



**Ministério da Educação**  
**Universidade Federal do Piauí**  
**Curso: Sistemas de Informação**  
**4º Período**  
**Disciplina:**  
**Redes de Computadores I**



**DISCENTE:**

Raildom da Rocha Sobrinho

**SEGUNDA AVALIAÇÃO - 2025**

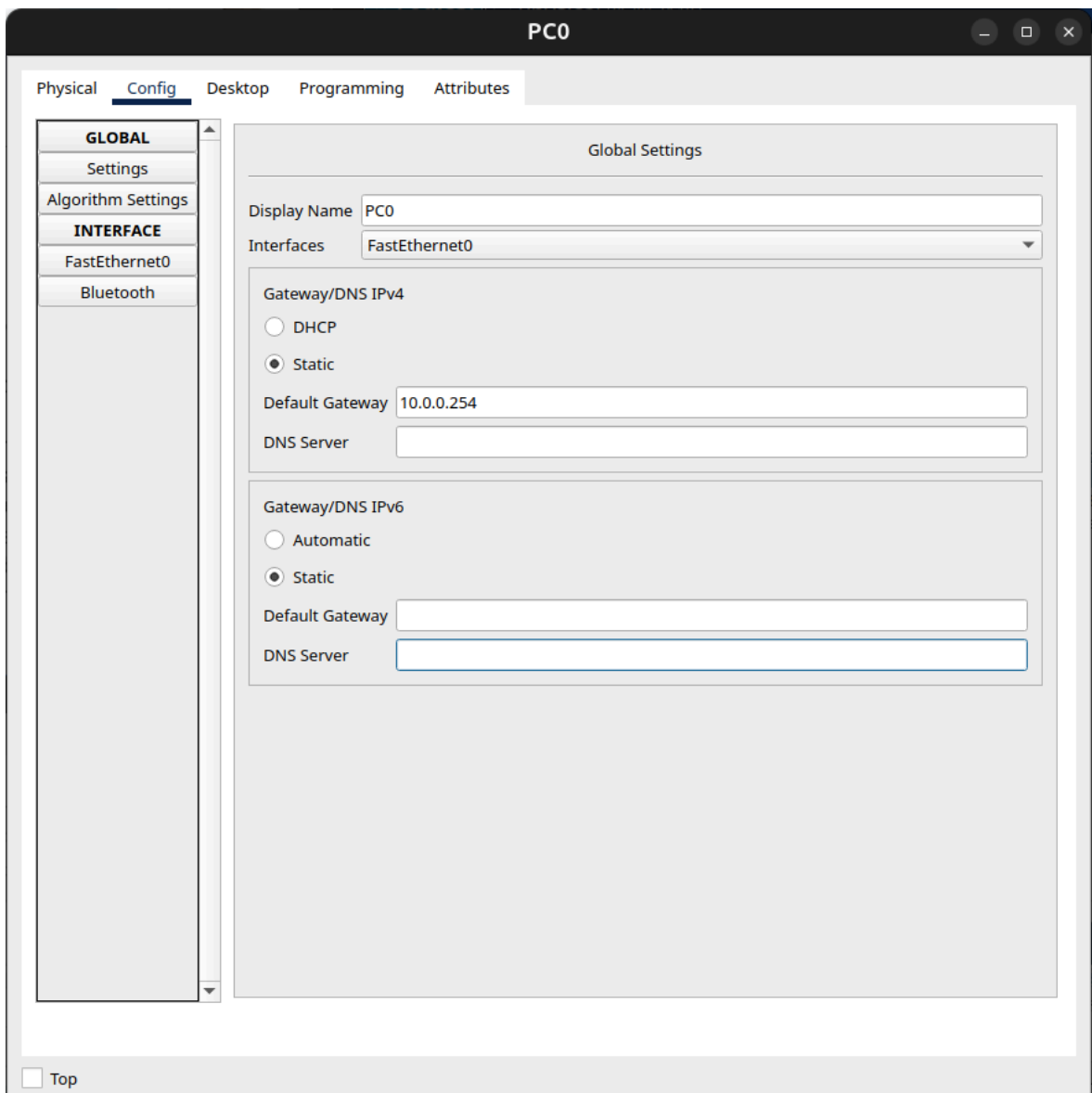
JUNHO - 2025

## **OBSERVAÇÕES:**

1. Repositório da avaliação no GitHub: [https://github.com/Raildom/network\\_2.git](https://github.com/Raildom/network_2.git)
2. Link do vídeo no YouTube: <https://youtu.be/wymXpuCf5CA>
3. As questões 4, 5 e 6 foram feitas usando o Docker Compose
4. Na sétima questão, foram abordados cabeçalhos que não aparecem nas capturas de tela fornecidas, pois pertencem aos protocolos IP, ICMP e ARP. Apesar de não serem exibidos explicitamente ao utilizar o comando tcpdump para visualizar os protocolos capturados, esses cabeçalhos estão presentes nos pacotes transmitidos e fazem parte da estrutura padrão de cada protocolo mencionado.

## **Primeira questão**

Considerando que toda a topologia da rede já está previamente estabelecida e que a única solicitação é garantir que todos os hosts consigam se comunicar entre si através do NAT, deve-se, em primeiro lugar, inserir o gateway padrão em cada computador. Para isso, basta clicar no computador desejado, acessar a aba "Config", depois "Settings" e, no campo "Default Gateway", inserir o endereço IP do gateway correspondente à rede daquele host.



Para configurar o NAT, é necessário acessar o roteador e utilizar a interface de linha de comando (CLI). Primeiramente, deve-se entrar no modo privilegiado com o comando `enable` e, em seguida, acessar o modo de configuração global com `configure terminal`. Após isso, utiliza-se o comando `ip nat inside source list 1 interface Gig0/0/1 overload` para definir a regra de tradução de endereços, especificando a lista de acesso número 1 e a interface de saída com sobrecarga (overload). Em seguida, cria-se a lista de acesso com o comando `access-list 1 permit any`, que permite o tráfego de qualquer endereço IP. Depois disso, configura-se a interface interna, Gig0/0/0, com o comando `ip nat inside`, e a interface externa, Gig0/0/1, com o comando `ip nat outside`. Após concluir a configuração, pode-se sair do modo de configuração. Para verificar as traduções NAT ativas, utiliza-se o comando `show ip nat translations`. No entanto, é importante observar que esse comando só exibirá resultados se houver comunicação ativa entre as duas redes, como demonstrado por meio de um teste de conectividade (ping) após a configuração.

The screenshot shows a terminal window titled "Router0" with tabs for "Physical", "Config", "CLI", and "Attributes". The "CLI" tab is active, displaying the "IOS Command Line Interface". The terminal shows the following commands and output:

```
Router>
Router>
Router>
Router>
Router>
Router>
Router>
Router>
Router>
Router>
Router>
Router>
Router>
Router>
Router>enable
Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#ip nat inside source list 1 interface Gig0/0/1 overload
Router(config)#access-list 1 permit any
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console

Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#interface Gig0/0/0
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#interface Gig0/0/1
Router(config-if)#ip nat outside
Router(config-if)#exit
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console

Router#sh ip nat translations
Pro Inside global      Inside local      Outside local      Outside global
icmp 200.1.1.254:1024  10.0.0.2:2        200.1.1.2:2        200.1.1.2:1024
icmp 200.1.1.254:1     10.0.0.3:1        200.1.1.2:1        200.1.1.2:1
icmp 200.1.1.254:2     10.0.0.1:2        200.1.1.2:2        200.1.1.2:2
icmp 200.1.1.254:4     10.0.0.1:4        200.1.1.2:4        200.1.1.2:4
icmp 200.1.1.254:5     10.0.0.2:5        200.1.1.2:5        200.1.1.2:5
icmp 200.1.1.254:6     10.0.0.3:6        200.1.1.2:6        200.1.1.2:6

Router#
```

At the bottom of the terminal window, there are "Copy" and "Paste" buttons. Below the terminal window, there is a "Top" button with a checkbox.

