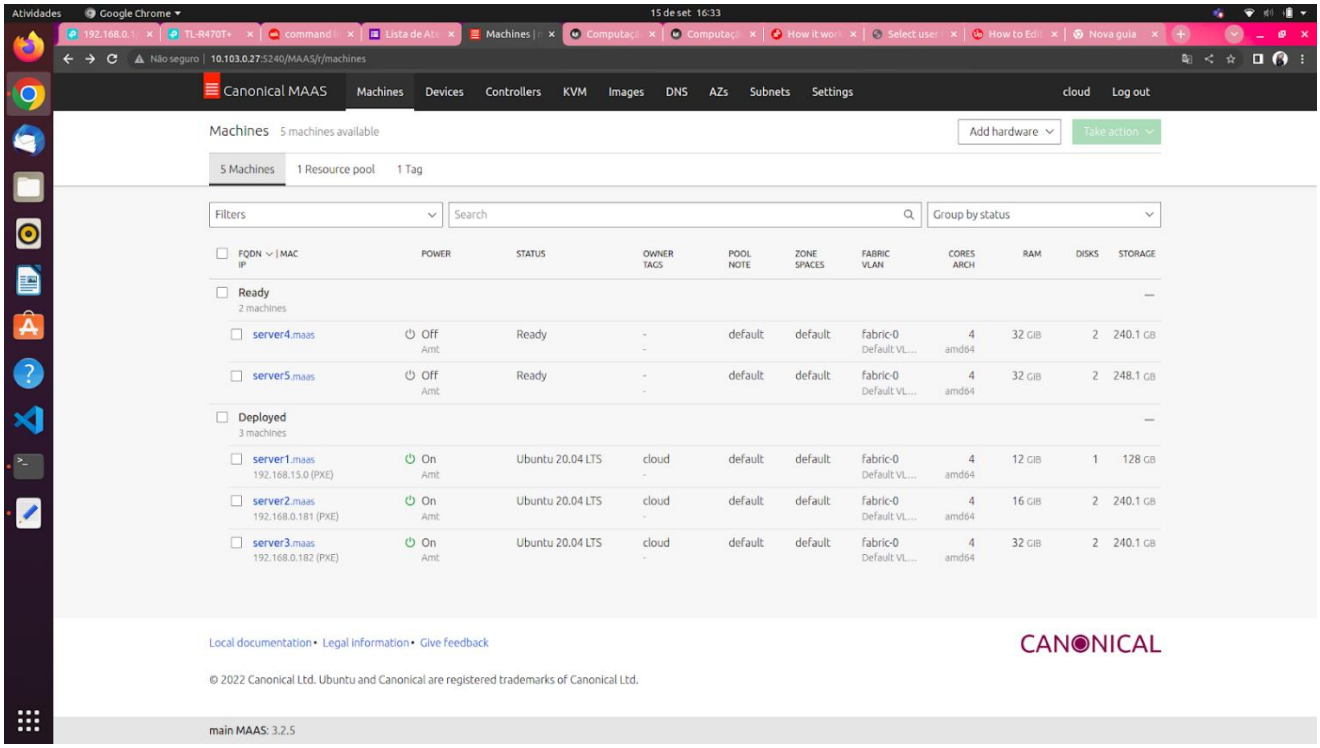
ROTEIRO 2

Alunos: Bernardo Cunha Capoferri e Lívia Sayuri Makuta

CHECK-POINT1:

1. Teste o acesso criando mais um túnel na string de conexão, caso esteja tudo certo, faça a tarefa abaixo

2. De um print da tela do Dashboard do MAAS com as 3 máquinas e seus respectivos IPs.



3. Mantendo o túnel aberto, rode o comando WGET para acesso a aplicação Django no server 2 e cole a saída do comando.

wget http://localhost:8001/admin

--2022-09-15 16:41:17-- http://localhost:8001/admin

Resolvendo localhost (localhost)... 127.0.0.1

Conectando-se a localhost (localhost)|127.0.0.1|:8001... conectado.

A requisição HTTP foi enviada, aguardando resposta... 301 Moved Permanently

Localização: /admin/ [redirecionando]

--2022-09-15 16:41:17-- http://localhost:8001/admin/

Reaproveitando a conexão existente para localhost:8001.

A requisição HTTP foi enviada, aguardando resposta... 302 Found

Localização: /admin/login/?next=/admin/ [redirecionando]

--2022-09-15 16:41:17-- http://localhost:8001/admin/login/?next=/admin/

Reaproveitando a conexão existente para localhost:8001.

A requisição HTTP foi enviada, aguardando resposta... 200 OK

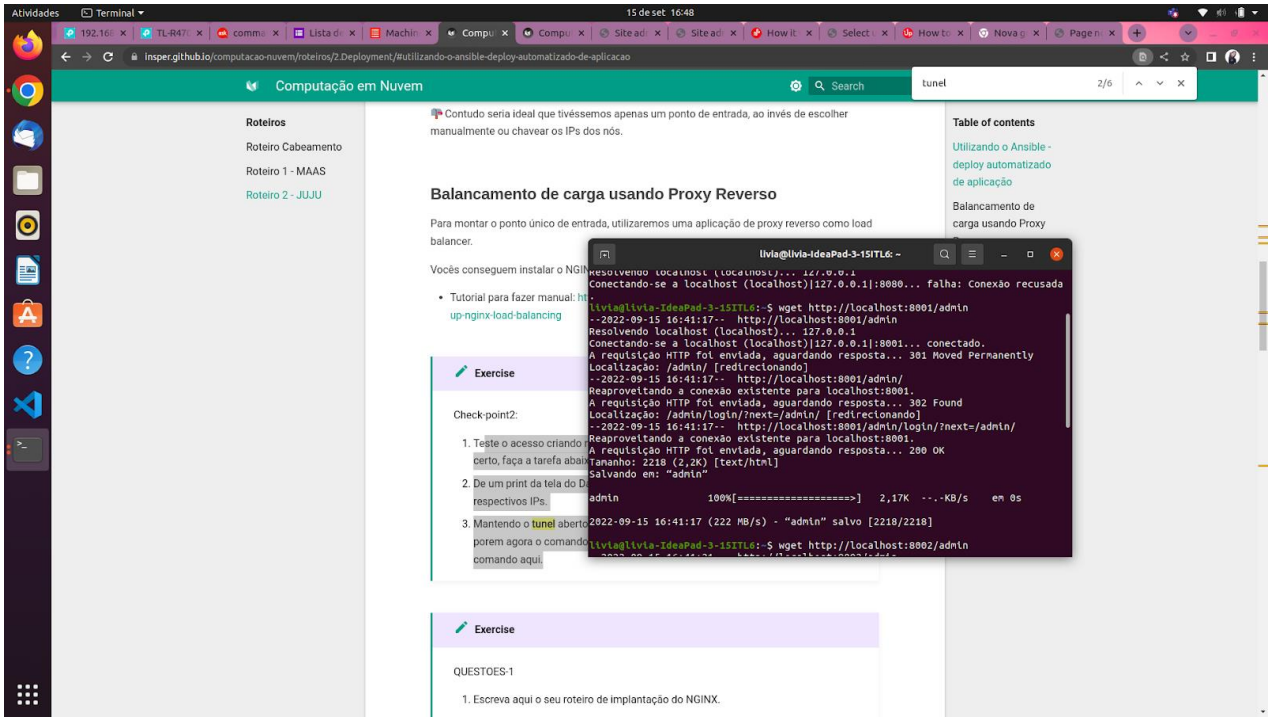
Tamanho: 2218 (2,2K) [text/html]

