

ROTEIRO 3

Alunos: Bernardo Cunha Capoferri e Lívia Sayuri Makuta.

QUESTÕES-1

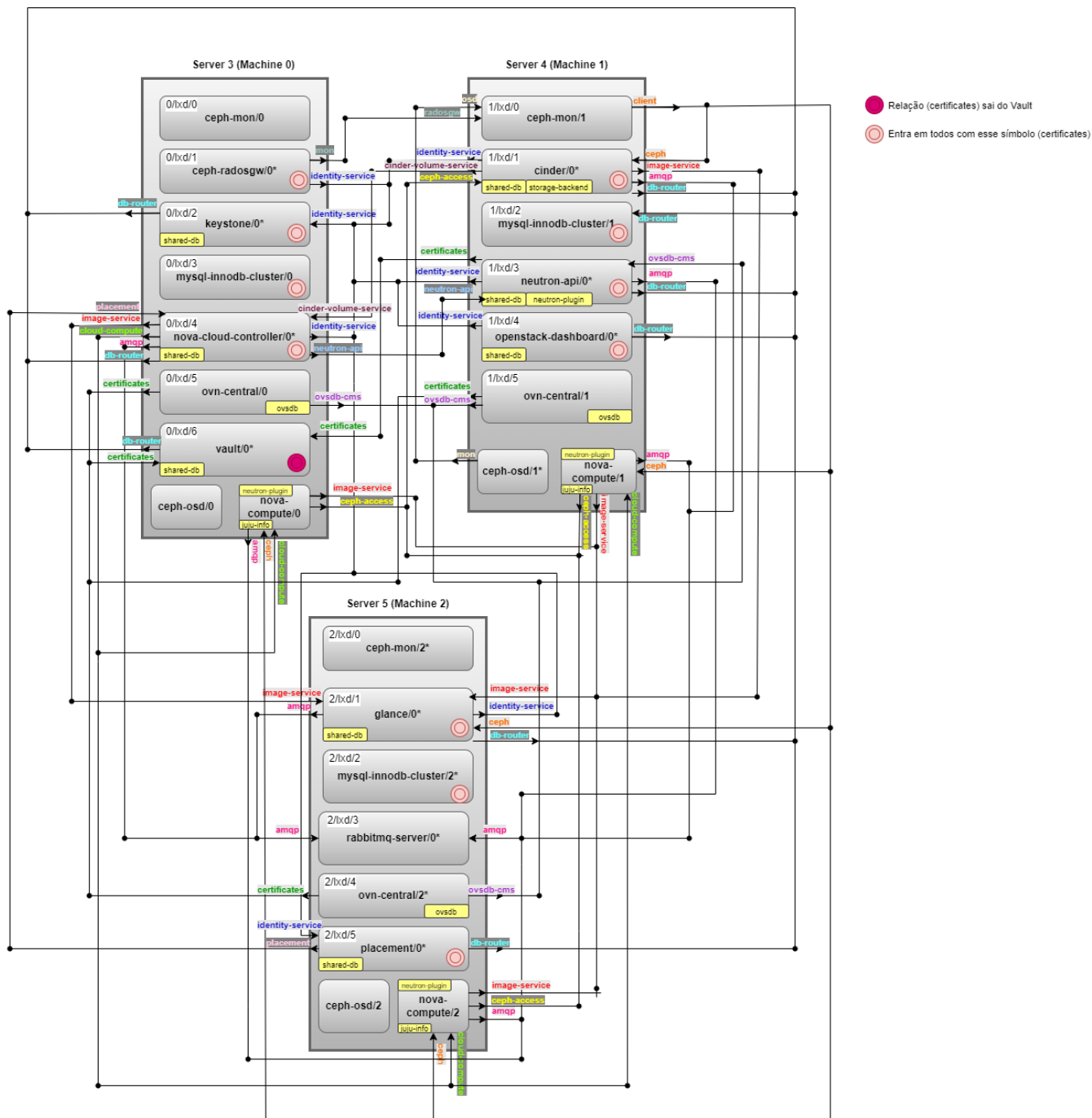
1. Conforme visto no arquivo de instalação, os serviços são instalados usando LXD nos nodes. O que é LXD?

O LXD é um sistema de container que também pode ser usado como um gerenciador de máquina virtual. O LXD faz uso do LXC que é um método de nível de sistema operacional usado para rodar várias instâncias isoladas (containers) de sistemas de Linux em um único host de controle. Lembra um pouco a ideia do Docker, mas esse é mais especializado na implantação de aplicativos.

Inclusive o LXD e o Docker são complementares, de tal forma que o Docker pode ser rodado e gerenciado dentro de containers de máquinas LXD. A diferença é que o Docker hospeda contêineres de aplicação (encapsula um app e sua identidade) enquanto o LXD hospeda contêineres de máquina, agindo como se fosse uma máquina virtual de Linux.

2. Baseado na distribuição do bundle.yaml e no juju status, faça um desenho de como é a sua arquitetura de solução:

- Coloque um retângulo grande para cada máquina física;
- Coloque um retângulo para cada lxd (container) dentro da respectiva máquina.
- Dentro de cada retângulo, escreva o nome do serviço.
- Realize a ligação entre os serviços baseando-se nas relações.



