Web Server Languages

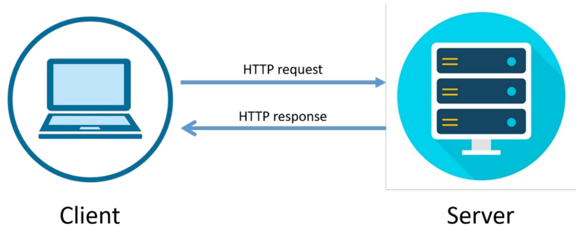
Simeon Monov, Hristo Karaperev

HTTP (HyperText Transfer Protocol)

An application-layer protocol for transmitting hypermedia documents, such as HTML

HTTP

- Designed for communication between web browsers and web servers, but it can also be used for other purposes
- HTTP follows a classical client-server model, which a client opening a connection to make a request, then waiting until it receives a response
- Foundation of any data exchange on the Web
- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data)



HTTP (2)

- The message sent by the client, usually a Web browser, are called requests and the messages sent by the server as an answer are called responses
- Designed in the early 1990s, it has evolved over time
- HTTP is a protocol that is sent over TCP (Transmission Control Protocol)

Components of HTTP systems

HTTP is a client-server protocol:

- requests are sent by one entity, the user-agent (or a proxy on behalf of it)
- each individual request is sent to a server, which handles it an provides and answer called the response
- between the client and the server there are numerous entities, collectively called proxies, which perform different operations and act as gateways or caches, for example

Components of HTTP systems - the Client(user-agent)

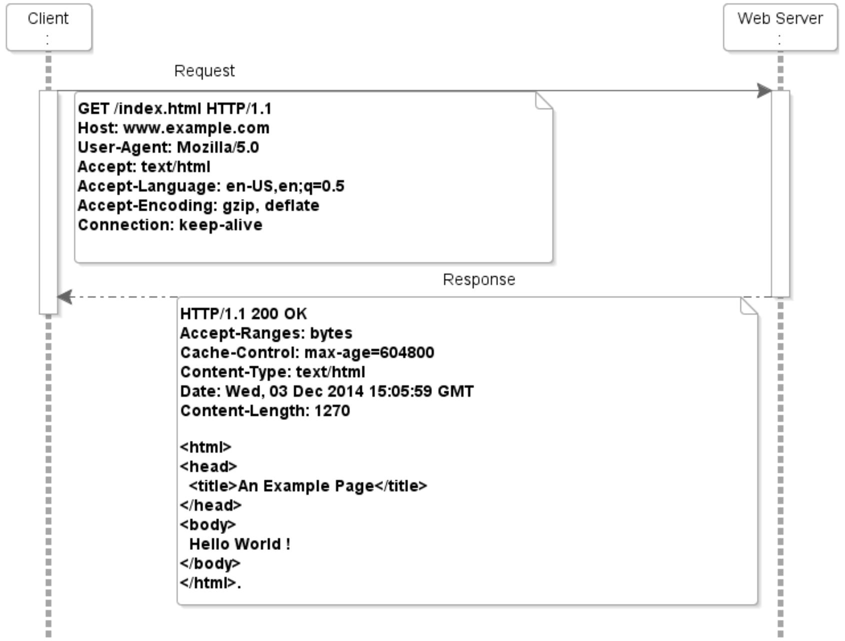
- User-agent is any tool that acts on behalf of the user
- User-agent is always the entity initiating the request, it is never the server (although some mechanisms have been added over the years to simulate server-initiated messages)

Components of HTTP systems - the Web server

- On the opposite side of the communication channel is the server
- A server appears as a single machine virtually, but it may actually be a collection of servers sharing the load or other software such as caches, databases, totally or partially generating the document on demand
- A server is not necessarily a single machine, several server software instances can be hosted on the same machine. With HTTP/1.1 and the Host HTTP header, they may share the same IP address

Aspects of HTTP - it is simple

- It is designed to be simple and human-readable
- HTTP messages can be read and understood by humans, providing easier resting and reduced complexity for newcomers