## Segunda questão

Para que o host 10.0.0.5 consiga pingar para o host 50.0.0.5. Primeiramente, deve-se definir o gateway padrão em cada computador. Para isso, é necessário acessar o computador correspondente, clicar na aba "Config", em seguida em "Settings", e, no campo "Default Gateway", inserir o endereço IP do gateway da rede à qual o host pertence.

Após a configuração dos gateways nos hosts, é preciso definir as rotas estáticas em todos os roteadores da rede. Para isso, deve-se acessar cada roteador, clicar na aba "Config", depois em "Routing", e, por fim, em "Static". Nessa seção, devem ser inseridas todas as rotas necessárias para permitir a comunicação entre as redes, tanto de ida quanto de volta.

A seguir a tabela de roteamento de todos os roteadores:

ROTEADOR 1:

Network Address
40.0.0.0/8 via 20.0.0.2
30.0.0.0/8 via 20.0.0.2
50.0.0.0/8 via 20.0.0.2

ROTEADOR 2:

Network Address
40.0.0.0/8 via 30.0.0.2
50.0.0.0/8 via 30.0.0.2
10.0.0.0/8 via 20.0.0.1

ROTEADOR 3:

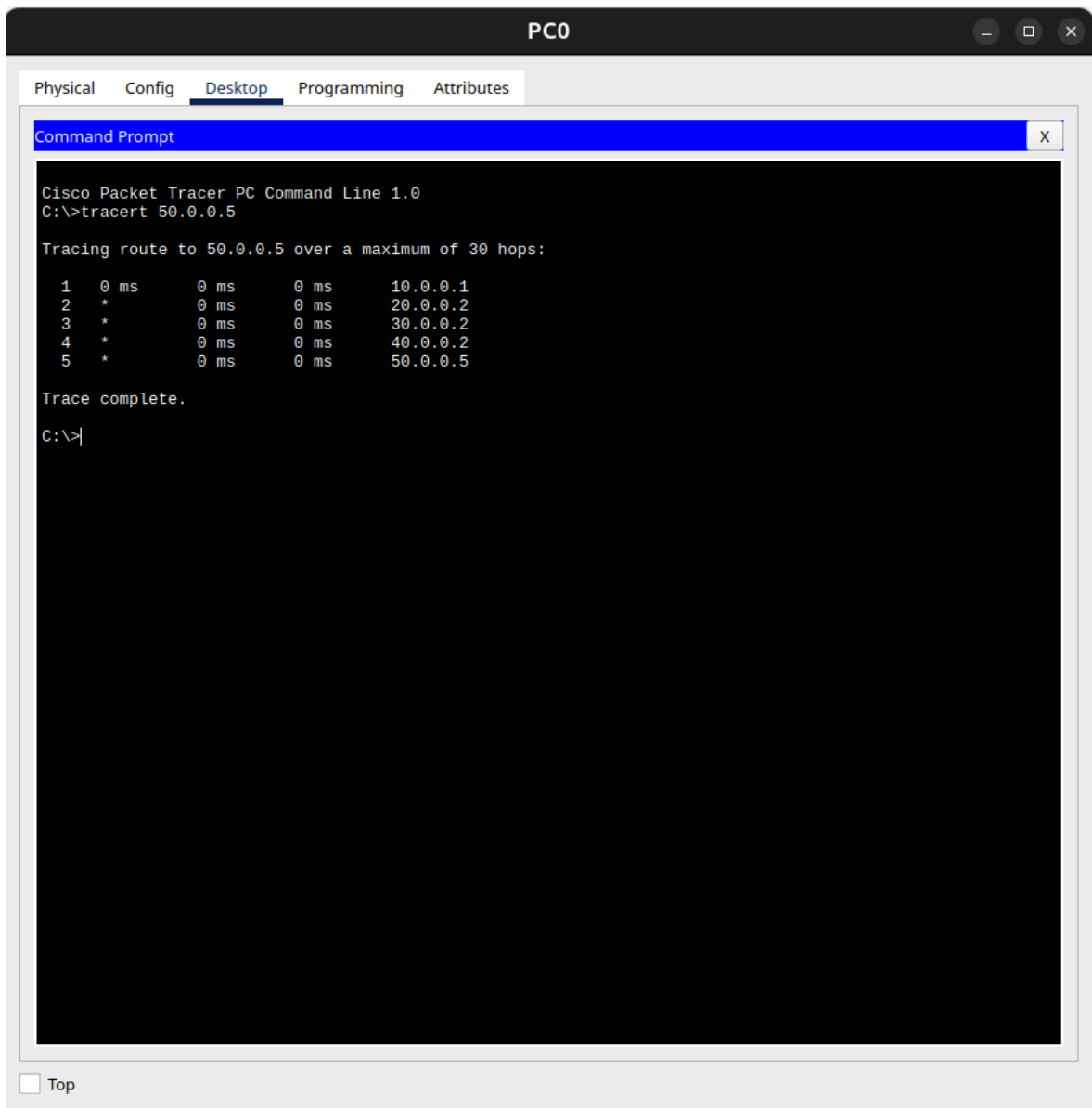
Network Address
50.0.0.0/8 via 40.0.0.2
20.0.0.0/8 via 30.0.0.1
10.0.0.0/8 via 30.0.0.1

ROTEADOR 4:

Network Address
30.0.0.0/8 via 40.0.0.1
20.0.0.0/8 via 40.0.0.1
10.0.0.0/8 via 40.0.0.1

### Terceira questão

Para demonstrar o funcionamento do comando traceroute no Cisco Packet Tracer, deve-se selecionar um dos computadores da rede, acessar a aba "Desktop", em seguida clicar em "Command Prompt" e, no terminal que se abre, digitar o comando tracert seguido do endereço IP do destino cuja rota se deseja visualizar. Esse procedimento permite acompanhar, passo a passo, o caminho percorrido pelos pacotes até alcançarem o host de destino, exibindo os roteadores intermediários por onde os dados transitam.



## Quarta questão

**Pré-requisitos:** Docker e Docker compose instalados.

**Estrutura dos arquivos:**

- docker-compose.yaml: arquivo principal que define e orquestra as redes e serviços (contêineres).
- host/Dockerfile: Define a imagem Docker para os hosts clientes.
- router/Dockerfile: Define a imagem Docker para os roteadores.
- web1/ , web2/ , web3/: Diretórios para cada servidor web, contendo:
  - Dockerfile: Define a imagem Docker para o servidor Nginx.
  - index.html: página HTML específica do site.
  - nginx.conf: Configuração do Nginx específica do site.

O arquivo docker-compose.yaml define os seguintes componentes:

**Redes:** Quatro redes customizadas do tipo bridge são criadas, cada uma com sua própria sub-rede IP definida usando IPAM (IP Address Management):

- lan1: 192.168.1.0/24
- lan2: 192.168.2.0/24
- lan3: 192.168.3.0/24
- lan4: 192.168.4.0/24

**Roteadores** (router1, router2, router3 ): Três contêineres baseados na imagem router/Dockerfile funcionam como roteadores.

- **Build:** Usam o contexto ./router e o router/Dockerfile.
- **Privilégios:** privileged: true é necessário para permitir a modificação de configurações do kernel, como o encaminhamento IP.
- **Redes e IPs:** Cada roteador conecta-se às suas LANs com IPs estáticos:
  - router1: lan1 (192.168.1.2)
  - router2: lan1 (192.168.1.3), lan2 (192.168.2.2), lan3 (192.168.3.2)
  - router3: lan1 (192.168.1.4), lan3 (192.168.3.3), lan4 (192.168.4.2)
- **Comandos de Inicialização:**
  - Habilitam o encaminhamento IP: sysctl -w net.ipv4.ip\_forward=1  
Obs: Necessário para que o contêiner atue como roteador.
  - Adicionam rotas estáticas: ip route add <rede\_destino> via <gateway> para direcionar o tráfego entre as LANs. Ex: router1 tem rotas para lan2 / lan3 via router2 (192.168.1.3) e para lan4 via router3 (192.168.1.4).
  - Configuram /etc/hosts: adicionam entradas para www1.empresa.com, www2.empresa.com e www3.empresa.com para resolução local de nomes.
  - Mantêm o contêiner ativo: tail -f /dev/null.