Salvando em: “admin”

admin 100%[===================>] 2,17K --.-KB/s em 0s

2022-09-15 16:41:17 (222 MB/s) - “admin” salvo [2218/2218]

Comando que também pode ser visto na imagem abaixo:



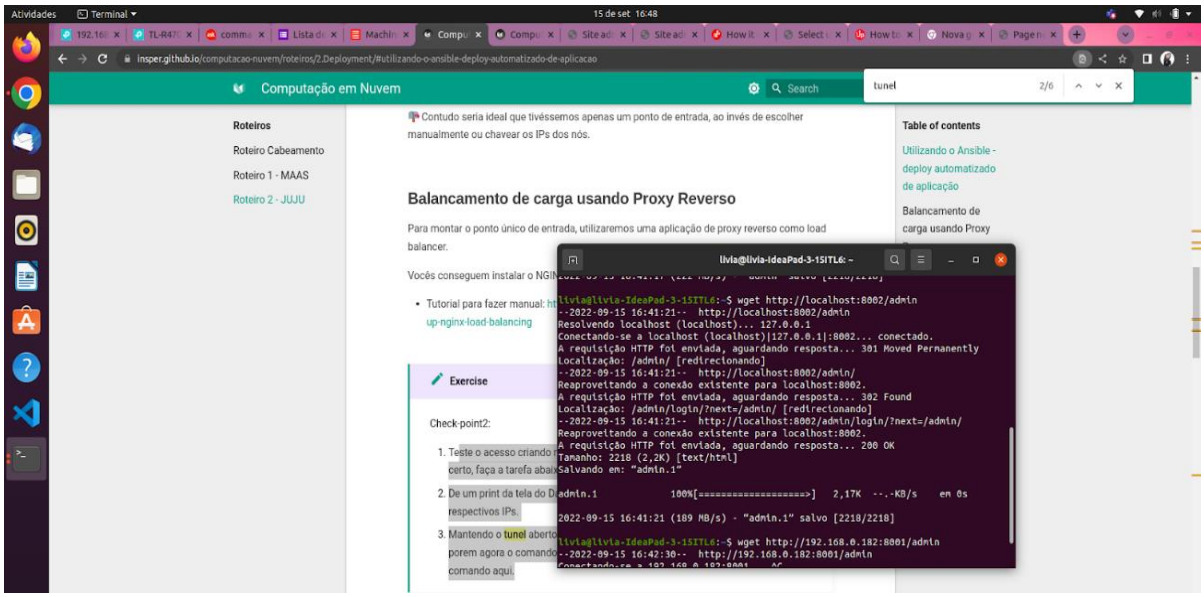
4. Mantendo o túnel aberto, rode o comando WGET para acesso a aplicação Django no server 3 e cole a saída do comando.

wget http://localhost:8002/admin
--2022-09-15 16:41:21-- http://localhost:8002/admin
Resolvendo localhost (localhost)... 127.0.0.1
Conectando-se a localhost (localhost)|127.0.0.1|:8002... conectado.
A requisição HTTP foi enviada, aguardando resposta... 301 Moved Permanently
Localização: /admin/ [redirecionando]
--2022-09-15 16:41:21-- http://localhost:8002/admin/
Reaproveitando a conexão existente para localhost:8002.
A requisição HTTP foi enviada, aguardando resposta... 302 Found
Localização: /admin/login/?next=/admin/ [redirecionando]
--2022-09-15 16:41:21-- http://localhost:8002/admin/login/?next=/admin/
Reaproveitando a conexão existente para localhost:8002.
A requisição HTTP foi enviada, aguardando resposta... 200 OK
Tamanho: 2218 (2,2K) [text/html]
Salvando em: “admin.1”

admin.1 100%[===================>] 2,17K --.-KB/s em 0s

2022-09-15 16:41:21 (189 MB/s) - “admin.1” salvo [2218/2218]

