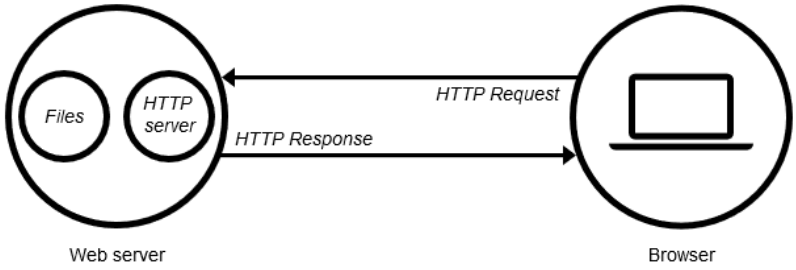
Web Introduction

Sunday, May 24, 2020 9:31 AM

"Web server" can refer to hardware or software, or both of them working together.

1. On the hardware side, a web server is a computer that stores web server software and a website's component files (e.g. HTML d ocuments, images, CSS stylesheets, and JavaScript files). It is connected to the Internet and supports physical data interchange with other devices connected to the web.
2. On the software side, a web server includes several parts that control how web users access hosted files, at minimum an *HTTP server*. An HTTP server is a piece of software that understands URLs (web addresses) and HTTP (the protocol your browser uses to view webpages). It can be accessed through the domain names (like mozilla.org) of websites it stores, and delivers their content to the end-user's device.

At the most basic level, whenever a browser needs a file which is hosted on a web server, the browser requests the file via HTTP. When the request reaches the correct web server (hardware), the *HTTP server* (software) accepts request, finds the requested document (if it doesn't then a 404 response is returned), and sends it back to the browser, also through HTTP



To publish a website, you need either a static or a dynamic web server.

A **static web server**, or stack, consists of a computer (hardware) with an HTTP server (software). We call it "static" because the server sends its hosted files "as-is" to your browser.

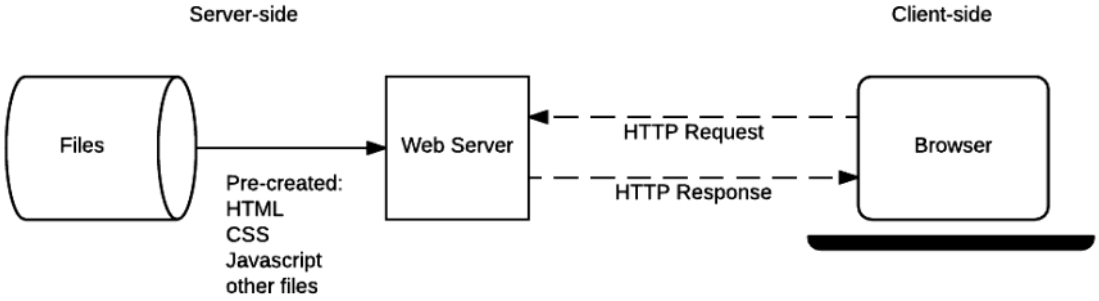
A **dynamic web server** consists of a static web server plus extra software, most commonly an *application server* and a *database*. We call it "dynamic" because the application server updates the hosted files before sending them to your browser via the HTTP server.

Static vs. Dynamic Website

Sunday, May 24, 2020 9:35 AM

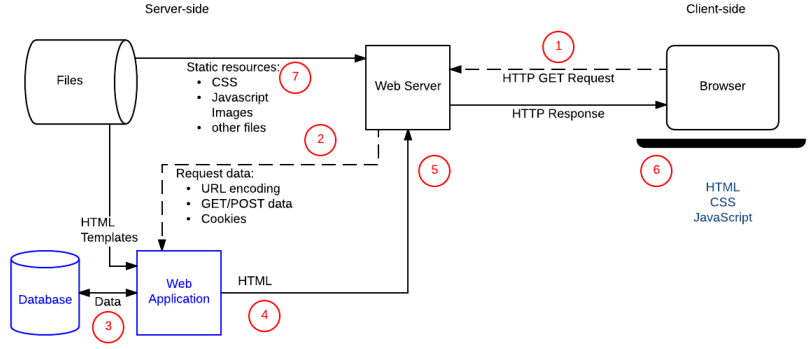
Static sites

The diagram below shows a basic web server architecture for a *static site* (a static site is one that returns the same hard-coded content from the server whenever a particular resource is requested). When a user wants to navigate to a page, the browser sends an HTTP "GET" request specifying its URL.
The server retrieves the requested document from its file system and returns an HTTP response containing the document and a success status (usually 200 OK). If the file cannot be retrieved for some reason, an error status is returned (see client error responses and server error responses).



Dynamic sites

A dynamic website is one where some of the response content is generated *dynamically*, only when needed. On a dynamic website HTML pages are normally created by inserting data from a database into placeholders in HTML templates (this is a much more efficient way of storing large amounts of content than using static websites).
A dynamic site can return different data for a URL based on information provided by the user or stored preferences and can perform other operations as part of returning a response (e.g. sending notifications).
Most of the code to support a dynamic website must run on the server. Creating this code is known as "**server-side programming**" (or sometimes "**back-end scripting**").
The diagram below shows a simple architecture for a *dynamic website*. As in the previous diagram, browsers send HTTP requests to the server, then the server processes the requests and returns appropriate HTTP responses.
Requests for *static* resources are handled in the same way as for static sites (static resources are any files that don't change —typically: CSS, JavaScript, Images, pre-created PDF files etc).



Requests for dynamic resources are instead forwarded (2) to server-side code (shown in the diagram as a *Web Application*). For "dynamic requests" the server interprets the request, reads required information from the database (3), combines the retrieved data with HTML templates (4), and sends back a response containing the generated HTML (5,6).

Let's now turn our attention to the code involved in server-side and client-side programming. In each case, the code is significantly different:

- They have different purposes and concerns.
- They generally don't use the same programming languages (the exception being JavaScript, which can be used on the server- and client-side).
- They run inside different operating system environments.

Code running in the browser is known as **client-side code** and is primarily concerned with improving the appearance and behavior of a rendered web page. This includes selecting and styling UI components, creating layouts, navigation, form validation, etc. By contrast, server-side website programming mostly involves choosing *which content* is returned to the browser in response to requests. The server-side code handles tasks like validating submitted data and requests, using databases to store and retrieve data and sending the correct data to the client as required.
Client-side code is written using HTML, CSS, and JavaScript — it is run inside a web browser and has little or no access to the underlying operating system (including limited access to the file system).
Web developers can't control what browser every user might be using to view a website — browsers provide inconsistent levels of compatibility with client-side code features, and part of the challenge of client-side programming is handling differences in browser support gracefully.
Server-side code can be written in any number of programming languages — examples of popular server-side web languages include PHP, Python, Ruby, C#, and NodeJS(JavaScript). The server-side code has full access to the server operating system and the developer can choose what programming language (and specific version) they wish to use.
Developers typically write their code using **web frameworks**. Web frameworks are collections of functions, objects, rules and other code constructs designed to solve common problems, speed up development, and simplify the different types of tasks faced in a particular domain.
Again, while both client and server-side code use frameworks, the domains are very different, and hence so are the frameworks. Client-side web frameworks simplify layout and presentation tasks while server-side web frameworks provide a lot of “common” web server functionality that you might otherwise have to implement

yourself (e.g. support for sessions, support for users and authentication, easy database access, templating libraries, etc.).