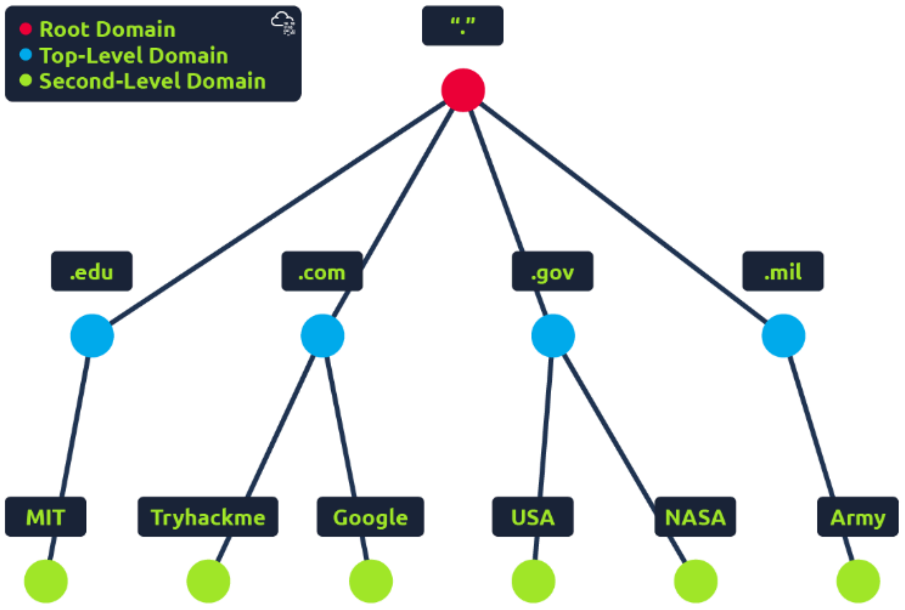
DNS

DNS (Domain Name System) provides a simple way for us to communicate with devices on the internet without remembering complex numbers. Much like every house has a unique address for sending mail directly to it, every computer on the internet has its own unique address to communicate with it called an IP address. An IP address looks like the following 104.26.10.229, 4 sets of digits ranging from 0 - 255 separated by a period. When you want to visit a website, it's not exactly convenient to remember this complicated set of numbers, and that's where DNS can help. So instead of remembering 104.26.10.229, you can remember tryhackme.com instead.

DNS Hierarchy



1. Root Domain (.)

- The root domain is the top of the DNS hierarchy and is represented by a dot (.).
- It doesn't appear in normal URLs, but every domain name technically ends with it (e.g., google.com.).

Example:
www.google.com. → the final dot is the root domain.

2. Top-Level Domain (TLD)

- The TLD is the part immediately to the left of the root domain.
- It tells the general purpose or country of the domain.

Types of TLDs:

1. gTLD (Generic TLD): .com, .edu, .gov, .mil, .org
2. ccTLD (Country Code TLD): .bd (Bangladesh), .uk, .in, .us

Example:
In tryhackme.com → .com is the TLD.

3. Second-Level Domain (SLD)

- The SLD comes immediately to the left of the TLD.
- This is usually the main name of the website or organization.

Example:
tryhackme.com → tryhackme is the Second-Level Domain
google.com → google is the Second-Level Domain

4. Subdomain

- A subdomain comes to the left of the Second-Level Domain.
- It is used to organize services or sections of a website.

Examples:
admin.tryhackme.com → admin is a subdomain
mail.google.com
jupiter.servers.tryhackme.com (multiple subdomains allowed)

DNS Record Types

DNS records tell the internet how to handle requests for a domain. Different record types serve different purposes.

1. A Record (Address Record)

- An A record maps a domain name to an IPv4 address.
- It tells browsers where a website is hosted.

Example: google.com → 104.26.10.229

2. AAAA Record

- An AAAA record maps a domain name to an IPv6 address.
- It is the IPv6 version of an A record.

Example: google.com → 2606:4700:20::681a:be5

3. CNAME Record (Canonical Name)

- A CNAME record points one domain to another domain name, not directly to an IP.
- DNS then resolves the target domain to an IP.

Example: store.tryhackme.com → shops.shopify.com
Then DNS looks up shops.shopify.com to get its IP.