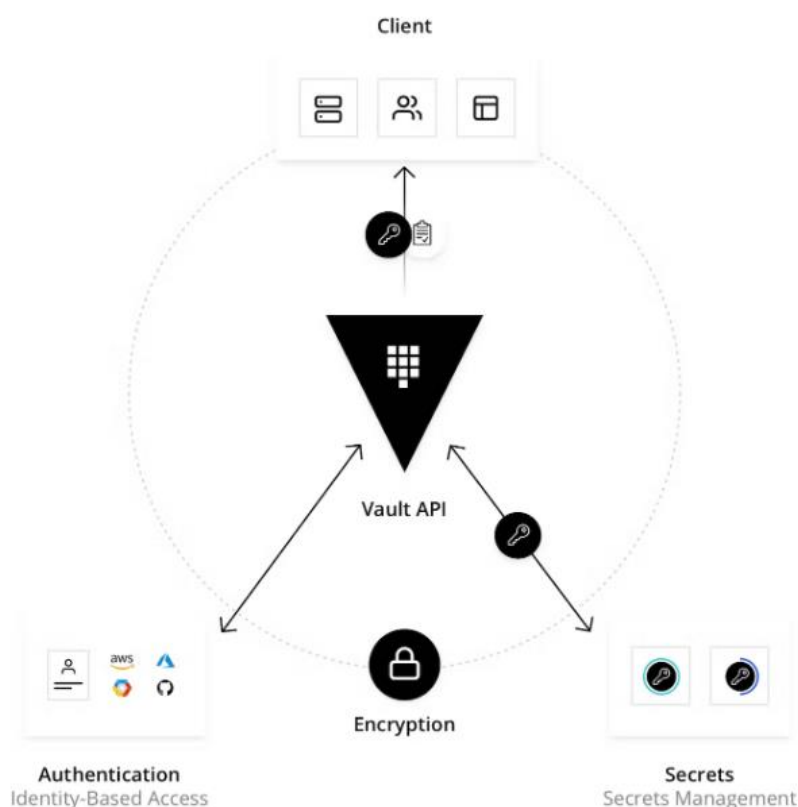
3. Vault é um servidor de secrets. Como ele funciona e para o que ele é usado?

O Vault é um servidor de secrets que protege, guarda e controla rigidamente o acesso a tokens, senhas, certificados e chaves API, dessa forma ele é uma ferramenta para acessar secrets de maneira segura. Assim, ele garante uma interface unificada para qualquer serviço e controla rigidamente o acesso aos secrets por meio de métodos de autenticação e autorização, além de manter um log detalhado e auditável.

E isso é de extrema importância já que um sistema moderno hoje em dia requer acesso a várias senhas, e pode ser uma tarefa difícil entender quem está acessando quais segredos e para quê está acessando. Dessa forma, ter a geração de novas chaves (*key rolling*), armazenamento seguro e logs de auditoria detalhados são essenciais, e é aí que o *Vault* é usado. Antes que segredos ou dados confidenciais sejam fornecidos, o *Vault* primeiro precisa validar e autorizar os clientes (usuários, máquinas, aplicativos).

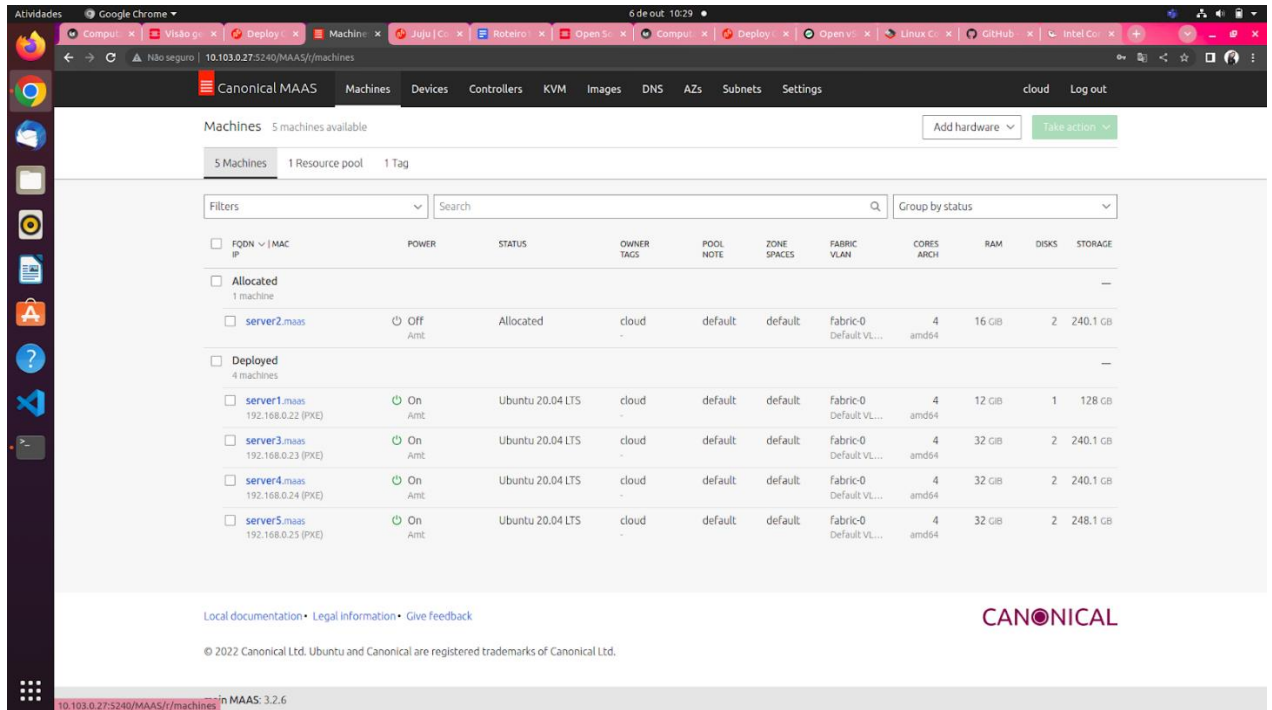
Por fim, o *Vault* funciona principalmente por meio de 4 estágios: autenticação, validação, autorização e acesso. Sendo assim, primeiro o cliente irá submeter sua identidade na etapa de autenticação para que esse servidor de secrets determine se o cliente é aquilo que ele diz que é, e depois que ele for autenticado um token é gerado e associado a uma política - que é o que vai determinar e restringir as ações e a acessibilidade a caminhos para cada cliente. Depois, o *Vault* irá validar o cliente através de recursos terceiros como o Github, LDAP, APPRole entre outros. Feito isso, o cliente vai ser autorizado a acessar determinados endpoints de API através do acesso com seu token do *Vault*. Por fim, o *Vault* concederá o acesso a segredos, chaves e recursos de criptografia emitindo um token com base nas políticas associadas à identidade do cliente.