**Hosts** (host1, host2, host3): Três contêineres baseados na imagem host/Dockerfile representam máquinas clientes.

- **Build:** Usam o contexto ./host e o host/Dockerfile.
- **Capacidades:** cap\_add: - NET\_ADMIN é necessário para permitir a modificação da tabela de roteamento (configurar gateway padrão).
- **Dependências** (depends\_on): Garantem que o roteador da sua LAN seja iniciado antes do host.
- **Redes e IPs:** Cada host conecta-se à sua LAN com IP estático:
  - host1: lan2 (192.168.2.10)
  - host2: lan3 (192.168.3.10)
  - host3: lan4 (192.168.4.10)
- **Comandos de Inicialização:**
  - Configuram o gateway padrão: ip route replace default via <ip\_do\_roteador\_local>. O replace evita erros caso o Docker já tenha adicionado uma rota.
  - Configuram /etc/hosts: adicionam entradas para os servidores web.
  - Mantêm o contêiner ativo: tail -f /dev/null.

**Servidores Web** (web1, web2, web3): Três contêineres baseados nas imagens webX/Dockerfile atuam como servidores web Nginx.

- **Build:** Usam os contextos ./web1, ./web2, ./web3 e seus respectivos Dockerfiles.
- **Capacidades:** cap\_add: - NET\_ADMIN para configurar o gateway padrão.
- **Dependências** (depends\_on): Garantem que o roteador da sua LAN seja iniciado antes.

- **Redes e IPs:** Cada servidor conecta-se à sua LAN com IP estático:
  - web1: lan2 (192.168.2.100)
  - web2: lan3 (192.168.3.100)
  - web3: lan4 (192.168.4.100)
- **Mapeamento de Portas:** Expõem a porta 80 do contêiner para uma porta específica na máquina host, permitindo acesso externo:
  - web1: 8081:80
  - web2: 8082:80
  - web3: 8083:80
- **Comandos de Inicialização:**
- Configuram o gateway padrão: ip route replace default via <ip\_do\_roteador\_local>.
- Configuram /etc/hosts: adicionam entradas para os servidores web.
- Iniciam o Nginx: nginx -g 'daemon off;' em modo foreground para manter o contêiner ativo.

## Como usar:

### 1. Iniciar o Ambiente:

- Navegue até a pasta questao-04 no terminal.
- Execute: docker compose up --build.

### 2. Verificar Contêineres:

- docker ps (deve listar 9 contêineres rodando).

### 3. Testar Conectividade Interna:

- Acesse um host: docker exec -it host1 bash
- Pingue outros IPs: ping 192.168.3.10, ping 192.168.4.100, etc.

### 4. Acessar Sites Internamente:

- Ainda dentro de um host (ex: host1):
  - links www1.empresa.com
  - links www2.empresa.com
  - links www3.empresa.com
- Use 'q' para sair do links.

### 5. Acessar sites da máquina host:

- Abra seu navegador web e acesse:
  - http://localhost:8081 (Site A)
  - http://localhost:8082 (Site B)
  - http://localhost:8083 (Site C)
- Ou no terminal, acesse o diretório “questao-04” e digite:
  - links http://192.168.2.100:80 (site A)
  - links http://192.168.3.100:80 (site B)
  - links http://192.168.4.100:80 (site C)

### 6. Parar o Ambiente:

- No terminal, na pasta questao-04, execute: docker compose down.

## Quinta questão



Observe que a quinta questão possui a exata mesma estrutura da questão anterior, mudando somente algumas coisas no Dockerfile do router e em linhas de comandos relacionadas aos routers no arquivo .yaml (arquivo do docker compose). Por conta disso, não será explicado novamente a estrutura do programa, apenas as diferenças entre as duas questões (considerando que a 5ª questão é uma variação da 4ª com algumas modificações necessárias para mudar de roteamento estático para o roteamento RIP).

### **O Dockerfile em router/Dockerfile foi modificado para suportar o FRR. As alterações incluem:**

O router/Dockerfile foi modificado para suportar o FRR, adicionando o pacote frr à lista de pacotes instalados com apt-get install -y iproute2 procps net-tools frr, criando o usuário frr com useradd -r -s /sbin/nologin -M -g frr frr e os grupos frr e frrvty com groupadd -r frr e groupadd -r frrvty, associando o usuário frr ao grupo frrvty usando usermod -aG frrvty frr para permissões de acesso ao FRR, e criando os diretórios /var/log/frr e /var/run/frr com mkdir -p, definindo permissões frr:frr para /var/log/frr e frr:frrvty para /var/run/frr utilizando chown e chmod.

### **Criação de Arquivos de Configuração do FRR:**

A configuração do FRR foi introduzida no host com a criação dos arquivos zebra.conf e ripd.conf, organizados nos diretórios ./router/frr\_configs/router1, ./router/frr\_configs/router2 e ./router/frr\_configs/router3, que são montados em /etc/frr nos contêineres dos roteadores, onde zebra.conf configura o daemon zebra para roteamento básico, definindo o hostname do roteador (ex.: hostname router1), senhas (password zebra, enable password zebra) e habilitando logs em stdout (log stdout), enquanto ripd.conf configura o daemon ripd para o protocolo RIP, especificando as redes conectadas a serem anunciadas (ex.: network 192.168.1.0/24) e também habilitando logs em stdout.

### **Modificações nos Comandos dos Roteadores no docker-compose.yaml:**

O docker-compose.yml foi atualizado para suportar o roteamento RIP, removendo os comandos de roteamento estático (ip route add) e adicionando a inicialização dos daemons FRR, iniciando zebra com /usr/lib/frr/zebra -d -f /etc/frr/zebra.conf & e ripd com /usr/lib/frr/ripd -d -f /etc/frr/ripd.conf & em background para sincronização, criando diretórios /var/log/frr e /var/run/frr no contêiner com permissões ajustadas para frr:frr usando chown (evitando frr:frrvty para /var/run/frr para prevenir erros), garantindo a existência de /etc/frr/zebra.conf e /etc/frr/ripd.conf com touch e definindo permissões frr:frr e chmod 640, além de usar tail -f /dev/null para manter o contêiner ativo após a inicialização dos daemons.

### **Montagem dos Arquivos de Configuração do FRR:**

Cada roteador no docker-compose.yml monta um diretório específico do host em /etc/frr. Por exemplo:

- router1: ./router/frr\_configs/router1:/etc/frr
- router2: ./router/frr\_configs/router2:/etc/frr
- router3: ./router/frr\_configs/router3:/etc/frr

Isso permite que os arquivos zebra.conf e ripd.conf sejam carregados pelos daemons FRR no contêiner.

