UNIVERSIDADE REGIONAL INTEGRADA DO ALTO URUGUAI E DAS MISSÕES
PRÓ-REITORIA DE ENSINO, PESQUISA E PÓS-GRADUAÇÃO CÂMPUS DE ERECHIM E FREDERICO WESTPHALEN
DEPARTAMENTO DE ENGENHARIAS E CIÊNCIA DA COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

IÈVES SOLANO PALOSCHI, MATHEUS LUIS MENEGAT, RONALDO ANTÔNIO MANFREDINI JÚNIOR

ESTUDO DO PROTOCOLO TCP/IP EM UM LOAD BALANCER BASEADO EM NODE.JS E DOCKER

ERECHIM - RS
2020

Protocolos utilizados no projeto

Protocolo IPv4

O IPv4 e seu sucessor, IPv6 são os principais protocolos utilizados atualmente para identificar os computadores nas redes e garantir que as informações cheguem corretamente aos destinos.

Foi inicialmente implementado pela ARPANET, em 1983 e utiliza blocos de 32 bits para referenciar os endereços (4 octetos). Isso permite 4.294.967.296 endereços, sendo um problema atualmente por conta de ter menos endereços do que dispositivos conectados.

Um endereço IPv4 pode ser representado tanto na forma decimal, separado por períodos (ex: 172.16.254.1), na notação CIDR (ex: 10101100.00010000.11111110.00000001) ou na notação hexadecimal (ex: 0xC00002EB).

No projeto, o IPv4 foi utilizado para endereçar a comunicação entre os contêineres e estabelecer a rede que comportou os nodos.

Protocolo HTTP

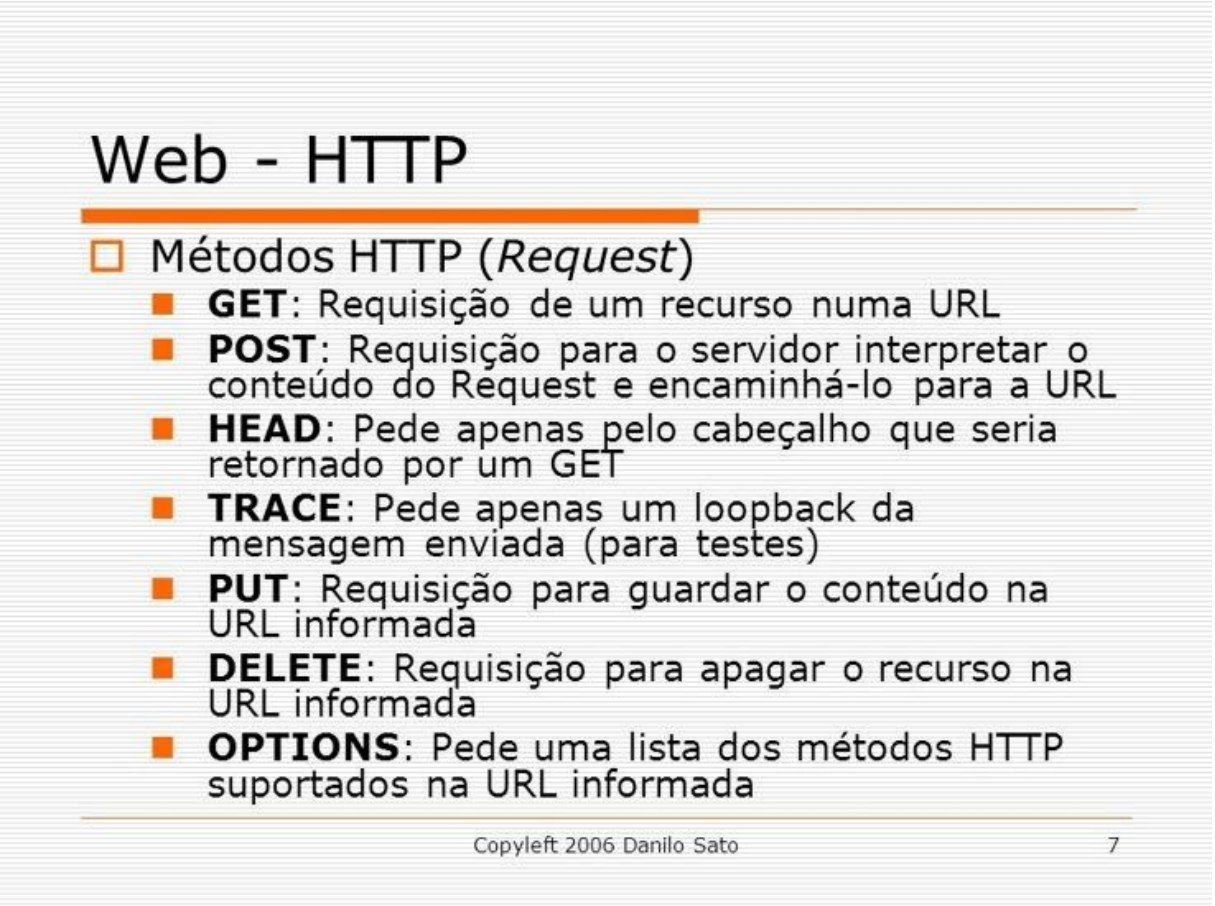
Protocolo de transferência de Hipertexto (Hypertext Transfer Protocol), é um dos vários protocolos de camadas de aplicação da internet, que conversam entre cliente e servidor através de mensagens. O HTTP define a estrutura e o modo como essas mensagens são trocadas.