4. MX Record (Mail Exchange)

- MX records specify the mail servers responsible for receiving emails for a domain.
- They include a priority value (lower number = higher priority).

Example: tryhackme.com → alt1.aspmx.l.google.com (priority 10)

5. TXT Record

- TXT records store text-based information for a domain.
- Commonly used for security and verification.

Common uses:
- SPF records (prevent email spoofing)
- Domain ownership verification
- DKIM / DMARC email security

Example: v=spf1 include:_spf.google.com ~all

What Happens When You Make a DNS Request

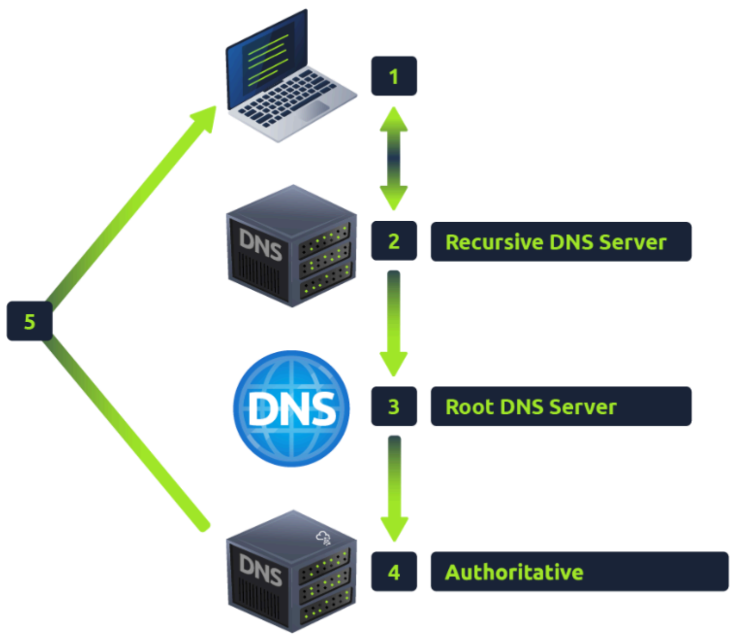
Step 1: Local Cache Check
- Your computer first checks its local DNS cache.
- If the IP is found, the process stops here.

Step 2: Recursive DNS Server
- If not found locally, the request goes to a Recursive DNS Server (usually your ISP or public DNS like Google).
- The recursive server also checks its own cache.
- If found, it returns the IP to your computer.

Step 3: Root DNS Server
- If the recursive server doesn't know the answer, it asks a Root DNS Server.
- The root server does not give the IP.
- It tells which TLD server to ask (e.g., .com).

Step 4: TLD (Top-Level Domain) Server
- The TLD server knows where the Authoritative DNS server is.
- It returns the address of the authoritative server.

Step 5: Authoritative DNS Server
- This server holds the actual DNS records (A, AAAA, MX, etc.).
- It sends the final IP address back to the recursive server.
- The result is cached using TTL and sent to your computer.



HTTP

HTTP (HyperText Transfer Protocol)

- HTTP is the protocol used to transfer web data between a browser and a web server.
- It is used to send and receive HTML pages, images, videos, CSS, JavaScript, etc.
- HTTP sends data in plain text, so anyone intercepting traffic can read it.
- Developed by Tim Berners-Lee (1989–1991).

  Port: 80
  Security: Not secure

HTTPS (HyperText Transfer Protocol Secure)

- HTTPS is the secure version of HTTP.
- It uses encryption (TLS/SSL) to protect data during transmission.
- Prevents eavesdropping, data tampering, and impersonation.
- Ensures you are talking to the real server, not a fake one.

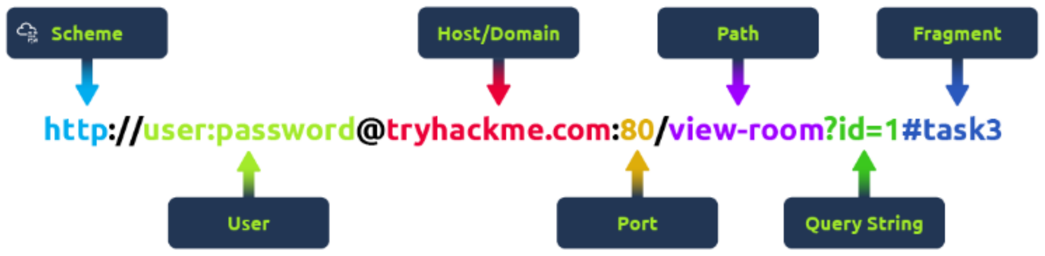
  Port: 443
  Security: Secure (encrypted)

URL (Uniform Resource Locator)

A URL tells the browser how and where to access a resource on the internet.

