**Fundamentos de Redes para Devops – Dia 1 – 25/08/2025**

**1. Introdução ao Treinamento**

Overview de fundamentos de rede para Devops

**Dia 2 – 25/08/2025**

**2. Como instalar o Linux**

https://medium.com/@RedsonBr140/como-instalar-qualquer-distro-linux-sem-pendrive-a547c8017d6c

**Dia 3 – 23/08/2025**

**3. Introdução ao TCP/IP**

O modelo TCP/IP é um conjunto de protocolos de comunicação usado para interligar dispositivos em redes, como a internet. Ele é dividido em quatro camadas, cada uma com funções específicas:

**1. Camada de Aplicação (L7 – Application Layer)**

* Essa camada é onde ocorrem as interações entre o usuário e a rede.
* Engloba os protocolos e aplicações que usam a rede para enviar e receber dados.
* Exemplos de protocolos:
  + HTTP/HTTPS (navegação na web)
  + FTP (transferência de arquivos)
  + SMTP, POP3, IMAP (e-mails)
  + DNS (resolução de nomes)

**Responsabilidade:** Facilitar a comunicação entre aplicações e a rede.

**2. Camada de Transporte (L4 – Transport Layer)**

* Garante a entrega confiável ou não dos dados entre dispositivos.
* Controla a fragmentação, envio e recebimento dos pacotes.
* Dois principais protocolos:
  + **TCP (Transmission Control Protocol):** Confiável, com controle de erros, usado em e-mails e navegação.
  + **UDP (User Datagram Protocol):** Mais rápido, mas sem garantia de entrega, usado em streaming e VoIP.

**Responsabilidade:** Gerenciar a conexão entre origem e destino, garantindo ou não a confiabilidade.

**3. Camada de Rede (L3 – Network Layer)**

* Determina a melhor rota para o envio dos pacotes.
* Usa endereços IP para identificar remetente e destinatário.
* Responsável pelo roteamento e encaminhamento dos pacotes.
* Protocolos importantes:
* IP (Internet Protocol): Define os endereços e roteia pacotes.
* ICMP (Internet Control Message Protocol): Diagnóstico de rede (ping).
* ARP (Address Resolution Protocol): Converte IP em MAC.

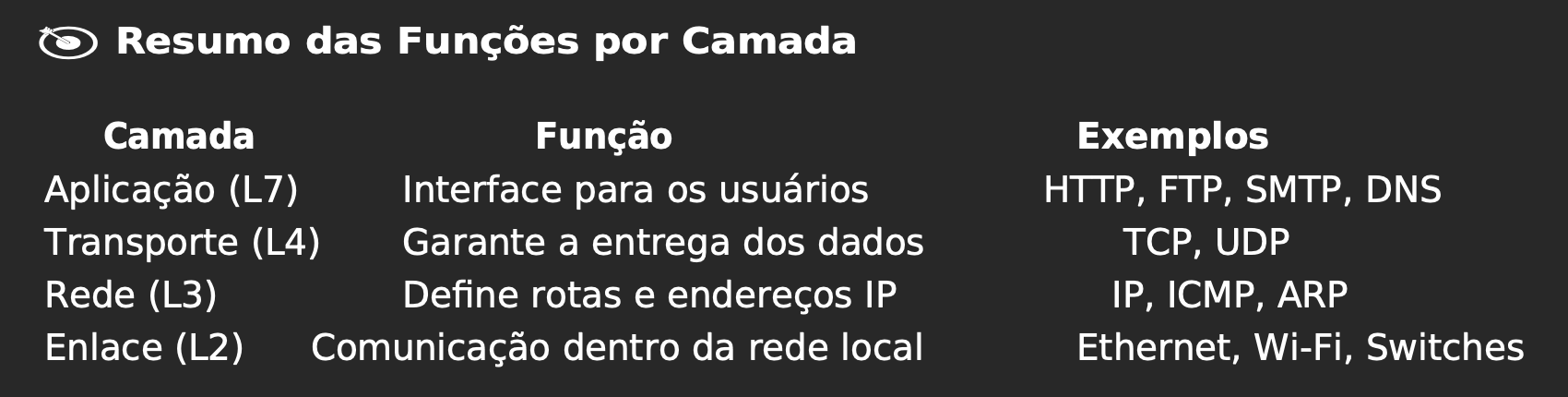
**Responsabilidade:** Roteamento e endereçamento dos pacotes na rede.

