

TRYBE

Modulo IV – Ciência da Computação

Estrutura de computador, Python, programação orientada a objetos, algoritmos, estruturas de dados.

Bloco 34: Arquitetura de Computadores e Redes

1) Arquitetura de computadores

História

Alan Turing é considerado o pai da computação.

ENIAC primeiro computador, 1946, exército US.

Programa mais velho: série de Bernoulli, máquina analítica, Ada Lovelace.

Computadores Modernos

Modelo de Von Neuman possui dois elementos:

- **memória principal** (como a memória RAM), onde podemos registrar e ler instruções e dados;
- **processador** (CPU), responsável por buscar tais informações, realizar os cálculos e armazenar os resultados novamente na memória.

Unidade	Símbolo	Valor
bit	1b	menor unidade
byte	1B	8b (bits)
kilobyte	1KB	1024B (bytes)
megabyte	1MB	1024KB
gigabyte	1GB	1024MB
terabyte	1TB	1024GB
petabyte	1PB	1024TB
exabyte	1EB	1024PB
zetabyte	1ZB	1024EB
yotabyte	1YB	1024ZB

unidades de medidas

Lógica binária

Bit: menor unidade em um sistema digital, com valor 0 ou 1.

Transistores: corrente passando é true 1, não passando false 0.

Portas lógicas: NOT AND OR XOR com mesma entrada e saída diferente, formando circuitos e mais operações.

Memória Principal – RAM

Grande biblioteca onde:

- cada prateleira possui endereço único como identificação (**ADRESSES ou ADDR das células**);
- capacidade total é **quantidade de célula x capacidade de armazenamento** de cada uma;
- endereço é um conjunto de n° binários, com número variável de dígitos.

Processador – CPU

CPU: Unidade Central de Processamento. Funciona com memória principal.

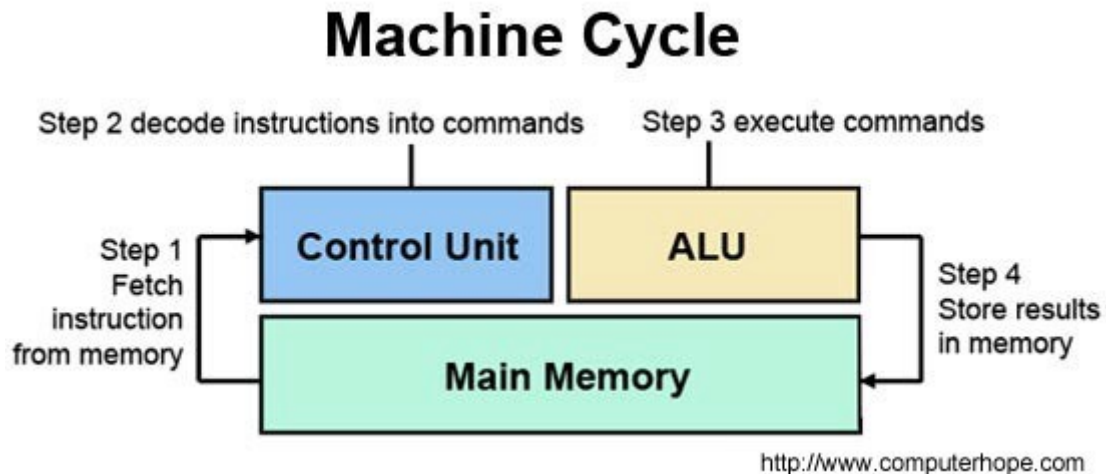
Elementos:

- Componente **ALU** (unidade lógico-aritmética);
- **Registradores**, células internas de memória para dados usados no processamento (operações são números e conjunto delas chamam **instrução**);

- “**clock**”, sinal alternando baixas e altas tensões a cada fração de tempo, gerado por um componente eletrônico, para gerenciar todas atividades. Frequência em Hertz.
 $1\text{Hz} = 1 \text{ operação} / \text{segundo}$.