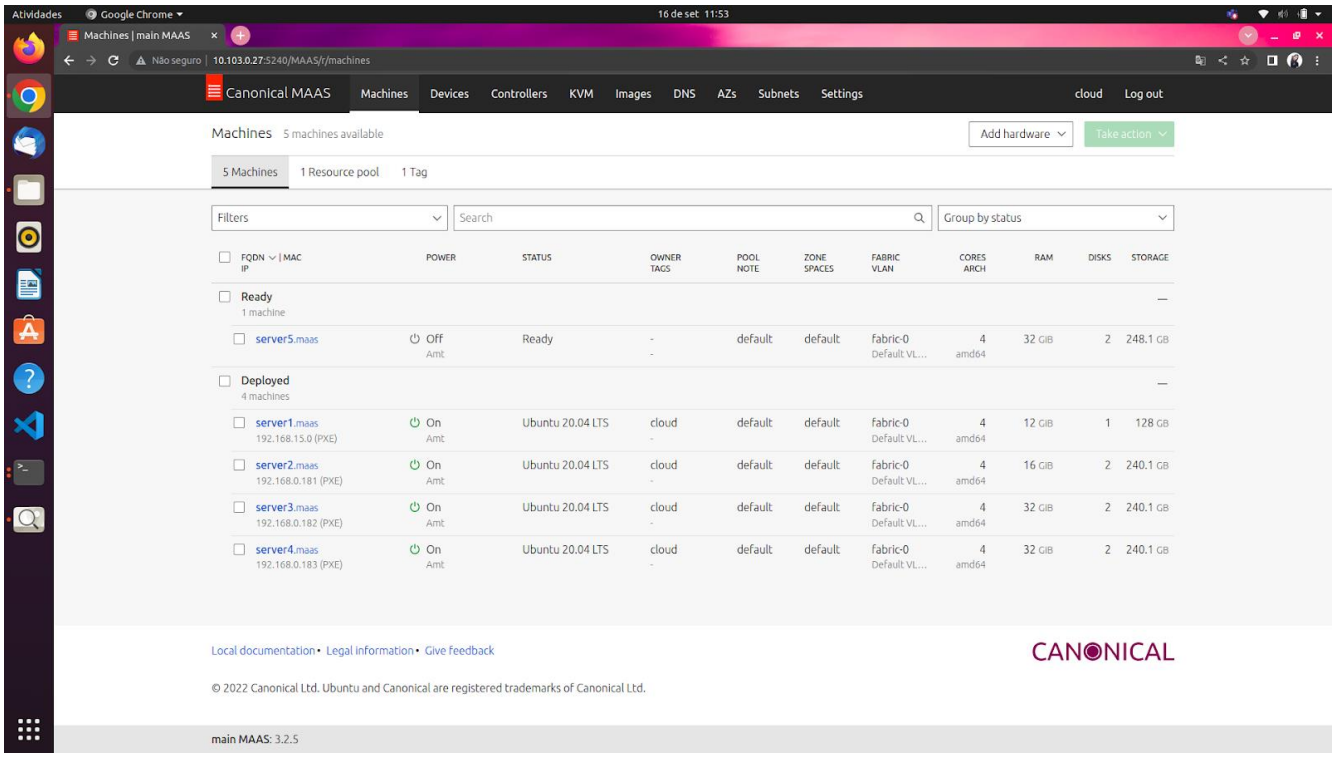
Comando que também pode ser visto na imagem abaixo:



CHECK-POINT2:

1. Teste o acesso criando mais um túnel na string de conexão, caso esteja tudo certo, faça a tarefa abaixo.

2. De um print da tela do Dashboard do MAAS com as 4 máquinas e seus respectivos IPs.



3. Mantendo o túnel aberto, rode o comando WGET para acesso a aplicação Django, porém agora o comando deve apontar para o load-balancer, cole a saída do comando aqui.

wget http://localhost:8007/admin

--2022-09-16 11:31:04-- http://localhost:8007/admin

Resolvendo localhost (localhost)... 127.0.0.1

Conectando-se a localhost (localhost)|127.0.0.1|:8007... conectado.

A requisição HTTP foi enviada, aguardando resposta... 301 Moved Permanently

Localização: /admin/ [redirecionando]

--2022-09-16 11:31:04-- http://localhost:8007/admin/

Reaproveitando a conexão existente para localhost:8007.

A requisição HTTP foi enviada, aguardando resposta... 302 Found

Localização: /admin/login/?next=/admin/ [redirecionando]

--2022-09-16 11:31:04-- http://localhost:8007/admin/login/?next=/admin/

Reaproveitando a conexão existente para localhost:8007.

A requisição HTTP foi enviada, aguardando resposta... 200 OK

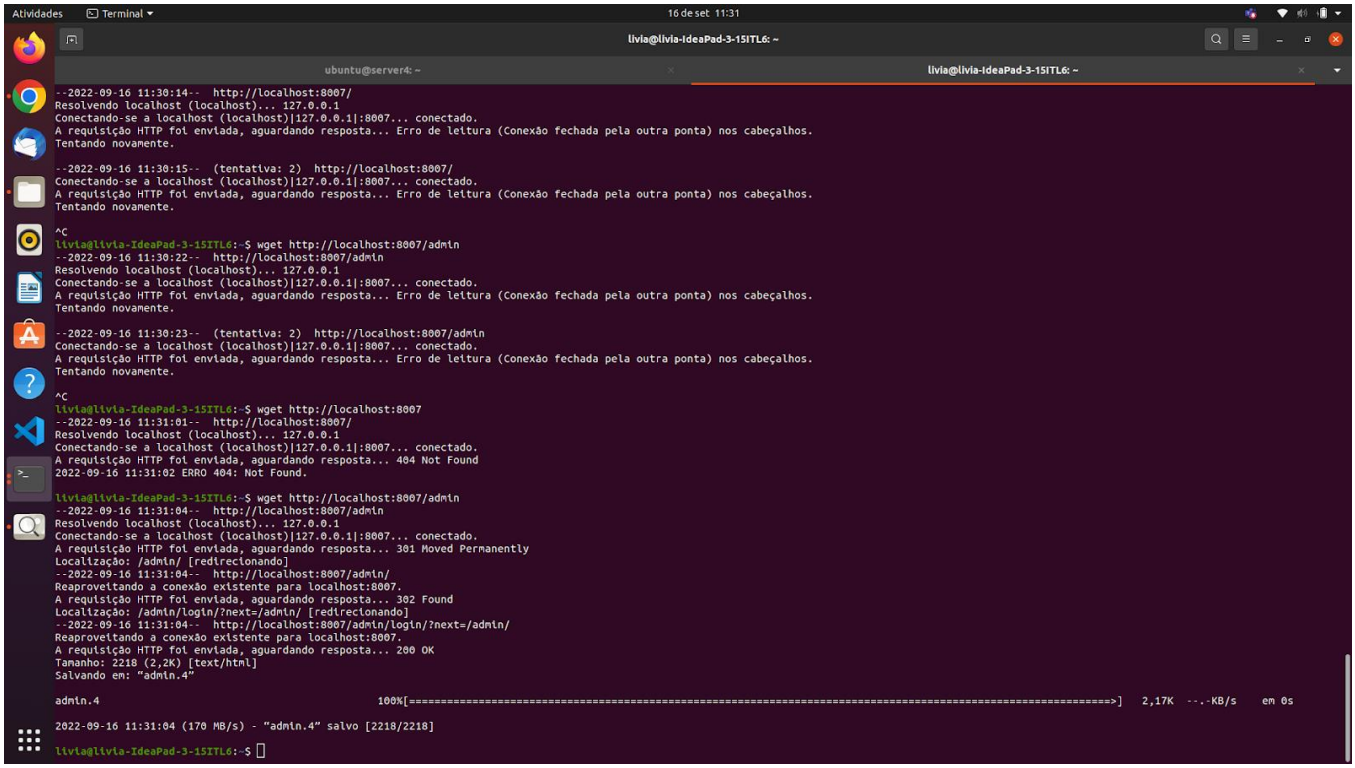
Tamanho: 2218 (2,2K) [text/html]

