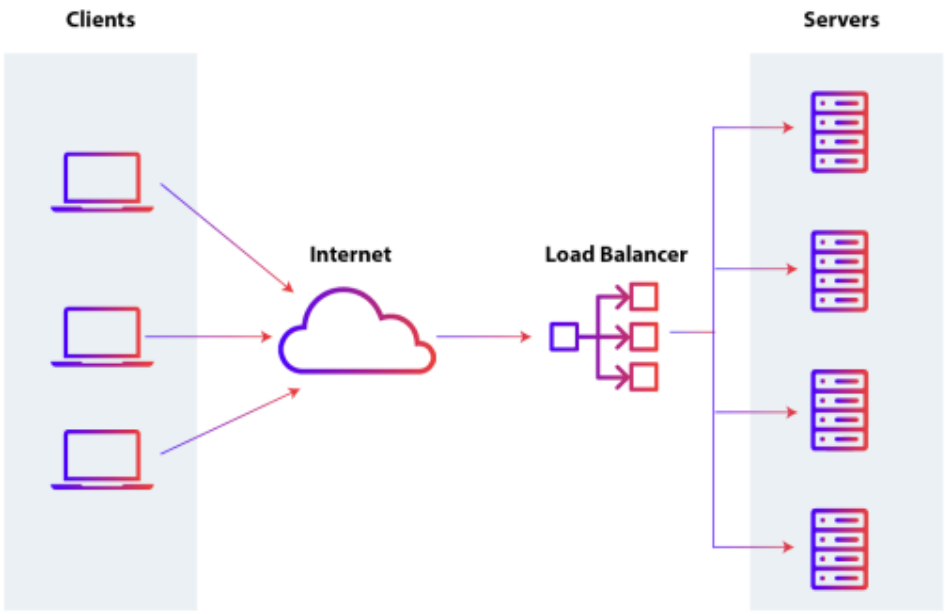
Instituto Politécnico de Tomar

Relatório de Projeto Final A e B

Tema : **Cloud Computing e Virtualization**



Por : **José Ngoma Júnior**

Aluno número: **25162**

1. Introdução

Este projeto visa criar uma infraestrutura de balanceamento de carga em em dois ambientes, projeto A utilizando Vagrant e projeto B utilizando contêineres Docker e ambos usando servidores Nginx para a distribuição de solicitações entre múltiplos servidores PHP. O objetivo principal é garantir a distribuição de carga, alta disponibilidade e escalabilidade dos serviços PHP em um ambiente local.

Problema a ser resolvido

No cenário de aplicação web, a alta demanda de solicitações pode sobrecarregar um único servidor, resultando em lentidão ou indisponibilidade. Para mitigar este problema, utilizamos balanceadores de carga que distribuem as solicitações entre vários servidores backend, garantindo uma utilização equilibrada dos recursos e melhorando a disponibilidade e performance do sistema.

Solução Proposta

Projeto A

Para o projeto A, a solução proposta envolve o provisionamento de duas máquinas php controladas por um servidor rodando o Nginx que efectua o balanceamento de carga carga de modo a garantir a alta disponibilidade e uma melhor performance em um ambiente Vagrant com os seguintes componentes:

- Dois servidores PHP servindo páginas web.

- Um balanceador de carga Nginx distribuindo as solicitações entre os servidores PHP.

Provisionamento

O provisionamento é o processo de configurar automaticamente uma máquina virtual (VM) após sua criação, garantindo que ela esteja em um estado desejado e pronta para uso. O Vagrant permite o uso de diversas ferramentas de provisionamento para instalar pacotes, atualizar sistemas, configurar serviços e executar scripts de configuração. O projeto em questão já possui uma configuração sólida e scripts prontos para serem executados com o simples comando “vagrant up".

Escalabilidade

Para lidar com o aumento de tráfego, podemos utilizar técnicas de escalabilidade vertical e horizontal.

- **Escalabilidade vertical**: Aumentar os recursos do servidor, como mais processamento e mais RAM.

- **Escalabilidade horizontal**: Adicionar mais servidores para distribuir o tráfego.

Na minha solução, para escalabilidade vertical, adicionei manualmente mais RAM a cada máquina. Para escalabilidade horizontal, utilizei o Nginx para balanceamento de carga entre dois servidores.

Balanceamento de Carga

O balanceamento de carga é um método de distribuição de tráfego, atuando geralmente como um proxy que permite a distribuição de tráfego entre diferentes servidores. Um proxy é um intermediário entre um recurso e uma petição.

Funcionamento do Sistema

Nesta parte do projeto, tenho três instâncias executando no meu Hypervisor, que neste caso é o VirtualBox. Uma dessas instâncias atua como balanceador de carga, enquanto as outras duas são os receptores das petições.

