

Cloud Computing - Roteiro 3

Arthur Barreto e Enricco Gemha

Questões 1

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

Inspirada pelo sucesso do Docker, a Canonical, empresa responsável pelo sistema operacional Ubuntu Linux, lançou sua própria solução de container, também como uma camada sobre o LXC. Essa solução foi chamada de LXD. Como qualquer solução de container, o LXD se propõe a isolar processos e limitar o consumo de recursos como CPU e memória.

Uma breve nota no tempo, para 2008, quando o Linux passou a incluir a primeira solução robusta de container por padrão no Kernel, que ficou conhecida como LXC (Linux Container). Em 2013, foi criado o Docker como uma camada sobre o LXC. O Docker se popularizou por simplificar a utilização de containers LXC ao oferecer uma API Rest para desenvolvedores. O foco do Docker é a arquitetura de micro-serviços, e é por isso que um container Docker executa apenas um processo por padrão.

Voltando ao LXD, ele também oferece uma API Rest para desenvolvedores. No entanto, a grande diferença entre eles é que enquanto o Docker trava rodar apenas um processo por container, o LXD foca em rodar um Linux completo por container, ou seja, permitindo rodar múltiplos processos em um único container LXD. Assim, as imagens oficiais do LXD são de distros Linux.

Apesar de comparações entre LXD e Docker, eles não são exatamente competidores, já que ambos podem trabalhar juntos. Por exemplo, podemos rodar containers Docker dentro de um container LXD.

2. O que é o bundle.yaml?

'bundle.yaml' é um arquivo de descrição no formato YAML com uma gama fixa de elementos. Para o Juju, esse arquivo descreve um conjunto de Charms, que são combinados e configurados para automatizar uma solução multi-charm. Nele são definidas informações como as máquinas a serem "deployadas", nome dos charms e a relação entre eles. Por exemplo, um bundle de WordPress pode incluir charms de: Wordpress, mysql e o charm de relação entre eles. Isso permite que o Juju continue gerenciando o deploy, já que as informações são facilmente acessadas por ele.

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

Vault é um servidor que gerencia credenciais, desenvolvida pela Hashicorp. Ela armazena segredos, como chaves SSH, dados de acesso a bancos de dados e tokens de API. Ele criptografa dados confidenciais em trânsito e em repouso usando chaves de criptografia gerenciadas e protegidas de maneira centralizada, tudo por meio de um único fluxo de trabalho e uma única API. Para acessar um segredo, um usuário precisa primeiro autenticar-se no Vault. Uma vez que está autenticado, o usuário pode solicitar o segredo desejado. O Vault irá então fornecer o segredo ao ~~usuário~~ usuário, criptografado. O usuário pode então descriptografar o segredo usando sua própria chave privada. É possível acessar o Vault através do CLI no terminal, uma Web GUI ou através de chamadas de API.

O Vault também permite a rotação de segredos, que é o processo de substituição de um segredo antigo por um novo. O Vault pode ser configurado para rotacionar segredos automaticamente em intervalos regulares. Isso ajuda a proteger os segredos contra ataques, pois torna mais difícil para um invasor obter acesso a um segredo válido.

Checkpoint 1

1. Dê 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**.
5. Da aba network topology no **OpenStack**.

Google Chrome File Edit View History Bookmarks Profiles Tab Window Help

Google Agenda - Seman... x | Computação em Nuvem x | Visão geral da Instância x | Machines | clouc MAAS x | Access the dashboard - x | ChatGPT x | + v

Not Secure | 10.103.0.27.5240/MAAS/r/machines

6º Semestre Coding Tools Portfolio Insper Monkeytype | A mi... Games IA Cloud How to integrate... + A - Planilhas Goo...