Salvando em: “admin.4”

admin.4 100%[==========>] 2,17K --.-KB/s em 0s

2022-09-16 11:31:04 (170 MB/s) - “admin.4” salvo [2218/2218]

Comando que também pode ser visto na imagem abaixo:



QUESTÕES-1

1. Escreva aqui o seu roteiro de implantação do NGINX.

Para implantar o NGINX, entre no servidor desejado - no nosso caso entramos no servidor 4 -, e instale o NGINX com o comando abaixo:

```
sudo apt-get install nginx
```

Depois de instalá-lo, vamos configurar um balanceador de carga *round robin*. Mas antes, o que é esse balanceador de carga *round robin*? Ele é uma maneira simples de distribuir requisições do cliente para um grupo de servidores em uma base cíclica, considerando que os servidores são similares e que aguentam cargas parecidas. Assim, através da configuração deste balanceador, se temos dois servidores (2 e 3) que vão receber as requisições do cliente, a primeira requisição seria mandada para o 2, já a segunda seria mandada para o 3, a quarta mandada novamente para o 2, e a quinta para o 3, e assim por diante.

Para configurar esse modelo, será necessário usar o módulo upstream do NGINX. A maneira de fazer isso é abrir um arquivo do NGINX que segue o seguinte caminho no seu computador:

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

Esse comando irá abrir um documento que precisa ser modificado. No início desse documento, ele vem com a definição de um server genérico que podemos comentar (basta seguir o padrão de comentário que está no documento) e depois no final desse documento podemos adicionar as seguintes linhas que incluem o módulo upstream:

```
upstream backend { server 192.168.0.181; server 192.168.0.182;}

server { location / { proxy_pass http://backend; } }
```

Lembrando que no nosso caso, adicionamos o servidor 2 e o servidor 3 como backend, já que esses servidores referenciam o backend do Django que subimos. Importante lembrar que os IPs que adicionamos foram os IPs desses dois servidores, então substitua-os para os IPs dos seus servidores que estão rodando o backend.

Por fim, reinicie o NGINX:

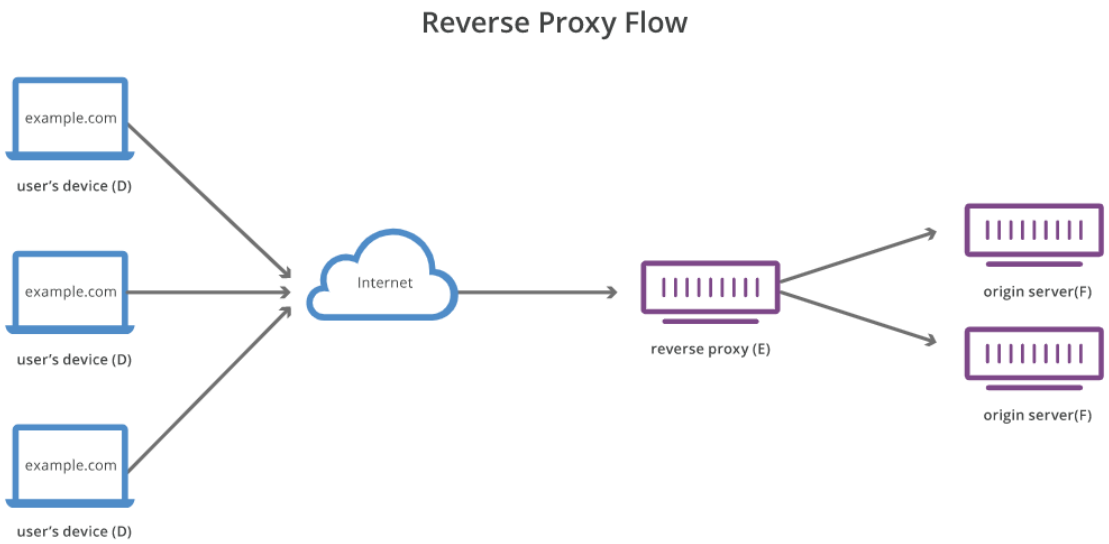
```
sudo service nginx restart
```

Pronto, agora você já tem o NGINX configurado e pronto para ser usado como um balanceador de carga.

2. Explique o conceito por traz do Reverse Proxy. Vocês já fizeram algo parecido?

O proxy reverso é um servidor que fica na frente de um ou mais servidores da web e que intercepta as requisições dos clientes. Dessa forma, quando um cliente envia solicitações para o servidor de origem do site, essas solicitações vão ser interceptadas na borda de rede (onde o dispositivo ou a rede local que contém o dispositivo se comunicam com a internet, por exemplo, o roteador do usuário é uma borda de rede) pelo servidor proxy reverso. Assim, é esse servidor que vai receber as requisições e enviar as respostas para o servidor de origem.