**4. Camada de Enlace (L2 – Data Link Layer)**

* Garante a transmissão de dados entre dispositivos conectados fisicamente.
* Trabalha com endereços MAC para comunicação dentro de uma rede local.
* Controla erros na transmissão dentro de um mesmo segmento de rede.
* Protocolos e tecnologias:
* Ethernet (IEEE 802.3)
* Wi-Fi (IEEE 802.11)
* PPP (Point-to-Point Protocol)
* Switches, VLANs

**Responsabilidade:** Comunicação dentro da rede local, usando endereços físicos (MAC).

**Resumo das Funções por Camada**



**4. A vida de um pacote**

O ciclo de vida de um pacote em uma rede TCP/IP descreve todas as etapas que ele percorre, desde a

origem até o destino. Isso envolve encapsulamento, roteamento, transmissão e decapsulamento. Vamos

ver isso em detalhes:

**1. Criação do Pacote (Camada de Aplicação - L7)**

Tudo começa quando um usuário ou uma aplicação deseja enviar dados. Por exemplo:

* Um navegador acessa um site via HTTP.
* Um programa envia um e-mail via SMTP.
* Um usuário faz download de um arquivo via FTP.

Nesse momento, os dados são formatados de acordo com o protocolo de aplicação (exemplo: HTTP, DNS, FTP).

**2. Encapsulamento (Camada de Transporte - L4)**

A camada de transporte recebe os dados da aplicação e adiciona informações de controle:

* Se o protocolo for TCP, ele adiciona:
* Número de sequência e reconhecimento.
* Controle de erros (checksum).
* Garantia de entrega.

Se o protocolo for UDP, ele apenas adiciona:

* o Porta de origem e destino.
* o Checksum (opcional).

Exemplo:

* O navegador envia uma solicitação HTTP.
* O protocolo TCP adiciona as informações de controle.

**3. Endereçamento e Roteamento (Camada de Rede – L3)**

Nesta etapa, os pacotes são preparados para viajar na rede:

1. O protocolo IP adiciona:

* Endereço IP de origem (do remetente).
* Endereço IP de destino (para onde o pacote vai).
* TTL (Time to Live): Limita o número de roteadores pelos quais o pacote pode passar antes de ser descartado.

2. Se o destino estiver fora da rede local, o pacote é enviado para o gateway (roteador).

**4. Transmissão Física (Camada de Enlace - L2)**

Agora, os pacotes são convertidos em quadros (frames) para serem transmitidos:

* O protocolo de enlace (como Ethernet ou Wi-Fi) adiciona:
  + Endereço MAC de origem (placa de rede do remetente).
  + Endereço MAC de destino (pode ser o roteador, se for necessário roteamento).
  + Verificação de erros (CRC - Cyclic Redundancy Check).

Se for uma rede local (LAN): O pacote pode ir direto ao destino.

Se for para outra rede: Ele passa pelo roteador.

**5. Roteamento (Passagem por Múltiplos Roteadores)**

Se o destino está em outra rede, o pacote passa por diversos roteadores:

1. O roteador lê o endereço IP de destino.

2. Decide a melhor rota e envia o pacote para o próximo roteador.

3. A cada roteador, o endereço MAC muda, mas o endereço IP permanece.

Se o TTL chegar a 0 antes de alcançar o destino, o pacote é descartado e um erro ICMP "Time Exceeded" é enviado ao remetente.