Canonical MAAS Machines Devices Controllers KVM Images DNS AZs Subnets Settings

Add hardware v Take action v

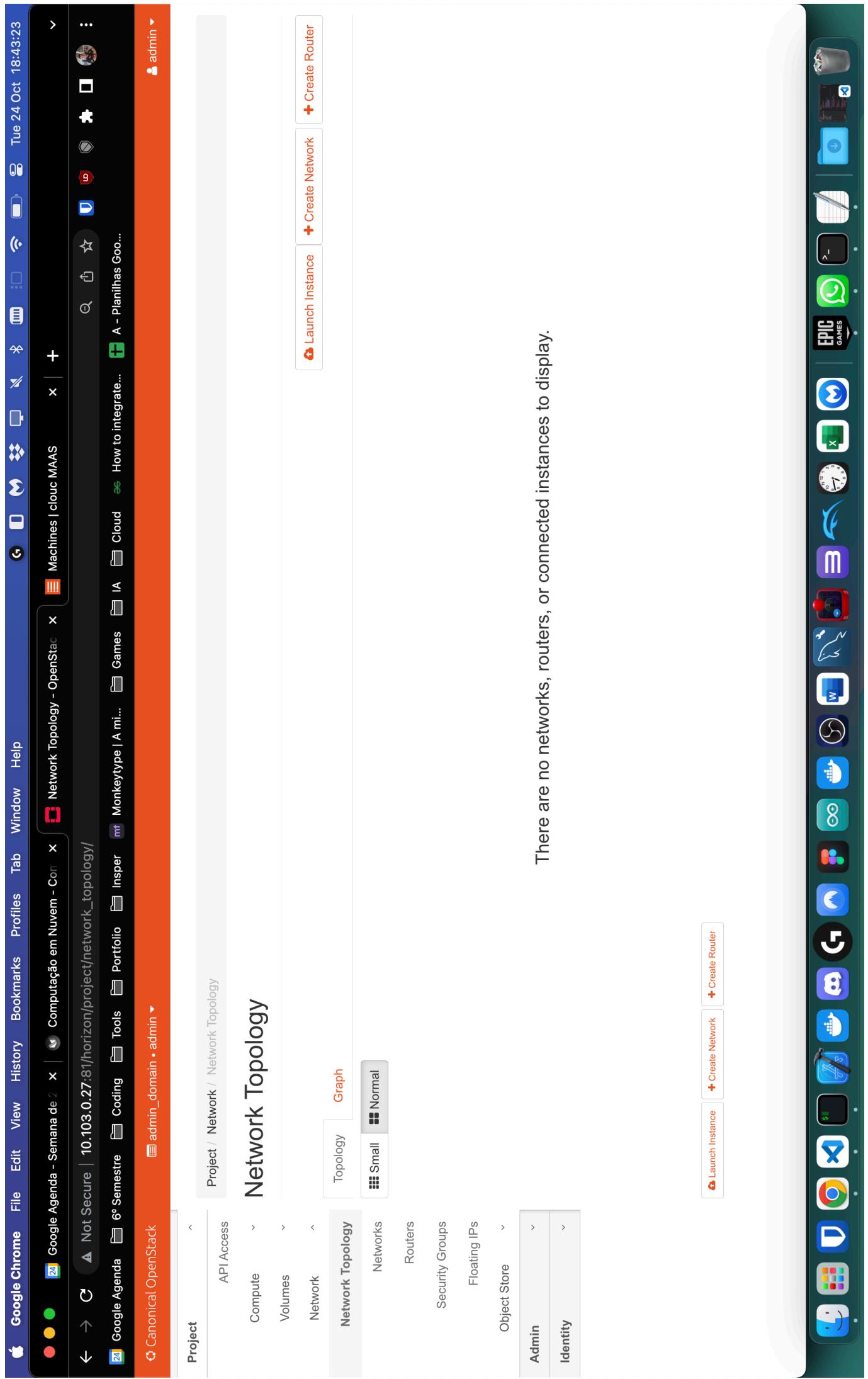
Machines 5 machines available

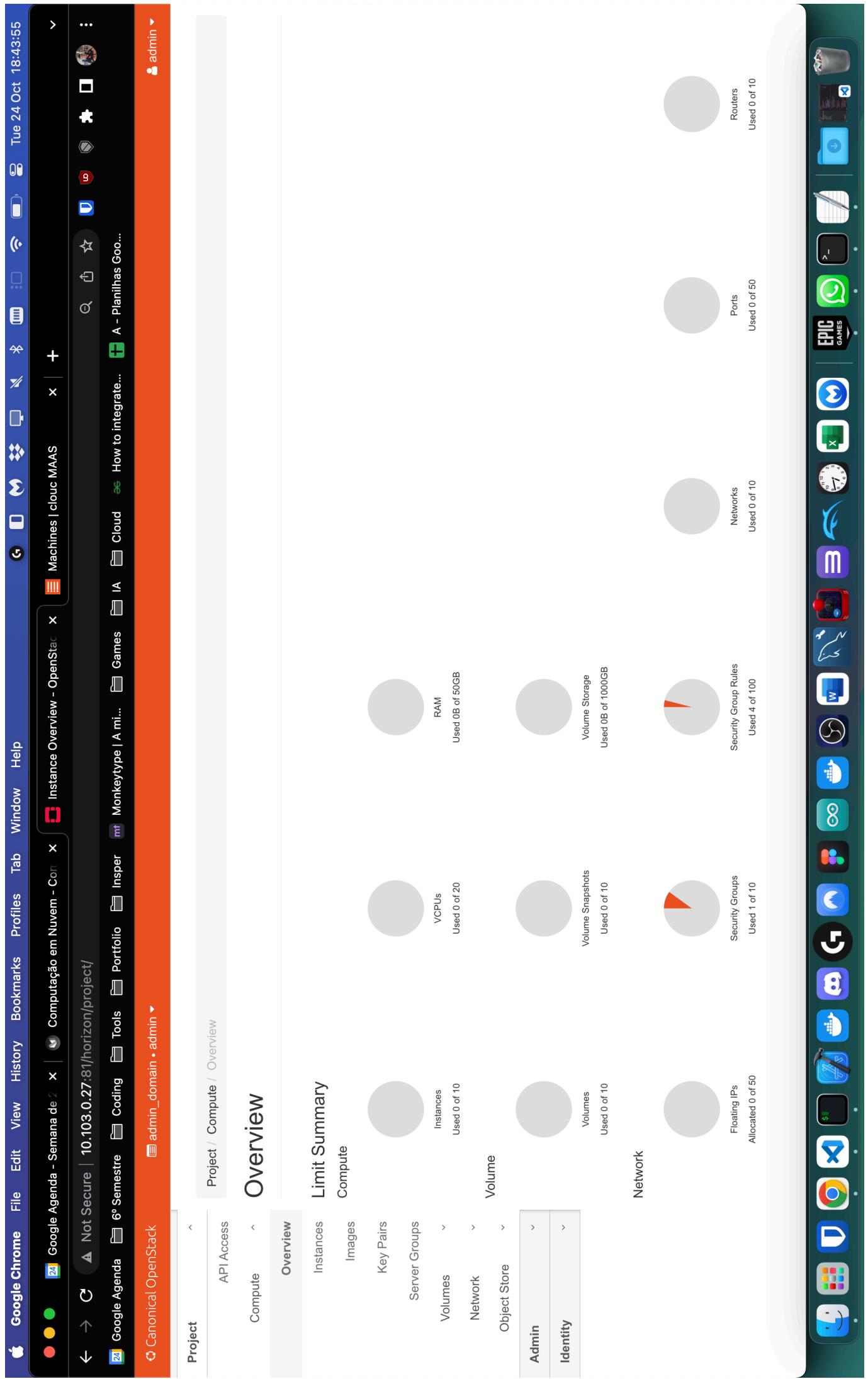
5 Machines 1 Resource pool 1 Tag

Filters							Search				Group by status			
<input type="checkbox"/> FQDN v MAC IP		POWER		STATUS		OWNER TAGS		ZONE SPACES		FABRIC VLAN		CORES ARCH		
Deployed														
<input type="checkbox"/>	server5.maas	On	Amt	Ubuntu 20.04 LTS	cloud	default	default	fabric-0 Default VI...	4	32 GiB	2	248.1 GB	—	
<input type="checkbox"/>	server4.maas	On	Amt	Ubuntu 20.04 LTS	cloud	default	default	fabric-0 Default VI...	4	32 GiB	2	240.1 GB	—	
<input type="checkbox"/>	server3.maas	On	Amt	Ubuntu 20.04 LTS	cloud	default	default	fabric-0 Default VI...	4	32 GiB	2	240.1 GB	—	
<input type="checkbox"/>	server1.maas	On	Amt	Ubuntu 22.04 LTS	cloud	default	default	fabric-0 Default VI...	4	12 GiB	1	128 GB	—	
<input type="checkbox"/>	Allocated		1 machine											
<input type="checkbox"/>	server2.maas	Off	Amt	Allocated	cloud	default	default	fabric-0 Default VI...	4	16 GiB	2	240.1 GB	—	



The screenshot shows a Mac desktop environment with a dark-themed window manager. A Google Chrome window is open, displaying the OpenStack Horizon dashboard under the project 'Compute'. The dashboard shows a list of instances, with one named 'Ubuntu' selected. The instance details page is visible, showing