Web Server Languages

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
- HTTP is a stateless protocol, meaning that the server does not keep any data (state) between two requests
- 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

There are more computers between a browser and the server handling the requests (routers, modems, and more) but thanks to the layered design of the Web, these are hidden in the network and transport layers. HTTP is on top, at the application layer

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)

To display a Web page, the browser sends an original request to fetch the HTML document that represents the page. It then parses the file, making additional requests corresponding to execution scripts, layout information (CSS) to display, and sub-resources contained within the page (images or videos). The browser combines these resources to present the Web page. Scripts executed by the browser can fetch more resources in later phases and the browser updates the page accordingly

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

Components of HTTP systems - proxies

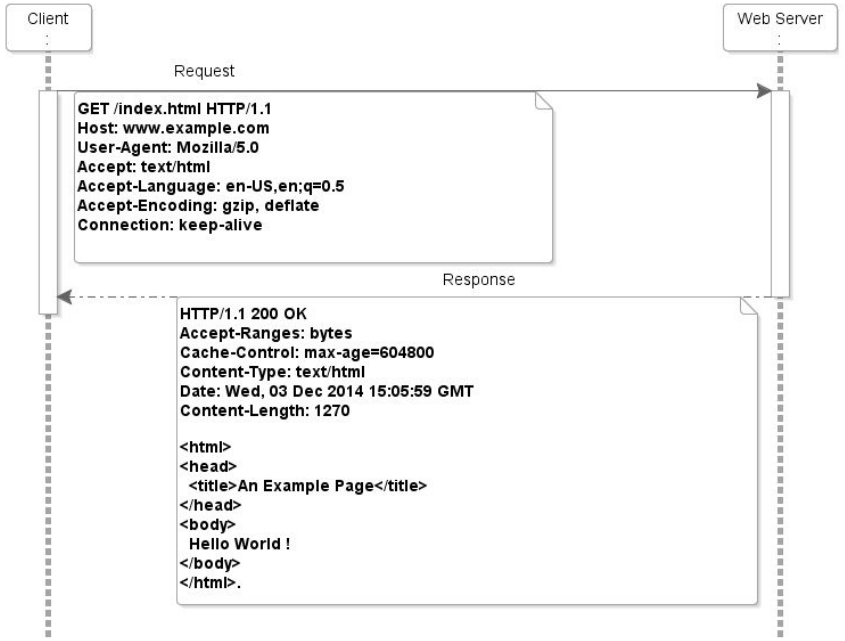
- Between the client and the server, numerous computers and machines relay the HTTP messages
- Due to the layered structure of the Web, most of these computers operate at the transport, network or physical levels, becoming transparent at the HTTP layer
- Those operating at the application layer are generally called proxies
- Proxies can be transparent (only forwarding the request without altering it) or non-transparent (change the request in some way before forwarding it to the server)

Components of HTTP systems - proxies (2)

- Proxies can have functions line:
  - caching
  - filtering (like an antivirus scan)
  - load balancing (allowing multiple servers to serve requests)
  - authentication (to control access to resources)
  - logging

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



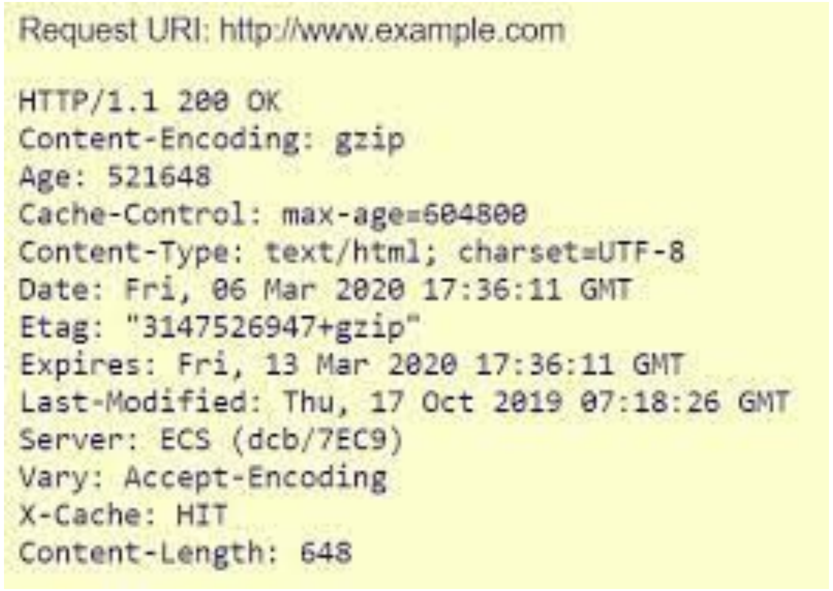
```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