**6. Recepção no Destino**

Quando o pacote chega ao destino final:

1. A Camada de Enlace (L2) verifica se o MAC de destino corresponde ao dispositivo correto.

2. A Camada de Rede (L3) verifica se o IP é o certo.

3. A Camada de Transporte (L4) reagrupa os pacotes (se for TCP).

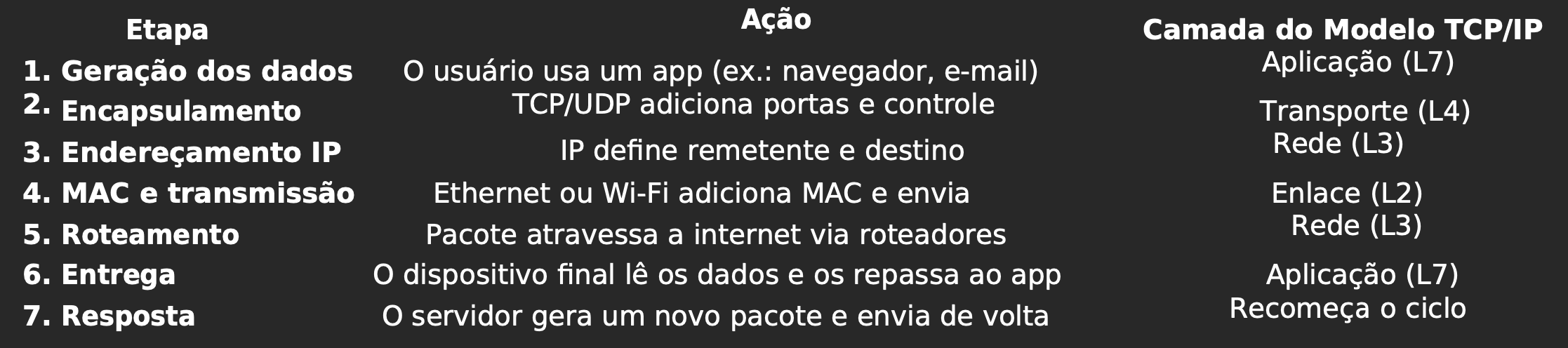
4. A Camada de Aplicação (L7) lê os dados e exibe para o usuário.

Exemplo: O navegador recebe a resposta do site e exibe a página.

**7. Resposta do Servidor**

Se a comunicação precisa de uma resposta (como um site carregando), o servidor cria um novo pacote e envia de volta para o cliente, repetindo o mesmo ciclo.

**Resumo do Ciclo de Vida de um Pacote**



**Exemplo Prático: Acessando um Site**

1. Você digita {HYPERLINK "http://www.google.com" \t "\_new"} no navegador (Camada de Aplicação).

2. O TCP cria pacotes para a requisição HTTP (Camada de Transporte).

3. O IP define que os pacotes vão para o servidor do Google (Camada de Rede).

4. O Wi-Fi ou Ethernet envia os pacotes pela rede (Camada de Enlace).

5. O roteador e os servidores da internet encaminham os pacotes.

6. O servidor do Google recebe, processa e responde.

7. O ciclo se repete até o site ser carregado.

**5. HTTP Message (Request vs Response)**

O protocolo HTTP funciona com mensagens de requisição (request) e mensagens de resposta (response).

**Request (Requisição)**

Quando um cliente (como um navegador) quer acessar um recurso, ele envia uma requisição HTTP para um servidor.

**Estrutura da Requisição:**

GET /index.html HTTP/1.1

Host: www.exemplo.com

User-Agent: Mozilla/5.0

Accept: text/html

* Linha de requisição: Contém o método (GET), o recurso (/index.html) e a versão do HTTP
* (HTTP/1.1).
* Cabeçalhos: Informações extras como o host, user-agent e tipos de conteúdo aceitos.
* Corpo (Opcional): Usado em POST e PUT para enviar dados.