A imagem abaixo (retirada do site <https://www.vaultproject.io/docs/what-is-vault>) ilustra bem isso:

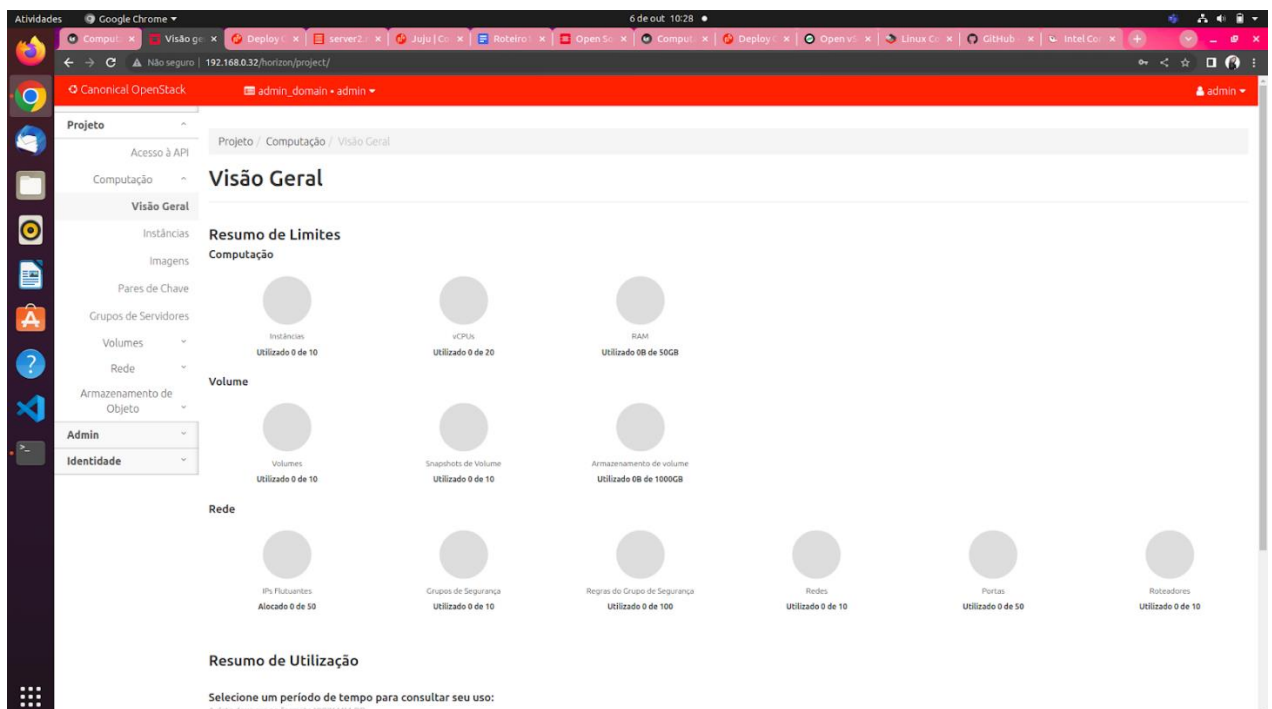


CheckPoint-1

1. De um print das Telas abaixo:
2. Do Dashboard do MAAS com as máquinas.



3. Da aba compute overview no OpenStack.



4. Da aba compute *instances* no OpenStack.

Projeto / Computação / Instâncias

Instâncias

ID de Instância = Filtro [Disparar Instância](#)

Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Age	Actions
Sem itens para exibir.										

5. Da aba network *topology* no OpenStack.

Projeto / Rede / Topologia de Rede

Topologia de Rede

[Disparar Instância](#) [Criar Rede](#) [Criar Roteador](#)

Topologia **Gráfico**

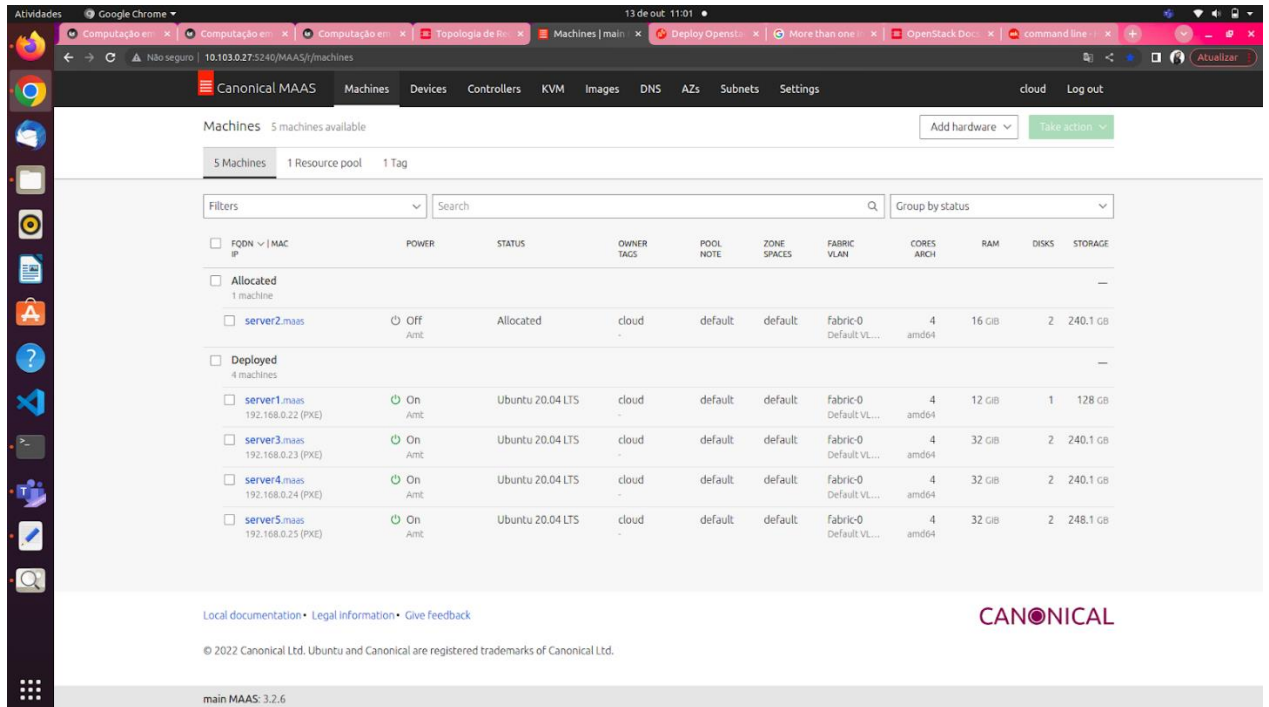
☐ Pequeno ☒ Normal

Não existem redes, roteadores, ou instâncias conectadas para exibir.