Na prática o HTTP funciona da seguinte forma, o cliente faz uma requisição de uma página ao servidor através do protocolo HTTP e o servidor responde ao cliente com a página solicitada.

Métodos HTTP

Esse protocolo define um conjunto de métodos que podem ser invocados pelo cliente, que se assemelham a comandos para enviar ao servidor.

As mensagens de request e response seguem um padrão, uma linha informando o método, seguido de uma linha ou mais linhas que passa um cabeçalho e por fim o corpo da mensagem.



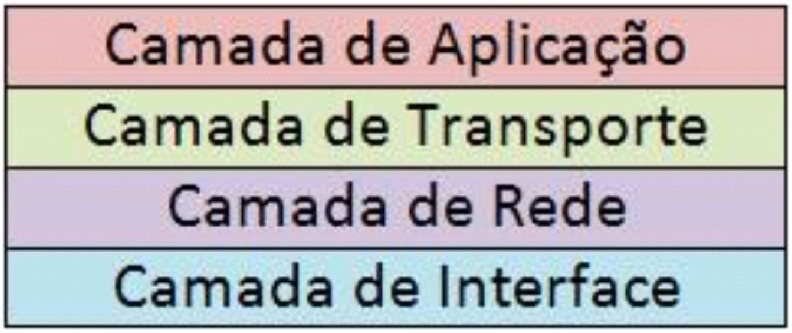
Protocolo TCP/IP

Esse protocolo é um padrão de comunicação que reúne um conjunto de protocolos tais como: TCP, IP, FTP, TELNET, ICMP, ARP E NFS.

As informações que trafegam na rede necessitam dele, por isso ele é utilizado como protocolo primário da rede na internet. Este modelo tem uma vantagem: por ter os processos de comunicação bem definidos e divididos em cada camada, qualquer alteração poderá ser feita isoladamente, não precisando reescrever todo o protocolo. O TCP/IP tem como principal característica a transmissão de dados em máquinas que diferem em suas arquiteturas .

Protocolo é uma espécie de linguagem utilizada para que dois computadores consigam se comunicar. Mesmo que dois computadores estejam ligados a uma mesma rede, eles não vão conseguir se comunicar sem o uso de um idioma comum. E na internet essa é a função do TCP/IP, um conjunto de protocolos criado para ser um recurso da ARPANET. Para que dessa forma, os computadores conseguem enviar e receber informações que vão desde um email até uma página que se carrega no navegador.

Então o TCP/IP é um conjunto de protocolos. Esse grupo é dividido em quatro camadas: aplicação, transporte, rede e interface. Cada uma delas é responsável pela execução de tarefas distintas. Essa divisão em camadas é uma forma de garantir a integridade dos dados que trafegam pela rede.