**Response (Resposta)**

O servidor recebe a requisição e responde com um código de status e os dados solicitados.

**Exemplo de Resposta:**

HTTP/1.1 200 OK

Content-Type: text/html

Content-Length: 342

<html>...</html>

* Linha de status: Mostra o código HTTP (200 OK, 404 Not Found, etc.).
* Cabeçalhos: Tipo de conteúdo, tamanho da resposta, etc.
* Corpo: O conteúdo real da página ou recurso solicitado.

**Resumindo:**

* O cliente envia um request para o servidor.
* O servidor processa e responde com um status e os dados solicitados.

**Codigos de erros importantes:**

### **✅ 1xx – Informativo (menos comuns, mas úteis em proxies/gateways)**

* 100 Continue: o cliente pode continuar com a requisição.
* 101 Switching Protocols: troca de protocolo (ex: HTTP para WebSocket).

### ✅ **2xx – Sucesso**

* 200 OK: requisição bem-sucedida.
* 201 Created: recurso criado com sucesso (usado em POST).
* 204 No Content: requisição bem-sucedida, mas sem corpo de resposta (útil para DELETE ou PUT).

### ⚠️ **3xx – Redirecionamento**

* 301 Moved Permanently: redirecionamento permanente (importante para SEO).
* 302 Found: redirecionamento temporário.
* 304 Not Modified: usado com cache — indica que o conteúdo não foi modificado.

### ❗ **4xx – Erro do cliente**

* 400 Bad Request: requisição malformada (frequente em APIs).
* 401 Unauthorized: autenticação necessária (ex: token inválido ou ausente).
* 403 Forbidden: acesso negado mesmo com autenticação.
* 404 Not Found: recurso inexistente.
* 408 Request Timeout: cliente demorou para enviar a requisição.
* 429 Too Many Requests: limitação de taxa (rate limiting) — importante em ambientes com throttling ou controle de APIs.

### 🔥 **5xx – Erro do servidor**

* 500 Internal Server Error: erro genérico no servidor.
* 502 Bad Gateway: erro de proxy/reverso (ex: NGINX não consegue contato com o backend).
* 503 Service Unavailable: serviço temporariamente indisponível (pode ser durante deploy ou overload).
* 504 Gateway Timeout: timeout entre proxy (como NGINX) e backend.

### 🛠️ **Contextos em que DevOps deve prestar atenção:**

* **CI/CD:** falhas ao testar APIs ou realizar deploys (ex: 500, 403, 401).
* **Logs de aplicação/Nginx/Apache:** identificar gargalos e falhas.
* **Monitoramento com Prometheus, Grafana, ELK:** alertas baseados em HTTP 5xx.
* **Gateway/API Gateway (ex: Kong, Traefik):** redirecionamentos, autorização e erros de upstream.

**Dia 2 – 15/08/2025**

**6. DEVELOPER MOZILLA: A BÍBLIA DO HTTP**

A **Mozilla Developer Network (MDN):** é um dos melhores recursos para aprender sobre HTTP e desenvolvimento web.

{HYPERLINK "https://developer.mozilla.org/pt-BR/docs/Web/HTTP" \t "\_new"}

O que você encontra lá?

• Estrutura das requisições HTTP

• Códigos de status HTTP

• Métodos (GET, POST, PUT, DELETE, etc.)

• Cabeçalhos HTTP (User-Agent, Content-Type, Authorization, etc.)

• Segurança e HTTPS

Se você quer entender HTTP profundamente, a MDN é sua melhor fonte.

**7. HTTP PERSISTENT VS NON-PERSISTENT**

O HTTP pode ser **persistente** ou **não persistente**, dependendo de como ele gerencia conexões.

**HTTP Não Persistente**

Cada requisição HTTP abre uma nova conexão TCP e fecha após receber a resposta.

**Problema:** Ineficiente, pois cria conexões repetidamente.

**HTTP Persistente**

Mantém a conexão aberta para múltiplas requisições, economizando tempo.