configuration options like 'Flavor', 'Image Name', 'Availability Zone', and 'Status'. The status is listed as 'Running' with a green icon. The 'Actions' column contains buttons for 'Launch Instance', 'Stop', 'Delete', and 'Reboot'. The browser's address bar shows the URL '10.103.0.27:81/horizon/project/instances/'. The top of the screen features a menu bar with 'Google Chrome', 'File', 'Edit', 'View', 'History', 'Bookmarks', 'Profiles', 'Tab', 'Window', and 'Help'. A system tray on the right side of the screen displays icons for battery, signal strength, and system status. The bottom of the screen features a dock with various application icons, including Finder, Mail, Safari, and system utilities.





Checkpoint 2

1. Dê 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**.
5. Da aba network topology no **OpenStack**.
6. Enumere as diferenças encontradas entre os prints das telas no Checkpoint 1 e o Checkpoint 2.

- Em overview, notamos que ele passou de mostrar somente um security group e suas 4 regras para mostrar:

1 instance criada; 1 vCPU utilizada nesta instance; 1 GB de RAM;

dois floating IPs (acabei criando duas rezes); mais 4 regras adicionadas ao default security group; Duas metaworks adicionadas (internal ~~ext~~ external); 4 portas de rede adicionadas; 1 roteador criado entre a rede interna e externa.

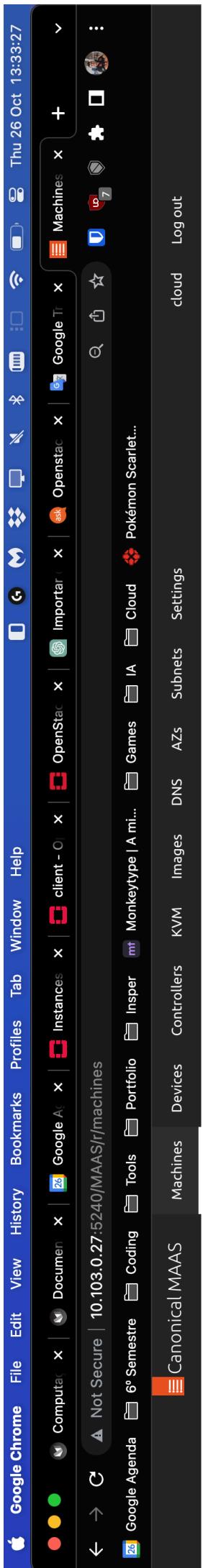
- No Dashboard do Maas vemos que nada mudou.