Vale lembrar que um servidor proxy reverso é diferente do servidor proxy de encaminhamento (forward proxy em inglês). Isso porque o proxy de encaminhamento fica na frente do cliente e impede que nenhum servidor de origem se comunique diretamente com aquele cliente, já o proxy reverso fica na frente do servidor de origem e garante que nenhum cliente se comunique diretamente com esse servidor de origem. Isso pode ser ilustrado pela imagem abaixo retirada do site: https://www.cloudflare.com/pt-br/learning/cdn/glossary/reverse-proxy/ .



E fizemos algo parecido na aula que é exatamente o load balancer (balanceador de carga em português) - já que ele vai fazer o gerenciamento das solicitações do cliente para distribuir o tráfego de entrada entre os servidores a fim de evitar que um único servidor fique sobrecarregado. Mas além disso, antes tivemos que configurar a NAT para poder acessar o Maas sem estar conectado com o cabo na rede interna, e isso de certa forma é parecido com o proxy reverso, já que nesse caso o roteador NAT através das portas vai saber qual servidor fez a requisição e irá mandar a resposta da solicitação para ele.

3. Na instalação toda, você alocou 4 máquinas físicas, duas para o Django, uma para o Postgres e uma para o NGINX.

Isso, nós alocamos uma máquina (o server1) para ser o banco de dados com o Postgres, e duas máquinas: o server2 e o server3, para o framework Django e uma máquina (o server4) para o NGINX.

4. Considerando que é um Hardware próprio, ao contrário do modelo "Public Cloud", esse modelo de arquitetura é bom ou ruim em termos de custos?

Em termos de custos é ruim, porque é quem possui o hardware que precisaria arcar com manutenção e possíveis problemas que podem surgir, além de que se o proprietário quiser expandir seus recursos na nuvem, ele que terá que comprar mais equipamentos, instalar o hardware, etc.

Em contrapartida, uma nuvem pública provê os principais serviços de infraestrutura para dar suporte à escalabilidade, segurança e resiliência das cargas de trabalho que incluem redes, computação, armazenamento, segurança e gerenciamento, de maneira a atender a demanda de quem contrata sem os custos de propriedade. Além disso, a empresa que é contratada para prover esses serviços de nuvem possui times com especialistas preparados para isso. Ou seja, dependendo, é muito melhor contratar um modelo de “Public Cloud” do que utilizar um hardware próprio.

QUESTÕES-2

1. Qual o S.O. utilizado na máquina Juju? Quem o instalou?

O sistema operacional utilizado na máquina Juju foi o Ubuntu 20.04. LTS. E quem o instalou foi o Maas que utilizou o sistema operacional que estava no servidor main.

2. O Juju requisitou uma máquina automaticamente para o MaaS via API. Por quais motivos ter uma API é importante para uma aplicação?

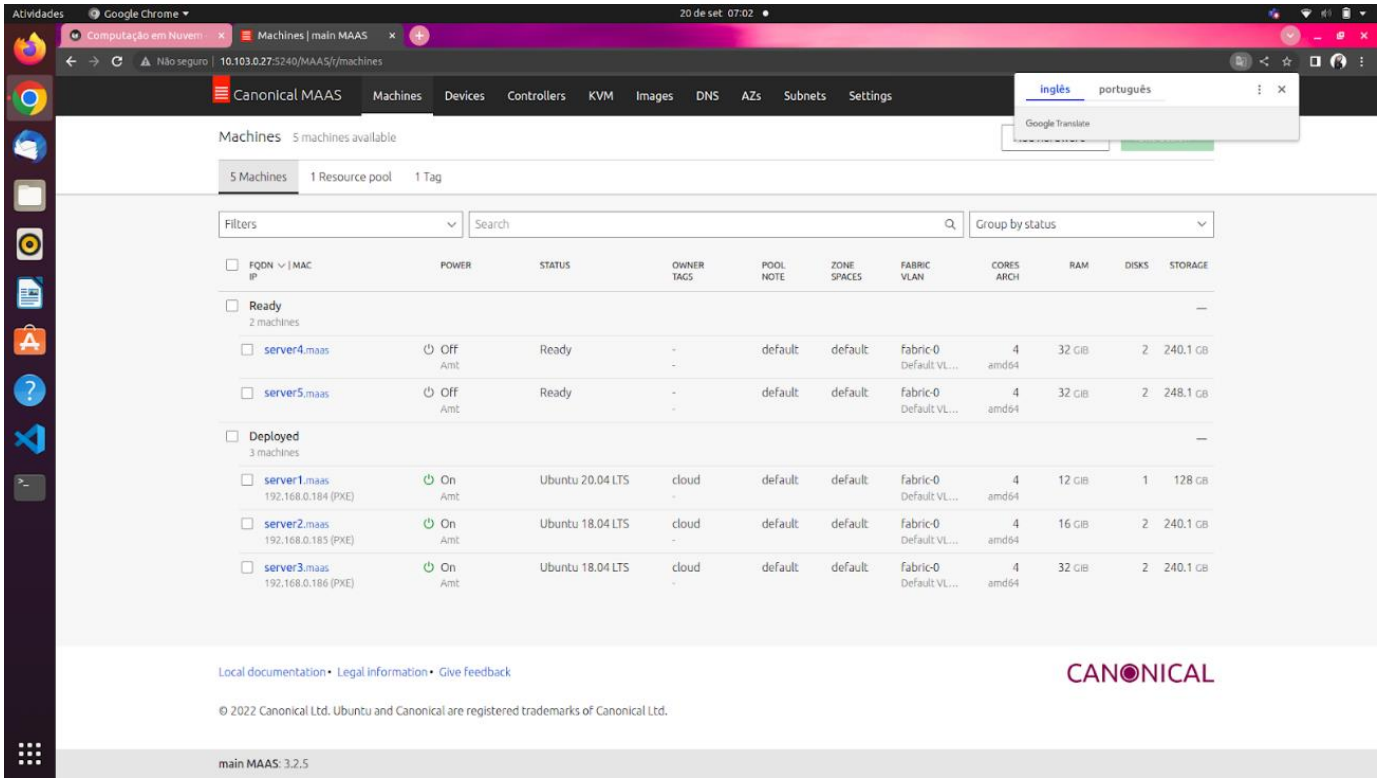
Porque o API (*Application Programming Interface*) é um conjunto de rotinas e padrões de programação para acessar um aplicativo ou plataforma na web. E a API é criada por empresas exatamente para que outros criadores de softwares possam desenvolver produtos associados ao seu serviço, ou seja, é uma forma de comunicação entre dois serviços ou programas. Por exemplo, o Google Maps possui uma API que é utilizada por muitos outros sites e aplicações que utilizam dos seus dados para usar esse serviço da maneira que preferirem - um hotel por exemplo pode colocar dentro do próprio site um mapa do Google com sua localização para que o usuário verifique qual o melhor caminho para chegar lá.