Loop do CPU:

- 1) realiza a leitura de algumas instruções pré-definidas,
- 2) executa elas,
- 3) passa a buscar e conseqüentemente a executar as instruções na memória.

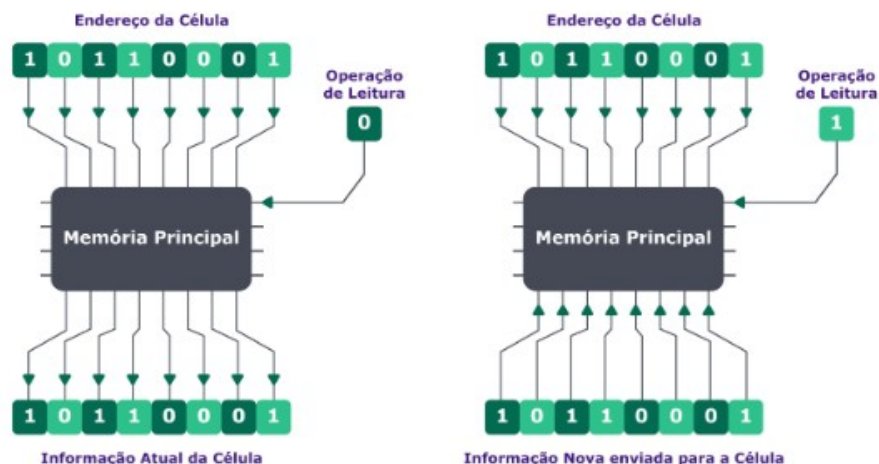


Barramentos

2 Operações: load (step 1 acima) & store (step 4).

3 Barramentos, vias que ligam CPU com memória:

- **Endereço (ADDR)** da célula de memória para aquela operação;
- **Dados (DATA)** da informação da memória (bidirecional);
- **Controle (CTRL)** indicando a "direção" dos dados para a operação, ou seja, se os dados serão transferidos da CPU para a memória (escrita) ou da memória para a CPU (leitura).



// barramento de controle determina o sentido do barramento de dados: 0 leitura, 1 escrita

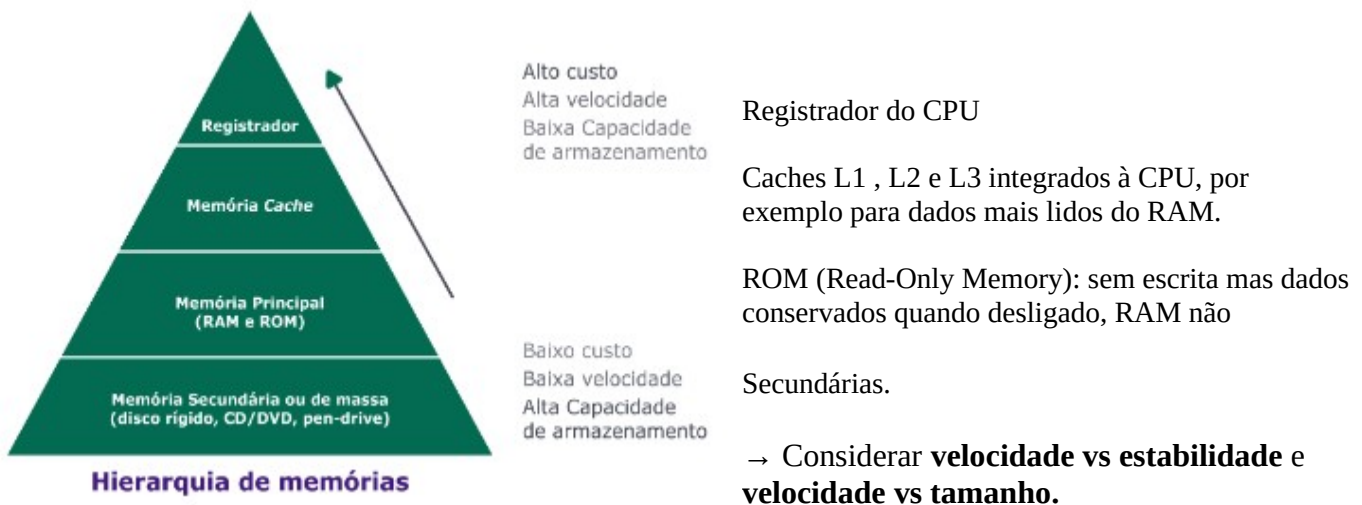
Tipos de Memória

2 grupos:

- memórias **primárias**: armazenamento temporário, acesso rápido, menor volume de dados, não conseguem seguir desligados;
- memórias **secundárias**: permanente, acesso mais lento, grandes volumes.

Hierarquia das memórias

Faz com que os **dados que são acessados com mais frequência** sejam armazenados em **memórias de acesso mais rápido**.



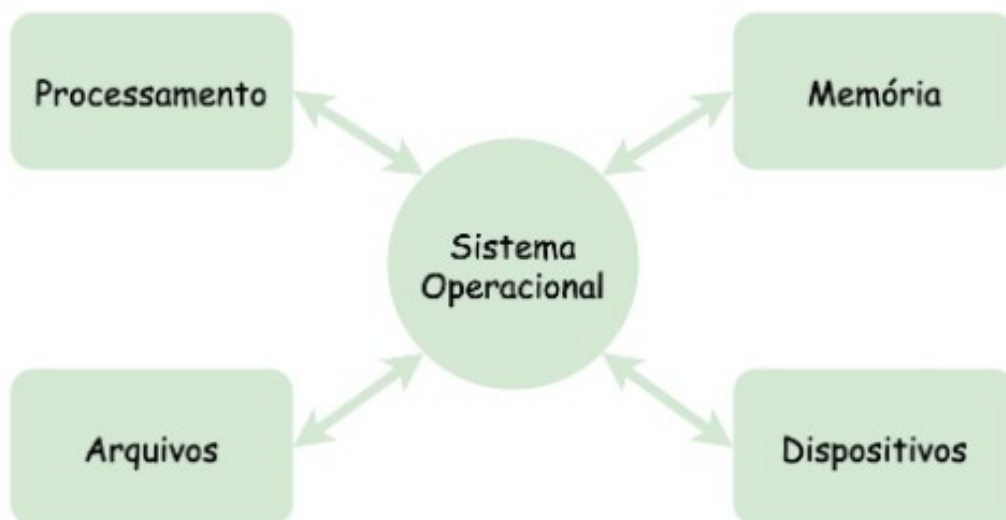
Sistema Operacional & Gerenciamentos

Interface para facilitar o acesso ao funcionamento, permitindo a invenção e o uso de computadores pessoais (Steves: Jobs / Wozniak, Bill Gates).



Todo SO trabalha como um **intermediário**, realizando a **gerência do hardware e dos softwares**, controlando os processos, arquivos, memória, rede e os dispositivos conectados ao computador.

O SO **distribui os processos** a serem executados pelo CPU, "cérebro" do computador.
Comandos terminal *ps auxww* e *top* (fotografia e tempo real dos processos).



SO executa processo de **alocação de memória** (troca de dados com o processador, a memória principal e a memória secundária, procurando espaços vazios).

Vid detalhes:

3 antigos esquemas: *single job*, *contiguous*, *fixed partitions*, *dynamic partitions*, alocados via *first fit*, *best fit*.

Hoje: **virtual memory**:

- *page memory allocation & keeping track with job, page map and memory mab tables & page replacement policy*;
- *segmented memory allocation*;
- *hybrid allocation*.

SO também **controla arquivos do computador** e permite operações CRUD neles, gerindo também o **sistema de permissões**.