**Vantagem:** Reduz latência, pois não precisa abrir uma nova conexão toda vez.

**HTTP/1.0** → Não persistente por padrão.

**HTTP/1.1 e HTTP/2** → Usa conexões persistentes por padrão (Keep-Alive).

**8. COMO INSTALAR O WIRESHARK**

O Wireshark é uma ferramenta para capturar e analisar pacotes de rede.

**🔹 Instalar no Windows**

1. Baixe o instalador no site oficial → https://www.wireshark.org/download.html

2. Execute o instalador (.exe) e siga as instruções.

3. Selecione a opção para instalar o Npcap (driver necessário).

4. Finalize a instalação e abra o Wireshark.

**🔹 Instalar no Linux (Debian/Ubuntu)**

sudo apt update

sudo apt install wireshark -y

**Opcional:** Para rodar sem sudo, adicione seu usuário ao grupo wireshark: sudo usermod -aG wireshark $USER

Depois, reinicie o computador.

**🔹 Instalar no macOS**

Se tiver o Homebrew, rode:

brew install wireshark

Agora você pode capturar pacotes e analisar o tráfego da rede!

**9. O QUE É E PARA QUE SERVE UM ARQUIVO HAR?**

O **HAR (HTTP Archive)** é um arquivo que registra todas as requisições HTTP feitas pelo navegador durante a navegação em um site.

📌 Para que serve?

• Depuração de problemas de carregamento.

• Análise de performance da página.

• Identificação de erros HTTP.

• Diagnóstico de tempo de resposta de APIs.

📥 Como gerar um arquivo HAR no Chrome?

1. Abra o DevTools: Pressione F12 ou Ctrl + Shift + I.

2. Vá até a aba "Network".

3. Recarregue a página (F5) para capturar as requisições.

4. Clique com o botão direito e selecione "Save as HAR with Content".

💡 Dica: Você pode abrir esse arquivo no Wireshark ou ferramentas online para analisar o tráfego da rede.

**10. COOKIES**

Cookies de sessão são pequenos arquivos de texto temporários que os sites armazenam no navegador do usuário **apenas enquanto a sessão estiver ativa**, ou seja, **enquanto o navegador estiver aberto ou até o usuário sair do site**.

### ✅ Para que servem?

Eles ajudam a manter o estado da sessão do usuário enquanto ele navega pelo site. Por exemplo:

* **Login temporário**: mantém o usuário autenticado durante a navegação.
* **Carrinho de compras**: armazena os itens escolhidos até a finalização da compra.
* **Preferências temporárias**: idioma, layout, filtros selecionados etc.

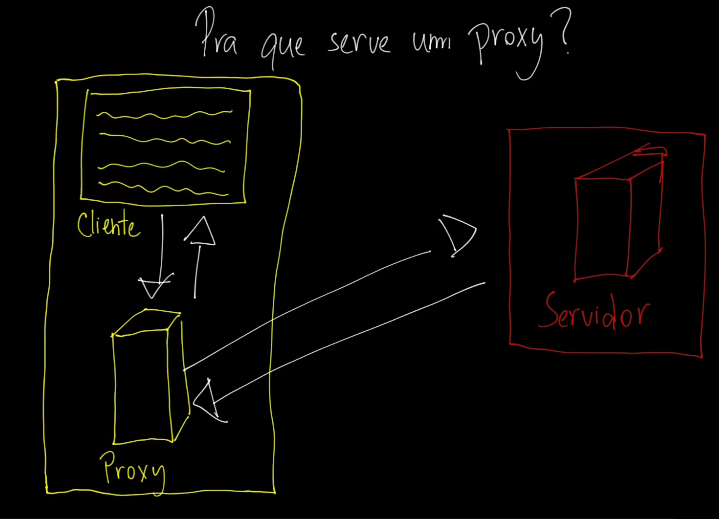
### 🔍 Características principais:

| Característica | Descrição |
| --- | --- |
| **Duração** | Até o navegador ser fechado (sessão encerrada) |
| **Armazenamento** | No navegador, mas não gravado de forma permanente no disco |
| **Segurança** | Podem ser mais seguros se usados com HTTPS |
| **Controle do servidor** | O servidor pode criar e ler esses cookies |

