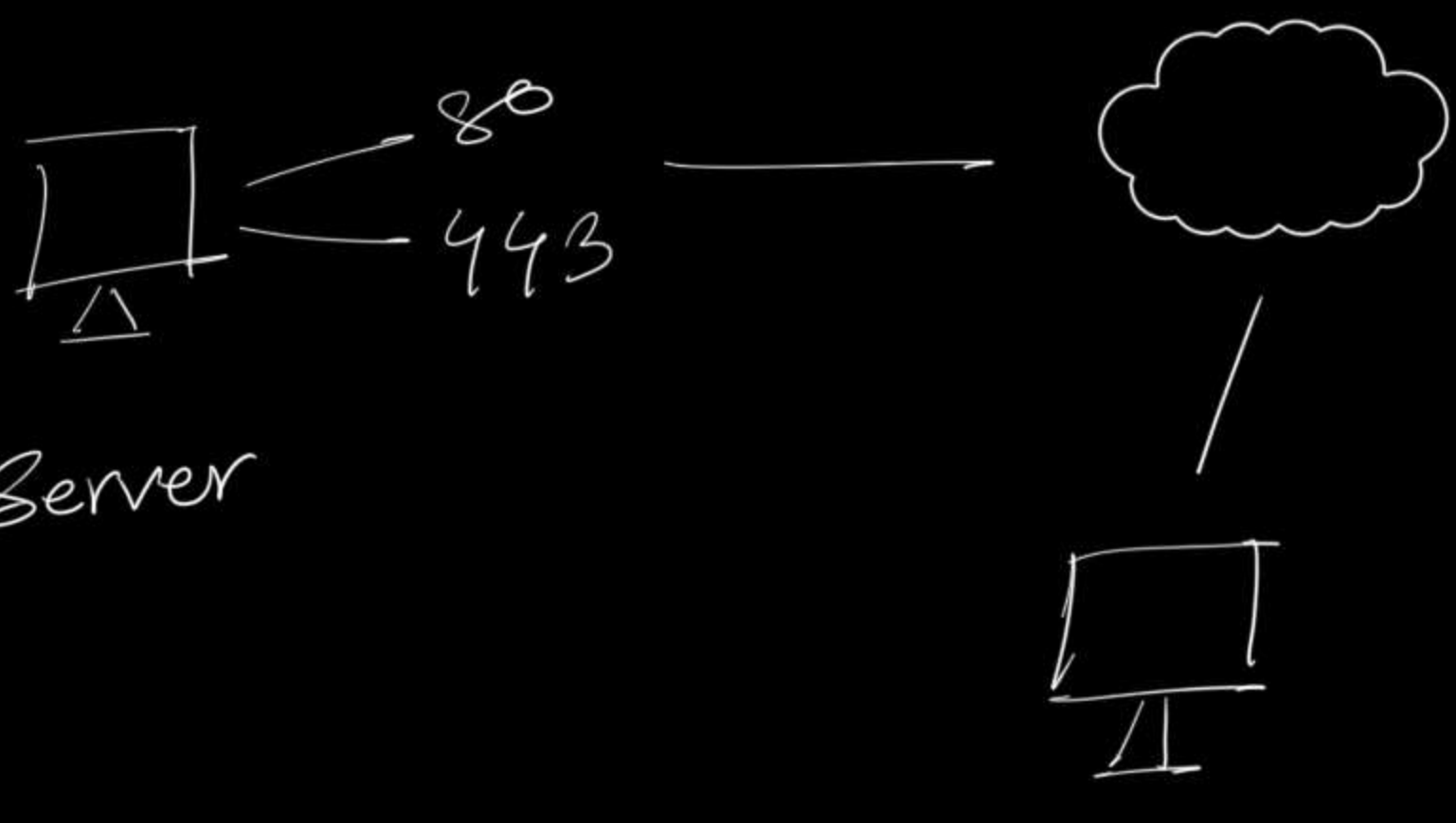
Computers listening for HTTP, Websocket, GRPC or any other kinds of request through a open port

↳ accessable over the internet

So, clients can connect & send data



HTTP

- Stateless → No memory of past interaction each request carries all the necessary information for the server to process
  - ↳ authentication token

Because http is stateless we need to use cookies, sessions or cookies to maintain the continuity of the interaction like login.

- Client-Server model → Client is typically web browser initiating connection by sending req to the server. Client is responsible to provide all the necessary information the server needs

On the other hand server host resources like html contents, api's etc & waits for incoming request from the client.

HTTPS works like the same with more security features like encryption, TLS etc.

The main thing to remember here is that client & servers established some kind of network connection & messages are sent & received

Messages

REQUEST

```
PUT /api/users/12345 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/93.0.4577.82
Safari/537.36
Content-Type: application/json
Content-Length: 123
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Accept: application/json
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: https://example.com/dashboard
Cookie: sessionId=abc123xyz456; lang=en-US

// a blank line here means, everything has been sent

{
  "firstName": "John",
  "lastName": "Doe",
  "email": "john.doe@example.com",
  "age": 30
}
```

RESPONSE

```
HTTP/1.1 200 OK
Date: Fri, 20 Sep 2024 12:00:00 GMT
Content-Type: application/json
Content-Length: 85
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: no-store
X-Request-ID: abcdef123456
Strict-Transport-Security: max-age=31536000;
includeSubDomains; preload
Set-Cookie: sessionId=abc123xyz456; Path=/; Secure;
HttpOnly
Vary: Accept-Encoding
Connection: keep-alive

// a blank line here means, everything has been sent

{
  "message": "User updated successfully",
  "userId": 12345,
  "status": "success"
}
```

PUT → request method

/api/users/12345 → resource URL, we are requesting from the server

HTTP/1.1 → http version

`Host: example.com` → host / our domain

`User to Cookie` → headers

blank line means headers are over

`? ?` → request body

`200 ok` → status code & message

Methods

Get Post Put Patch Delete Options

For CORS (Cross Origin Resource Sharing) there is two type of flow

- Simple request: Suppose ours frontend is at domain a & server is at domain b

The request sent by the browser/client contains host which is the servers domain & origin which is the frontend domains.

When the server sends the response it add the frontends domain in Access-Control-Allow-Origin. If the browser don't find this in the response header the browser won't allow.

- Preflight request: It is a request before the original request browser does to enquire some capabilites. The request qualifies as a preflight if:

  - The method is not Get, Post or Head. Has be Put or Delete etc.

  or

- The request include non-simple (general or request) headers. Has to be Authorization or Custom headers etc.

Or

- The request have a content-type other than application from urlencoded / multipart / text plain.

- The Preflight request is made with Option method.

Caching

Is a technique to store responses to reuse

The idea here is, the browser stores response after a get request along with last modified time & ETag. For the next get request the browser sends the time & ETag with request header & server checks if the resource is updated science or time is expired.

If not, the server sends response 304 & the browser loads from its cache.

Content Negotiation

A mechanism, using which client & server aggry on the best format to exchange data.

The idea is that the client can let the server know its preference & the server response accordingly (data format, language)

HTTP based Compression

We use it to reduce the content size.

Multipart Request

Usually use for sending large files from client to the server.

The binary data of the file is transfered part by part.

Streaming Response

Use for to receive large files from server in chunks.