- Em Instances criamos uma instância chamada "client".

- Em Network Topology podemos verificar como nossa rede dentro do OpenStack ficou.

7. Explique como cada recurso foi criado. *após criarmos as redes interna e ext.*

Ao ~~criarmos~~ criarmos os flavors "m1.tiny", "m1.small", "m1.medium", "m1.large", nós acabamos por criar as possibilidades de alocar N vCPUs para cada instância, de 1 a 4, a depender do flavor. De maneira análoga, podemos alocar de 1 a 8 GB de RAM, a depender do flavor. Voltando um pouco atrás no roteiro, também configuramos a rede na qual o Openstack irá operar. Neste caso, configuramos a rede externa e interna, utilizando nosso conhecimento em sub-redes. Configuramos a external network para o IP 172.16.0.0/20, com gateway em 172.16.0.1, e a internal network configurada em 192.169.0.0. O interessante é que configuramos como gateway da rede interna o provider-router, no endereço 192.169.0.1. Para configurar em qual IP dessa rede ~~a~~ determinada instância pode ser acessada, é necessário definir um floating IP para aquela instância, o que fizemos.

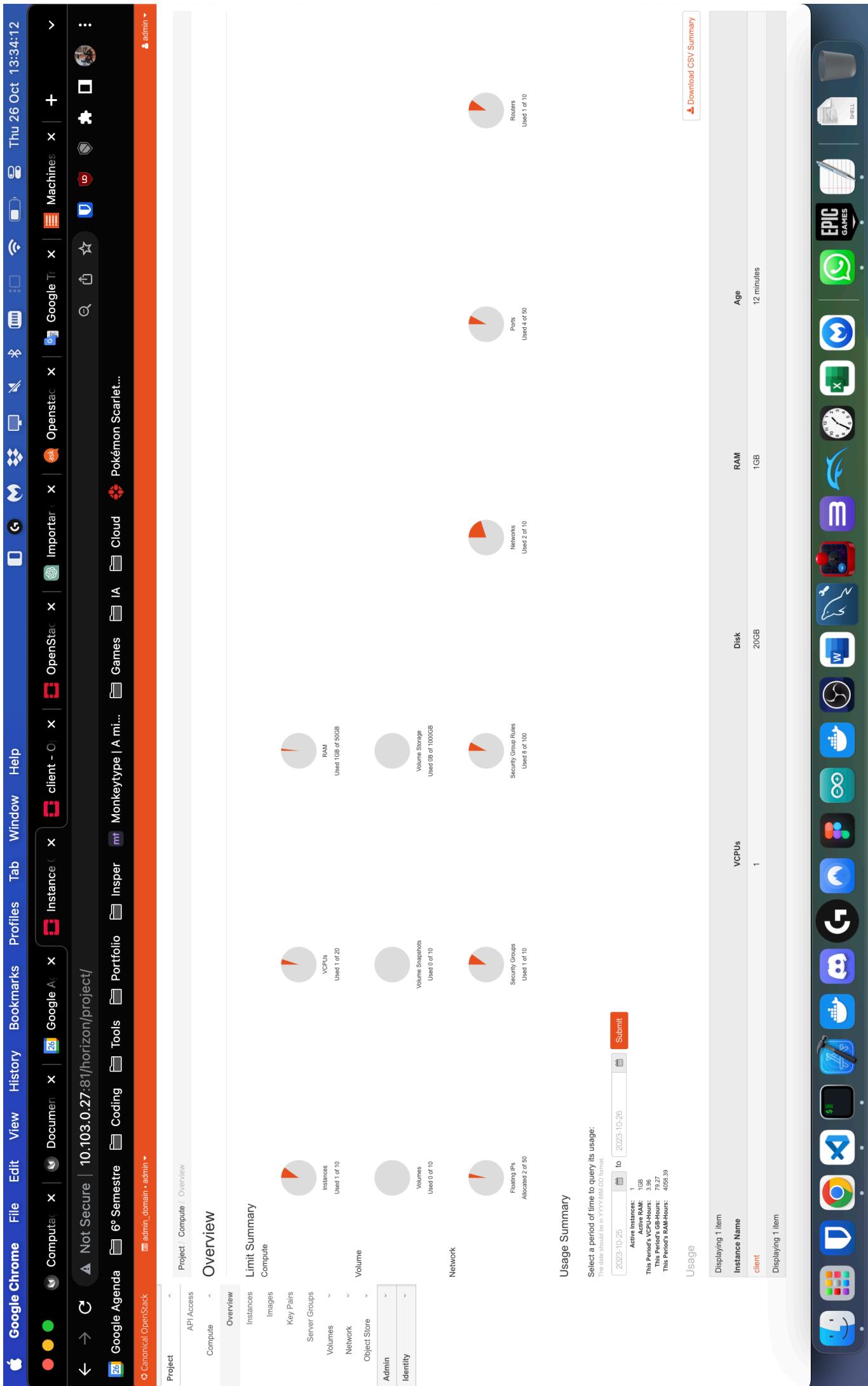


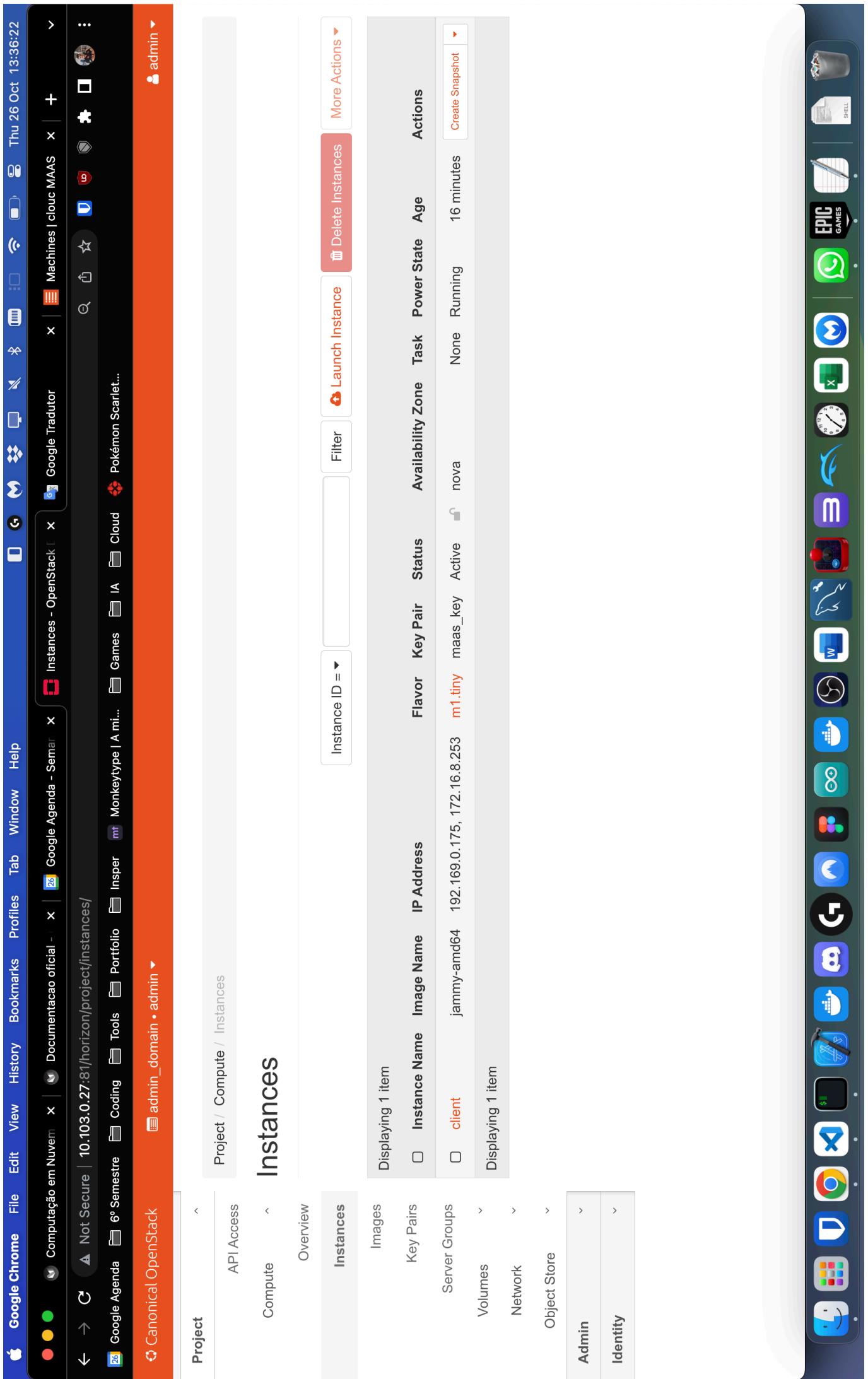
| local documentation • | legal information • | Give feedback

cloud MAAS: 3.3.4

CANONICAL







Thu 26 Oct 13:37:04

Google Chrome File Edit View History Bookmarks Profiles Tab Window Help

Computação em Nuvem - Con | Google Agenda - Semana de 2 | Network Topology - OpenStack | Google Tradutor | Machines | clouc MaaS | +

Not Secure | 10.103.0.27.81/horizon/project/network_topology/#/close | 9 | D | Q | Machines | clouc MaaS | +

Google Agenda | 6º Semestre | Coding | Tools | Portfolio | Inspire | Monkeytype | A mi... | Games | IA | Cloud | Pokémon Scarlet...