Quando as petições são feitas ao servidor no endereço http://192.168.44.10 o balanceador de carga se encarrega a distribuir a carga entre os servidores. Para isso foi necessário provisionar ou configurar as duas máquinas receptoras e instalar o Nginx na máquina que realizará o balanceamento de carga. Também foi preciso editar o arquivo de configuração do servidor Nginx.

A estrutura do projeto é a seguinte:

Passos para Configuração

1. **Acessar a máquina do servidor**:

- Executar o comando **vagrant ssh** para liberar o shell do Linux e executar comandos no bash.

- É preciso certificar que o Hypervisor esteja rodando. Caso contrário, inicializá-lo com o comando **vagrant up**.

2. **Criação do novo arquivo de configuração**:

- sudo touch /etc/nginx/conf.d/load-balancer.conf

3. **Remoção do bloqueio virtual padrão**:

- sudo rm -r /etc/nginx/sites-enabled/default

4. **Reiniciar o Nginx**:

- sudo /etc/init.d/nginx restart

5. **Modificar o arquivo de configuração**:

- sudo nano /etc/nginx/conf.d/load-balancer.conf

```
upstream nodes {
    server 192.168.44.11;
    server 192.168.44.12;
}

server {
    listen 80;
    location / {
        proxy_pass http://nodes;
    }
}
```

6. **Guardar as alterações e reiniciar o Nginx**:

- sudo /etc/init.d/nginx restart

Com estas configurações feitas teremos o nosso balanceador de carga funcionando, alternando entre os dois servidores, usando o algoritmo (padrão) Round Robin.

Projeto B

O balanceamento de carga é frequentemente utilizado entre vários serviços replicados para se alcançar escalabilidade. O balanceador de carga atua como uma camada intermediária entre o cliente e os servidores, distribuindo as requisições de forma eficiente. Existem diversos algoritmos para distribuição de requisições, sendo o mais comum o algoritmo Round Robin. Este algoritmo distribui as requisições de forma sequencial entre os servidores disponíveis.

Além do Round Robin, também podemos configurar o balanceador de carga para que todas as requisições de um cliente sejam direcionadas para um único servidor utilizando o hash. Essa abordagem é útil quando os serviços mantêm estados, garantindo que todas as requisições de um mesmo cliente sejam processadas pelo mesmo servidor. Outros algoritmos disponíveis distribuem as requisições para o servidor com menos conexões ativas, otimizando o uso dos recursos.

Com o balanceador de carga, busca-se sempre maior escalabilidade e robustez. Dessa forma, se um dos servidores falhar, os demais podem continuar a operar sem que o usuário perceba qualquer interrupção no serviço.

Proxy Reverso

Um exemplo de proxy reverso pode ser ilustrado com uma página web que possui as URLs `lojadosaber/oportunidades` e `lojadosaber/clientes`. Podemos configurar o proxy para direcionar as requisições para diferentes servidores, ao invés de sobrecarregar um único servidor. Essa técnica é amplamente utilizada em arquiteturas de microsserviços, onde as lógicas de negócios são separadas entre diferentes serviços.

- **Least Connections (`least_conn`)**: Este parâmetro envia o tráfego para o servidor com menos conexões ativas.
- **IP Hash (`ip_hash`)**: Este parâmetro cria um hash com o IP da requisição, garantindo que as requisições de um mesmo cliente sejam sempre direcionadas para o mesmo servidor.

Foco do projeto

Nesta segunda parte do projeto, o foco foi resolver questões relacionadas à escalabilidade utilizando balanceamento de carga e alta disponibilidade. Também foi utilizado o Azure Container Registry para a publicação e gerenciamento de contêineres.