```
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

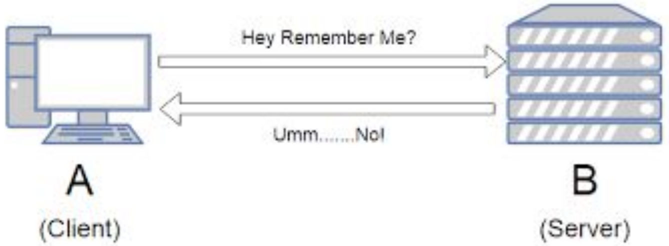


Request URI: http://www.example.com

```
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
```

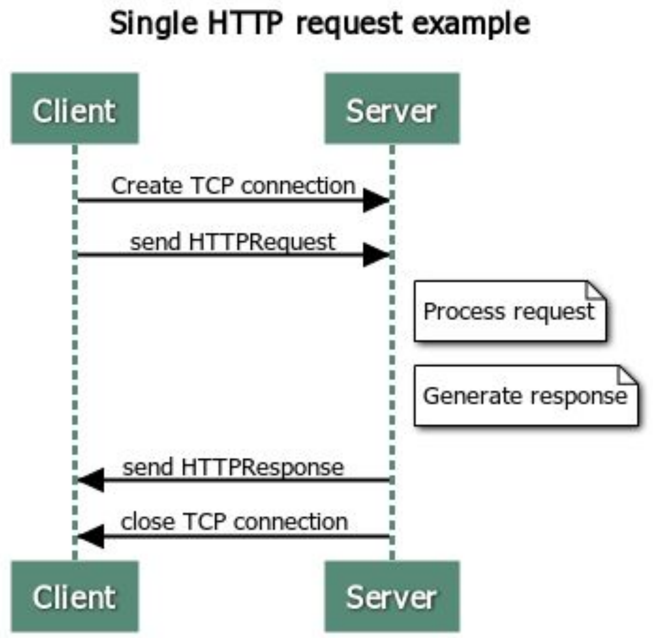
Aspects of HTTP - it is stateless

- There is no link between two requests being successively sent out on the same connection
- This has the prospect of being problematic if users are attempting to interact with pages coherently, for example, using e-commerce shopping baskets
- While the core of HTTP is stateless, HTTP cookies allow the user of stateful sessions



Aspects of HTTP - connections

- Connections are controlled at the transport layer, and fundamentally out of scope for HTTP
- HTTP requires a reliable transport protocol so messages are not lost
- Before client and server exchange HTTP messages they must establish a TCP connection



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

If HTTP pipelining is activated, several request can be sent without waiting for the first response to be fully received.

HTTP messages - requests

An HTTP request messages consists of:

- an HTTP method (usually a verb like GET, POST that defines the operation the client wants to perform)
- the path of the resource to fetch (the URL of the resource stripped from elements the are obvious from the context, for example without the protocol - http://)
- the version of the HTTP protocol
- Optional headers to convey additional information to the server
- Optional body, for some methods like POST, which contains the resource sent

URL (Uniform Resource Locator)

- The most common form of resource identifier
- Describes the specific location of a resource on a particular server
- Components of URLs:
  - Scheme - first part of the URL, which indicates the protocol that the browser must use to request the resource
  - Authority - second part of the URL, which is separated from the scheme by “://”. It includes the domain (the Web server) and the port (the technical “gate” used to access the Web server, that can be omitted if the Web server uses the standard ports for HTTP)
  - Path to resource - path to the resource on the Web server
  - Parameters - extra parameters the user can use to send information to the Web server
  - Anchor - represents a “bookmark” inside the resource, giving the browser directions to show that part of the resource

HTTP messages - responses

An HTTP response messages consists of:

- the version of the HTTP protocol they follow
- a status code, indicating if the request was successful or not
- a status message, a short description of the status code
- HTTP headers, similar to requests
- Optionally, a body containing the fetched resource

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

Web Servers (2)

At the most basic level:

- if a browser needs a file that is hosted on a web server, it requests the file via HTTP
- when the request reaches the correct (hardware) web server, the (software) HTTP server accepts the request, finds the requested document and sends it back to the browser also via HTTP

Static and dynamic web server

1. Static web server consists of a computer with an HTTP server. It is called static because the server sends its hosted files as-is to the user browser
2. Dynamic web server consists of a static web server and extra software like an application server and a database. It is called dynamic because the application server updates the hosted files before sending content to the user browser via the HTTP server

Hosting files

A web server can store the website’s files, namely the HTML documents and their related assets, including images, CSS stylesheets, JavaScript files, fonts and video

One could host these files on their own PC but it is more convenient to use a dedicated web server because:

- the server is more available (up and running)
- the server is always connected to the Internet (excluding downtimes and system troubles)
- the server can have the same IP address all the time
- the server is typically maintained by a third party

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. In the context of web applications, these requests are often in the form of HTTP requests. The server processes these requests and directs them to the appropriate components or services within the application
- 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. They establish connections with databases, retrieve data, update records and handle transactions
- Session management - some applications require the ability to maintain state information for individual clients as they interact with the application. Application servers typically offer session management capabilities
- Security - application servers often include built-in security features such as authentication and authorization mechanisms. Additionally they may support encryption to protect data transmitted between the client and server

Key functions and features (3)

- Load balancing - application servers can be configured with load balancing when an application experiences high traffic for example. This distributes incoming requests evenly across multiple server instances, improving performance
- Integration - application servers facilitate integration with other systems and services (messaging queues, external APIs, third-party libraries). This is crucial for creating robust applications
- 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

Key functions and features (4)

- Scalability - application servers are designed with scalability in mind, allowing companies to add more server instances in response to increased traffic or workload
- Monitoring and management - application servers often come equipped with tools for monitoring performance. These tools provide information about the application server health and help in identifying and addressing problems

Differences between Web and Application servers

Functionality:

- Web server - handles HTTP messages, focuses on serving static content
- 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, .NET, Python, 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
- WildFly
- Node.js
- IIS
- Jetty

Popular web application frameworks

- Spring Framework
- Django
- Express.js
- ASP.NET
- Laravel
- 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.

Spring Framework

- Java platform that provides infrastructure support for developing Java applications
- Open source first released in 2003
- Provides extensions for building Enterprise Java applications
- Enables developers to develop enterprise-class applications using POJOs (Plain Old Java Objects)
- Organized in modular fashion, meaning you can use just the modules that you need
- Spring’s web framework is a well-designed MVC framework
- Inversion of Control
- Aspect Oriented Programming

Spring Framework - Inversion of Control

- Application classes should be as independent as possible to increase the possibility to reuse the classes
- Inversion of Control concept and Spring’s implementation Dependency Injection helps gluing the application together while keeping the classes independent
- Dependency Injection - all it means is that the class B which is dependent on class A will get the class A injected into it by Spring

Spring Framework - Aspect Oriented Programming

- Functions that span multiple points of an application are called cross-cutting concerns (logging, caching, security, etc.)
- Dependency Injection helps you decouple the application’s objects from each other, while Aspect Oriented Programming helps you decouple cross-cutting concerns from objects they affect
- Provides us to enable functionality adding annotations to classes, fields or functions

Spring Boot

- Tool that helps with developing web applications with the Spring Framework by providing:
    - Autoconfiguration - applications are initialized with pre-set dependencies that you don’t have to configure manually (it automatically configures the Spring Framework and the third-party packages based on your settings and best practices)
    - Opinionated approach to configuration - following its own judgement, Spring Boot chooses which packages to install and which default values to use, rather than requiring the user to set up everything manually. For example the Spring Web starter dependency allows the developer to build web applications with minimal configuration just by adding the dependency in the project
    - The ability to create standalone applications - it allows you to create standalone applications that run on their own without relying on an external web server. You can just launch your application on any platform by running the produced .jar file