Camada de Aplicação

Esta camada é responsável por fazer a comunicação entre os programas e os protocolos de transporte no TCP/IP.

Exemplo quando você solicita ao seu cliente de e-mail para fazer o download das mensagens que estão armazenados no servidor, você está fazendo uma solicitação à camada de aplicação do TCP/IP, que neste caso é servido pelo protocolo SMTP. Quando você abre uma página no seu navegador, ele vai requerer ao TCP/IP, na camada de aplicação, servido pelo protocolo HTTP, por isso que as páginas iniciam-se com http://.

Essa camada de aplicação possui protocolos importantes e conhecidos, como o HTTP, FTP, DNS e DHCP.

- O HTTP é utilizado para a comunicação de dados da internet – WWW;
- O FTP é utilizado para a transferência de arquivos de modo interativo;
- O DNS é utilizado para resolver o nome de um host em endereço IP;
- O DHCP é utilizado para oferecer dinamicamente endereços de rede.

Camada Transporte

Esta camada é responsável por receber os dados enviados pela camada de aplicação e transformá-los em pacotes menores, a serem repassados para a camada de internet. Ela garante que os dados chegarão sem erros e na sequência correta.

É formado por dois protocolos o TCP (Transmission Control Protocol) e o UDP (User Datagram Protocol).

O UDP realiza apenas a multiplexação para que várias aplicações possam acessar o sistema de comunicação de forma coerente. Não possui confirmação de entrega e é geralmente usado na transmissão de informações de controle.

O TCP realiza, além da multiplexação, uma série de funções para tornar a comunicação entre origem e destino mais confiável. São responsabilidades do protocolo TCP: o controle de fluxo e erro, a sequência e a multiplexação de mensagens.

CAMADA DE INTERNET

Fica responsável pelo endereçamento e roteamento do pacote, fazendo a conexão entre as redes locais. Adiciona ao pacote o endereço IP de origem e o de destino, para que ele saiba qual o caminho deve percorrer.

Na transmissão, esse pacote de dados recebido da camada de transporte é dividido em pedaços chamados datagramas. Os datagramas são enviados para a camada de interface com a rede, onde são transmitidos pelo cabeamento da rede através de quadros.

INTERFACE

A tarefa dessa camada é receber e enviar pacotes pela rede. Os protocolos utilizados nessa camada dependem do tipo de rede que está sendo utilizado. Atualmente, o mais comum é o Ethernet, disponível em diferentes velocidades.

O modelo utilizado por essa protocolo é o de cliente/servidor, onde um computador envia solicitações, como carregar uma página da web, a outro computador. O TCP (Protocolo de Controle de Transmissão) é o responsável por quebrar uma mensagem em partes menores, enviando-as pela internet. O computador que recebe esses pacotes de informação utiliza outra ferramenta do TCP para reunir estes dados na mensagem original.

O IP (Protocolo de Interconexão), é importante para que os pacotes cheguem ao destino certo, é ele que fornece o endereço certo para a entrega das informações. Nesse processo, os fragmentos da mensagem original podem até tomar rotas diferentes, mas chegarão ao mesmo destino para que a informação esteja completa.

Projeto

O projeto apresentado neste documento, bem como sua documentação está disponível para acesso público no Github por meio do link: https://github.com/MatheeMene/final_SO

As informações contidas nesse documento tratam-se de um estudo do protocolo TCP/IP e como ele atua na rede Docker implementada no projeto.

Para iniciar as operações, certifique-se de que tenha Docker e Node instalados na máquina. Ambos podem ser encontrados nos links:

https://nodejs.org/en/
https://docs.docker.com/engine/install/

Após instalados, clonar o repositório do Github https://github.com/MatheeMene/final_SO e rodar o comando:

npm install

Uma das primeiras necessidades do projeto foi tornar fixo o IP do contêiner *master*. Uma das ferramentas do Docker *daemon* é a criação de redes para os containers. Conseguimos obter a rede para nosso projeto usando o comando:

docker network create --subnet=172.18.0.0/16 balancer-net

Argumento	Significado
network	Indica que os argumentos seguintes tratarão da criação/modificação de alguma rede do Docker.
create	Indica que iremos criar uma rede.
--subnet=172.18.0.0/16	Aqui indicamos o endereço da nossa rede.
balancer-net	Nome da rede que iremos criar.