**Fundamentos de Redes para Devops – Dia 2 – 13/08/2025**

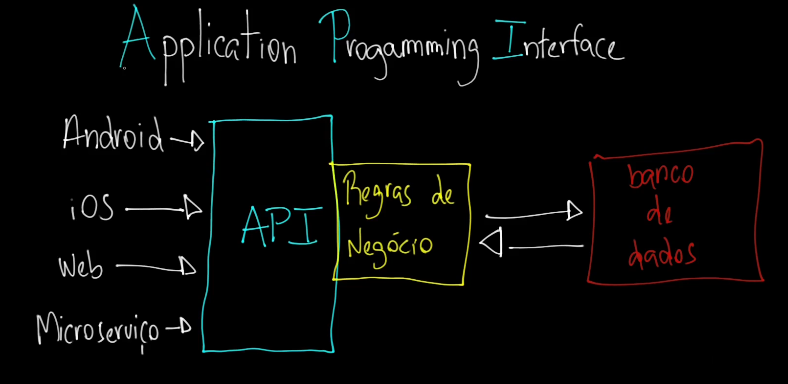
**11. O que é e para que serve um Proxy?**

Um **proxy** é um intermediário entre o seu computador e a internet. Ele recebe as suas requisições e as envia para o destino final. Serve para **esconder seu IP**, **filtrar conteúdo**, **aumentar a segurança** ou **acessar sites bloqueados**.



**12. O que é e para que serve uma API?**

Uma **API (Interface de Programação de Aplicações)** permite que programas diferentes **se comuniquem entre si**. Por exemplo, um app de clima usa uma API para pegar dados atualizados de previsão do tempo de outro sistema.



**13. Compreendendo como consumir uma API**

Consumir uma API significa **fazer uma requisição** (como um pedido) para ela e **receber uma resposta**, geralmente em formato **JSON**. Por exemplo, você envia um pedido de informações de um produto e a API responde com os dados.

**14. Um projeto de API que desenvolvi**

Você pode contar sobre um projeto onde criou uma API para, por exemplo, **cadastrar e listar clientes**. A API tinha rotas como /clientes, e usava métodos como GET (buscar), POST (criar), PUT (editar) e DELETE (excluir).

**15. Ferramenta curl pt. 1**

O curl é um **comando de terminal** usado para fazer requisições HTTP. Com ele, você pode **testar uma API**, **baixar arquivos**, ou **ver o conteúdo de uma página** diretamente pelo terminal.

| Opção | O que faz | Exemplo |
| --- | --- | --- |
| -X | Define o método HTTP (GET, POST, PUT, DELETE...) | curl -X GET https://api.exemplo.com |
| -H | Adiciona um cabeçalho HTTP (ex: tipo de conteúdo) | curl -H "Content-Type: application/json" |
| -d | Envia dados no corpo da requisição (POST, PUT) | curl -d '{"nome":"João"}' |
| -i | Mostra os cabeçalhos da resposta | curl -i https://api.exemplo.com |
| -s | Modo silencioso (sem barra de progresso) | curl -s https://api.exemplo.com |
| -o | Salva a saída em um arquivo | curl -o pagina.html https://site.com |
| -O | Salva com o nome original do arquivo do servidor | curl -O https://site.com/arquivo.zip |
| -L | Segue redirecionamentos automaticamente | curl -L http://site.com |
| -u | Autenticação básica (usuário:senha) | curl -u user:1234 https://api.exemplo.com |
| -I | Só exibe os cabeçalhos da resposta (HEAD) | curl -I https://site.com |

**16. Ferramenta curl pt. 2 - Interagindo com API usando curl**

Você pode usar o curl para **enviar dados para uma API**.