Dicas diversas

Como acessar via node no terminal informações de hardware:

node

```
const sistema = require('os')
```

```
sistema.arch()
```

```
sistema.cpus()
```

```
sistema.version()
```

```
sistema.homedir()
```

```
sistema.freemem()
```

```
sistema.release()
```

Sem variável, podia escrever diretamente *os.function()*

Todas functions possíveis listadas na doc <https://nodejs.org/api/os.html> .

2) Arquitetura de redes

Internet

Implementação de uma **rede de computadores**. Sem cérebro central, vários data centers e nuvem.

Cabo submarino mais eficiente do que ida e volta usando satellite que teria interferência solar.

Tempo download > upload: rede estruturada para ter mais velocidade para o user mandar.

5Generation: novo equip. Internet das coisas: controlar coisas pela internet, misturado com AI.

Redes de computadores

Conjuntos de software (trafegando infos) e **hardware** (componente físicos) **que permitem a comunicação** entre diversos dispositivos.

Tamanhos diversos: *Personal, Local, Neighbourhood, Metropolitan, World Area Network*.



Pacotes

Para **trafegar dados via rede**: converter dados para binários, pedaços são os "pacotes".

Contém dados de envio: quem enviou, destino, indicações em caso de perda ou reenvio.

Protocolos

Conjuntos de **regras que controlam como os dados são trocados**. Essa padronização garante que quem recebe info sabe ler info.

Modelo de Rede

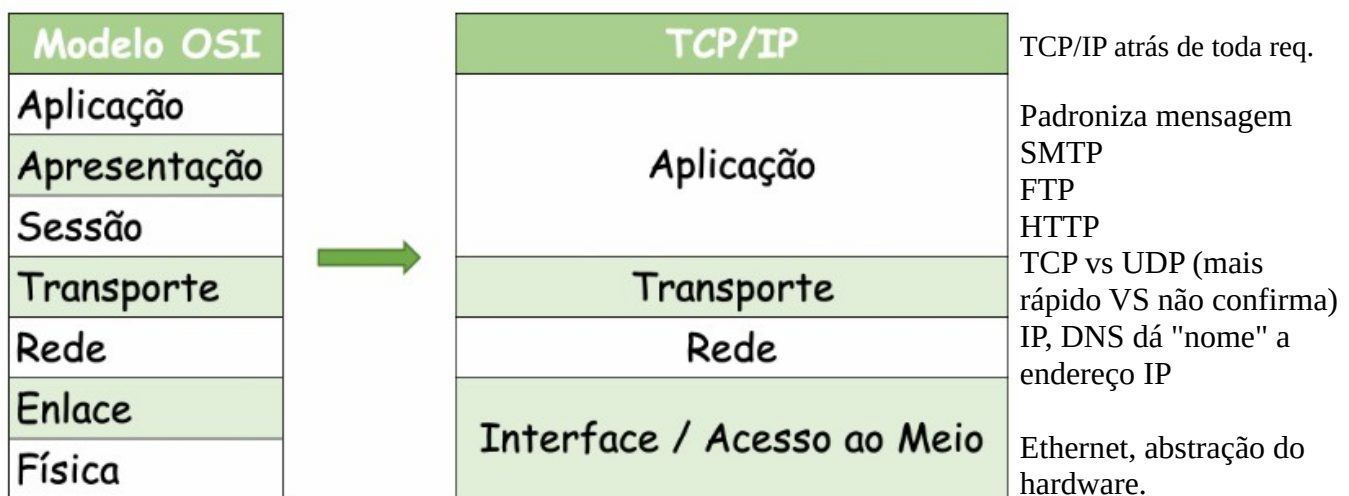
Modelo é o conjunto de camadas de protocolos, cada um definindo regras específicas para a parte pela qual é responsável.