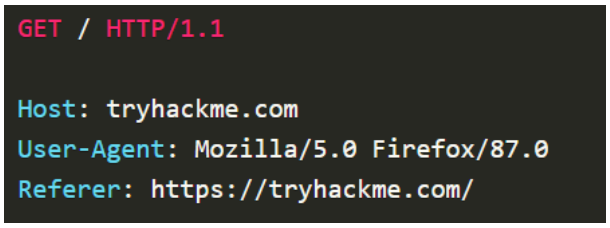
Main parts of a URL:

1. Scheme – Protocol used (http, https, ftp)
2. User (optional) – Username/password (rare, insecure)
3. Host/Domain – Server name or IP (e.g. tryhackme.com)
4. Port – Service port (HTTP → 80, HTTPS → 443)
5. Path – Resource location (/blog/page)
6. Query String – Extra data (?id=1)
7. Fragment – Page section (#top)



HTTP Request (Client → Server)

Example request:



```
GET / HTTP/1.1

Host: tryhackme.com
User-Agent: Mozilla/5.0 Firefox/87.0
Referer: https://tryhackme.com/
```

Line 1: This request is sending the GET method (more on this in the HTTP Methods task), requesting the home page with / and telling the web server we are using HTTP protocol version 1.1.

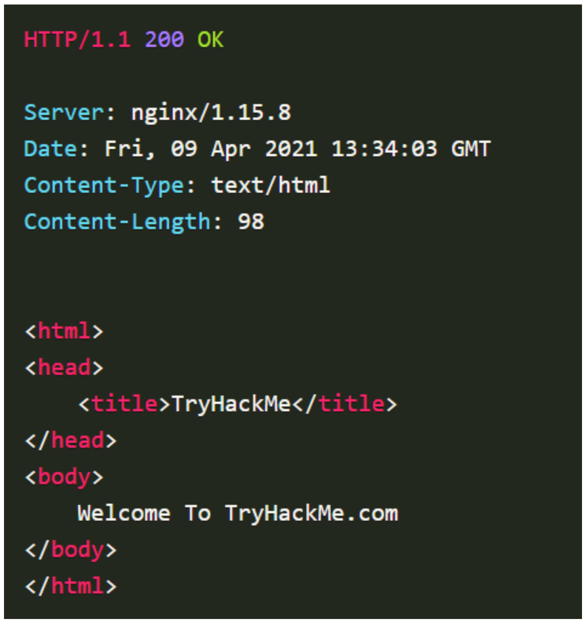
Line 2: We tell the web server we want the website tryhackme.com

Line 3: We tell the web server we are using the Firefox version 87 Browser

Line 4: We are telling the web server that the web page that referred us to this one is https://tryhackme.com

Line 5: HTTP requests always end with a blank line to inform the web server that the request has finished.

Example response:



```
HTTP/1.1 200 OK

Server: nginx/1.15.8
Date: Fri, 09 Apr 2021 13:34:03 GMT
Content-Type: text/html
Content-Length: 98


<html>
<head>
    <title>TryHackMe</title>
</head>
<body>
    Welcome To TryHackMe.com
</body>
</html>
```

Line 1: HTTP/1.1 → Protocol version
200 OK → Request successful

Line 2: Server → Server software and version

Line 3: Date → Server time

Line 4: Content-Type → Data type (HTML, image)

Line 5: Content-Length → Size of data

Line 6: Blank line → Headers end

Line 7-13: Body → Actual webpage/data

HTTP Methods

HTTP methods tell the server what action the client wants to perform.

GET
- Used to retrieve data from a server.
- Does not change data.
- Example: Loading a webpage, viewing a profile.

POST
- Used to send data to the server.
- Often creates new records.
- Example: Creating a user account, submitting a form.

PUT
- Used to update existing data.
- Replaces or modifies a record.
- Example: Updating user profile information.