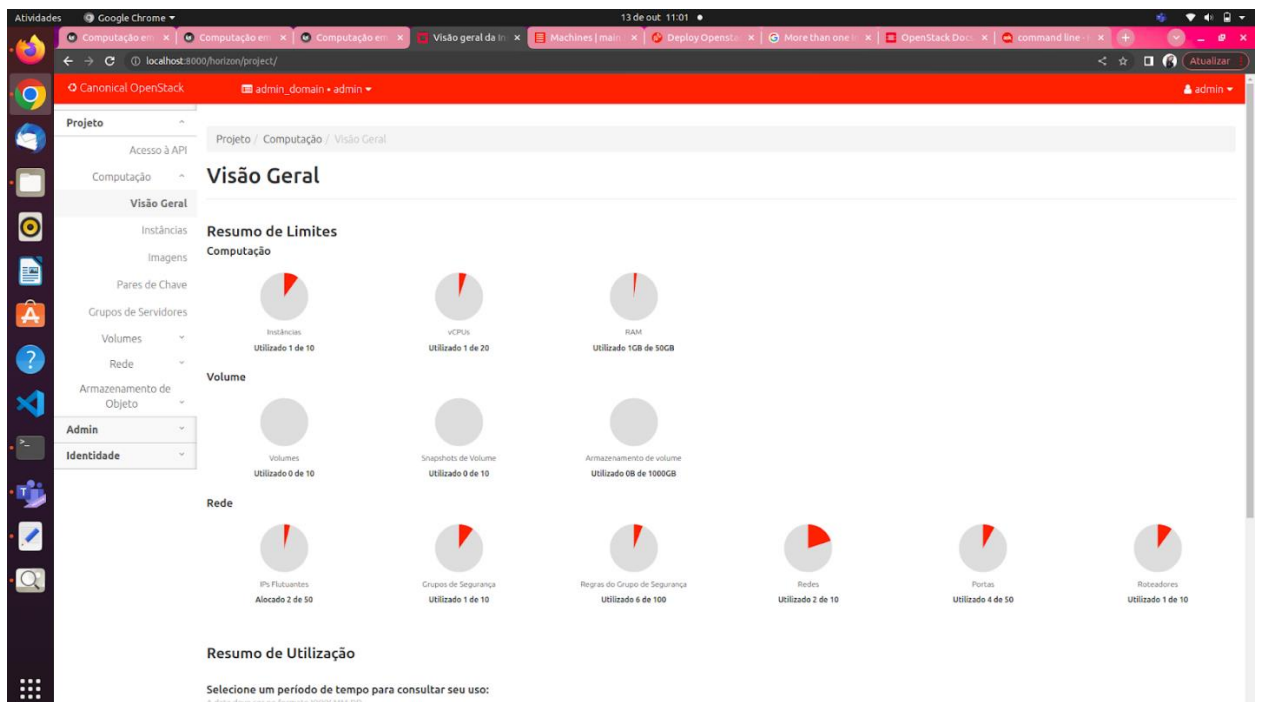
[Disparar Instância](#) [Criar Rede](#) [Criar Roteador](#)

CheckPoint-2

1. De um print das Telas abaixo:
2. Do Dashboard do MAAS com as máquinas.



3. Da aba compute overview no OpenStack.



4. Da aba compute *instances* no OpenStack.

Projeto / Computação / Instâncias

Instâncias

ID de Instância = Filtro [Disparar Instância](#) [Excluir Instâncias](#) [Mais Ações](#)

Exibindo 1 item

Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Age	Actions
focal-1	focal-ami64	192.169.0.196, 192.168.8.239	m1.tiny	mykey	Ativo	nova	Nenhum	Executando	13 minutos	Criar Snapshot

Exibindo 1 item

Obs: erramos o nome da instância, mas essa focal-1 corresponde a instância cliente.

5. Da aba network *topology* no OpenStack.

Projeto / Rede / Topologia de Rede

Topologia de Rede

[Disparar Instância](#) [Criar Rede](#) [Criar Roteador](#)

Topologia Gráfico

Pequeno Normal

10.0.0.0/24 10.0.0.0/24

focal-1

ID: 352f578b-d5a6-4956-8503-1c1ede960000

STATUS: Ativo

[Ver Detalhes da Instância](#) [Abrir Console](#) [Excluir Instância](#)

[Disparar Instância](#) [Criar Rede](#) [Criar Roteador](#)

6. Enumere as diferenças encontradas entre os prints das telas no Checkpoint1 e o Checkpoint-2.

Dentre as diferenças encontradas entre os prints das telas no Checkpoint1 e no Checkpoint2 temos:

1 - Já na página de Overview vemos que alguns recursos que antes não estavam sendo utilizados agora estão, como: vCPUs, memória RAM, instâncias, entre outros. Além de que agora redes foram adicionadas (tanto externa quanto interna) e roteadores também.

2 - Na aba instâncias, por sua vez, podemos ver que agora existe uma criada: a focal-1 (erramos o nome, era para ser client) que utilizou o tipo de instância (flavor) com os menores recursos: a tiny.

3 - Por fim, na topologia de rede podemos ver que existem duas redes: uma externa e uma interna, e entre elas temos um roteador virtual que é o gateway que permite a comunicação entre essas duas redes. Além disso, na rede interna existe uma instância, que é a client, e que permite que aplicações sejam subidas nela, funcionando como uma máquina virtual.