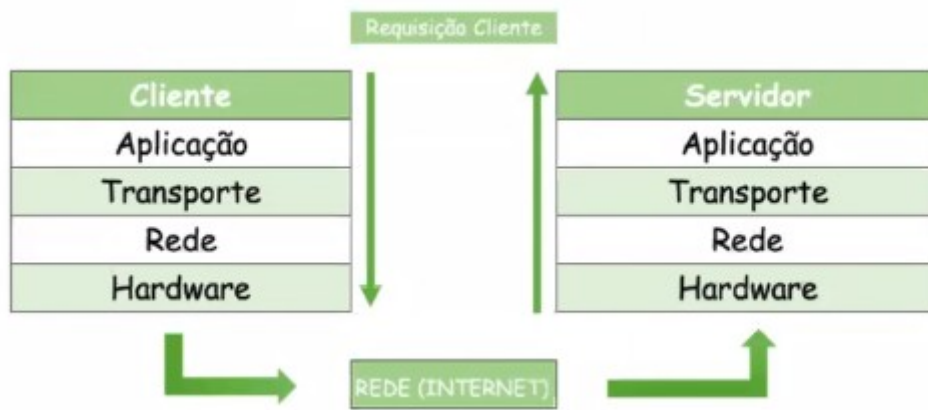
Modelo ISO/OSI (Open Systems Interconnection)

7 camadas: Física hardware, Enlace corrigir erros, rede endereço, transporte ordem dos pacotes, sessão comunicação entre 2 máquinas, apresentação inclui criptografia, aplicação efetivamente recebe info.

&

Modelo Internet – TCP/IP (protocolos de comunicação: *Transmission Control Protocol* e *Internet Protocol*).





Dicas diversas / mão na massa

→ **cURL** para realizar uma chamada HTTP (cURL utiliza TCP com controle de fluxo e não UDP) terminal:

`curl your-url`

`curl -X GET your-url`

`cURL -k` ou `-insecure` (prosseguir a request mesmo sabendo que a conexão não é "confiável")

→ **dgram** para criar servidor UDP

```
const dgram = require("dgram");
const server = dgram.createSocket("udp4");
server.on("message", (message) => {
  console.log(message.toString());
});
server.bind(3001, () => console.log("Servidor UDP rodando"));
```

`.write` -> Escreve algo dentro do túnel da comunicação.

`.on` -> Pega os dados que estão sendo trafegado no túnel de comunicação.

`.listen` executa o servidor no NET/Express etc

`.bind` executa o servidor no Dgram

// Quando na criação do servidor dgram fazemos um `.createSocket("udp4")` estamos informando que a versão protocolo IP será a 4.

→ **NET** para criar server TCP, sintaxe parece socket.io

```
const net = require("net");
const server = net.createServer((socket) => {
  socket.write("Olá cliente");
  socket.on("data", (data) => {
    console.log(data.toString());
  });
});
server.listen(3000, () => console.log("Servidor TCP rolando"));
// Para ver a troca, node index.js num terminal e no outro, cliente, telnet 127.0.0.1 8080
```

→ **Inspect – Network** para ver requisições

Headers para ver tipos de objetos esperados, e eventualmente falha de segurança

→ **google scholar** para acompanhar notícias, por exemplo sobre IoT (internet of things)

→ Saber o IP externo da sua máquina no <https://www.meuip.com.br/>

3) Redes de computadores, ferramentas e segurança

Segurança tanto **nas comunicações** quanto **nas máquinas** mesmo.

Criptografia e certificados

Algoritmos de criptografia oferecem um jeito de codar mensagens para que eles circulem entre pessoas autorizadas. **Chaves** de criptografia: um n° permite embaralhar, outro desembaralhar.

Autoridades certificadoras: garantem que tal página é reconhecida por proteger comunicações. Uma empresa analisa página e entrega certificado de segurança.

Protocolos Seguros

Protocolos projetados para fornecer segurança (garantir identidades, evitar interceptações).

SSL (Secure Sockets Layer) e **TLS (Transport Layer Security)** implementam uma camada de segurança na rede. TLS é sucessor de SSL.

HTTPS (Hyper Text Transfer Protocol Secure) é o HTTP com mais uma camada que utiliza o protocolo SSL/TLS.