## Como usar:

### 1. Iniciar o Ambiente:

- Navegue até a pasta questao-05 no terminal
- Execute: `docker compose up --build`

### 2. Permissões necessárias:

**OBS:** Em determinados computadores, a liberação dessa permissão não se faz necessária.

- `sudo chown -R nome do usuário:nome do usuário ./router/frr_configs` (EX: `sudo chown -R raeldom:raeldom ./router/frr_configs`)
- `sudo chmod -R u+rw ./router/frr_configs`

### 3. Visualizar as rotas RIP:

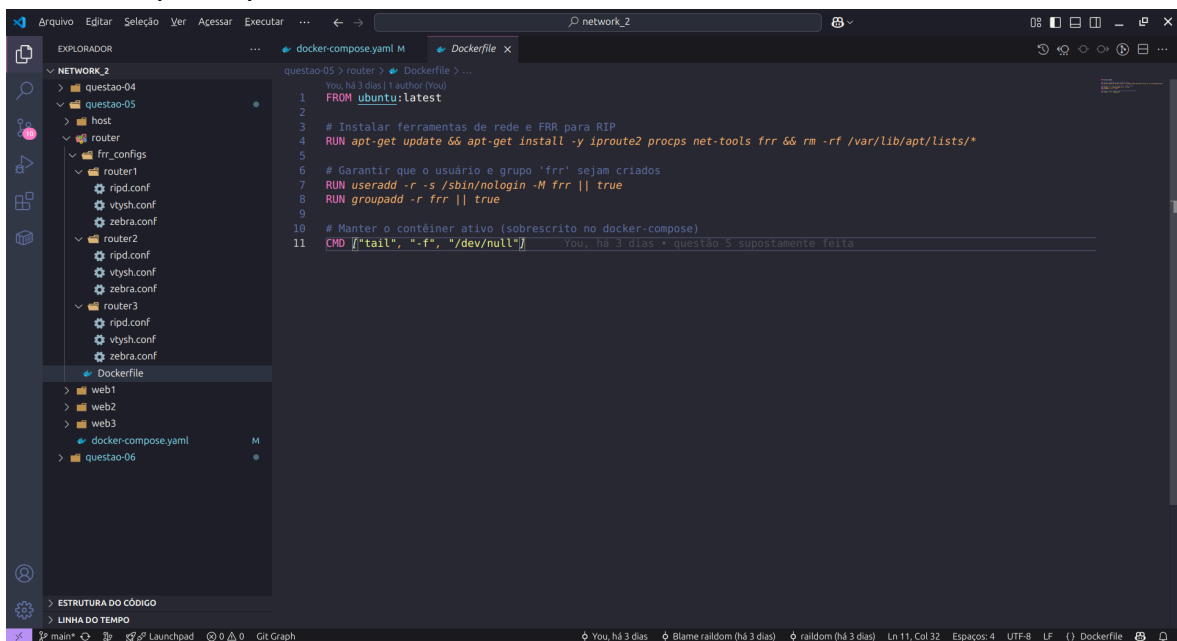
- `docker exec router1 vtysh -c 'show ip route'`
- `docker exec router2 vtysh -c 'show ip route'`
- `docker exec router3 vtysh -c 'show ip route'`

### 4. Parar o Ambiente:

- No terminal, na pasta questao-05, execute: `docker compose down`.

## Alterações realizadas nos arquivos:

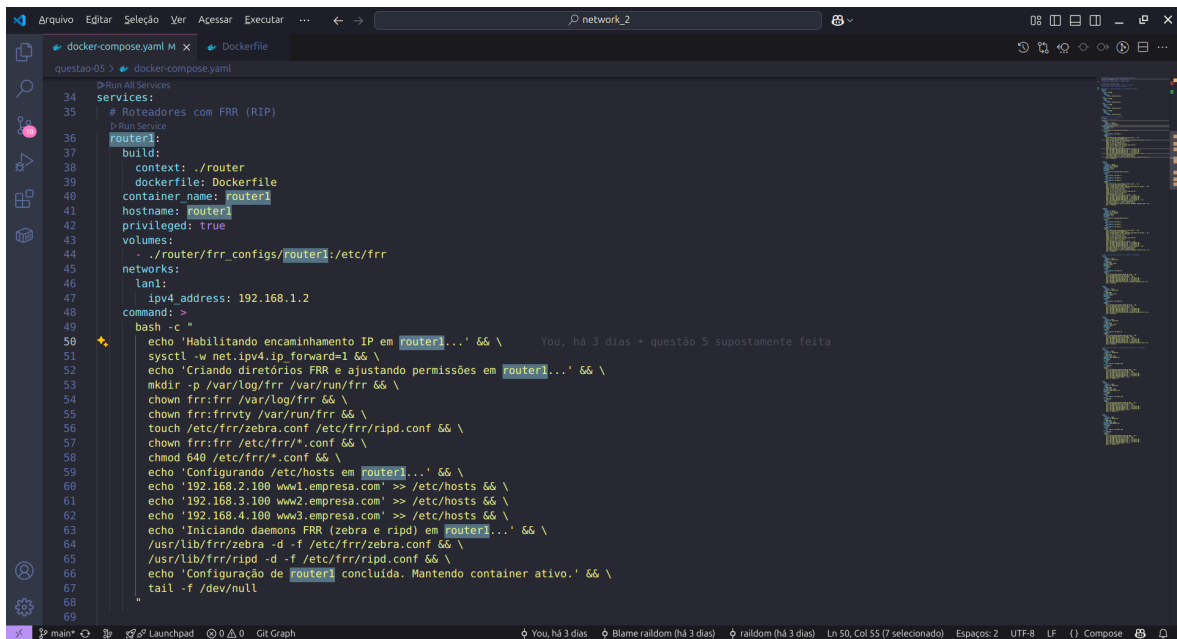
### Dockerfile (router):



```
1 FROM ubuntu:latest
2
3 # Instalar ferramentas de rede e FRR para RIP
4 RUN apt-get update && apt-get install -y iproute2 procps net-tools frr && rm -rf /var/lib/apt/lists/*
5
6 # Garantir que o usuário e grupo 'frr' sejam criados
7 RUN useradd -r -s /sbin/nologin -M frr || true
8 RUN groupadd -r frr || true
9
10 # Manter o contêiner ativo (sobrescrito no docker-compose)
11 CMD ["tail", "-f", "/dev/null"]
```

Note que, no caso do Dockerfile do router a única diferença são dois novos comandos usados para garantir que o usuário e grupo 'frr' sejam criados.

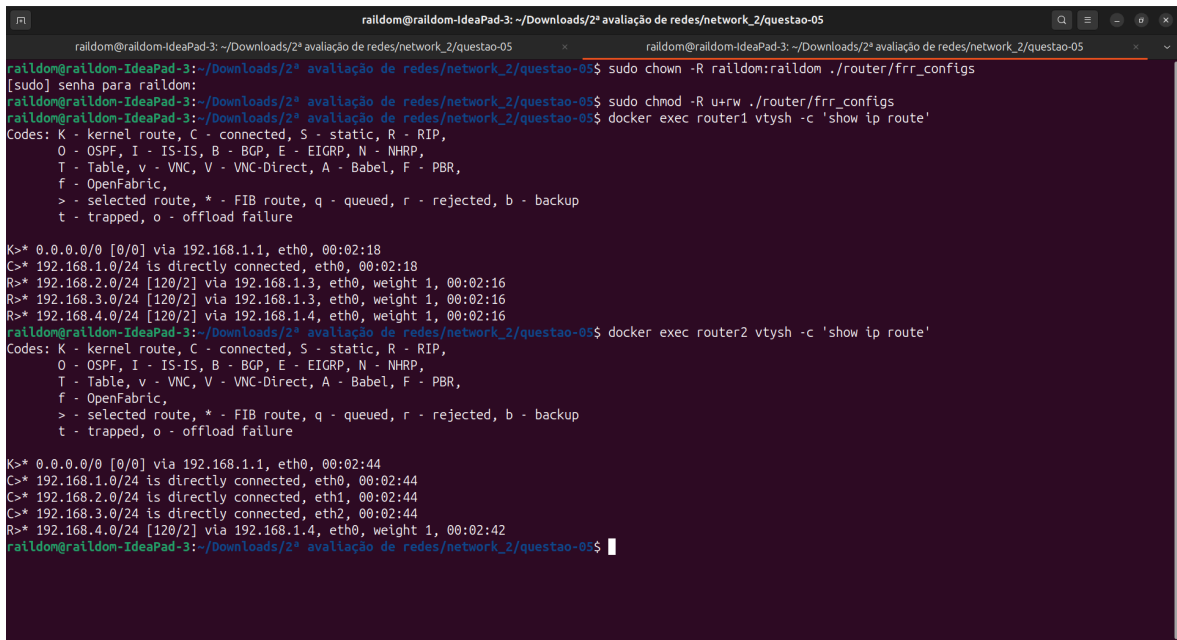
### Docker-compose.yml:



```
34 services:
35   # Roteadores com FRR (RIP)
36   router1:
37     build:
38       context: ./router
39       dockerfile: Dockerfile
40     container_name: router1
41     hostname: router1
42     privileged: true
43     volumes:
44       - ../router/frr_configs/router1:/etc/frr
45     networks:
46       lan1:
47         ipv4_address: 192.168.1.2
48     command: >
49     bash -c *
50     echo 'Habilitando encaminhamento IP em router1...' && \
51     sysctl -w net.ipv4.ip.forward=1 && \
52     echo 'Criando diretórios FRR e ajustando permissões em router1...' && \
53     mkdir -p /var/log/frr /var/run/frr && \
54     chown frr:frr /var/log/frr && \
55     chown frr:frr /var/run/frr && \
56     touch /etc/frr/zebra.conf /etc/frr/ripd.conf && \
57     chown frr:frr /etc/frr/*.conf && \
58     chmod 640 /etc/frr/*.conf && \
59     echo 'Configurando /etc/hosts em router1...' && \
60     echo '192.168.2.100 www1.empresa.com' >> /etc/hosts && \
61     echo '192.168.3.100 www2.empresa.com' >> /etc/hosts && \
62     echo '192.168.4.100 www3.empresa.com' >> /etc/hosts && \
63     echo 'Iniciando daemons FRR (zebra e ripd) em router1...' && \
64     /usr/lib/frr/zebra -d -f /etc/frr/zebra.conf && \
65     /usr/lib/frr/ripd -d -f /etc/frr/ripd.conf && \
66     echo 'Configuração de router1 concluída. Mantendo container ativo.' && \
67     tail -f /dev/null
68
69
```