Solução Proposta

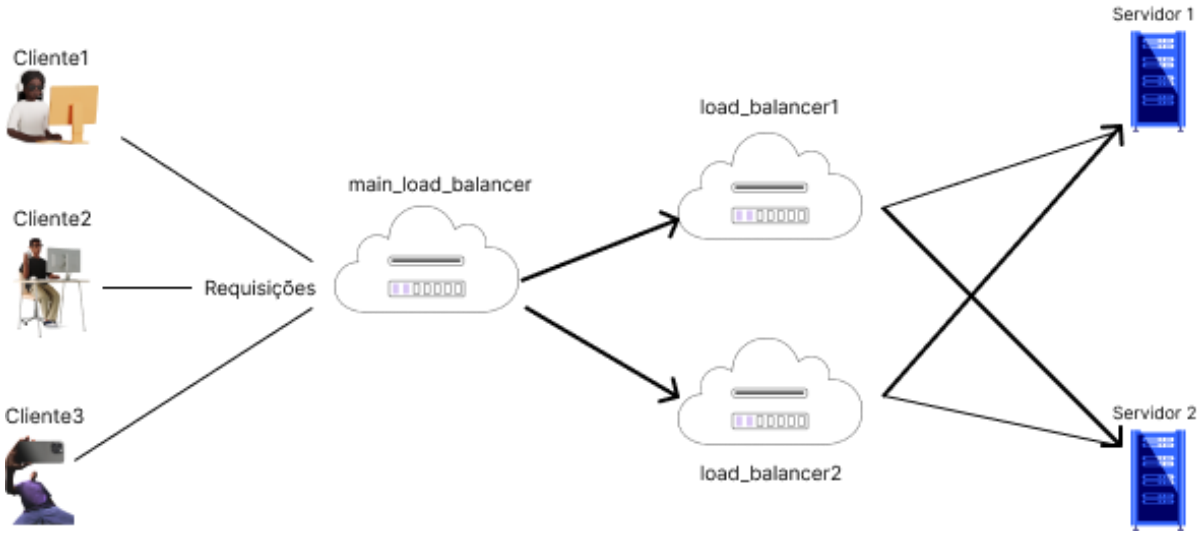
A solução proposta envolve a configuração de um ambiente Docker com os seguintes componentes:

- **Dois servidores PHP** (projectb1 e projectb2), servindo páginas web simples. Onde projectb1 e projectb2 são imagens prontas com todas as configurações necessárias e a configuração de base de dados.

- **Dois balanceadores de carga Nginx** (load_balancer1 e load_balancer2), cada um distribuindo as solicitações entre os servidores PHP.

- Um balanceador de carga principal Nginx(main_load_balancer), distribuindo as solicitações entre os dois balanceadores de carga Nginx.

A estrutura do projeto é a seguinte:



4. Configuração e Implementação

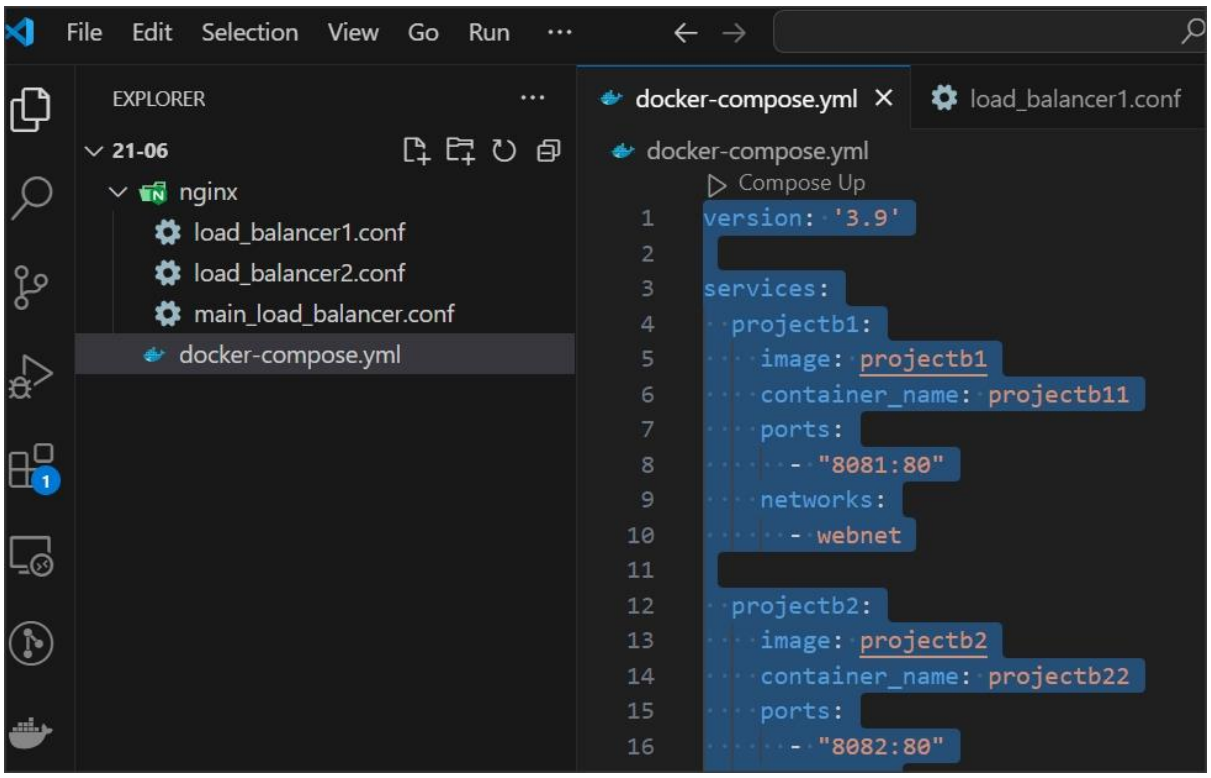
4.1. Docker Compose

O arquivo **docker-compose.yml** define todos os serviços e suas respectivas redes, garantindo que todos os contêineres possam se comunicar:

```yaml
version: '3.9'

services:
  projectb1:
    image: projectb1
    container_name: projectb11
    ports:
      - "8081:80"
    networks:
      - webnet

  projectb2:
    image: projectb2
    container_name: projectb22
    ports:
      - "8082:80"
    networks:
      - webnet

  load_balancer1:
    image: nginx:latest
    volumes:
      - ./nginx/load_balancer1.conf:/etc/nginx/nginx.conf
    ports:
      - "8084:80"
    depends_on:
      - projectb1
      - projectb2
    networks:
      - webnet

  load_balancer2:
    image: nginx:latest
    volumes:
      - ./nginx/load_balancer2.conf:/etc/nginx/nginx.conf
    ports:
      - "8083:80"
```

```
    depends_on:
      - projectb1
      - projectb2
    networks:
      - webnet

  main_load_balancer:
    image: nginx:latest
    volumes:
      - ./nginx/main_load_balancer.conf:/etc/nginx/nginx.conf
    ports:
      - "80:80"
    depends_on:
      - load_balancer1
      - load_balancer2
    networks:
      - webnet

networks:
  webnet:
```

Estrutura de pastas



Onde projectb1 e projectb2 são imagens do projeto.

4.3. **Configuração do Nginx**

Os arquivos de configuração do Nginx estão no diretório `nginx`:

nginx/main_load_balancer.conf:

```
events {
    worker_connections 1024;
}

http {
    upstream load_balancers {
        server load_balancer1:80 max_fails=3 fail_timeout=30s;
        server load_balancer2:80 max_fails=3 fail_timeout=30s;
    }

    server {
        listen 80;

        location / {
            proxy_pass http://load_balancers;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

nginx/load_balancer1.conf:

```
events {
    worker_connections 1024;
}
http {
    upstream php_servers {
        server projectb1:80;
        server projectb2:80;
    }
    server {
        listen 80;

        location / {
            proxy_pass http://php_servers;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

nginx/load_balancer2.conf:

```
events {
    worker_connections 1024;
}
```

```
http {
    upstream php_servers {
        server projectb1:80;
        server projectb2:80;
    }

    server {
        listen 80;

        location / {
            proxy_pass http://php_servers;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

5. Benefícios da Solução

- **Distribuição de Carga**:A carga é distribuída entre vários servidores PHP, evitando a sobrecarga de um único servidor.

- **Alta Disponibilidade**: Se um servidor PHP ou um balanceador de carga falhar, o sistema continua a operar utilizando os servidores restantes.

- **Escalabilidade**: Novos servidores PHP e balanceadores de carga podem ser facilmente adicionados à configuração existente.

6. **Possíveis Melhorias**

6.1. **Escalabilidade**

Para melhorar a escalabilidade:

- **Auto-scaling**: Utilizar ferramentas como Kubernetes para auto-escalar contêineres com base na demanda.

- **Cloud-based Load Balancing**:Utilizar balanceadores de carga na nuvem que oferecem escalabilidade automática e gerenciamento simplificado.

6.2. **Alta Disponibilidade**

Para melhorar a alta disponibilidade:

- **Redundância Geográfica**: Implementar servidores em múltiplas regiões geográficas para evitar falhas regionais.

- **Database Replication**:Utilizar replicação de banco de dados para garantir a disponibilidade dos dados.

6.3. **Performance**

Para melhorar a performance:

- **Cache de Conteúdo**: Implementar caching em diversos níveis, como no Nginx, para reduzir a carga nos servidores PHP.

- **Otimização de Código**: Garantir que o código PHP seja eficiente e que consultas ao banco de dados sejam otimizadas.

7. Conclusão

Este ptrabalho demonstrou a implementação de uma solução de balanceamento de carga utilizando Vagrant no projeto A e Docker no projeto B e ambas usando Nginx, para a distribuição de solicitações entre múltiplos servidores PHP para garantir alta disponibilidade e escalabilidade. Através das melhorias sugeridas, o sistema pode ser ainda mais robusto e eficiente, atendendo a demandas maiores e garantindo um serviço confiável e de alta performance.

Esta configuração é adequada para ambientes de desenvolvimento e testes locais. Para produção, recomenda-se o uso de soluções mais avançadas e robustas de balanceamento de carga e gerenciamento de contêineres.