DELETE
- Used to remove data from the server.
- Example: Deleting a user or a post.

HTTP Status Codes

HTTP status codes tell the client what happened to its request.

Status Code Categories

1xx – Information
- Request received, continue sending data
- Rarely used

2xx – Success
- Request was successful
- Server processed it correctly

3xx – Redirection
- Resource has moved
- Client must go to a different URL

4xx – Client Error
- Problem is on the client side
- Bad request, no permission, or page not found

5xx – Server Error

- Problem is on the server side
- Server failed to process a valid request

Common HTTP Status Codes (Must Remember)

200 – OK ● Request succeeded	401 – Unauthorized ● Login required ● Authentication missing or incorrect
201 – Created ● New resource created ● Example: New user account, blog post	403 – Forbidden ● Access denied ● Logged in or not, no permission
301 – Moved Permanently ● Page moved permanently ● Search engines update links	404 – Not Found ● Page or resource does not exist
302 – Found ● Page moved temporarily ● May change again	405 – Method Not Allowed ● HTTP method not supported ● Example: Sending GET instead of POST
400 – Bad Request ● Request is invalid or missing data	500 – Internal Server Error ● Server crashed or misconfigured
	503 – Service Unavailable ● Server overloaded or under maintenance

Common Request Headers (Client → Server)

Host

- Tells the server which website you want.
- Needed when one server hosts multiple sites.

User-Agent

- Identifies the browser and version.
- Helps the server send compatible content.

Content-Length

- Tells the server how much data is being sent.
- Prevents missing or incomplete data.

Accept-Encoding

- Tells the server which compression formats the browser supports.
- Example: gzip, deflate.

Cookie

- Sends stored user data back to the server.
- Used for sessions, login, preferences.

Common Response Headers (Server → Client)

Set-Cookie

- Stores data in the browser.
- Sent back on future requests.

Cache-Control

- Tells the browser how long to cache the response.
- Improves performance and reduces traffic.

Content-Type

- Defines what kind of data is being sent.
- Example: text/html, application/json, image/png.

Content-Encoding

- Tells how the data was compressed.
- Example: gzip.

HTTP Cookies

A cookie is a small piece of data stored in the browser by a website to remember the user.
Cookies exist because HTTP is stateless (the server doesn't remember you by default).

How Cookies Work (Step-by-Step)

1. Client requests a page

- Browser sends: GET / HTTP/1.1

2. Server responds

- Sends a page (login form, name form, etc.)

3. Client submits data

- Browser sends: POST / with form data

4. Server sets a cookie

- Response includes:
- Set-Cookie: name=adam

5. Browser stores the cookie

- Saved locally in the browser

6. Future requests

- Browser automatically sends:
- Cookie: name=adam

7. Server recognizes the user

- Shows personalized content (e.g., “Welcome back adam”)

Important Cookie Headers

Set-Cookie (Server → Client)

- Tells the browser to store data
  Example: Set-Cookie: sessionid=abc123

Cookie (Client → Server)

- Sends stored data back
  Example: Cookie: sessionid=abc123

How a Website Loads (End-to-End Flow)

1. User enters a URL

- Example: https://tryhackme.com

2. DNS Lookup

- Browser uses DNS to convert the domain name into an IP address

3. Connect to Web Server

- Browser connects to the server using the IP (via TCP, usually port 80/443)

4. HTTP Request

- Browser sends an HTTP request (GET/POST, headers, cookies)

5. Server Response

- Server replies with: HTML, CSS, JavaScript, Images, etc.

6. Browser Renders Page

- Browser processes the files and displays the website

Load Balancers

Purpose:

Distribute incoming traffic across multiple servers to improve performance, availability, and fault tolerance.