Canonical OpenStack | admin_domain • admin ▾ | Launch Instance | + Create Network | + Create Router

Project API Access

Compute Volumes Network

Network Topology

Topology Graph

Network Topology

Small Normal

Routers

Security Groups

Floating IPs

Object Store

Admin Identity

Network Topology

Project / Network / Network Topology

Launch Instance + Create Network + Create Router

client Instance

Provider... Router

192.169.0.175

192.169.0.1

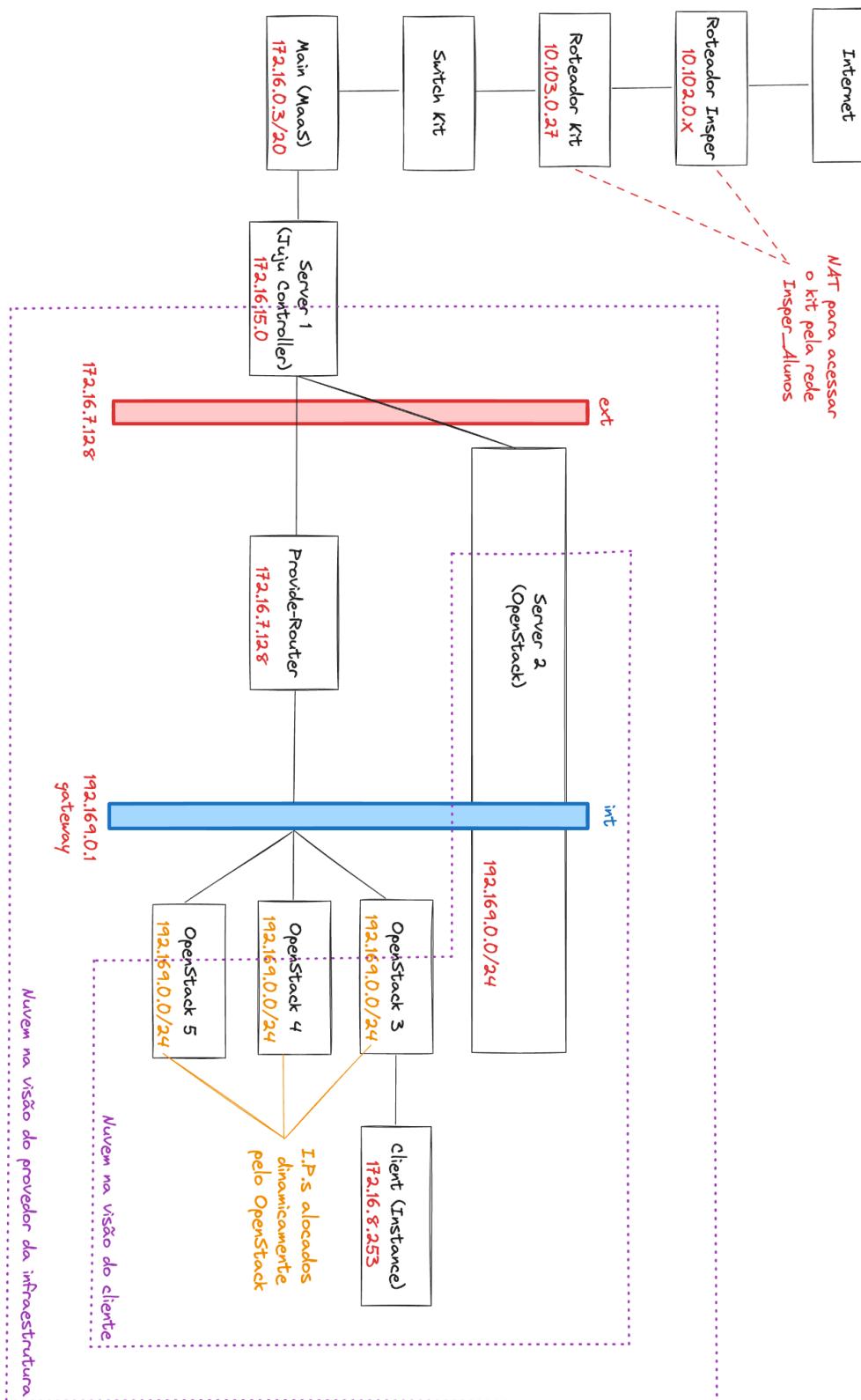
int_net 192.169.0.0/24

ext_net 172.16.0.0/20

```
graph LR; Router1[Router] --- Client[client Instance]; Router1 --- Provider[Provider... Router]; Router2[Router] --- ext_net[ext_net 172.16.0.0/20]; Router2 --- Router1
```

Questões 2

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



Uso da Infraestrutura

Objetivos

1. Aprender a utilizar a nuvem Openstack para aplicações mundo real;
2. Aprofundar conceitos sobre deploy de aplicações em nuvem.

Pré-requisitos:

1. Terminar o roteiro anterior (OpenStack).

Django na Nuvem VM criada

Agora que conseguimos corrigir o erro apontado na conclusão do H2, vamos levantar a mesma aplicação Django mas sem precisar sacrificar 4 máquinas inteiras.

