HTTP

Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems.

From RFC 7231

- RFC 7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content
  - Defines the message semantics, independently of how messages are communicated.

- RFC 7230 - Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing
  - Defines a way to represent messages as bytes and communicate those messages.

- RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2)
  - Defines an alternative way to represent messages and communicate them over the network.

- HTTP is a generic interface protocol for information systems. It is designed to hide the details of how a service is implemented by presenting **a uniform interface to clients that is independent of the types of resources provided.**

- HTTP is a stateless request/response protocol that operates by exchanging messages across a reliable transport- or session-layer "connection"
    - An HTTP "client" is a program that establishes a connection to a server for the purpose of sending one or more HTTP requests.
    - An HTTP "server" is a program that accepts connections in order to service HTTP requests by sending HTTP responses.

From RFC 7230

- The term "user agent" refers to any of the various client programs that initiate a request, including (but not limited to) browsers, spiders (web-based robots), command-line tools, custom applications, and mobile apps.

- The term "origin server" refers to the program that can originate authoritative responses for a given target resource

- The terms "sender" and "recipient" refer to any implementation that sends or receives a given message, respectively.

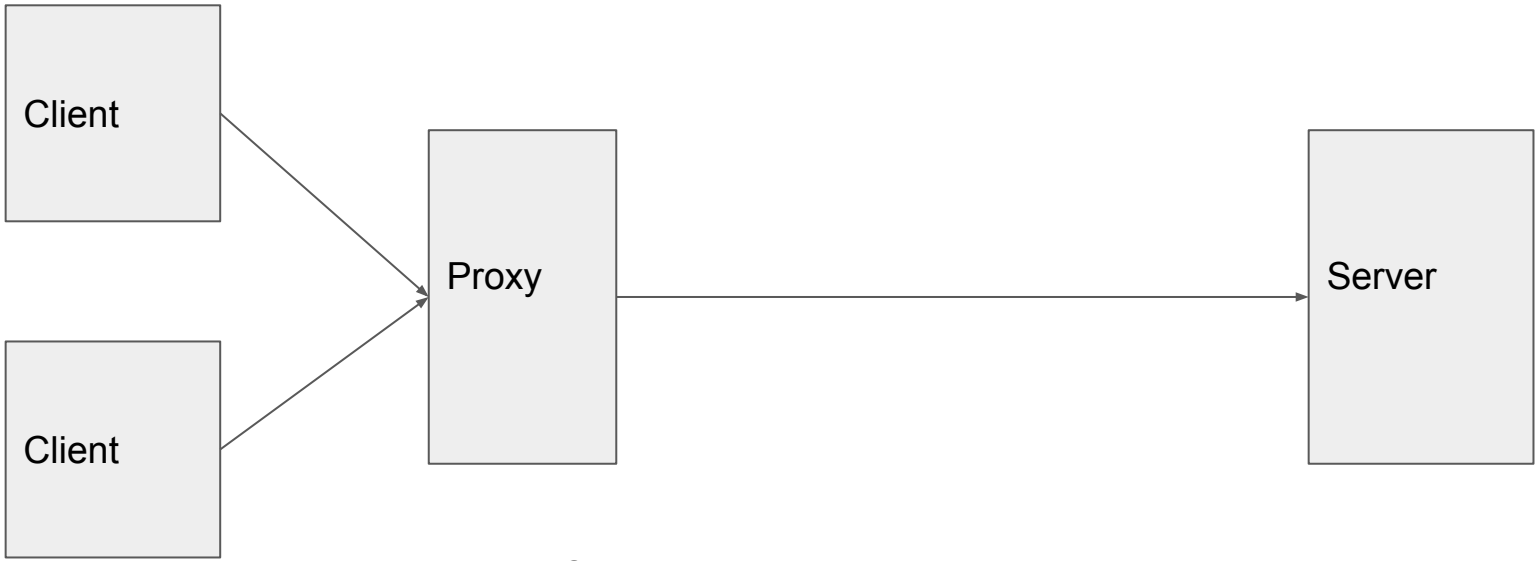
From RFC 7230

Intermediates

- Proxy
- Gateway (Reverse Proxy)
- Tunnel

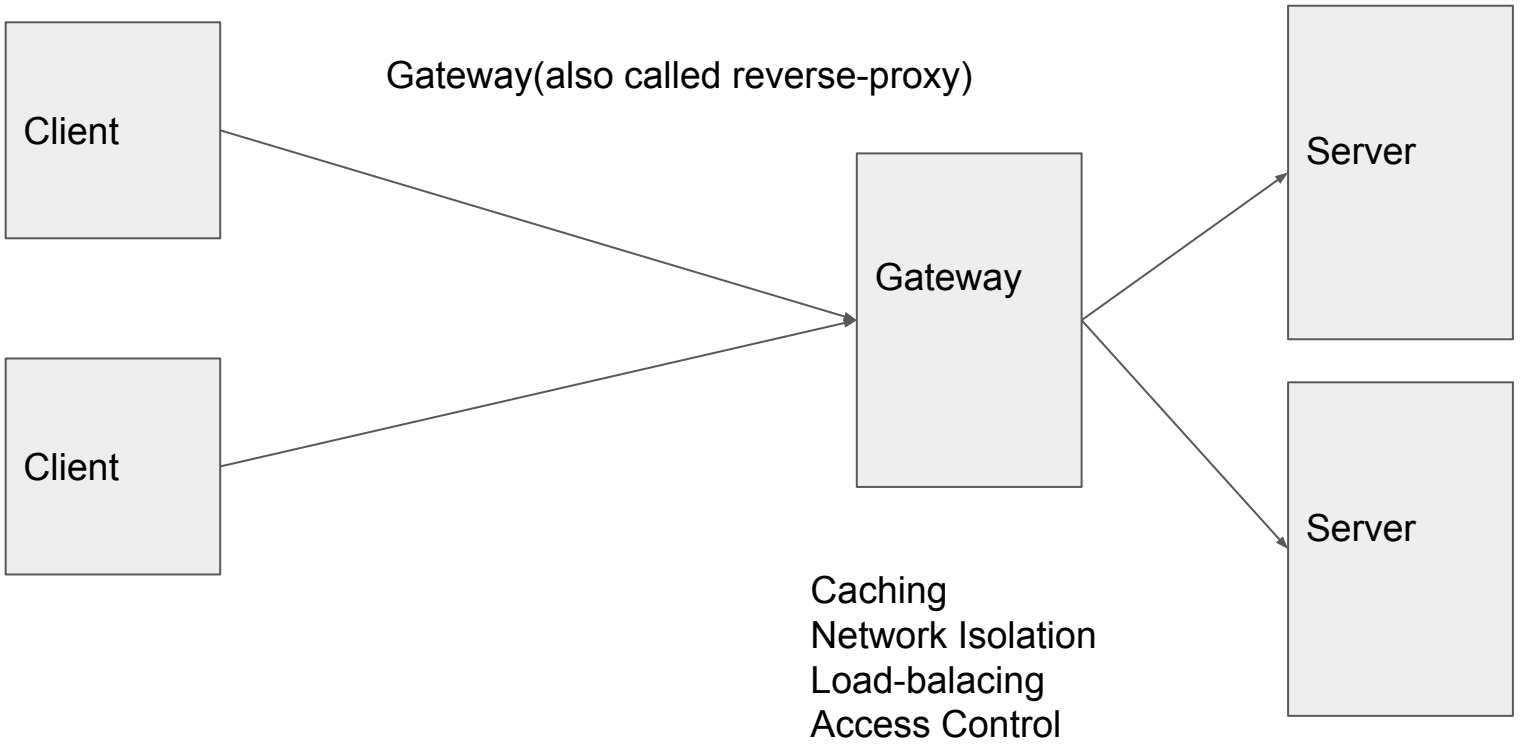
From RFC 7230

Intermediaries - Proxy



Caching
Network Isolation

Intermediaries - Gateway (also called reverse-proxy)



Uniform Interface

HTTP provides a uniform interface for interacting with a resource, regardless of its type, nature, or implementation, via the manipulation and transfer of representations .

From Fielding Dissertation

Message Structure

Request message:

- Method - defines the operation being requested by the message.
- Request URI - defines the resource where the operation should be performed.
- Request headers - provide extra information about the request.
- Request payload (headers + body)

Response message:

- status code - comunicates the operation outcome.
- response headers - provide extra information about the response.
- response payload (headers + body)

Methods

Main Methods defined on RFC 7231

- `GET`
- `POST`
- `PUT`
- `DELETE`
- `HEAD`
- `CONNECT`
- `OPTIONS`
- `TRACE`

However, more HTTP methods can be define in additional specifications

- E.g. RFC 5789 - PATCH Method for HTTP

Methods Properties

Safe methods

Request methods are considered "safe" if their defined semantics are essentially read-only; i.e., the client does not request, and does not expect, any state change on the origin server as a result of applying a safe method to a target resource. Likewise, reasonable use of a safe method is not expected to cause any harm, loss of property, or unusual burden on the origin server.

The purpose of distinguishing between safe and unsafe methods is to allow automated retrieval processes (spiders) and cache performance optimization (pre-fetching) to work without fear of causing harm.

From RFC 7231

Method Properties

Idempotent methods

A request method is considered "idempotent" if the intended effect on the server of multiple identical requests with that method is the same as the effect for a single such request. Of the request methods defined by this specification, PUT, DELETE, and safe request methods are idempotent.

Idempotent methods are distinguished because the request can be repeated automatically if a communication failure occurs before the client is able to read the server's response. For example, if a client sends a PUT request and the underlying connection is closed before any response is received, then the client can establish a new connection and retry the idempotent request. It knows that repeating the request will have the same intended effect, even if the original request succeeded, though the response might differ.

From RFC 7231

Status Codes

The first digit of the status-code defines the class of response. The last two digits do not have any categorization role. There are five values for the first digit:

- 1xx (Informational): The request was received, continuing process
- 2xx (Successful): The request was successfully received, understood, and accepted
- 3xx (Redirection): Further action needs to be taken in order to complete the request
- 4xx (Client Error): The request contains bad syntax or cannot be fulfilled
- 5xx (Server Error): The server failed to fulfill an apparently valid request

From RFC 7231