**Exemplo:**

curl -X POST -H "Content-Type: application/json" -d '{"nome":"João"}' http://api.exemplo.com/clientes

Esse comando envia dados no formato JSON para a rota /clientes.

| Opção | Função |
| --- | --- |
| -X POST | Define o método HTTP como POST |
| -H | Define um cabeçalho HTTP (ex: tipo de conteúdo enviado) |
| -d | Define os **dados** que serão enviados no corpo da requisição |

### 🧪 **Exemplo 1 – Enviar dados JSON para uma API**

curl -X POST https://api.exemplo.com/usuarios \

-H "Content-Type: application/json" \

-d '{"nome":"Ana", "email":"ana@email.com"}'

🔎 Esse comando envia um JSON com nome e e-mail para a rota /usuarios.

### 🧪 **Exemplo 2 – Enviar dados como formulário (formulário padrão)**

curl -X POST https://api.exemplo.com/login \

-d "usuario=admin&senha=1234"

🔎 Aqui os dados são enviados como application/x-www-form-urlencoded, formato padrão de formulários HTML.

### 🧪 **Exemplo 3 – Usando autenticação**

curl -X POST https://api.exemplo.com/dados \

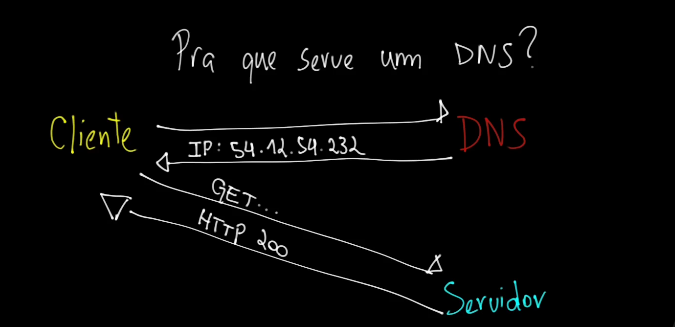
-u admin:1234 \

-H "Content-Type: application/json" \

-d '{"info":"teste"}'

**17. Para que serve o DNS?**

O **DNS (Sistema de Nomes de Domínio)** serve para **traduzir nomes de sites** (como google.com) em **endereços IP** (como 142.250.64.110) que os computadores usam para se comunicar.



### 🔧 **Principais funções do DNS**

### 🧭 **Resolução de nomes (Name Resolution)**

* + Converte nomes de sites como google.com em **endereços IP** como 142.250.64.110.
  + É a função mais conhecida do DNS.

### ✉️ **Roteamento de e-mails**

* + Usa registros do tipo **MX** (Mail Exchange) para indicar **para onde os e-mails de um domínio devem ser enviados**.
  + Ex: o DNS de meudominio.com pode apontar os e-mails para os servidores do Gmail.

### 🧪 **Verificação e autenticação de domínios**

* + Usado para provar que você é o dono de um domínio, como em **verificações de Google, Facebook, SSL, etc.**
  + Utiliza registros do tipo **TXT** com códigos de verificação.

### 🔐 **Segurança e autenticação de e-mails**

* + Ajuda a **evitar spam e falsificação de e-mail** (phishing) com registros como:
    - **SPF** (Sender Policy Framework)
    - **DKIM** (DomainKeys Identified Mail)
    - **DMARC** (Domain-based Message Authentication)

### 🪞 **Redirecionamento de nomes**

* + Com **CNAME** e **ALIAS**, o DNS pode redirecionar um subdomínio para outro nome.
  + Ex: www.seusite.com pode apontar para seusite.com ou para outro serviço externo.

### 🌎 **Balanceamento de carga e geolocalização**

* + Alguns provedores DNS conseguem **direcionar usuários para servidores diferentes** com base na localização geográfica, para melhorar velocidade e desempenho.
  + Usado em **CDNs (Content Delivery Networks)**.