Tipos de ataques

DoS / DDoS

DDoS (*Distributed Denial of Service*), ataque "bombardeando" aplicação com diversos acessos simultâneos, com o objetivo tirar serviço do ar.

Brute Force

Indivíduos, robôs ou scripts maliciosos que tentam diversas combinações de usuário e senha. Prevenir com senhas mais fortes e proteger servidores com camada extra.

Firewall e Proxy

Definição de firewalls

Portas corta fogo para impedir ameaças entrarem/sairem.

Iptables e Netfilter

Netfilter: sistema padrão para filtragem de pacotes do linux.

Iptables: firewall padrão do linux, para configurar Netfilter. *Compara o tráfego de rede que recebe ou envia com um conjunto de regras pré estabelecidas* (sobre características esperadas, ação em cada caso, regras por protocolo, porta de origem/destino, endereço IP).

Fail2ban

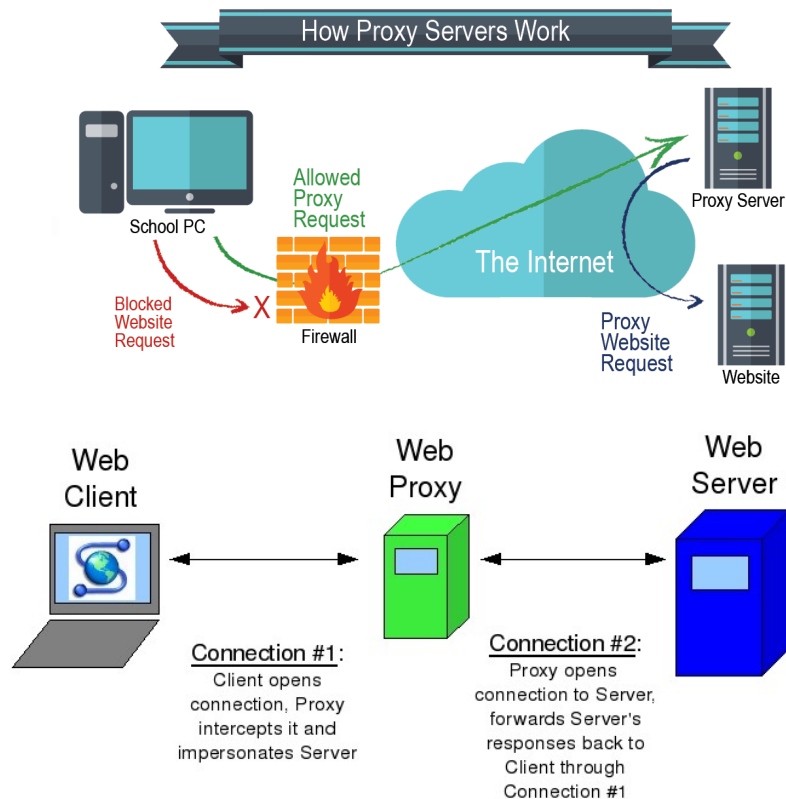
IPS (*Intrusion Prevention System*) que monitora os logs e rejeita IPS com comportamento suspeito, gerando *noiptables*.

Proxy

Mais uma camada de segurança **entre a internet e os dispositivos da rede**.

Espécie de **filtro de tráfego**, condicionando acesso ou não do user para umas urls.

Tem papel complementar ou substitutivo do firewall.



Dicas diversas / mão na massa

- Monitorar seu app e nº de requisições, ver se picos correspondem com mkt ou ataques.
- Quanto mais a estrutura do app se mantém, mais é vulnerável pois atacantes conhecem.

→ **ping** (packet internet grouper) para medir tempo de resposta da conexão do computador com outros dispositivos da rede local ou da internet (usa protocolo ICMP).

`$ ping google.com`

(pode pingar por nome ou por url)

`$ ping -c 4 google.com`

(- c para especificar o nº de pacotes)

→ **traceroute** para descobrir o caminho dos pacotes desde origem até destino. Pode até observar se é rede dedicada.

