "_")}"
99  }
100
101 resource "juju_application" "mysql_innodb_cluster" {
102   model = juju_model.model.name
103   units = 3
104   placement = join(",", sort([
105     juju_machine.juju_machine[0].machine_id,
106     juju_machine.juju_machine[1].machine_id,
107     juju_machine.juju_machine[2].machine_id,
108   ]))
109
110   charm {
111     name = "mysql-innodb-cluster"
112     channel = "8.0/stable"
113   }
114 }
```

B.2 Arquivo do Módulo known_host

```
1  variable "host" {
2    type = string
3    nullable = false
4  }
5
6  variable "key" {
7    type = string
8    nullable = false
9  }
10
11 resource "random_uuid" "host_uuid" {
12   provisioner "local-exec" {
13     when = create
```

```
    command = <<EOT
      umask 177;
      flock $HOME/.ssh/known_hosts sh -c "
        TMP=\$(mktemp \$HOME/.ssh/known_hosts.XXXXXXXXXX);
        cp \$HOME/.ssh/known_hosts \$TMP;
        echo '${var.host} ${replace(var.key, "\n", " ")} # Added by a
        ↪  terraform module. ${self.result}' >> \$TMP;
        mv -f --no-copy \$TMP \$HOME/.ssh/known_hosts;
      ";
    EOT
  }

  provisioner "local-exec" {
    when = destroy
    command = <<EOT
      umask 177;
      flock $HOME/.ssh/known_hosts sh -c "
        TMP=\$(mktemp \$HOME/.ssh/known_hosts.XXXXXXXXXX);
        grep -v "${self.result}" \$HOME/.ssh/known_hosts > \$TMP;
        mv -f --no-copy \$TMP \$HOME/.ssh/known_hosts;
      ";
    EOT
  }
}
```

B.3 Arquivo do Módulo `await_ssh`

```
variable "host" {
  type = string
  nullable = false
}

variable "identity" {
  type = string
  nullable = false
}

resource "terraform_data" "awaiter" {
  provisioner "local-exec" {
```

```
13      command = <<EOT
14        while ! nc -zw 5 ${var.host} 22; do sleep 1; done
15        while ! ssh -i ${var.identity} ubuntu@${var.host} true; do sleep
          ↪  5; done
16      EOT
17    }
18  }
```