Dessa forma, criamos uma rede com IP 172.18.0.0, com máscara de rede 255.255.0.0. Essa configuração permite mais de 65000 hosts, o que é mais do que suficiente para abrigar os contêineres que iremos criar. O próximo passo será criar o contêiner master, que irá orquestrar o balanceamento de carga.

Antes de criá-lo, precisamos de uma imagem específica que atenda às nossas necessidades. Para isso, criamos a Dockerfile que se encontra no projeto.

Comando	Significado
FROM archlinux:latest	Começamos informando de qual imagem pré existente no Docker Hub queremos partir. Nesse caso, escolhemos uma

	imagem do SO Arch Linux.
COPY package*.json ./	Aqui copiamos todos os arquivos que iniciem com *package* e tenham a extensão *.json* para dentro do contêiner.
RUN pacman -Sy	Sincroniza o banco local do gerenciador de pacotes.
RUN pacman -S --noconfirm nodejs npm RUN pacman -S --noconfirm net-tools RUN pacman -S --noconfirm vim RUN pacman -S --noconfirm iputils RUN pacman -S --noconfirm iproute2	Com essa série de comandos, instalamos alguns utilitários como node, npm, net-tools, vim, iputils, iproute2.
RUN npm i	Instala as dependências do projeto, especificadas no *package.json*.
COPY . .	Copia os arquivos do projeto para dentro do contêiner.
EXPOSE 3000	Expõe a porta 3000 para o host.
CMD [“node”, “master.js”]	Comando que inicia as operações. Como o contêiner master e nodo precisam basicamente das mesmas bibliotecas e configurações, para criar o contêiner nodo só precisaremos alterar a última parte do comando de “master.js” para “nodo.js”.

Para criar a imagem do contêiner, basta rodar o comando:

```
docker build -t "master:latest" .
```

Argumento	Significado
build	Informa que os argumentos seguintes irão usar o módulo de criação de imagens do Docker
-t “master:latest”	Indica a *tag* da imagem. Para criar o contêiner nodo, só precisamos modificar para “nodo:latest”.
.	Local da Dockerfile que será utilizada.

Por fim, rodamos o contêiner master:

```
docker run --ip 172.18.0.22 -it --net balancer-net master:latest
```

Destrinchando o comando, podemos notar alguns detalhes:

Argumento	Significado
run	Roda o contêiner com os argumentos inseridos
--it	Trata-se de uma abreviação de *--iteractive* + *--tty*. Com essa flag, seremos direcionados diretamente para dentro do contêiner, o que nos permitirá ver logs e acompanhar o funcionamento.
--net	Aqui iremos especificar a rede que criamos. Nesse caso, passamos “balancer-net” como argumento.
master:latest	Por fim, especificamos o nome da imagem do contêiner que acabamos de criar. O sufixo “latest” indica ao Docker que desejamos a última versão criada dessa imagem.

Acessando o container, conseguimos ver que as placas de rede estão funcionando e configuradas conforme desejado.

```
→ final_SO git:(master) ✗ docker exec -it 770b9da71f88 /bin/bash
[root@770b9da71f88 /]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.18.0.22  netmask 255.255.0.0  broadcast 172.18.255.255
        ether 02:42:ac:12:00:16  txqueuelen 0  (Ethernet)
        RX packets 1112  bytes 192155 (187.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1090  bytes 103942 (101.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@770b9da71f88 /]#
```

Dessa forma, nosso orquestrador está quase pronto para receber novos nodos e carga. A última parte é rodar o script statusFetcher.js. Por tratar-se de um projeto que usa conteinerização, ou seja, temos as instâncias rodando diretamente no kernel do sistema,

