Task 1: Write a blog on Difference between HTTP1.1 vs HTTP2

In order to Know about the difference between HTTP we need to know about what is http and its uses

* Protocol Definition: HTTP is a protocol used for transmitting hypertext and other data over the World Wide Web.
* Client-Server Communication: It operates on a client-server model, where a client (such as a web browser) makes requests to a server, and the server responds with the requested information.
* Stateless Nature: HTTP is stateless, meaning each request from a client to a server is independent, and the server does not retain any information about the client between requests.
* Port Number: HTTP typically uses port 80 for communication, but it can use other ports as well, like 8080 or 8000, depending on the configuration.
* Uniform Resource Identifier (URI): HTTP uses URIs to identify resources on the web. URLs (Uniform Resource Locators) are a common form of URIs.
* Request Methods: HTTP defines various request methods, including GET (retrieve data), POST (submit data), PUT (update a resource), DELETE (remove a resource), etc.
* Header Fields: HTTP messages contain header fields that provide information about the request or response, such as the type of data being sent, the length of the content, and the desired action.
* Status Codes: Responses from the server include status codes indicating the result of the request (e.g., 200 OK, 404 Not Found, 500 Internal Server Error).
* HTTP Cookies: Cookies are small pieces of data sent from a server and stored on the client's side, used for tracking and maintaining information about a user's session.
* Persistent Connections: HTTP supports persistent connections, allowing multiple requests and responses to be sent over a single connection, reducing latency.
* SSL/TLS for Security: HTTPS (Hypertext Transfer Protocol Secure) is a secure version of HTTP that uses SSL/TLS protocols to encrypt data transmitted between the client and server.
* HTTP/1.1 Pipelining: Allows multiple requests to be sent without waiting for each response, improving performance by reducing latency.
* Caching: HTTP supports caching mechanisms, allowing clients to store copies of resources locally to reduce the need for repeated requests to the server.
* Content Negotiation: Clients and servers can negotiate the format and language of the data being exchanged based on the preferences specified in the request headers.
* WebSockets: While HTTP is traditionally request-response-based, WebSockets provide a full-duplex communication channel over a single, long-lived connection, allowing for real-time data exchange between the client and server.

**What is HTTP/1:**

1. Introduction: HTTP/1.1 is a version of the Hypertext Transfer Protocol, designed to improve upon the original HTTP/1.0.
2. Standardization: HTTP/1.1 is defined by the Internet Engineering Task Force (IETF) in RFC 2616, which was later updated by RFC 7230-7235.
3. Persistent Connections: HTTP/1.1 introduced persistent connections, allowing multiple requests and responses to be sent over a single TCP connection, reducing latency.
4. Host Header: It introduced the mandatory "Host" header in the HTTP request, enabling the hosting of multiple websites on the same IP address.
5. Request Pipelining: HTTP/1.1 supports pipelining, allowing multiple requests to be sent without waiting for each response, improving the efficiency of data transmission.
6. Chunked Transfer Encoding: This encoding mechanism allows the server to send data in chunks, which is useful for streaming content or for responses with unknown lengths.
7. Range Requests: HTTP/1.1 supports range requests, enabling clients to request only specific portions of a resource, which is useful for resuming downloads or streaming.
8. Caching Improvements: It introduced more robust caching mechanisms, including the ability to cache partial responses and cache control headers like "Cache-Control."
9. Content Negotiation: HTTP/1.1 allows clients and servers to negotiate the content format and language based on the preferences specified in the request headers.
10. Status Code 100 Continue: HTTP/1.1 introduced the "100 Continue" status code, allowing a client to check if a server is willing to accept the request before sending the entire message.
11. Connection Header: It introduced the "Connection" header with values like "keep-alive" or "close" to control whether the connection should be kept open for multiple requests.
12. Improved Error Handling: HTTP/1.1 refined the status codes for better error handling, providing more specific codes for different types of errors.
13. Transfer-Encoding and Content-Length: HTTP/1.1 clarified the interaction between the "Transfer-Encoding" and "Content-Length" headers to avoid ambiguities in message framing.
14. Hosted Virtual Servers: With the introduction of the "Host" header, HTTP/1.1 facilitated the hosting of multiple virtual servers on a single physical server.
15. Backward Compatibility: HTTP/1.1 is designed to be backward compatible with HTTP/1.0, allowing servers and clients that support HTTP/1.1 to communicate with those that only support HTTP/1.0.

