Overview of HTTP

HTTP Where ? why?

HTTP found on web. **web application talk with HTTP language** to communicate.

WHY?
it's **reliable** data transmit protocol , that mean it make sure that your data not damaged through transmit, for that it's good for web servers & users because you can access data without worrying about its **integrity** .

Client & Server

- web content (Resources) **live in** web servers , and that servers talk **HTTP** so it's called **HTTP servers**
- it store the data and provide it when requested by **HTTP client** like **web browser** to appear to user.

Resources on Web

as we say how has web resources live in web servers .

LIKE WHAT?
like static file on the web server's file system .(that the simplest kind of web resources) and this file can contain any type of data (HTML file , pics, audio, video)

- resources also can be a software program which not a static file , its **dynamic** that show content on demand based on identity of who request. like **social media** or E-commerce

Media Type

each type can transmit through HTTP has a **label** describe its **data format** called **MIME**

a web server attach a MIME to all HTTP object will sent , and when **web browser receive it it check that label to see if it know how to handle that object format**

```
content-type:image/jpeg
```

```
MIME label in HTTP header
```

URIs (uniform resource identifier)

there's a tons of objects on the web, to make client specify what exactly he need to fetch in this wide ocean, each resource has a name can **identify it uniquely**

- URI split into (URLs and URNs)

URLs (uniform resource locator) (most common)

it describe the specific location of a resource on a particular server giving its location telling you how to fetch it right.

`http://www.joes-hardware.com/specials/saw-blade.gif`

- `http://` --> is the scheme that describe what protocol is used .
- `www.joes-hardware.com` --> second part is the server name (address)
- `/specials/saw-blade.gif` --> rest is specific resource location on `www.joes-hardware.com`

URNs (uniform resource name)

serve a unique name for a particular piece of content , independent of where its located right now but it didn't used yet still experimental

Transaction

client and server communicate through a chain of transactions **(requests & responses)**, a request from client to server and response from server to client.

this communication happens through formatted blocks called **HTTP messages**

Methods

every message sent had a **method** to tell server what action to perform .

GET -> send response from server to client (when client need to fetch something)
PUT -> store date from client to server
DELETE -> delete the named resource from a server
POST -> send data into server
HEAD -> send just the HTTP headers from response

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
```

```
Accept: text/html
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Date: Mon, 18 Feb 2025 12:34:56 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Type: text/html
Content-Length: 1234

<html>
<head><title>Example Page</title></head>
<body>
<h1>Welcome to Example.com</h1>
<p>This is a simple webpage.</p>
</body>
</html>
```

Status Code

each response message comeback with status code tell the client something.
200 -> OK. document return correctly
302 -> redirection, go some place else to get resources
404 -> not found.

multiple objects, multiple servers and multiple requests

some web pages that has a rich content depend on much resources to appear that look and that **didn't** fetched in one request , it **need many of them**

- the resources didn't have to be in the same server , it can be split into different servers to fetch from e.g. the header from server and pics from another.

Messages

HTTP requests and responses called **HTTP messages**

HTTP messages is :

- simple
- plain text
- line-oriented structure so it's easy to read

message split into **three parts** :

1. **start line** : first line of the message tell **what to do** (request as a command) and **what happen** (response like status)
2. **headers** : is a `key:value` pair
3. **body** : contain any kind of data

Connection (how messages move ??)

after see an overview about messages , no to see how it can move

- across **TCP-IP** stack
  HTTP itself in this stack reside on application layer which it more abstracting than any other , you didn't need to dig down to use it or to see network things , you can use it simply and **TCP** will take care of this

WHY TCP?

- it have error-free data transportation
- in-order delivery
  so you can send data without worry about integrity of data .

> in network terms when say HTTP is **layered over** TCP mean HTTP **uses** TCP to transport its message data.

Connection , IP Address and Port Number

before an HTTP can start send anything it need first to **establish** a TCP/IP connection between client and server with **IP and Port Number**

in TCP you need IP address of server computer and Port that associated with the specific software run on that server computer

`http://207.200.83.29:80/index.html

`http://www.netscape.com:80/index.html` -> URL here is the IP but using DNS to make it easy to read

`http://www.netscape.com/index.html` -> sometimes when the port number was popular with specific protocol like `80 HTTP` it can ignored and didn't write it

> with IP and Port number now its easy to establish a TCP/IP connection with another device to use HTTP

scenario steps

1. the browsers extract the server's hostname from the URL
2. the browser convert that hostname into server's IP address
3. the browser extract the port from the URL (if found)
4. then browser establish TCP connection with web server
5. the browser sends an HTTP request message to the server
6. the server sends an HTTP response back to the browser
7. the connection is closed , and the browser displays the document

another components of the web

Proxies -> intermediate device sit between client and server
caches -> HTTP storehouse that keep copies of popular web page close to client
gateways -> special web server that connect to other applications (a gateway between two different protocols like from HTTP to FTP)
tunnels -> special proxies that blindly forward HTTP communication
agents -> web clients that make HTTP requests like web browsers and crawlers bots