Observe que na configuração de todos os roteadores no compose, a diferença é mínima, mudando apenas o nome do roteador.

**Resultado esperado:**



```
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-05
[sudo] senha para raildom:
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-05$ sudo chown -R raildom:raildom ./router/frr_configs
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-05$ sudo chmod -R u+rwx ./router/frr_configs
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-05$ docker exec router1 vttysh -c 'show ip route'
Codes: K - kernel route, C - connected, S - static, R - RIP,
O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
f - OpenFabric,
> - selected route, * - FIB route, q - queued, r - rejected, b - backup
t - trapped, o - offload failure

K>* 0.0.0.0/0 [0/0] via 192.168.1.1, eth0, 00:02:18
C>* 192.168.1.0/24 is directly connected, eth0, 00:02:18
R>* 192.168.2.0/24 [120/2] via 192.168.1.3, eth0, weight 1, 00:02:16
R>* 192.168.3.0/24 [120/2] via 192.168.1.3, eth0, weight 1, 00:02:16
R>* 192.168.4.0/24 [120/2] via 192.168.1.4, eth0, weight 1, 00:02:16
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-05$ docker exec router2 vttysh -c 'show ip route'
Codes: K - kernel route, C - connected, S - static, R - RIP,
O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
f - OpenFabric,
> - selected route, * - FIB route, q - queued, r - rejected, b - backup
t - trapped, o - offload failure

K>* 0.0.0.0/0 [0/0] via 192.168.1.1, eth0, 00:02:44
C>* 192.168.1.0/24 is directly connected, eth0, 00:02:44
C>* 192.168.2.0/24 is directly connected, eth1, 00:02:44
C>* 192.168.3.0/24 is directly connected, eth2, 00:02:44
R>* 192.168.4.0/24 [120/2] via 192.168.1.4, eth0, weight 1, 00:02:42
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-05$
```

**Sexta questão**

Observe que a sexta questão possui a exata mesma estrutura da quarta questão, diferenciando-se apenas em alguns pontos específicos do Dockerfile do roteador e em linhas de comandos relacionadas aos roteadores no arquivo .yaml (arquivo do Docker Compose). Por essa razão, a estrutura do programa não será explicada novamente nesta questão. Serão abordadas apenas as modificações necessárias para a transição do roteamento estático, utilizado na quarta questão, para o protocolo de roteamento dinâmico OSPF, implementado nesta sexta questão.

**O Dockerfile em router/Dockerfile foi modificado para suportar o FRR. As alterações incluem:**

Adicionar a instalação do pacote frr junto com iproute2, procs e net-tools para habilitar roteamento dinâmico com OSPF, criando apenas o usuário e grupo frr sem o grupo frrvty ou associação do usuário a ele, e omitindo a criação ou configuração de permissões dos diretórios /var/log/frr e /var/run/frr, que foram delegados ao docker-compose.yml.

### **Criação de Arquivos de Configuração do FRR:**

A Questão 06 introduz a criação de arquivos de configuração do FRR nos diretórios ./router/frr\_configs/router1, router2 e router3, montados em /etc/frr nos contêineres dos roteadores, incluindo zebra.conf para configurar o daemon zebra com hostname e senhas, ospfd.conf para definir redes OSPF (ex.: network 192.168.1.0/24 area 0) e vtysh.conf para o terminal vtysh, diferente da Questão 04, que usava roteamento estático e não requeria arquivos de configuração do FRR. O docker-compose.yml garante a existência desses arquivos com touch, define permissões frr:frr e aplica chmod

### **Modificações nos Comandos dos Roteadores no docker-compose.yml:**

Os comandos dos roteadores no docker-compose.yml foram modificados para suportar o roteamento dinâmico com OSPF, removendo os comandos de roteamento estático (ip route add) e adicionando a inicialização dos daemons FRR zebra e ospfd com /usr/lib/frr/zebra -d -f /etc/frr/zebra.conf e /usr/lib/frr/ospfd -d -f /etc/frr/ospfd.conf em foreground, cria diretórios /var/log/frr e /var/run/frr com permissões frr:frr para o primeiro e frr:frrvty para o segundo (apesar de frrvty não ser criado no Dockerfile), garante a existência de zebra.conf, ospfd.conf e vtysh.conf em /etc/frr com touch, ajusta suas permissões para frr:frr e chmod, mantém a configuração de /etc/hosts para resolver www1.empresa.com (192.168.2.100), www2.empresa.com (192.168.3.100) e www3.empresa.com (192.168.4.100), habilita o encaminhamento IP com sysctl -w net.ipv4.ip\_forward=1, e usa tail -f /dev/null para manter o contêiner ativo.

### **Montagem dos Arquivos de Configuração do FRR:**

O docker-compose.yml foi atualizado para montar diretórios específicos do host contendo arquivos de configuração do FRR (zebra.conf, ospfd.conf e vtysh.conf) em /etc/frr nos contêineres dos roteadores, com router1 montando ./router/frr\_configs/router1:/etc/frr, router2 montando ./router/frr\_configs/router2:/etc/frr e router3 montando ./router/frr\_configs/router3:/etc/frr, permitindo que os daemons zebra e ospfd acessem essas configurações para o roteamento OSPF.

### **Como usar:**

#### **1. Iniciar o Ambiente:**

- Navegue até a pasta questao-06 no terminal
- Execute: docker compose up --build

#### **2. Permissões necessárias:**

**OBS:** Em determinados computadores, a liberação dessa permissão não se faz necessária.

- sudo chown -R nome do usuário:nome do usuário ./router/frr\_configs (EX: sudo chown -R raeldom:raeldom ./router/frr\_configs)
- sudo chmod -R u+rw ./router/frr\_configs

