**Serving Up Web Server Basics:**

When discussing how a Web server works, it is not enough to simply outline a diagram of how low-level network packets go in and out of a Web server. To give such a nuts-and-bolts explanation some sort of practical value, it must be placed in context.

Years ago, when Web servers were first prototyped, they served simple HTML documents and images. Today, they are frequently used for much more!

It's a safe assumption that most Internet users believe a Web site's success or failure is due to its content and functionality rather than the server used to power it. However, the choice of the correct server, and understanding its capabilities and limitations is an important step on the road to success.

**So, what does a Web server do?**

It serves static content to a Web browser at a basic level. This means that the Web server receives a request for a Web page such as

http://www.Webcompare.com/index.html

and maps that Uniform Resource Locator (URL)to a local file on the host server.

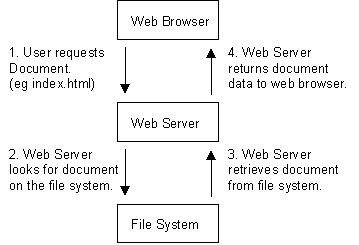
In this case, the file

index.html

is somewhere on the host file system. The server then loads this file from disk and serves it out across the network to the user's Web browser.

This entire exchange is mediated by the browser and server talking to each other using Hypertext Transfer Protocol (HTTP).

This workflow is shown in the figure below:



That's all there is to it.

**But if it's that simple, then why such an in-depth Analysis?**

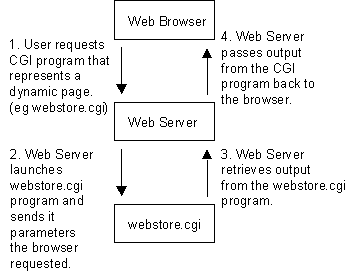
Because this simple arrangement, which allows the serving of static content such as HyperText Markup Language (HTML)and image files to a Web browser was the initial concept behind what we now call the World Wide Web. The beauty of its simplicity is that it has led to much more complex information exchanges being possible between browsers and Web servers.

Perhaps the most important expansion on this was the concept of **dynamic content (i.e., Web pages created in response to a user's input, whether directly or indirectly)**.

The oldest and most used standard for doing this is **Common Gateway Interface (CGI)**. This is a pretty meaningless name, but it basically defines how a Web server should run programs locally and transmit their output through the Web server to the user's Web browser that is requesting the dynamic content.

For all intents and purposes the user's Web browser never really has to know that the content is dynamic because **CGI is basically a Web server extension protocol.**

The figure below shows what happens **when a browser requests a page dynamically generated from a CGI program.**

****

**The second important advance**, and the one that makes e-commerce possible, was the introduction of **HyperText Transmission Protocol, Secure (HTTPS)**.

This protocol allows secure communication to go on between the browser and Web server. In a nutshell, this means that it is safe for user and server to transmit sensitive data to each another across what might be considered an insecure network. What happens when the data arrives at either end is another matter, however, and should not be ignored.

The simplicity of the above arrangements is deceptive or misleading, and underestimating its complexities often leads to bad decisions being made about the design of a Web-hosting infrastructure.

It is too easy to focus on the design of the Web pages themselves and the technologies used to create dynamic content, such as Java, Javascript, Perl, C/C++, and ASP, and to subsequently miss the fact that each of these technologies can be aided, or hindered, by the platform on which they are to be run -- the Web server itself.. In other words, **explaining how a Web server works involves discussing more than just how a Web server serves documents.**

**What Is HTTP, and How Does It Work?**

Put simply, HTTP is the protocol that allows Web browsers and servers to communicate. It forms the basis of what a Web server must do to perform its most basic operations.

HTTP started out as a very simple protocol, and even though it has had numerous enhancements, it is still relatively simple. As with other standard Internet protocols, control information is passed as plain text via a TCP connection.

In fact, HTTP connections can actually be made using standard "telnet" commands.

**For example:**

/home/chughes > **telnet** www.extropia 80

GET /index.html **HTTP/1.0**

*<- Extra char return needed*

**Note:** Port 80 is the default port a Web server "listens" on for connections.

In response to this HTTP GET command, the Web server returns to us the page "index.html" across the telnet session, and then closes the connection to signify the end of the document.

The following is part of the sample response:

**<HTML>**

**<HEAD>**

**<TITLE>eXtropia Homepage</TITLE>**

**[...]**

**</HEAD>**

**</HTML>**

But this simple request/response protocol was quickly outgrown, and it wasn't long before HTTP was refined into a more complex protocol (currently version 1.1). Perhaps the greatest change in HTTP/1.1 is its support for persistent connections.

In HTTP/1.0, a connection must to be made to the Web server for each object the browser wishes to download. Many Web pages are very graphic intensive, which means that in addition to downloading the base HTML page (or frames), the browser must also retrieve a number of images. Many of them may actually be quite small and merely sliced up to provide some hard-coded formatting framework to the rest of the HTML page.

Establishing a connection for each one is wasteful, as several network packets have to be exchanged between the Web browser and Web server before the image data can ever start transmitting. In contrast, opening a single TCP connection that transmits the HTML document and then each image one-by-one is more efficient, as the negotiation of starting new TCP connections is eliminated.

Now, there is more to a Web server than its function a communications protocol.

Ultimately, a Web server serves up content. This content must be identified in a way such that a Web browser can download and display that content in correct manner. The primary mechanism for deciding how to display content is the MIME type header.

**Multipurpose Internet Mail Extension (MIME)** types tell a Web browser what sort of document is being sent. Such type identification is not limited to simple graphics or HTML. In fact, more than 370 MIME types are distributed with the Apache Web server by default in the mime.types configuration file. And even this list does not represent the entire universe of possible MIME types! MIME types are distinguished using a type/subtype syntax associated with a file extension. Here is a brief snippet from an Apache mime.typesfile.

|  |  |
| --- | --- |
| text/xml | xml |
| video/mpeg | mpeg mpg mpe |
| video/quicktime | qt mov |

From this, we can see that files containing MPEG video content end with file extensions such as mpeg, mpg, or mpe. So a file with the name "southpark.mpeg" would be served up as being an MPEG video file.