How it works:

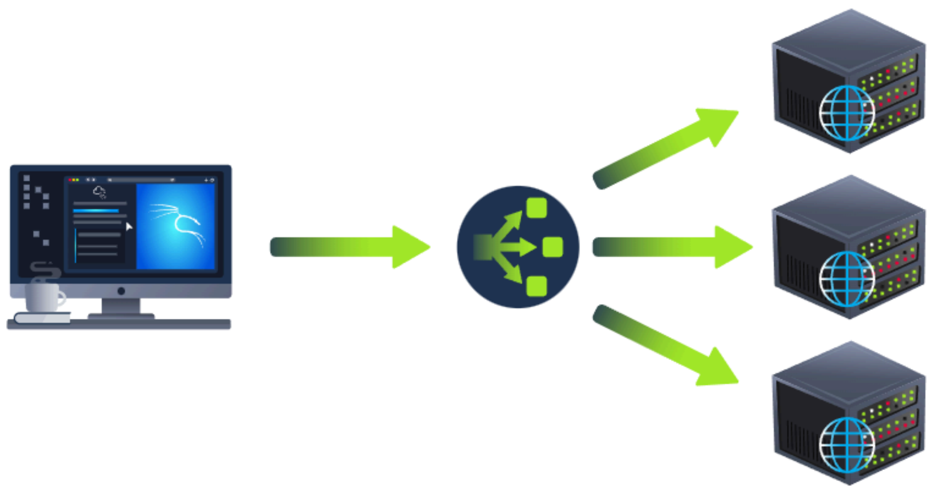
1. Client sends request to the Load Balancer
2. Load balancer forwards it to one of many backend servers
3. Response is sent back to the client

Common Algorithms:

- Round-Robin → Sends requests to servers one by one
- Weighted / Least-Load → Sends request to the least busy server

Health Checks:

- Load balancer regularly checks server health
- If a server fails, traffic is stopped to that server



CDN (Content Delivery Network)

Purpose:
Reduce latency and server load by serving static content from the nearest location.

What it delivers:CSS, JavaScript, Images, Videos

How it works:

1. User requests a file
2. CDN routes request to the closest server geographically
3. Faster load times + reduced main server traffic

Databases

Purpose:
Store and retrieve application data (users, posts, passwords, settings).

Used for:

- Login systems
- User profiles
- Posts, comments, records

Common Databases:

1. MySQL
2. PostgreSQL
3. MSSQL
4. MongoDB

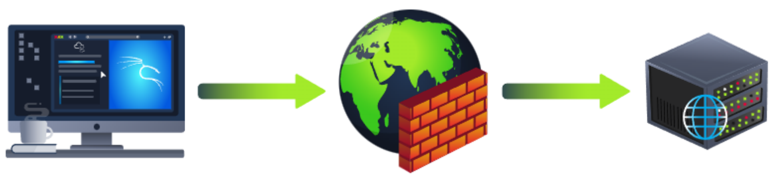
Databases can be single servers or clusters for speed and reliability.

WAF (Web Application Firewall)

Purpose:
Protect web applications from attacks before they reach the server.

Sits Between:

Client → WAF → Web Server



What it blocks:

- SQL Injection
- XSS
- Malicious bots
- Excessive requests (DoS)

Extra Protection:

- Rate limiting (e.g., X requests per IP per second)
- Drops suspicious requests instantly

Web Server

A web server is software that listens for incoming requests and uses HTTP/HTTPS to deliver web content to clients (browsers).

Common Web Servers:

- Apache
- Nginx
- IIS (Windows)
- Node.js

Root Directory (Document Root):

- Linux (Apache/Nginx): /var/www/html
- Windows (IIS): C:\inetpub\wwwroot

Example:

Request → http://www.example.com/picture.jpg
Server sends → /var/www/html/picture.jpg

Virtual Hosts (VHosts)

Host multiple websites on one web server using different domain names.

How it works:

1. Server checks the Host header in the HTTP request
2. Matches it with a virtual host configuration
3. Serves the correct website

Example Mapping:

one.com → /var/www/website_one
two.com → /var/www/website_two

Static vs Dynamic Content

Static Content

- Never changes
- Same for every user
- Sent directly from server to browser

Examples: Images, CSS, JavaScript, Static HTML

Dynamic Content

- Changes based on request/user/input
- Generated at runtime
- Generated by backend code

Examples:

- Blog homepage (latest posts)
- Search results
- User profile page

Frontend vs Backend

Frontend

- What the user sees
- HTML, CSS, JavaScript
- Runs in the browser

Backend

- Runs on the server
- Handles logic, data processing, database access
- Hidden from the user

Backend / Scripting Languages

Used to create dynamic & interactive websites

Common Languages:

- PHP
- Python
- Node.js
- Ruby
- Perl

What they can do:

- Process user input
- Talk to databases
- Call external APIs
- Generate HTML dynamically