HTTP (Hypertext Transfer Protocol) is the foundation of communication on the World Wide Web. It allows web browsers and servers to exchange data, enabling the retrieval and display of web pages. Over the years, HTTP has evolved, with HTTP/1.1 being the most widely used version for a long time. However, with the introduction of HTTP/2, significant improvements were made to enhance the performance and efficiency of web communication. In this blog, we will explore the differences between HTTP/1.1 and HTTP/2.

Protocol Design:

HTTP/1.1 is a text-based protocol where requests and responses are sent in plain text. Each request/response requires a separate TCP connection, which can lead to latency and overhead.

HTTP/2 is a binary protocol that uses a single, multiplexed connection. It allows multiple requests and responses to be sent concurrently over a single connection, reducing latency and improving efficiency.

Header Compression:

HTTP/1.1, headers are sent with each request and response, resulting in redundant data transmission. This can lead to increased bandwidth usage and slower performance.

HTTP/2 introduces header compression using the HPACK algorithm. It reduces the overhead of headers by compressing them before transmission, resulting in reduced bandwidth usage and improved performance.

Server Push:

HTTP/1.1, the server can only respond to client requests. If the server wants to send additional resources to the client, it needs to wait for the client to request them.

HTTP/2 introduces server push, where the server can proactively send additional resources to the client without waiting for a request. This reduces the number of round trips required and improves page load times.

Multiplexing:

HTTP/1.1, each request/response requires a separate TCP connection. This can lead to head-of-line blocking, where a slow request/response can block subsequent requests/responses.

HTTP/2 uses multiplexing, allowing multiple requests and responses to be sent concurrently over a single connection. This eliminates head-of-line blocking and improves overall performance.

Stream Prioritization:

HTTP/1.1, there is no built-in mechanism for prioritizing requests. All requests are treated equally, which can lead to inefficient resource allocation.

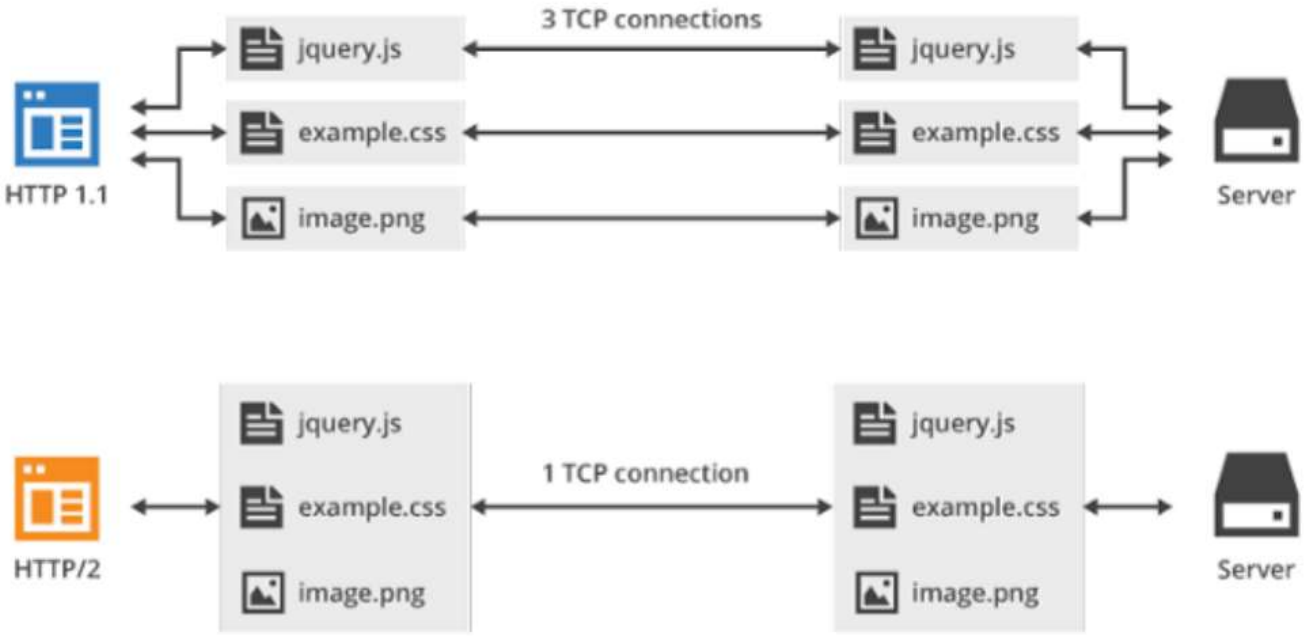
HTTP/2 introduces stream prioritization, allowing the client to assign priority to different requests. This enables more efficient resource allocation and improves the overall user experience.

Security:

HTTP/1.1 does not have built-in support for encryption. Secure communication requires the use of additional protocols such as HTTPS.

HTTP/2 has built-in support for encryption. Secure communication is the default, providing improved security and privacy.

In conclusion, HTTP/2 brings significant improvements over HTTP/1.1 in terms of performance, efficiency, and security. With features like multiplexing, header compression, server push, and stream prioritization, HTTP/2 enhances the web browsing experience by reducing latency, improving page load times, and optimizing resource allocation. As more websites and browsers adopt HTTP/2, the web will become faster and more efficient for users worldwide.


3 TCP connections
jquery.js
example.css
image.png
jquery.js
example.css
image.png
HTTP 1.1
Server
jquery.js
example.css
image.png
1 TCP connection
jquery.js
example.css
image.png
HTTP/2
Server

JavaScript object is a non-primitive data-type that allows you to store multiple collections of data.

Here is an example of a JavaScript object.

```
// object
const student = {
    firstName: 'ram',
    class: 10
};
```

JavaScript Object Declaration

The syntax to declare an object is:

```
const object_name = {
   key1: value1,
   key2: value2
}
```

Here, an object object_name is defined. Each member of an object is a **key: value** pair separated by commas and enclosed in curly braces {}.

```
// object creation
const person = {
    name: 'John',
    age: 20
};
console.log(typeof person); // object
```

You can also define an object in a single line.

```
const person = { name: 'John', age: 20 };
```

In the above example, name and age are keys, and John and 20 are values respectively.