Dessa maneira, as APIs permitem que um usuário use um aplicativo, software ou mesmo uma planilha para consultar, alterar e armazenar dados de diversos sistemas, sem que ele precise acessar isso diretamente. Então, o intuito de ter uma API é trocar dados entre diferentes sistemas que muitas vezes serve para automatizar processos manuais e/ou permitir criar funcionalidades.

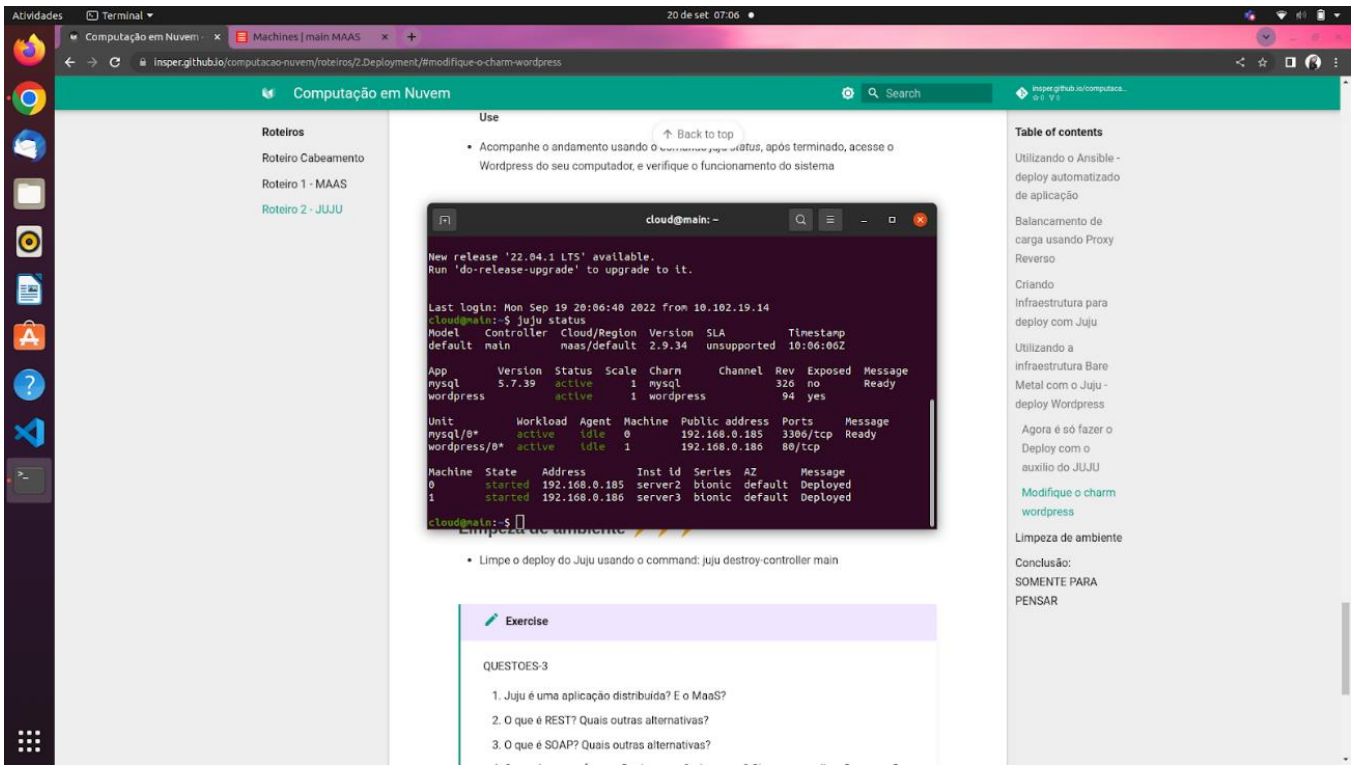
Assim, no caso do Juju, ele consegue acessar os dados que estão no Maas, como os dados das máquinas e servidores registrados nele, para poder fazer o deploy das máquinas - sabendo qual máquina ele vai alocar espaço para aplicação, onde ele vai subir a aplicação, etc. Dessa forma, o Juju é como o cliente, e vai fazer requisições via API para o Maas pedindo uma máquina que está no estado “pronta” para poder fazer o deploy do controlador do Juju. Então o Juju, com essa comunicação via API, depois de conferir quais máquinas vai usar, vai pedir pro Maas fazer o deploy em uma máquina (que está disponível) do serviço que o usuário pediu.