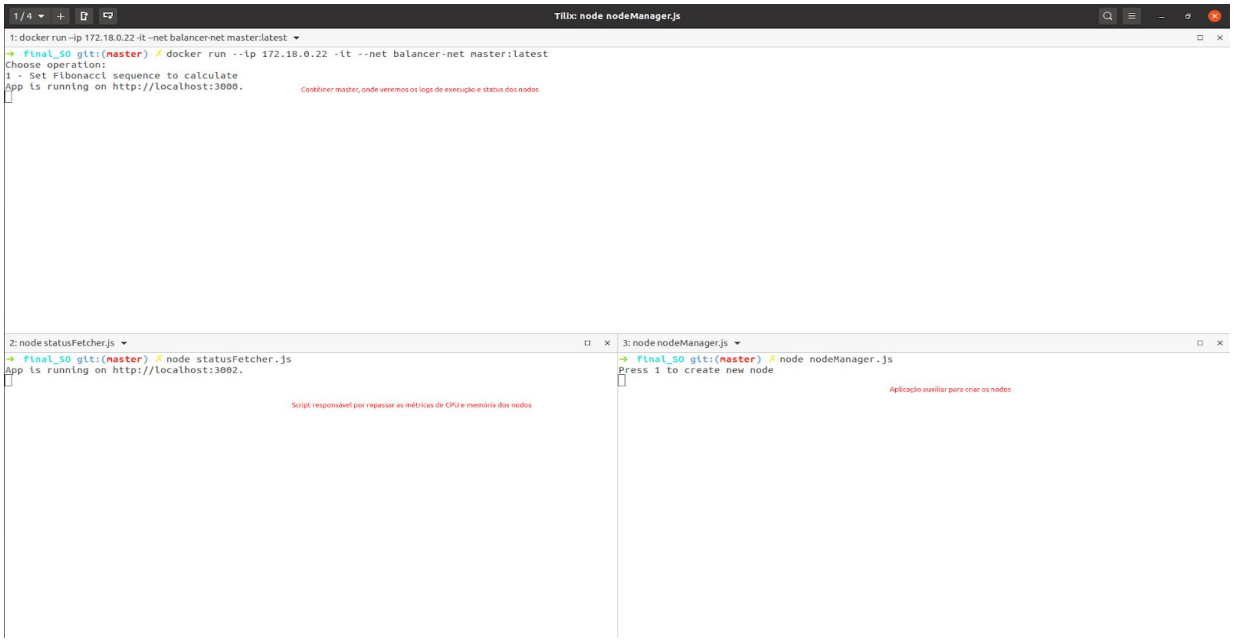
não conseguimos ter métricas de consumo de CPU e memória de dentro dos contêineres, pois eles estão compartilhando o mesmo hardware do restante do sistema. A forma de obtermos esses dados é utilizando ferramentas do próprio Docker, que está rodando no hospedeiro.

node statusFetcher.js

Com todos esses passos completos, podemos finalmente rodar o script nodeManager.js e começar a criar novos nodos para nosso balanceador de carga. Fazemos isso com o seguinte comando:

node nodeManager.js

O setup final deve ficar parecido com a captura de tela a seguir:



Dessa forma, para utilizar a aplicação, basta rodar alguns nodos na janela do nodeManager e enviar alguma carga usando a janela do contêiner mestre. Utilizamos a sequência de Fibonacci para simular carga por ser um algoritmo custoso em termos de processamento.

Referências

IBRAHIM, Issam. Conjunto de protocolos TCP/IP e suas escolhas. 2011. Disponível em: http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/666/1/CT_TELEINFO_XIX_2011_12.pdf. Acesso em: 28 nov. 2020.

MARTINS, Elaine. O que é TCP/IP? 2012. Disponível em: https://www.tecmundo.com.br/o-que-e/780-o-que-e-tcp-ip-.htm. Acesso em: 28 nov. 2020.

CISCO. Visão geral do TCP/IP. Disponível em: https://www.cisco.com/c/pt_br/support/docs/ip/routing-information-protocol-rip/13769-5.pdf. Acesso em: 28 nov. 2020.

GAIDARGI, Juliana. O que é TCP/IP e como funciona. 2018. Disponível em: https://www.infonova.com.br/artigo/o-que-e-tcp-ip-e-como-funciona/. Acesso em: 28 nov. 2020.

POSTEL, Jon. INTERNET PROTOCOL. 1981. Disponível em: https://tools.ietf.org/html/rfc791