7. Explique como cada recurso foi criado.

Antes de tudo começamos importando as imagens tanto do Bionic quanto do Ubuntu para ter um sistema operacional na nuvem e conseguir configurar as redes. E para isso baixamos as imagens do site oficial e depois as importamos para uma pasta que criamos, chamada cloud-images.

Feito isso, configuramos as redes conforme solicitado no roteiro. Dessa forma, uma rede externa foi montada e um roteador também foi configurado (esse será usado como provedor por todos da rede para ter um acesso público à todas as máquinas virtuais que serão criadas). A rede externa foi configurada com o gateway 192.168.0.1 e na subrede que começa em 192.168.7.0 e que vai até 192.168.8.255. Depois criamos a rede interna com o seguinte gateway: 192.169.0.1 e em um intervalo que começa no endereço de ip 192.169.0.10 e termina no ip 192.169.0.200. Por fim, configuramos o roteador que é provedor tanto para essa rede externa quanto para a rede interna.

Agora, para a máquina virtual ser criada, primeiro criamos 4 tipos de instâncias (flavors) que permitem escolher os recursos que serão alocados para a criação da máquina virtual. Esses 4 flavors eram: tiny, small, medium e large - que utilizam recursos de maneira crescente, por exemplo, o flavor tiny usa apenas 1 vCPU, 1Gb de RAM e 20 Gb de disco, enquanto o flavor small utiliza 1vCPU, 2 Gb de RAM e 20 Gb de disco. E antes de criar a máquina virtual, para acessar as instâncias nós tivemos que importar um par de chaves ssh

para a nuvem e depois criar regras para cada grupo de segurança - que é o que permite o ping e que o ssh flua para as instâncias.

Com tudo isso, finalmente criamos a máquina virtual client com a imagem do ubuntu e com o flavor tiny. Para isso, também foi necessário pedir um ip flutuante que é um ip público que conseguimos acessar - isto é endereçar ele nas rotas - ou seja, a máquina virtual tem um ip na rede interna e outro público que a rede externa conhece e pode endereçar. Por fim, logamos na máquina para confirmar que ela havia sido criada com sucesso.

QUESTÕES-2

1. Qual a função do Glance? O que o comando abaixo faz? "juju run-action glance-simplestreams-sync/0 sync-images --wait"

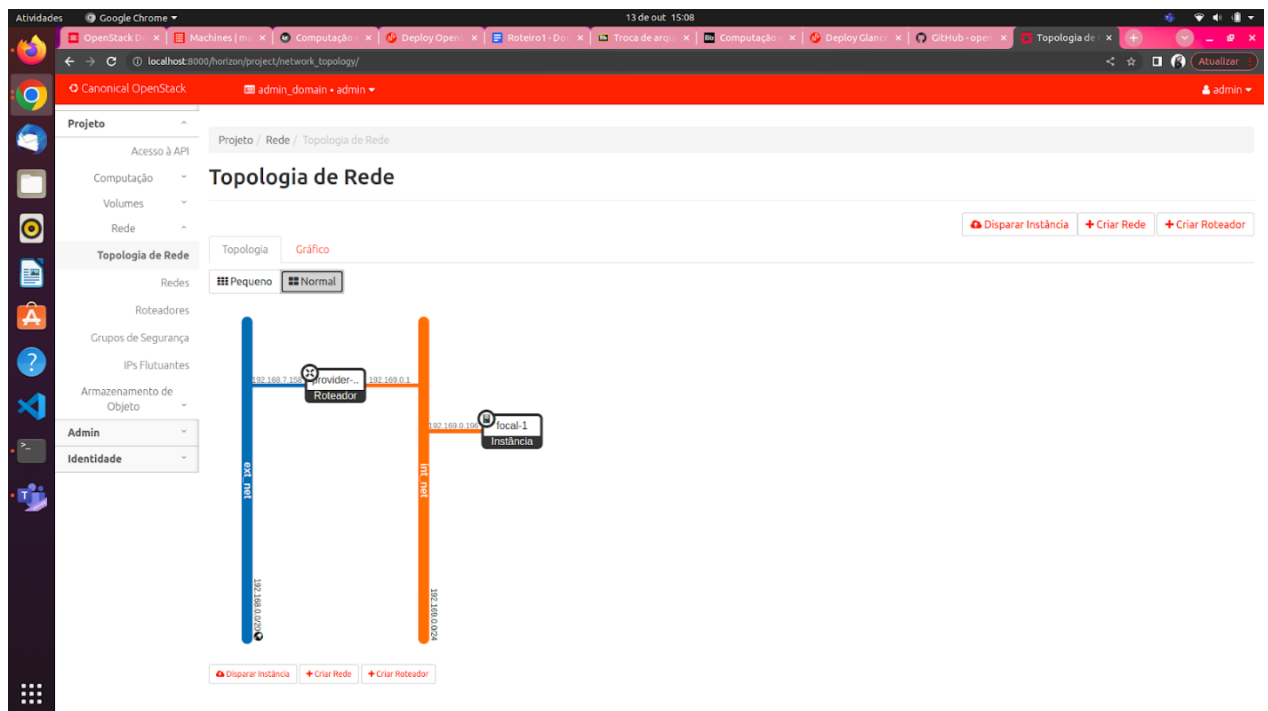
O Glance fornece serviços de imagem que incluem descobrir, registrar e recuperar imagens de máquinas virtuais. Além disso, ele possui uma API RESTful que permite consultar os metadados da imagem da VM, sendo que as imagens de VM disponibilizadas pelo Glance podem ser armazenadas em vários locais - desde sistemas de arquivos simples até sistemas de armazenamento de objetos.

E o comando "juju run-action glance-simplestreams-sync/0 sync-images --wait" executa uma ação Juju que permite uma operação específica de ser realizada por unidade. Nesse caso a ação será sincronizar uma imagem única mais atualizada do Ubuntu de uma mirror list na máquina 0.

2. Para que serve o Metadata gerada pelo glance?

O glance gera o metadata que nada mais é do que a informação sobre a imagem da VM que ele sincroniza, ou seja - são dados que descrevem dados, no nosso caso, dados que descrevem a imagem do Ubuntu.