CHECKPOINT-3

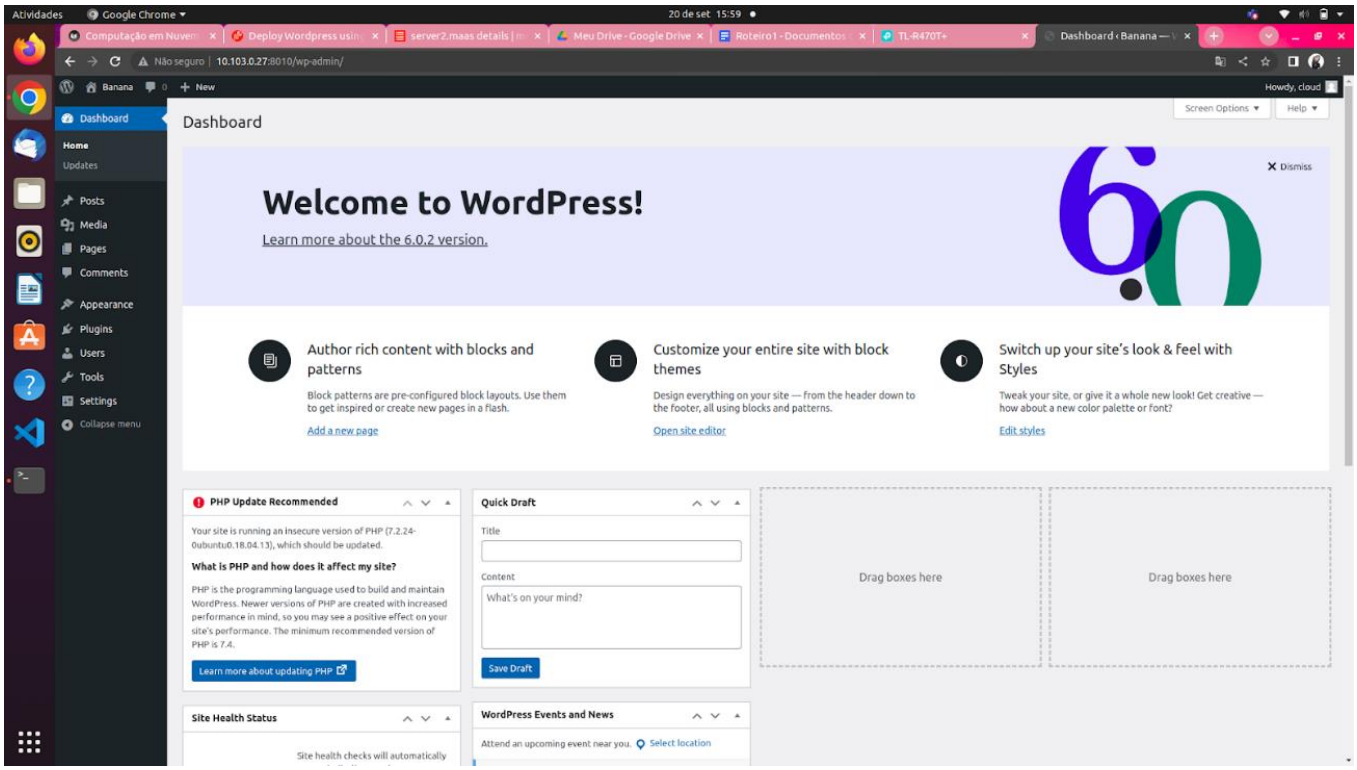
1. De um print da tela do Dashboard do MAAS com as máquinas e seus respectivos IPs.



2. Rode o comando "juju status" depois que o Wordpress estiver "active" e cole aqui.



3. **Entre no Dashboard da aplicacao Wordpress. Para isso acesse a rede interna do KIT, plugando o cabo diretamente no switch.**
4. **Instale a aplicação !!!! ANOTE A SENHA e coloque o seu e-mail.**
5. **De um print da tela do Dashboard do Wordpress que foi implantado.**



QUESTÕES-3

1. **Juju é uma aplicação distribuída? E o MaaS?**

Uma aplicação distribuída é qualquer aplicação que foi projetada para executar em mais de uma máquina, assim, um grupo de processos é executado em diferentes máquinas e trabalha de maneira coordenada e cooperativa para realizar uma determinada tarefa.

Assim, no caso da JUJU, podemos dizer que não é uma aplicação distribuída, isso porque ele apenas dá o deploy na aplicação, mas não executa essa aplicação em várias máquinas ao mesmo tempo – é como se fosse um motoboy, pois ele apenas entrega a requisição para quem de fato vai executar toda a instalação e rodar a aplicação.

Já o Maas faz o gerenciamento das características de hardware das máquinas que foram cadastradas nele e permite subir sistemas operacionais nessas máquinas. Além disso, também é possível subir aplicações como o Django. Nesse caso o Maas é uma aplicação distribuída, e isso porque ele roda em mais de uma máquina de maneira conjunta e executa de fato essas aplicações em várias máquinas concomitantemente. Por exemplo, o Maas roda o banco de dados em uma máquina, o Django em outra, o Juju em outra, e o Wordpress em outra, ou seja, executa várias aplicações diferentes em máquinas distintas ao mesmo tempo.

2. O que é REST? Quais outras alternativas?

O REST (*Representational State Transfer* em inglês ou Transferência de Estado Representacional em português) é um conjunto de princípios e definições que são necessárias para criar um projeto com interfaces definidas solidamente. E essa arquitetura permite a comunicação entre aplicações e é caracterizada pela separação de interesses do cliente e servidor e por serem “sem estado”.

Em relação a primeira característica, isso significa que a implementação do cliente e do servidor podem ser feitas de maneira independente sem que um saiba do outro, ou seja, o código do lado do cliente pode ser mudado sem que isso afete a operação do servidor e vice-versa - desde que o outro lado saiba qual formato de mensagem mandar para o outro. Com isso há melhoras na flexibilidade de interfaces nas plataformas e melhora na escalabilidade por simplificar os componentes do servidor.

Por sua vez, ser “sem estado”(stateless em inglês) significa que o servidor não precisa saber nada sobre qual estado o cliente está e vice-versa. Dessa forma, tanto o servidor quanto o cliente podem entender qualquer mensagem recebida mesmo sem ver as mensagens anteriores. E essa restrição é imposta por meio do uso de recursos ao invés de comandos, sendo que os recursos são os nomes na Web que descrevem qualquer objeto, documento ou coisa que seja necessário armazenar ou enviar para outros serviços.

Um exemplo é a arquitetura Web, e funciona da seguinte maneira: quando abrimos o navegador o REST vai estabelecer uma conexão TCP/IP com o servidor de destino, e irá enviar uma requisição GET HTTP a partir do endereço que foi informado, por sua vez, do outro lado o servidor vai enviar uma resposta HTTP para o navegador. Esse processo então

é repetido várias vezes. A questão é que o REST é mais flexível, leve e permite melhorar os resultados quando a metodologia ágil é aplicada.