#### **3. Visualizar as rotas RIP:**

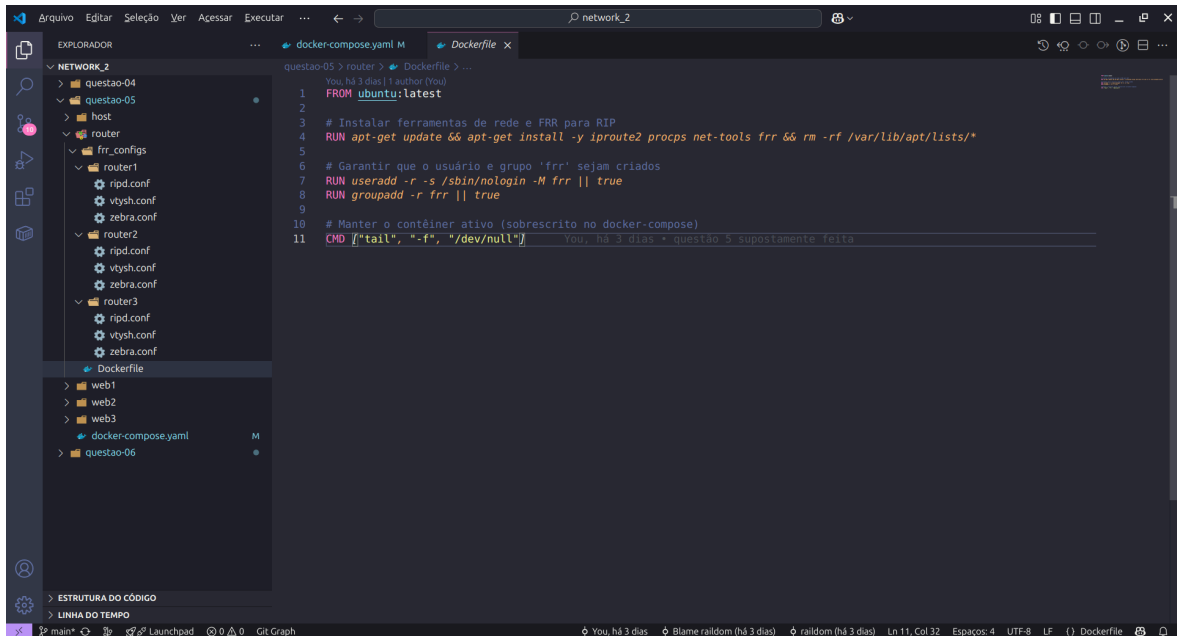
- docker exec router1 vtysh -c 'show ip route'
- docker exec router2 vtysh -c 'show ip route'
- docker exec router3 vtysh -c 'show ip route'

#### 4. Parar o Ambiente:

- No terminal, na pasta questao-05, execute: `docker compose down`.

#### Alterações realizadas nos arquivos:

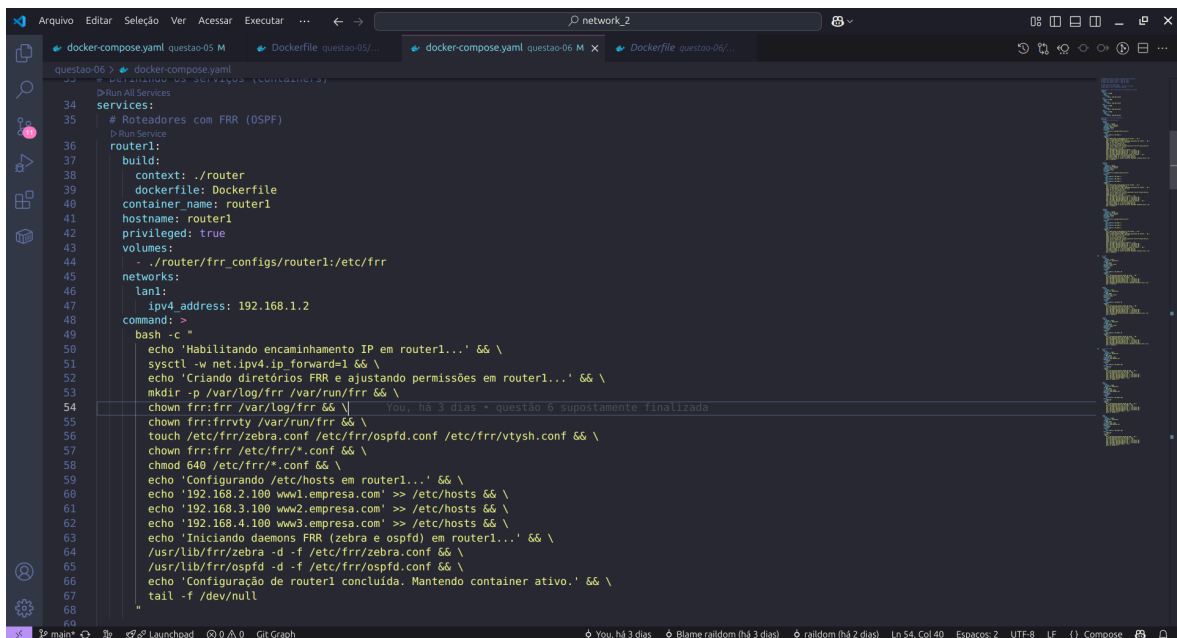
##### Dockerfile (router):



```
1 FROM ubuntu:latest
2
3 # Instalar ferramentas de rede e FRR para RIP
4 RUN apt-get update && apt-get install -y iproute2 procps net-tools frr && rm -rf /var/lib/apt/lists/*
5
6 # Garantir que o usuário e grupo 'frr' sejam criados
7 RUN useradd -r -s /sbin/nologin -M frr || true
8 RUN groupadd -r frr || true
9
10 # Manter o contêiner ativo (sobrescrito no docker-compose)
11 CMD ["tail", "-f", "/dev/null"]
```

Note que, no caso do Dockerfile do roteador, a única alteração consiste na adição de dois comandos extras utilizados para assegurar a criação do usuário e do grupo 'frr'.

##### Docker-compose.yml:



```
34 services:
35   # Roteadores com FRR (OSPF)
36   router1:
37     build:
38       context: ../router
39       dockerfile: Dockerfile
40     container_name: router1
41     hostname: router1
42     privileged: true
43     volumes:
44       - ../router/frr_configs/router1:/etc/frr
45     networks:
46       lan1:
47         ipv4_address: 192.168.1.2
48     command: >
49     bash -c "
50     echo 'Habilitando encaminhamento IP em router1...' && \
51     sysctl -w net.ipv4.ip_forward=1 && \
52     echo 'Criando diretórios FRR e ajustando permissões em router1...' && \
53     mkdir -p /var/log/frr /var/run/frr && \
54     chown frr:frr /var/log/frr && \
55     chown frr:frr /var/run/frr && \
56     touch /etc/frr/zebra.conf /etc/frr/ospfd.conf /etc/frr/vtysh.conf && \
57     chown frr:frr /etc/frr/*.conf && \
58     chmod 640 /etc/frr/*.conf && \
59     echo 'Configurando /etc/hosts em router1...' && \
60     echo '192.168.2.100 www1.empresa.com' >> /etc/hosts && \
61     echo '192.168.3.100 www2.empresa.com' >> /etc/hosts && \
62     echo '192.168.4.100 www3.empresa.com' >> /etc/hosts && \
63     echo 'Iniciando daemons FRR (zebra e ospfd) em router1...' && \
64     /usr/lib/frr/zebra -d -f /etc/frr/zebra.conf && \
65     /usr/lib/frr/ospfd -d -f /etc/frr/ospfd.conf && \
66     echo 'Configuração de router1 concluída. Mantendo container ativo.' && \
67     tail -f /dev/null
68
```

Perceba que, na configuração de cada roteador no arquivo *compose*, a modificação é sutil, limitando-se apenas à troca do nome atribuído a cada roteador.