3. Faça um desenho de como é a sua arquitetura de rede, desde a sua conexão com o Insper até a instância alocada.



4. O que é um Hypervisor? Qual o hypervisor do Openstack, da AWS e da Azure?

Um hypervisor - que também é chamado de monitor da máquina virtual - é um software ou hardware responsável por criar, executar máquinas virtuais e alocar recursos para elas. Assim, ele isola o sistema operacional do hypervisor e os recursos das máquinas virtuais, e permite a criação e o gerenciamento dessas máquinas. De maneira geral, é o hypervisor que permitirá o usuário acessar múltiplas máquinas virtuais de maneira otimizada em apenas uma peça de hardware.

E existem dois tipos de hypervisors: o do tipo 1 e o tipo 2. Os do tipo 1, também chamados de bare metal, são executados de maneira direta no hardware do host. Por sua vez, os do tipo 2 são executados como outros programas de computador, como se fosse uma camada de software em um sistema operacional.

O hypervisor do Openstack que estamos usando é o LXD (que utiliza o LXC), embora o Openstack também seja compatível com outros hypervisors, como: KVM, QEMU, UML, VMware vSphere, XEN, XENServer, Hyper-V e Virtuozzo.

Já o da AWS por anos foi o Xen, embora eles tenham criado um novo hypervisor baseado em KVM - o Nitro.

Por fim, o hypervisor da Azure é o Azure Hypervisor que é similar ao Microsoft Hyper V, mas foi customizado especificamente para a plataforma da Azure.

EXERCÍCIOS

1. Escreva um relatório dos passos utilizados.

Para levantar a aplicação Django com o banco de dados Postgres e fazer o uso do NGINX como load balancer, primeiro começamos criando máquinas virtuais - para exatamente corrigir o problema do final do roteiro 2 que consistia no fato de utilizar muitos recursos para uma única aplicação.

Assim, aproveitamos os flavors que já tínhamos e decidimos levantar todas as aplicações com o small - isso porque não queríamos que os recursos fossem muito pequenos, mas também não muito grandes. Sendo que, caso fosse necessário escalar a aplicação, teríamos mais recursos do que apenas o necessário, ao mesmo tempo não sendo muito exagerado. Assim, com o small as quatro máquinas virtuais (postgres-1, django-1, django-2 e nginx) foram criadas com os seguintes recursos: 1 vcpu/2Gb RAM/20Gb disk.

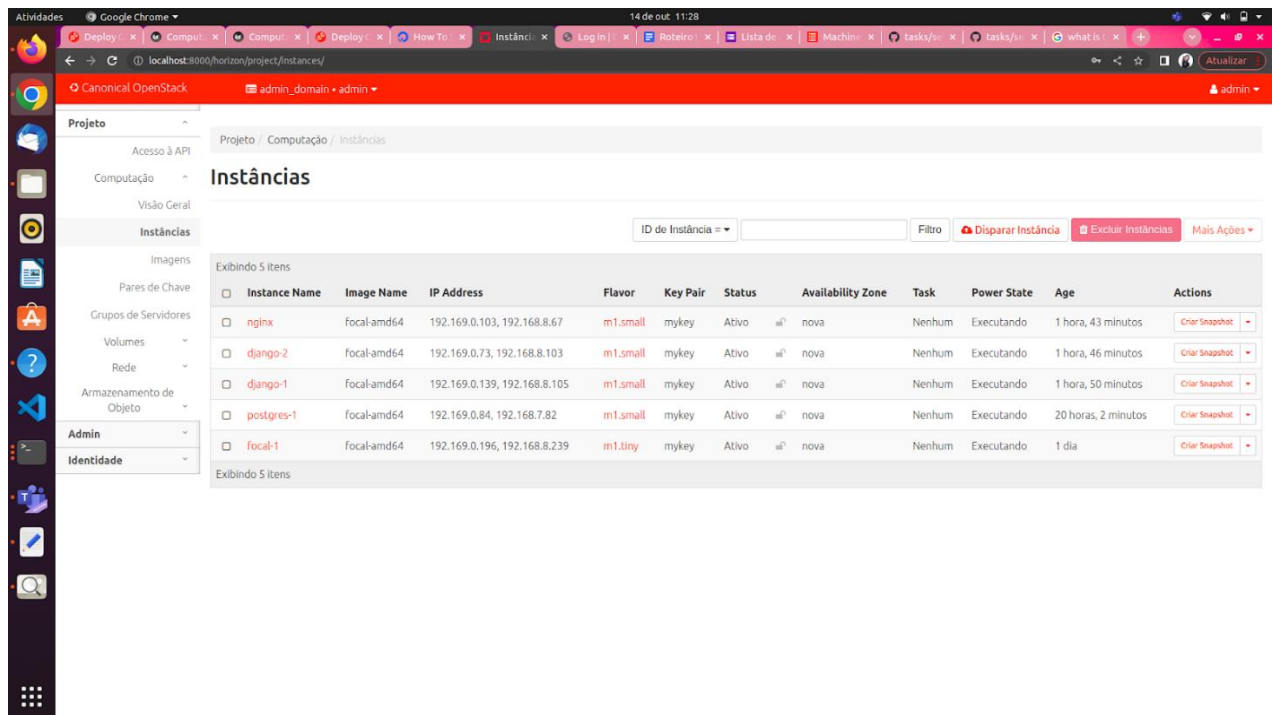
Feito isso, seguimos o roteiro 1 para criar o Postgres - configuramos o usuário, criamos um database, expomos o serviço para acesso e liberamos qualquer máquina dentro da subrede do kit. E o postgres-1 foi alocado pelo próprio Openstack para o server4.

Por sua vez, para levantar as duas aplicações do Django, seguimos o roteiro 2 e utilizamos o Ansible para automatizar o processo. Como já tínhamos o Ansible baixado no server main apenas pedimos para que as máquinas django-1 e django-2 fizessem o deploy do serviço tasks. Vale lembrar que nesse processo tivemos que mudar o documento hosts, e isso porque na nossa aplicação Django, o host estava configurado para ser o server1, mas o server1 muda dependendo da máquina na qual queremos subir a aplicação. Assim, em cada máquina (django-1 e django-2) alteramos o arquivo hosts para reconhecer o server1 como o ip público de cada máquina (django-1 e django-2). Por fim, o django-1 foi alocado no server3 e o django-2 no server2.