**What is HTTP/2**

* Introduction: HTTP/2 is the second major version of the Hypertext Transfer Protocol, designed to improve the performance and address limitations of HTTP/1.1.
* Standardization: HTTP/2 is specified by the Internet Engineering Task Force (IETF) in RFC 7540, published in May 2015.
* Binary Protocol: Unlike HTTP/1.1, which is a text-based protocol, HTTP/2 uses a binary protocol for more efficient parsing and reduced overhead.
* Multiplexing: One of the key features of HTTP/2 is multiplexing, allowing multiple requests and responses to be sent in parallel over a single connection, improving overall page load times.
* Header Compression: HTTP/2 employs header compression (HPACK) to reduce redundant header information, further decreasing latency and bandwidth usage.
* Stream Prioritization: It introduces stream prioritization, allowing for the prioritized delivery of certain resources, ensuring that critical resources are loaded first.
* Server Push: HTTP/2 supports server push, where the server can proactively push resources to the client's cache before the client requests them, optimizing page load times.
* Binary Framing Layer: HTTP/2 uses a binary framing layer to encapsulate messages, enabling more efficient communication by eliminating the ambiguity associated with parsing text-based protocols.
* Flow Control: HTTP/2 features flow control mechanisms at both the connection and stream levels, preventing a fast sender from overwhelming a slow receiver.
* Connection Reuse: Connection reuse is inherent in HTTP/2, reducing the need for repeatedly establishing and tearing down connections, leading to improved efficiency.
* Header Fields Reduction: The use of header compression and binary representation significantly reduces the amount of redundant header information, saving bandwidth.
* Backward Compatibility: HTTP/2 is designed to be fully backward compatible with HTTP/1.1, allowing existing web applications to migrate to the new protocol without requiring a complete overhaul.
* Security: While not a direct feature of the protocol, the use of HTTP/2 often encourages the adoption of secure connections (HTTPS) due to the associated performance benefits.
* Alleviation of Head-of-Line Blocking: Multiplexing in HTTP/2 helps alleviate the head-of-line blocking problem, where one slow request could block subsequent requests in HTTP/1.1.
* Wide Adoption: Many modern web browsers and servers support HTTP/2, contributing to its widespread adoption for delivering faster and more efficient web experiences.

Here is the difference between the http/1 and http/2

|  |  |  |
| --- | --- | --- |
| **Multiplexing** | No multiplexing, one request at a time | Supports multiplexing, multiple requests in parallel |
| **Header Compression** | Headers are not compressed | Uses header compression (HPACK) for reduced overhead |
| **Binary Protocol** | Text-based protocol | Binary protocol for more efficient data transfer |
| **Connection Handling** | Requires multiple connections for parallelism | Single connection supports parallel streams |
| **Prioritization** | No built-in support | Supports stream prioritization for better resource allocation |
| **Server Push** | Not supported | Allows server to push resources to the client proactively |
| **Header Size** | Larger header size due to redundancy | Smaller header size due to compression |
| **Latency** | Higher latency due to head-of-line blocking | Lower latency, as multiple streams can be processed concurrently |
| **Flow Control** | Only supports simple flow control | Supports more efficient flow control mechanisms |
| **Request Compression** | No support for request/response compression | Supports request/response body compression |
| **TLS Usage** | Optional (HTTP/1.1) | Encouraged and more secure with mandatory TLS |
| **Error Handling** | Errors can result in blocking subsequent requests | Supports independent error handling for streams |
| **Backward Compatibility** | Fully backward compatible with HTTP/1.0 | Backward compatible, can be used over HTTP/1.1 |
| **Resource Prioritization** | Lacks explicit prioritization | Prioritizes resources using dependencies and weights |
| **Header Fields** | No support for header field reordering | Supports header field reordering for efficiency |
| **Connection Usage** | Parallelism | Single connection for multiple parallel streams |
| **Adoption Timeline** | Introduced in 1997 | Published as a standard in 2015 |