### 🕵️ **Proteção e controle de acesso**

* + DNS pode ser usado para:
    - Bloquear sites perigosos
    - Filtrar conteúdos (controle parental)
    - Redirecionar domínios suspeitos
  + Exemplos: OpenDNS, Cloudflare Gateway.

**18. A estrutura do DNS**

O DNS tem uma **estrutura hierárquica**, com:

* **Root (.)** no topo
* **Domínios de topo** como .com, .org
* **Domínios** como google
* **Subdomínios** como www

| Nível | Exemplo | Descrição |
| --- | --- | --- |
| **Root (.)** | (raiz) | O topo da hierarquia. Representado por um ponto final invisível. |
| **TLD** | .com, .org | "Top-Level Domain" — domínios de topo, como .br, .gov, .net. |
| **Domínio** | google | Nome registrado dentro do TLD. |
| **Subdomínio** | www, mail | Partes adicionais usadas para separar serviços ou regiões. |

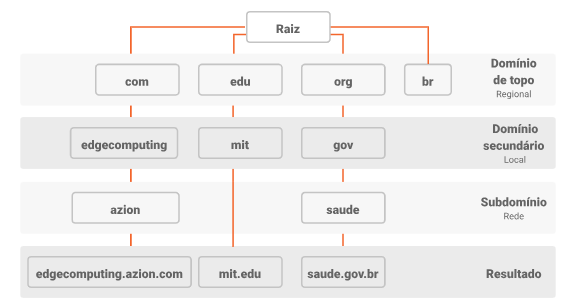
### 📍 **Exemplo prático:**

* Endereço: www.google.com
* Interpretação:
  + . → Raiz
  + com → TLD
  + google → Domínio
  + www → Subdomínio

### 🔄 Como essa estrutura funciona?

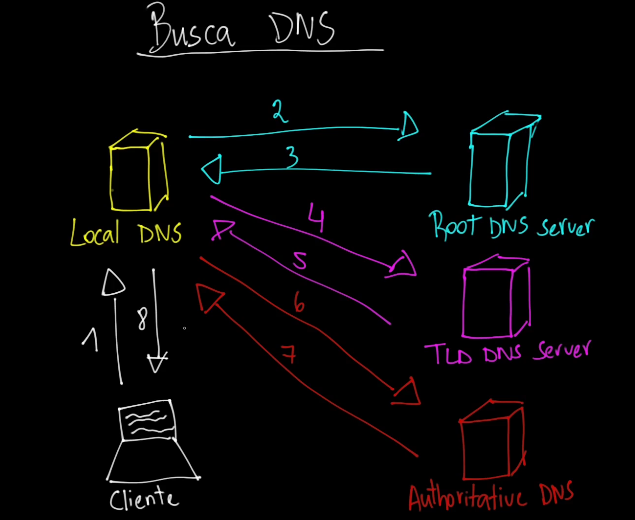
1. Você digita www.google.com.
2. O sistema começa a procurar:
   * Na **raiz (**.**)** → "Quem sabe sobre .com?"
   * No **TLD (**.com**)** → "Quem cuida do domínio google?"
   * No **domínio (**google.com**)** → "Quem responde por www?"

Cada nível aponta para o próximo, até chegar no servidor correto com o **IP final**.



**19. Como funciona uma busca DNS?**

Quando você digita um site, o navegador **pede ao DNS o IP correspondente**. Se ele não sabe, vai perguntando em ordem: primeiro ao servidor local, depois aos servidores DNS públicos, até encontrar e retornar o IP correto.



**20. DNS records**

Registros DNS indicam **como o domínio deve funcionar**. Exemplos:

* A: aponta para o IP do servidor
* MX: define onde chegam os e-mails
* CNAME: aponta para outro nome (alias)
* TXT: armazena textos, como verificações