Agora, para levantar o NGINX para funcionar como um load balancer e dividir a carga de acesso entre os dois nós, apenas seguimos o tutorial do site: <https://www.digitalocean.com/community/tutorials/how-to-set-up-nginx-load-balancing> . Nele alteramos os ips do backend que eram os das máquinas virtuais criadas para o django (lembrando de acrescentar a porta de cada um, no caso como o serviço é o django a porta era 8080).

Por fim, criamos um túnel e testamos a conexão de tudo que havia sido criado e entramos em uma das aplicações Django com o NGINX.

2. Anexe fotos e/ou diagramas contendo: arquitetura de rede do Django dentro do Openstack* (Dica: use o dashboard do Openstack), lista de VMs utilizadas com nome e IPs alocados, foto do Dashboard Django conectado na máquina Nginx/LB.

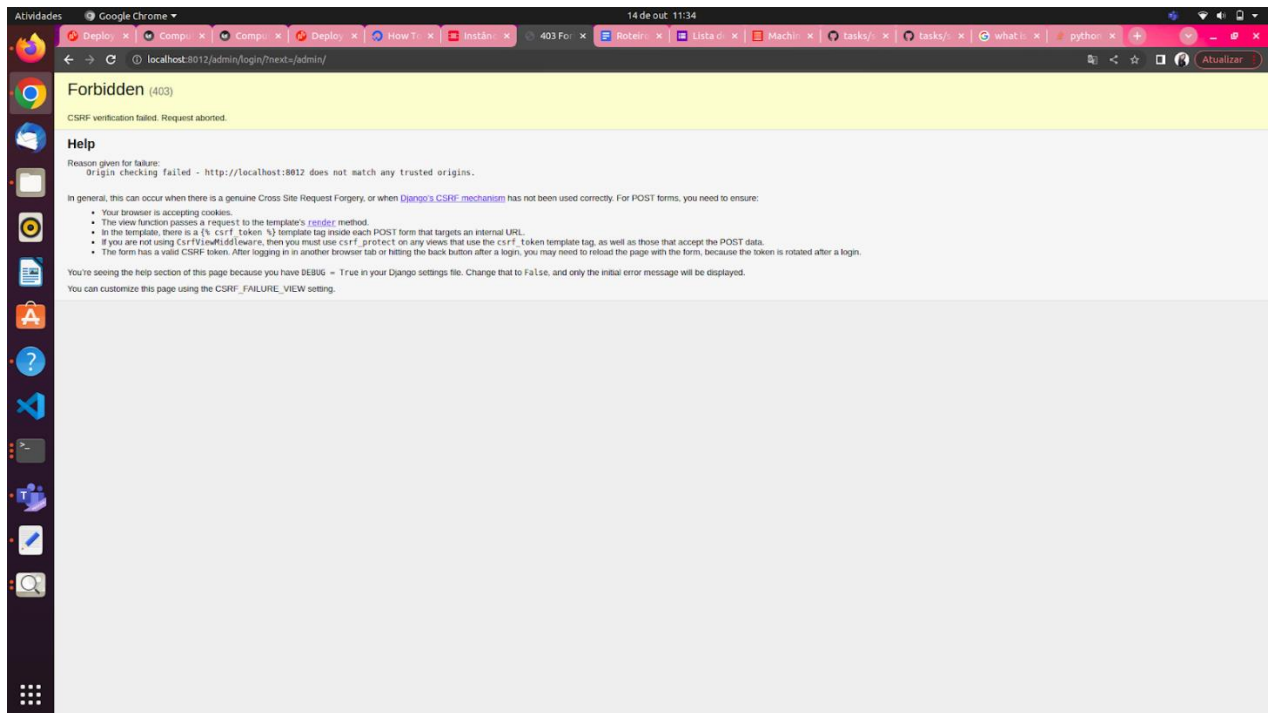


Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Age	Actions
nginx	focal-amd64	192.169.0.103, 192.168.8.67	m1.small	mykey	Ativo	nova	Nenhum	Executando	1 hora, 43 minutos	Criar Snapshot
django-2	focal-amd64	192.169.0.73, 192.168.8.103	m1.small	mykey	Ativo	nova	Nenhum	Executando	1 hora, 46 minutos	Criar Snapshot
django-1	focal-amd64	192.169.0.139, 192.168.8.105	m1.small	mykey	Ativo	nova	Nenhum	Executando	1 hora, 50 minutos	Criar Snapshot
postgres-1	focal-amd64	192.169.0.84, 192.168.7.82	m1.small	mykey	Ativo	nova	Nenhum	Executando	20 horas, 2 minutos	Criar Snapshot
focal-1	focal-amd64	192.169.0.196, 192.168.8.239	m1.tiny	mykey	Ativo	nova	Nenhum	Executando	1 dia	Criar Snapshot

Lista de VMs utilizadas com nome e IPs alocados



Arquitetura da rede do Django dentro do Openstack.



Dashboard Django conectado na máquina Nginx.

QUESTÕES-3

1. Dado que vocês trabalharam com Nuvem Pública e com Nuvem Privada, descreva com detalhes como você montaria uma Nuvem Híbrida. Como seria a troca de dados?

Uma nuvem híbrida é o resultado da conexão de um ambiente de nuvem privada com o de uma nuvem pública. Nesse caso, vamos adotar que a nuvem pública é quem terá a fonte dos dados, e isso pelo motivo de que a nuvem pública garante maior segurança e tem menores custos para o contratante - já que ele não precisará arcar com os custos de montar a estrutura, de implementar a segurança, garantir alta disponibilidade e manter o hardware que possui (contrato de energia, internet, etc), além de que em nuvem privada a escalabilidade é mais difícil do que na nuvem pública. Assim, as requisições seriam da nuvem privada para a pública, e como cada uma está em uma rede diferente, nesse processo teríamos um roteador NAT para trocar informações entre elas - o que seria possível por meio de portas (sockets) para cada aplicação.