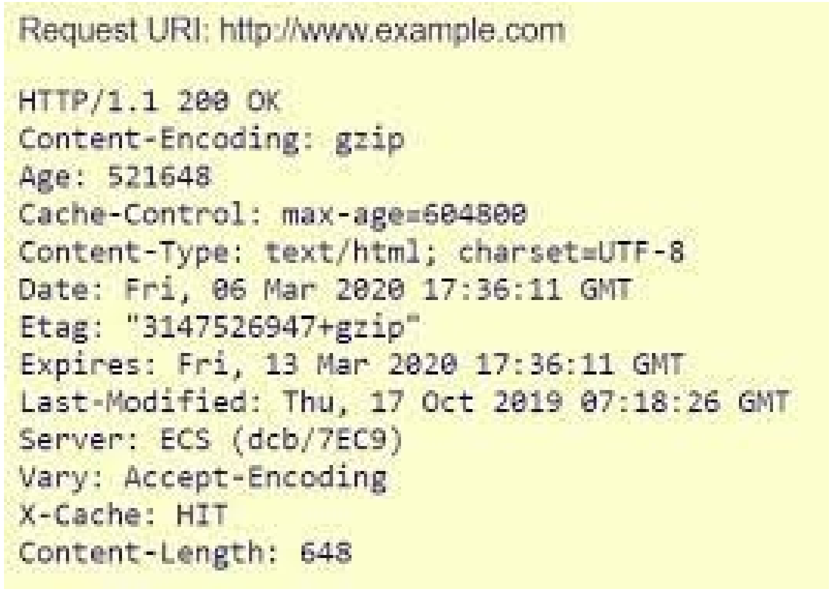
```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: max-age=604800
Content-Type: text/html
Date: Wed, 03 Dec 2014 15:05:59 GMT
Content-Length: 1270

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World !
</body>
</html>.
```

Aspects of HTTP - it is extensible

- Introduced in HTTP/1.0, HTTP headers make the protocol easy to extend and experiment with
- New functionality can be introduced by a simple agreement between a client and a server about new header semantics

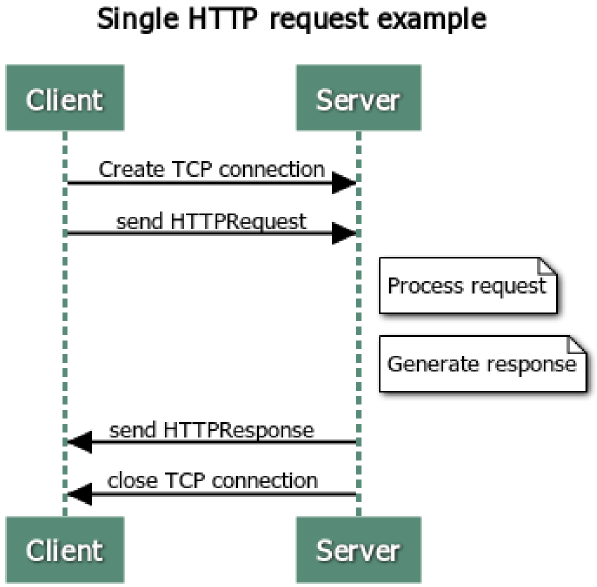


Request URI: http://www.example.com

```
HTTP/1.1 200 OK
Content-Encoding: gzip
Age: 521648
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Fri, 06 Mar 2020 17:36:11 GMT
Etag: "3147526947+gzip"
Expires: Fri, 13 Mar 2020 17:36:11 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (dcb/7EC9)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 648
```

HTTP flow

- When a client want to communicate with a server, either the final server or an intermediate proxy it performs the following steps:
  - Open a TCP connection
  - Send an HTTP message
  - Read the response sent by the server
  - Close or reuse the connection for further requests



HTTP messages - requests

An HTTP request messages consists of:

- Request line
- Zero or more request header fields (at least 1 header in case of HTTP/1.1)
- Empty line
- An optional request body

HTTP messages – request line

A request line consists of:

- Case-sensitive request method (e.g. GET, POST, DELETE, ...)
- A space
- The requested URL, used to find the resource on the server
- Another space
- The protocol version (e.g. HTTP/1.0 or HTTP/1.1)

```
GET /index.html HTTP/1.1
```

```
GET /images/logo.jpg HTTP/1.1
```

HTTP messages – request header fields

Each request header field consists of:

- Name, a colon, optional whitespace and the field value
- In HTTP/1.1 **Host** field is required

```
Host: www.example.com
Accepted-Language: en
```

HTTP messages – request body

Request body is optional and used to send additional data to the server (e.g. when doing a POST request).

```
field1=value1&field2=value2
```

HTTP messages – request methods

HTTP defines methods to indicate the desired action to be performed on the identified resource:

Method	Description
GET	Retrieve / load a resource
POST	Create / store a resource
PUT	Update a resource
DELETE	Delete (remove) a resource
PATCH	Update resource partially
HEAD	Retrieve the resource’s headers
OPTIONS	Return the HTTP methods that the server supports for the specified URL

HTTP messages - responses

An HTTP response messages consists of:

- Status line
- Zero or more response header fields
- An empty line
- An optional message body

HTTP messages – response status line

HTTP response status line consists of:

- HTTP protocol version
- A space
- Response status code
- Another space
- Reason phrase (can be empty)

```
HTTP/1.1 200 OK
```

HTTP messages – response header fields

Each response header field consists of:

- Name, a colon, optional whitespace and the field value

```
Content-type: text/html
```

HTTP messages – response status code

All HTTP response status codes are separated into five classes or categories:

- **1xx** informational response – 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 fulfil an apparently valid request

HTTP messages – response status code (2)

Example response status codes:

- 200 OK
- 200 Created
- 301 Moved Permanently
- 400 Bad Request
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 500 Internal Server Error
- 502 Bad Gateway

Web Servers

The term can refer to hardware or software, or both of them working together

1. Hardware web server - computer that stores web server software and website components and is connecting to the Internet
2. Software web server - it consists of several parts that control how users access hosted files. At a minimum the web server is an HTTP server (software that understands URLs and HTTP). It can be accessed through the domain names of the websites it stores and deliver the content of these websites to the user

Static and dynamic web server

1. Static web server consists of a computer with an HTTP server.
2. Dynamic web server consists of a static web server and extra software like an application server and a database.