E outras alternativas ao REST são: GraphQL, o gRPC e o Apache Kafka.

3. O que é SOAP? Quais outras alternativas?

O SOAP (*Simple Object Access Protocol* ou Protocolo de Acesso de Objetos Simples) é um protocolo padrão de mensagens baseado em XML que permite a troca de informações entre computadores, que foi projetado originalmente para que fosse possível estabelecer a comunicação para aplicações desenvolvidas em diferentes linguagens. Assim, ele define o conteúdo da mensagem e informa como processá-la e determina o conjunto de regras de codificação para os tipos de dados.

E como é um protocolo, ele possui regras integradas que aumentam sua complexidade e sobrecarga, o que também desacelera o carregamento das páginas. Mas, ao mesmo tempo, o SOAP proporciona conformidade integrada o que inclui segurança, consistência, isolamento e durabilidade, o que é importante quando se busca transações confiáveis de bancos de dados.

Outras alternativas ao SOAP são: BEEP (Blocks Extensible Exchange Protocol), CTS (Canonical Text Services Protocol), E-Business XML, JSON-RPC, JSON-WSP, entre outros.

4. O que é e o que faz um *Deployment Orchestrator*? Cite outras aplicações que não usamos nesse roteiro.

Um orquestrador de deploy é que vai automatizar o processo de colocar no ar alguma aplicação que teve seu desenvolvimento concluído, ou seja, ao invés de ter que subir a aplicação ou até mesmo a imagem de um sistema operacional em cada máquina, o orquestrador que terá esse papel, realizando a coordenação, a organização e subindo ao ar de maneira mais rápida, automatizada e reduzindo as falhas. Por exemplo, quando subimos o Wordpress e o mysql, não precisamos fazer a configuração de senha, login, entre outras coisas, tudo isso foi o Juju que fez.

Outras aplicações de um *Deployment Orchestrator* podem ser vistas como por exemplo no MidVision RapidDeploy, que não só orquestra implantações de aplicativos, como

também configura o middleware e atualizações de banco de dados, e isso não usando só os recursos de automação do produto deles, mas também deixando quem contrata livre para executar rotinas de automação específicas que foram roteirizadas internamente. Além disso, um orquestrador também pode permitir que várias aplicações sejam subidas de uma vez, definindo a ordem dos deploys e/ou as versões dos componentes que precisam ser lançados juntos, e se eles devem ser executados em série ou em paralelo.

E isso tudo que foi citado abrange um conceito de orquestração em nuvem (“Cloud Orchestration”), já que vai além da automação promovendo também a coordenação entre várias tarefas automatizadas. Em uma empresa, isso adiciona funções como redundância, escalabilidade, failover e failback, além de gerenciar dependências e agrupá-las em um único pacote, o que reduz a carga geral de TI.

5. Como é o o processo de interação entre um servidor API REST e uma client application?

O cliente da aplicação (por exemplo, o navegador da web) faz requisições para a API de maneira a obter alguma informação ou alterar algo dentro da sua aplicação, como o conteúdo de sua página. As informações solicitadas são então mandadas de volta ao navegador e exibidas na tela.

E essa API é chamada de REST (*Representational State Transfer)* pois ela segue a arquitetura REST, que como visto anteriormente em outra questão, é um conjunto de diretrizes que o software pode usar para se comunicar e tornar as integrações simples e escaláveis - como o próprio nome já diz, quando um cliente faz uma requisição de um recurso usando uma API REST, o servidor transfere de volta o estado atual do recurso em uma representação padronizada.

Um exemplo para ajudar a compreensão seria: imagine que quero criar um programa que se integre ao Youtube, o meu programa - que é o cliente - pode solicitar à API REST do Youtube informações sobre um vídeo específico (um recurso). A API do Youtube responderá a solicitação com o estado do recurso, que pode ser: o nome do vídeo, a data da publicação, a contagem das visualizações, entre outros. E esse recurso será empacotada de maneira que meu programa entenda e consiga analisar e usá-lo de maneira rápida. Por fim, esse meu programa também pode postar um vídeo (adicionar um novo recurso) ao meu canal pessoal no Youtube através dessa API.

6. Defina Aplicação Distribuída, Alta Disponibilidade e *Load Balancing*?

Uma aplicação distribuída é qualquer aplicação que foi projetada para executar em mais de uma máquina, assim, um grupo de processos é executado em diferentes máquinas e trabalha de maneira coordenada e cooperativa para realizar uma determinada tarefa.

Por sua vez, alta disponibilidade é a capacidade de garantir que serviços utilizados serão continuados, mesmo quando algo como o hardware ou o software falhem ou a energia seja interrompida. Em outras palavras: as funcionalidades do sistema não podem ser interrompidas, e é por isso que soluções de segurança de redes usam esse conceito. Trazendo isso para a implementação de um firewall, por exemplo, temos que mesmo se haja interrupção de energia e o hardware perca suas funções, vai existir um sistema em paralelo que possui configurações idênticas ao firewall original que estará pronto para assumir a operação.

Por fim*, Load Balancing* diz respeito à distribuição eficiente do tráfego de rede de entrada em um grupo de servidores backend, que também são chamados de server farm ou server pool. E esse conceito é interessante porque atualmente os sites precisam lidar com alto tráfego de solicitações simultâneas de usuários ou clientes e retornar o recurso que está sendo solicitado (uma imagem, um vídeo, um texto, etc) rapidamente e de maneira confiável. Sendo assim, é aí que entra o balanceador de carga, que é como se fosse um policial no trânsito, que fica na frente dos seus servidores e que distribui as solicitações entre os servidores para que essas sejam atendidas o mais rápido possível garantindo que nenhum dos servidores seja sobrecarregado. Assim, se um servidor ficar inativo, por exemplo, o balanceador de carga irá redirecionar o tráfego para os servidores online restantes, e caso mais um servidor seja adicionado ele também começa automaticamente a enviar solicitações para ele. Isso tudo garante alta disponibilidade e confiabilidade, além de flexibilidade e eficiência.