2. É possível somar todo o hardware disponível e disparar uma instância gigante (ex: mais memória do que disponível na melhor máquina)? Discorra sobre as possibilidades.

Sim, é possível somar todo o hardware disponível e disparar uma instância gigante. Para fazer isso existem dois jeitos diferentes: podemos tanto rodar vários processos em máquinas diferentes que juntas resultam em mais recursos, como somar os recursos das máquinas para rodar um único processo. O primeiro seria um paralelismo, já que os recursos em si são divididos em vários processos, dessa forma, cada máquina estaria somando a memória e o processador para executar aplicações diferentes (como por exemplo executar o Postgres em um processo e em uma máquina, e executar o Django em outro processo e em outra máquina). No segundo caso, por sua vez, todos os recursos são utilizados para um único processo, e normalmente isso é feito para distribuir a carga de um processamento particularmente pesado. Um exemplo é como aconteceu no filme do Avatar, onde para processar a imagem foram utilizados muitos recursos juntos em um único processo, a fim de montar todo o contexto e efeitos no mesmo momento em que a imagem era capturada ao vivo pela câmera.

3. Cite e explique pelo menos 2 circunstâncias em que a *Private Cloud* é mais vantajosa que a *Public Cloud*.

Em uma nuvem privada temos uma estrutura em nuvem na qual sua infraestrutura é implantada e usada exclusivamente por apenas uma organização. Além disso, esse tipo de plataforma em nuvem pode até estar no local fisicamente, como no datacenter da empresa, mas também pode ser operada por um fornecedor e ser terceirizada fora do local.

Uma das grandes diferenças de uma nuvem privada para uma nuvem pública é o nível significativo mais alto do controle sobre o sistema pela empresa. Sendo assim, em uma nuvem privada é a empresa que precisa cuidar da manutenção do hardware e de toda a infraestrutura. Além disso, pelos recursos do sistema estarem isolados em uma rede privada, há maior sigilo garantindo que ninguém de fora consiga acessar esses recursos.

Sendo assim, as principais vantagens de uma nuvem privada são o controle e o sigilo de dados. Isso porque como tudo está situado no local (mesmo que seja terceirizado), a empresa terá controle sobre a infraestrutura, além de ter flexibilidade - já que poderá personalizar a nuvem para atender especificamente às suas necessidades e adaptá-la para atender as conformidades atuais, o que consequentemente faz com que o desempenho de maneira geral seja previsível. E, em termos de sigilo, tem-se que será mais eficiente, pois não existem outras empresas utilizando aquele serviço e compartilhando recursos da

infraestrutura. Dessa forma, a própria empresa pode implementar soluções de segurança e lidar com o tráfego limitado às transações da própria empresa.

4. Openstack é um Sistema Operacional? Descreva seu propósito e cite as principais distribuições?

O OpenStack é um sistema operacional em nuvem que controla grandes conjuntos de recursos de computação, armazenamento e rede em um datacenter por meio de projetos (ferramentas), além de criar e gerenciar nuvens públicas e privadas. Para isso, todos os recursos são gerenciados e provisionados através de APIs com autenticação.

E além dessa funcionalidade de infraestrutura como serviço, também existem outros componentes que fornecem orquestração e gerenciamento de falhas e serviços para garantir alta disponibilidade.

5. Quais são os principais componentes dentro do Openstack? Descreva brevemente suas funcionalidades.

Existem vários componentes dentro do OpenStack para fazer o deploy, dentre esses os principais são: NOVA, ZUN, IRONIC, CYBORG, SWIFT, CINDER, MANILA, NEUTRON, OCTAVIA, DESIGNATE, KEYSTONE, PLACEMENT, GLANCE, BARBICAN, HEAT, SENLIN, MISTRAL, ZAQAR, BLAZAR, AODH, MAGNUM, SAHARA, TROVE, MASAKARI, MURANO, SOLUM, FREEZER, EC2API, HORIZON E SKYLINE. Dentre esses o que mais usamos foram os seguintes: Nova, Cinder, Neutron, Keystone, Glance, Ceph, Horizon, Mysql Inno Cluster e RabbitMQ.

O NOVA é usado para implementar serviços e bibliotecas associadas para prover acesso a serviços de recursos de computação de maneira massivamente escalável, sob demanda e que inclui bare metal, máquinas virtuais e contêineres. E esse componente depende de outros como: Keystone, Neutron e Glance.

Por sua vez o Cinder é um serviço de block storage para o Openstack que virtualiza o gerenciamento de dispositivos de armazenamento em bloco e provê aos usuários finais uma API para solicitar e consumir esses recursos sem precisar de nenhum conhecimento de onde seu armazenamento está realmente implantado ou em que tipo de dispositivo. E para isso uma implementação de referência (LVM) ou drivers de plug-in para outro armazenamento é utilizado. Além disso, o Cinder depende do Keystone.

Já o Neutron é um projeto de rede SDN que é focado em entregar a rede como um serviço (NaaS) em ambientes virtuais de computação. E assim como o Cinder, ele também depende do Keystone.

O Keystone cuida da parte de autenticação de cliente de API e também da descoberta de serviço e autorização distribuído por meio da implementação da API de identidade do Openstack. Sendo que ele suporta o LDAP, OAuth, OpenID Connect, SAML e SQL.

O Glance por sua vez fornece serviços de imagem que incluem descobrir, registrar e recuperar imagens de máquinas virtuais. Além disso, ele possui uma API RESTful que permite consultar os metadados da imagem da VM e recuperar a imagem real. E as imagens de VM disponibilizadas pelo Glance podem ser armazenadas em vários locais - desde sistemas de arquivos simples até sistemas de armazenamento de objetos.

Já o Horizon é a implementação da Canonical para o Dashboard da OpenStack, que oferece uma interface web para o usuário para os serviços da OpenStack.

Por fim, o Ceph foi utilizado pelo Glance como armazenamento e o Mysql Inno Cluster foi utilizado como um cluster de banco de dados e o RabbitMQ como um servidor de filas.

Fonte das informações: <https://www.openstack.org/software/project-navigator/openstack-components/#openstack-services> .