`$ traceroute google.com`

→ **IP interno vs IP externo**

Interno: endereço de um dispositivo conectado em uma rede local doméstica

Externo: endereço de um dispositivo conectado à rede externa ou rede mundial de computadores, a internet.

- comandos para configurar segurança

`$ brew cask install docker`

`$ docker ps`

`$ docker run --privileged -it ubuntu:20.04 bash`

`$ apt-get update && apt-get install iputils-ping`

`$ apt-get update && apt-get install traceroute`

`$ apt-get update && apt-get install iptables`

- Padrão de segurança: negar tudo e escrever regras sobre o que aceita.

→ iptables & seus [params](#)

\$ iptables -L

Para ver as regras já configuradas. Mostra as de entrada (INPUT), saída (OUTPUT) e encaminhados (FORWARD).

\$ iptables -A INPUT|FORWARD|OUTPUT

Para setar a chain para qual a regra deve ser alocada.

O iptables por padrão usa a table para armazenar as regras, e dentro dessa table existem as Chains, que são: INPUT, FORWARD, OUTPUT.

\$ iptables -A INPUT -p icmp

Para informar o protocolo no qual a regra vai ser aplicada.

\$ iptables -A INPUT -p icmp --icmp-type echo-request

Para informar o tipo de mensagem dentro do protocolo, no qual a regra vai ser aplicada.

\$ iptable -A INPUT -p icmp -j DROP|ACCEPT|REJECT

Para informar a ação que será executada caso o pacote cair na regra. A ordem desses comandos importa muito, aceitar antes de rejeitar para que casos se encaixem em um ou outro.

\$ iptable -A input -p icmp -m limit --limit 10/minutes -j DROP

Para informar um limite de tempo de aceitação do pacote, formato Quantidade/MedidaDeTempo. Boa estratégia cortar tempo de reação do hacker.

\$ iptables --flush

Para apagar todas as regras do iptables.

→ proxy [ngrok](#) para criar túnel para nosso localhost.

→ [openSSL](#) para gerar certificados.

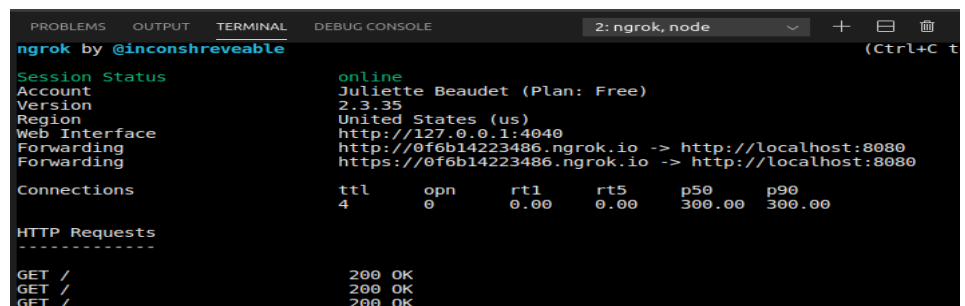
→ criar server [HTTPS](#) com pacote node.

4) Projeto – Explorando os protocolos

Dicas diversas do projeto

→ [Padrão de portas](#): 80 para req HTTP, 443 para HTTPS

→ [ngrok set-up](#)



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 2: ngrok, node
ngrok by @inconsheveable (Ctrl+C t)

Session Status online
Account Juliette Beaudet (Plan: Free)
Version 2.3.35
Region United States (us)
Web Interface http://127.0.0.1:4040
Forwarding http://0f6b14223486.ngrok.io -> http://localhost:8080
Forwarding https://0f6b14223486.ngrok.io -> http://localhost:8080

Connections
t1l opn rt1 rt5 p50 p90
4 0 0.00 0.00 300.00 300.00

HTTP Requests
-----
GET / 200 OK
GET / 200 OK
GET / 200 OK
```

→ em caso de problema de porta já usada inexplicadamente, lembrar do *killall node*.