Criando as instâncias necessárias

Vamos começar criando as duas instâncias de Django. Para isso, rode os comandos abaixo:

```
openstack server create --image jammy-amd64 --flavor m1.large \
--key-name maas_key --network int_net \
django1
```

```
openstack server create --image jammy-amd64 --flavor m1.large \
--key-name maas_key --network int_net \
django2
```

Em seguida, criamos a instância de banco de dados:

```
openstack server create --image jammy-amd64 --flavor m1.large \
--key-name maas_key --network int_net \
db
```

Por fim, criamos a instância de balanceador de carga:

```
openstack server create --image jammy-amd64 --flavor m1.large \
--key-name maas_key --network int_net \
lb
```

Assinalando os IPs flutuantes

Agora, vamos assinalar os IPs flutuantes para cada uma das instâncias:

```
IP_DJANG01=$(openstack floating ip create -f value --column floating_ip_address ext_net)
openstack server add floating ip django1 $IP_DJANG01
```

```
IP_DJANG02=$(openstack floating ip create -f value --column floating_ip_address ext_net)
openstack server add floating ip django2 $IP_DJANG02
```

```
IP_DB=$(openstack floating ip create -f value --column floating_ip_address ext_net)
openstack server add floating ip db $IP_DB
```

```
IP_LB=$(openstack floating ip create -f value --column floating_ip_address ext_net)
openstack server add floating ip lb $IP_LB
```

Configurando PostgreSQL (banco de dados)

Vamos acessar a instância de banco de dados:

```
ssh ubuntu@$IP_DB
```

Começamos assinalando o IP do nameserver no arquivo do netplan:

```
sudo nano /etc/netplan/50-cloud-init.yaml
```

E adicionamos o seguinte ao final do arquivo:

```
nameservers:
  addresses: 172.16.0.1
```

Instalando o PostgreSQL:

```
sudo apt update
sudo apt install postgresql postgresql-contrib -y
```

Agora, vamos criar o banco de dados e o usuário:

```
sudo su - postgres  
createuser -s cloud -W
```

Configure a senha **cloud**. Crie a database para o projeto Django:

```
createdb -O cloud tasks
```

Exponha o banco de dados para acesso externo:

```
nano /etc/postgresql/14/main/postgresql.conf
```

E dentro do arquivo, altere a linha:

```
listen_addresses = '*'
```

Altere também o arquivo de configuração de autenticação:

```
nano /etc/postgresql/14/main/pg_hba.conf
```

E adicione a seguinte linha:

```
host      all            all        172.16.0.0/20      trust
```

Saia do usuário postgres:

```
exit
```

Libere o firewall:

```
sudo ufw allow 5432/tcp
```

Por fim, reinicie o PostgreSQL:

```
sudo systemctl restart postgresql
```

Configurando Django

Vamos acessar a instância de Django 1:

```
ssh ubuntu@$IP_DJANGO1
```

Clonamos o repositório e instalamos:

```
git clone https://github.com/raulikeda/tasks.git  
cd tasks  
sudo apt update  
sudo apt install python3
```

Alteramos a linha do `views.py`:

```
sudo nano tasks/views.py
```

E alteramos a linha:

```
    return HttpResponse("Hello, world. You're at the tasks index in Django  
1.")
```

Adicionamos o `server1` ao arquivo de hosts:

```
sudo nano /etc/hosts
```

E adicionamos a seguinte linha:

```
<IP_DB> server1
```

Instalamos conteúdo do arquivo `./install.sh`:

```
./install.sh
```

Rebootamos a instância:

```
sudo reboot
```

Vamos testar se a aplicação está funcionando:

```
wget http://[IP]:8080/admin/
```

Para o Django2, resolvemos utilizar o Ansible para automatizar a instalação. Para isso rodamos os segundos comandos:

```
git clone https://github.com/raulikeda/tasks.git  
cd tasks
```

Alteramos a linha do `views.py`:

```
sudo nano tasks/views.py
```

E alteramos a linha:

```
return HttpResponse("Hello, world. You're at the tasks index in Django  
2.")
```

E rodamos o Ansible:

```
ansible-playbook tasks-install-playbook.yaml --extra-vars server=  
<IP_DJANGO2>
```

E testamos se a aplicação está funcionando:

```
ssh ubuntu@$IP_DJANGO2  
wget http://[IP]:8080/admin/
```

Configurando Nginx (load balancer)

Vamos acessar a instância de balanceador de carga:

```
ssh ubuntu@$IP_LB
```

Instalando o Nginx:

```
sudo apt update  
sudo apt install nginx
```

Abra o arquivo de configuração do Nginx:

```
sudo nano /etc/nginx/sites-available/default
```

E adicione o seguinte:

```
server { location / { proxy_pass http://backend; } }  
upstream backend { server <IP_DJANG01>:8080; server <IP_DJANG02>:8080; }
```

Reinicie o Nginx:

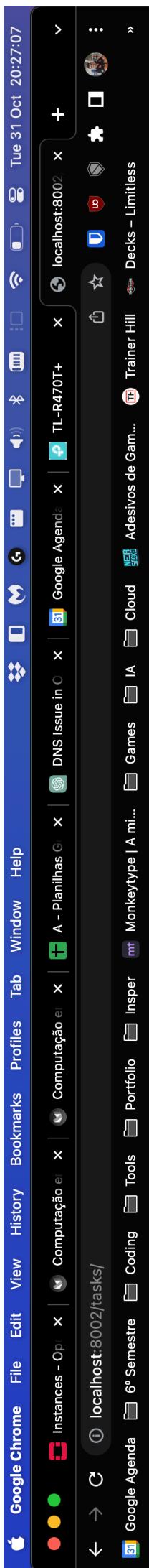
```
sudo service nginx restart
```

Testando a aplicação

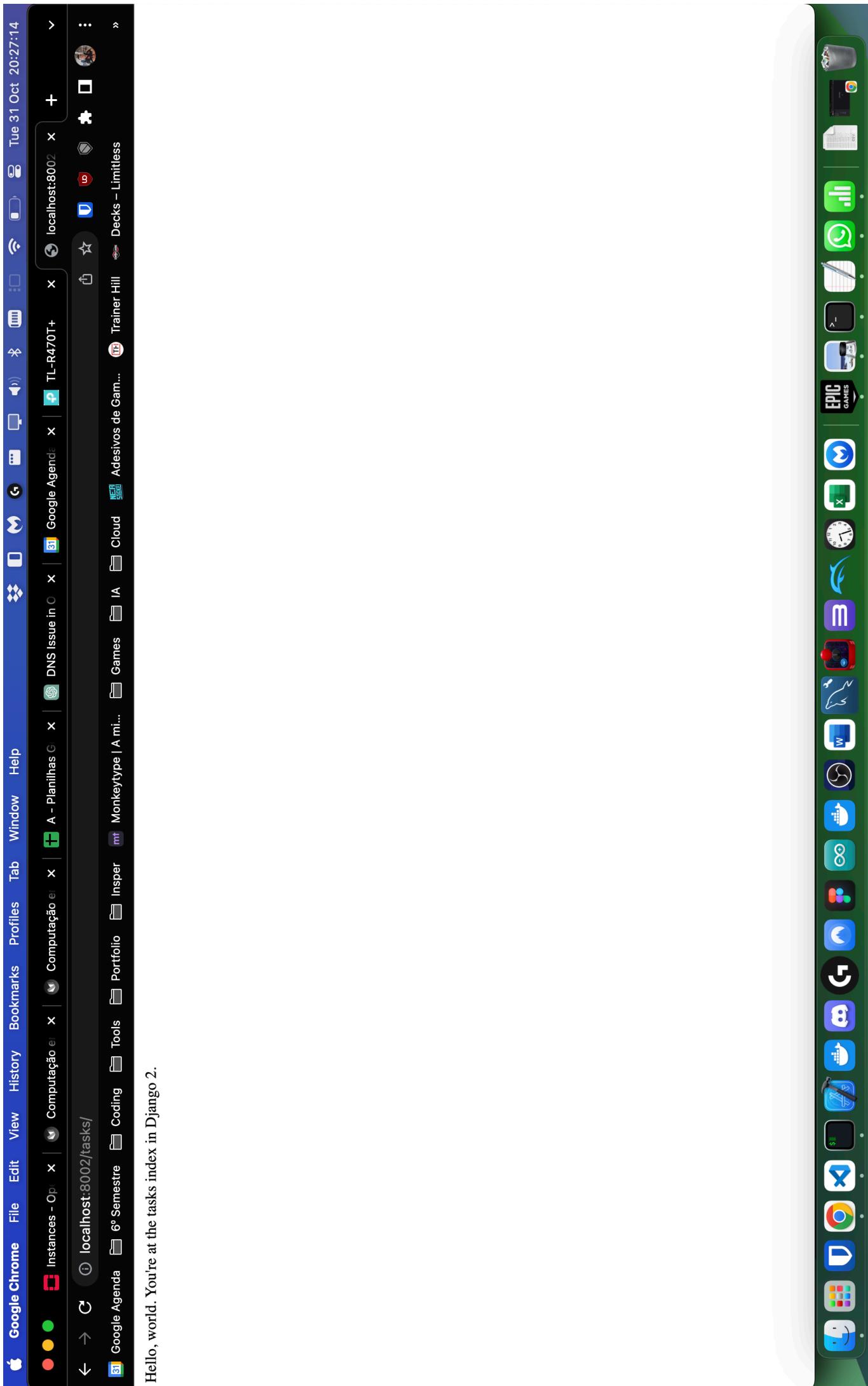
Por fim, vamos testar a aplicação. Para isso, fazemos o port forward do load balancer:

```
ssh cloud@roteador -L 8002:[IP_LB]:80
```

E, por fim, acessamos o endereço <http://localhost:8002/tasks/> no navegador.



Hello, world. You're at the tasks index in Django 1.



Google Chrome File Edit View History Bookmarks Profiles Tab Window Help

Network Topology - OpenStack | Computação em Nuvem - Con...

localhost:8002/admin/login/?next=/admin/ | 6º Semestre | Coding | Portfolio | Tools | Monkeytype! A mi...

Google Agenda | 6º Semestre | Cloud | IA | Games | Cloud | Adesivos de Gam...

Trainer Hill | Decks - Limitless

Tue 31 Oct 20:32:12

Django contrib admin default | +

Django site admin

Login | Django site admin

Computação em Nuvem - Con...

Monkeytype! A mi...

Trainer Hill

Decks - Limitless

Django administration

Username: cloud

Password:

Log in

The screenshot shows a cloud management interface with a navigation bar at the top. The main area displays a summary of various resources: RAM, VCPUs, Instances, Volume Storage, Security Group Rules, Networks, Floating IPs, Volumes, Volume Snapshots, Security Groups, and Ports. Each resource is represented by a pie chart indicating usage against limits. Below this, there's a section for 'Usage Summary' where users can select a time range to query usage data. A 'Download CSV Summary' button is also present. The interface is clean with a white background and orange accents.