Application server

- Crucial software framework or platform responsible for managing the execution of web applications
- It acts as an intermediary layer between the client and the backend systems, such as databases or external services
- The primary purpose of an application server is to handle incoming client requests, execute the necessary business logic and generate responses that are sent back to the client

Key functions and features

- **Request handling** - at the heart of the functionality of an application server is its ability to handle incoming requests and redirect them to respective components
- **Business logic execution** - application servers are where the core business logic of a application resides. This includes code for data processing, calculations and other tasks. Separating this logic from the presentation layer (the UI) is a fundamental principle of modern software architecture, often referred to as the MVC (Model-View-Controller) pattern

Key functions and features (2)

- **Database access** - for applications that rely on databases, application servers play a critical role in managing database interactions.
- **Session management** - some applications require the ability to maintain state information for individual clients as they interact with the application.
- **Security** - application servers often include built-in security features such as authentication and authorization mechanisms.

Key functions and features (3)

- **Load balancing** - application servers can be configured with load balancing when an application experiences high traffic for example.
- **Integration** - application servers facilitate integration with other systems and services (messaging queues, external APIs, third-party libraries).
- **Caching** - to improve performance and reduce load on databases, some application servers offer caching mechanisms to store frequently accessed data in memory for quick retrieval

Differences between Web and Application servers

Functionality:

- Web server - handles HTTP messages
- Application server - designed to execute dynamic code and applications

Content handling:

- Web server - ideal for serving static content efficiently and quickly
- Application server - ideal for running server-side code and generating dynamic content

Differences between Web and Application servers (2)

Scripting and programming:

- Web server - while they support server-side scripting languages like PHP, their capabilities are limited compared to dedicated application servers
- Application server - they support various programming languages like Java, Python, .NET, Ruby etc.

Differences between Web and Application servers (3)

Database connectivity:

- Web server - can connect to database, but their interaction is limited to basic database queries and operations
- Application server - they excel at database connectivity. They often provide support for connection pooling, transaction management and sophisticated database operations

Popular Application servers

- Apache Tomcat
- JBoss Enterprise Application Platform
- Node.js
- Jetty
- IIS
- uWSGI

Popular web application frameworks

- Spring Framework
- Django
- Express.js
- ASP.NET
- Ruby on Rails

MVC (Model-View-Controller)

- Pattern in software design
- Emphasizes separation between the software’s business logic and display
- There are three parts to the MVC design pattern
    - Model - manages data and business logic, defines data the app should contain
    - View - handles display, defines how the data should be displayed
    - Controller - routes commands to the model and view parts, contains logic that updates the model or the view (or both) in response to user input

REST

- Acronym to Representational State Transfer
- Architectural style (not a protocol or a standard) for distributed systems first presented in 2000
- Design principles:
  - Uniform interface
  - Client-server decoupling
  - Statelessness
  - Cacheability
  - Layered system architecture
  - Code on demand (optional)

REST - Uniform Interface

- All requests for the same resource, no matter where the request comes from, should look the same
- The API should make sure that the same piece of data, belongs only to one uniform resource identifier
- Resources should contain just as much information as the client needs

REST - Client-server decoupling

- Client and server applications must be independent of each other
- The client should only know information about the URI of the requested resource
- It should not be able to interact with the server application in other ways
- The server application should not modify the client application, but only send the requested data via HTTP

REST - Statelessness

- Each request needs to include all the information necessary for processing it
- Server applications should not store any data related to a client request

REST - Cacheability

- Resources should be cacheable on the client or on the server side
- Server responses need to contain information about whether caching is allowed
- Goal is to improve performance on the client side

REST - Layered system architecture

- APIs need to be designed so that neither the client nor the server can tell if they communicate with the end application or an intermediary because the messages between the client and the server go through different layers

REST - Code on demand

- REST APIs send static resources most of the time
- In some cases responses can contain executable code, and the code should only run on-demand

JSON (JavaScript Object Notation)

- standard text-based format for representing data based on the JavaScript object syntax
- commonly used for transmitting data in web applications
- it can be used independently of JavaScript
- JSON exists as a string, it needs to be converted by web applications if you want to access the data
- you can include the same basic data types inside JSON as you can in standard JavaScript object - strings, numbers, booleans, arrays, etc.