#### Resultado esperado:

```
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-06
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-06
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-06$ docker exec router1 vttysh -c 'show ip route'
Codes: K - kernel route, C - connected, S - static, R - RIP,
O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
T - Table, V - VNC-Direct, A - Babel, F - PBR,
f - OpenFabric,
> - selected route, * - FIB route, q - queued, r - rejected, b - backup
t - trapped, o - offload failure

K>* 0.0.0.0/0 [0/0] via 192.168.1.1, eth0, 00:00:21
O 192.168.1.0/24 [110/10] is directly connected, eth0, weight 1, 00:00:20
C>* 192.168.1.0/24 is directly connected, eth0, 00:00:21
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-06$ docker exec router2 vttysh -c 'show ip route'
Codes: K - kernel route, C - connected, S - static, R - RIP,
O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
T - Table, V - VNC-Direct, A - Babel, F - PBR,
f - OpenFabric,
> - selected route, * - FIB route, q - queued, r - rejected, b - backup
t - trapped, o - offload failure

K>* 0.0.0.0/0 [0/0] via 192.168.1.1, eth0, 00:00:33
O 192.168.1.0/24 [110/10] is directly connected, eth0, weight 1, 00:00:32
C>* 192.168.1.0/24 is directly connected, eth0, 00:00:33
O 192.168.2.0/24 [110/10] is directly connected, eth1, weight 1, 00:00:32
C>* 192.168.2.0/24 is directly connected, eth1, 00:00:33
O 192.168.3.0/24 [110/10] is directly connected, eth2, weight 1, 00:00:32
C>* 192.168.3.0/24 is directly connected, eth2, 00:00:33
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-06$
```

## Sétima questão

```
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-04
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-04$ sudo tcpdump -r captura-lan1.pcap
reading from file captura-lan1.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:14:47.002978 IP 192.168.4.10 > 192.168.1.2: ICMP echo request, id 34, seq 1, length 64
15:14:47.003010 IP 192.168.1.2 > 192.168.4.10: ICMP echo reply, id 34, seq 1, length 64
15:14:48.050010 IP 192.168.4.10 > 192.168.1.2: ICMP echo request, id 34, seq 2, length 64
15:14:48.050045 IP 192.168.1.2 > 192.168.4.10: ICMP echo reply, id 34, seq 2, length 64
15:14:49.073969 IP 192.168.4.10 > 192.168.1.2: ICMP echo request, id 34, seq 3, length 64
15:14:49.073999 IP 192.168.1.2 > 192.168.4.10: ICMP echo reply, id 34, seq 3, length 64
15:14:50.097250 IP 192.168.4.10 > 192.168.1.2: ICMP echo request, id 34, seq 4, length 64
15:14:50.097972 IP 192.168.1.2 > 192.168.4.10: ICMP echo reply, id 34, seq 4, length 64
15:14:52.465994 ARP, Request who-has 192.168.1.4 tell 192.168.1.2, length 28
15:14:52.465910 ARP, Request who-has 192.168.1.2 tell 192.168.1.4, length 28
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-04$ sudo tcpdump -r captura-lan2.pcap
reading from file captura-lan2.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:18:36.014050 IP 192.168.2.10 > 192.168.2.2: ICMP echo request, id 35, seq 1, length 64
15:18:36.014111 IP 192.168.2.2 > 192.168.2.10: ICMP echo reply, id 35, seq 1, length 64
15:18:37.041965 IP 192.168.2.10 > 192.168.2.2: ICMP echo request, id 35, seq 2, length 64
15:18:37.042021 IP 192.168.2.2 > 192.168.2.10: ICMP echo reply, id 35, seq 2, length 64
15:18:38.065985 IP 192.168.2.10 > 192.168.2.2: ICMP echo request, id 35, seq 3, length 64
15:18:38.066020 IP 192.168.2.2 > 192.168.2.10: ICMP echo reply, id 35, seq 3, length 64
15:18:39.089920 IP 192.168.2.10 > 192.168.2.2: ICMP echo request, id 35, seq 4, length 64
15:18:39.089952 IP 192.168.2.2 > 192.168.2.10: ICMP echo reply, id 35, seq 4, length 64
15:18:41.329910 ARP, Request who-has 192.168.2.10 tell 192.168.2.2, length 28
15:18:41.329938 ARP, Request who-has 192.168.2.2 tell 192.168.2.10, length 28
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-04$ sudo tcpdump -r captura-lan3.pcap
reading from file captura-lan3.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:23:16.408036 IP 192.168.3.10 > 192.168.3.2: ICMP echo request, id 36, seq 1, length 64
15:23:16.408093 IP 192.168.3.2 > 192.168.3.10: ICMP echo reply, id 36, seq 1, length 64
15:23:17.425940 IP 192.168.3.10 > 192.168.3.2: ICMP echo request, id 36, seq 2, length 64
15:23:17.425904 IP 192.168.3.2 > 192.168.3.10: ICMP echo reply, id 36, seq 2, length 64
15:23:18.450955 IP 192.168.3.10 > 192.168.3.2: ICMP echo request, id 36, seq 3, length 64
15:23:18.450989 IP 192.168.3.2 > 192.168.3.10: ICMP echo reply, id 36, seq 3, length 64
15:23:19.473980 IP 192.168.3.10 > 192.168.3.2: ICMP echo request, id 36, seq 4, length 64
15:23:19.474030 IP 192.168.3.2 > 192.168.3.10: ICMP echo reply, id 36, seq 4, length 64
15:23:21.905980 ARP, Request who-has 192.168.3.10 tell 192.168.3.2, length 28
15:23:21.905980 ARP, Request who-has 192.168.3.2 tell 192.168.3.10, length 28
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-04$ sudo tcpdump -r captura-lan4.pcap
reading from file captura-lan4.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:27:09.974473 IP 192.168.4.10 > 192.168.4.2: ICMP echo request, id 38, seq 1, length 64
15:27:09.974501 IP 192.168.4.2 > 192.168.4.10: ICMP echo reply, id 38, seq 1, length 64
15:27:11.025931 IP 192.168.4.10 > 192.168.4.2: ICMP echo request, id 38, seq 2, length 64
15:27:11.025963 IP 192.168.4.2 > 192.168.4.10: ICMP echo reply, id 38, seq 2, length 64
15:27:12.050944 IP 192.168.4.10 > 192.168.4.2: ICMP echo request, id 38, seq 3, length 64
15:27:12.050977 IP 192.168.4.2 > 192.168.4.10: ICMP echo reply, id 38, seq 3, length 64
15:27:13.073949 IP 192.168.4.10 > 192.168.4.2: ICMP echo request, id 38, seq 4, length 64
15:27:13.073981 IP 192.168.4.2 > 192.168.4.10: ICMP echo reply, id 38, seq 4, length 64
15:27:15.377902 ARP, Request who-has 192.168.4.10 tell 192.168.4.2, length 28
15:27:15.377910 ARP, Request who-has 192.168.4.2 tell 192.168.4.10, length 28
raildom@raildom-IdeaPad-3: ~/Downloads/2ª avaliação de redes/network_2/questao-04$
```

A imagem acima apresenta todas as informações capturadas por meio da ferramenta tcpdump durante a ocorrência de um fluxo de dados entre os contêineres. A seguir, são descritos e explicados os valores e cabeçalhos exibidos na captura:

## Protocolo IP

### Amostra Selecionada:

Linha da captura de LAN1 em 15:14:47.002978: IP 192.168.4.10 > 192.168.1.2: ICMP echo request, id 34, seq 1, length 64.

### Mapeamento:

Conexão LAN1: Host3 para Roteador1.

- 192.168.4.10 representa o Host3.
- 192.168.1.2 representa o Roteador1.

### Campos do Cabeçalho IP:

- Version: 4 (IPv4).
- Header Length: 20 bytes (valor padrão, sem opções adicionais).
- Type of Service (ToS): 0 (sem prioridade ou diferenciação).
- Total Length: 64 bytes.
  - Cabeçalho IP: 20 bytes
  - Cabeçalho ICMP: 8 bytes
  - Dados: 36 bytes
- Identification: usado para controle de fragmentação.
- Flags: "Don't Fragment", por ser um pacote pequeno.
- Fragment Offset: 0 (não fragmentado).
- Time to Live (TTL): Valor típico de 64.
- Protocol: 1 (ICMP).
- Header Checksum: validação do cabeçalho IP.
- Source IP Address: 192.168.4.10 (Host3).
- Destination IP Address: 192.168.1.2 (Roteador1).

## Protocolo ICMP

### Amostra Selecionada:

Solicitação: IP 192.168.4.10 > 192.168.1.2: ICMP echo request, id 34, seq 1, length 64

Resposta: IP 192.168.1.2 > 192.168.4.10: ICMP echo reply, id 34, seq 1, length 64

### Mapeamento:

Conexão LAN1: Host3 (origem) para Roteador1 (destino).

### Campos do Cabeçalho ICMP:

- Type:
  - Echo Request: 8
  - Echo Reply: 0
- Code: 0 (valor padrão para requisições e respostas de echo).
- Checksum: utilizado para verificação de integridade.
- Identifier: 34 (Usado para correlacionar a solicitação e a resposta).
- Sequence Number: 1 (incrementado a cada novo envio de pacote de ping).
- Length: 64 bytes (tamanho total do pacote, incluindo cabeçalhos).
  - Cabeçalho ICMP: 8 bytes
  - Dados (payload): 36 bytes (geralmente preenchimento padrão).
- Payload: preenchimento padrão.

## Protocolo ARP

### Amostra Selecionada:

Linha da captura de LAN1 em 15:14:52.465904: ARP, Request who-has 192.168.1.4 tell 192.168.1.2, length 28.

### Mapeamento:

Conexão LAN1: Roteador1 (192.168.1.2) solicita o endereço MAC de Host3 (192.168.1.4).

### Campos do Cabeçalho ARP:

- Hardware Type: 1 (Ethernet) (Indica que o enlace utiliza tecnologia Ethernet).
- Protocol Type: 0x0800 (IPv4) (Protocolo de camada de rede utilizado é o IP versão 4).
- Hardware Address Length: 6 bytes (Tamanho de endereços MAC em redes Ethernet).
- Protocol Address Length: 4 bytes (Tamanho de endereços IP (IPv4)).

- Operation:
  - Request (1): "Who has 192.168.1.4? Tell 192.168.1.2".
  - Reply (2): (não mostrado na imagem acima)
- Sender MAC Address: MAC de 192.168.1.2.
- Sender IP Address: 192.168.1.2 (Roteador1).
- Target MAC Address: 00:00:00:00:00:00 (desconhecido – padrão em requisições).
- Target IP Address: 192.168.1.4 (Host3).
- Length: 28 bytes (tamanho total de um pacote ARP).

A seguir, apresenta-se como ocorreu o tráfego de informações, bem como os resultados individuais capturados por meio do tcpdump nas 4 sub-redes:

#### LAN1: Host3 para router1 e web3 para web1

```
raildom@raildom-IdeaPad-3:~/Downloads/2ª avaliação de redes/network_2/questao-04$ sudo tcpdump -r captura-lan1.pcap
reading from file captura-lan1.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:14:47.002978 IP 192.168.4.10 > 192.168.1.2: ICMP echo request, id 34, seq 1, length 64
15:14:47.003010 IP 192.168.1.2 > 192.168.4.10: ICMP echo reply, id 34, seq 1, length 64
15:14:48.050010 IP 192.168.4.10 > 192.168.1.2: ICMP echo request, id 34, seq 2, length 64
15:14:48.050045 IP 192.168.1.2 > 192.168.4.10: ICMP echo reply, id 34, seq 2, length 64
15:14:49.073969 IP 192.168.4.10 > 192.168.1.2: ICMP echo request, id 34, seq 3, length 64
15:14:49.073990 IP 192.168.1.2 > 192.168.4.10: ICMP echo reply, id 34, seq 3, length 64
15:14:50.097949 IP 192.168.4.10 > 192.168.1.2: ICMP echo request, id 34, seq 4, length 64
15:14:50.097972 IP 192.168.1.2 > 192.168.4.10: ICMP echo reply, id 34, seq 4, length 64
15:14:52.465904 ARP, Request who-has 192.168.1.4 tell 192.168.1.2, length 28
15:14:52.465910 ARP, Request who-has 192.168.1.2 tell 192.168.1.4, length 28
```

#### LAN2: Host1 para router2 e web1 para web3

```
raildom@raildom-IdeaPad-3:~/Downloads/2ª avaliação de redes/network_2/questao-04$ sudo tcpdump -r captura-lan2.pcap
reading from file captura-lan2.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:18:36.014056 IP 192.168.2.10 > 192.168.2.2: ICMP echo request, id 35, seq 1, length 64
15:18:36.014111 IP 192.168.2.2 > 192.168.2.10: ICMP echo reply, id 35, seq 1, length 64
15:18:37.041965 IP 192.168.2.10 > 192.168.2.2: ICMP echo request, id 35, seq 2, length 64
15:18:37.042021 IP 192.168.2.2 > 192.168.2.10: ICMP echo reply, id 35, seq 2, length 64
15:18:38.065985 IP 192.168.2.10 > 192.168.2.2: ICMP echo request, id 35, seq 3, length 64
15:18:38.066028 IP 192.168.2.2 > 192.168.2.10: ICMP echo reply, id 35, seq 3, length 64
15:18:39.089928 IP 192.168.2.10 > 192.168.2.2: ICMP echo request, id 35, seq 4, length 64
15:18:39.089952 IP 192.168.2.2 > 192.168.2.10: ICMP echo reply, id 35, seq 4, length 64
15:18:41.329919 ARP, Request who-has 192.168.2.10 tell 192.168.2.2, length 28
15:18:41.329938 ARP, Request who-has 192.168.2.2 tell 192.168.2.10, length 28
```

#### LAN3: Host2 para router2 e web2 para web1

```
raildom@raildom-IdeaPad-3:~/Downloads/2ª avaliação de redes/network_2/questao-04$ sudo tcpdump -r captura-lan3.pcap
reading from file captura-lan3.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:23:16.408036 IP 192.168.3.10 > 192.168.3.2: ICMP echo request, id 36, seq 1, length 64
15:23:16.408093 IP 192.168.3.2 > 192.168.3.10: ICMP echo reply, id 36, seq 1, length 64
15:23:17.425948 IP 192.168.3.10 > 192.168.3.2: ICMP echo request, id 36, seq 2, length 64
15:23:17.425984 IP 192.168.3.2 > 192.168.3.10: ICMP echo reply, id 36, seq 2, length 64
15:23:18.450955 IP 192.168.3.10 > 192.168.3.2: ICMP echo request, id 36, seq 3, length 64
15:23:18.450989 IP 192.168.3.2 > 192.168.3.10: ICMP echo reply, id 36, seq 3, length 64
15:23:19.473989 IP 192.168.3.10 > 192.168.3.2: ICMP echo request, id 36, seq 4, length 64
15:23:19.474039 IP 192.168.3.2 > 192.168.3.10: ICMP echo reply, id 36, seq 4, length 64
15:23:21.905898 ARP, Request who-has 192.168.3.10 tell 192.168.3.2, length 28
15:23:21.905908 ARP, Request who-has 192.168.3.2 tell 192.168.3.10, length 28
```

#### LAN4: Host3 para router3 e web3 para web2

```
raildom@raildom-IdeaPad-3:~/Downloads/2ª avaliação de redes/network_2/questao-04$ sudo tcpdump -r captura-lan4.pcap
reading from file captura-lan4.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:27:09.974473 IP 192.168.4.10 > 192.168.4.2: ICMP echo request, id 38, seq 1, length 64
15:27:09.974503 IP 192.168.4.2 > 192.168.4.10: ICMP echo reply, id 38, seq 1, length 64
15:27:11.025931 IP 192.168.4.10 > 192.168.4.2: ICMP echo request, id 38, seq 2, length 64
15:27:11.025963 IP 192.168.4.2 > 192.168.4.10: ICMP echo reply, id 38, seq 2, length 64
15:27:12.050944 IP 192.168.4.10 > 192.168.4.2: ICMP echo request, id 38, seq 3, length 64
15:27:12.050977 IP 192.168.4.2 > 192.168.4.10: ICMP echo reply, id 38, seq 3, length 64
15:27:13.073949 IP 192.168.4.10 > 192.168.4.2: ICMP echo request, id 38, seq 4, length 64
15:27:13.073981 IP 192.168.4.2 > 192.168.4.10: ICMP echo reply, id 38, seq 4, length 64
15:27:15.377902 ARP, Request who-has 192.168.4.10 tell 192.168.4.2, length 28
15:27:15.377916 ARP, Request who-has 192.168.4.2 tell 192.168.4.10, length 28
```