Traditional web server Model

* In the traditional web server model, Node.js follows a non-blocking, event-driven architecture, allowing it to handle concurrent requests efficiently. Here's an overview of how the traditional web server model works in Node.js:
* 1. Incoming Request: When a client makes an HTTP request to a Node.js server, the server receives the request and begins processing it.
* 2. Event Loop: Node.js utilizes an event loop, which constantly checks for new events or tasks to execute.
* 3. Non-Blocking I/O: Node.js is designed to handle I/O operations asynchronously in a non-blocking manner. It means that when the server encounters an I/O operation, such as reading from a file or making a network request, it doesn't block the execution of subsequent code. Instead, it delegates the task to the underlying system, and the server continues to handle other requests.
* 4. Callbacks: To handle asynchronous operations, Node.js commonly uses callbacks. When an I/O operation is completed, the corresponding callback function is triggered, allowing the server to continue processing other tasks.
* 5. Event-driven Architecture: Node.js leverages an event-driven architecture, where events are emitted for various actions or triggers. These events can be predefined, such as a request arriving or a file being read, or they can be custom events defined by the developer.
* 6. Event Handlers: Event handlers, or listeners, are registered to respond to specific events. When an event occurs, the associated event handler is executed.
* 7. Response: Once the server has processed the request and generated the response, it sends the response back to the client.
* 8. Scalability: Due to its non-blocking and event-driven nature, Node.js excels in handling concurrent requests efficiently. It can handle a large number of connections with relatively low resource consumption.
* By utilizing the event-driven and non-blocking I/O model, Node.js is able to achieve high performance and scalability, making it well-suited for building fast and efficient web servers.

Example:

const http = require('http');

// Create a server instance

const server = http.createServer((req, res) => {

// Set the response header

res.writeHead(200, { 'Content-Type': 'text/plain' });

// Write the response content

res.write('Hello, World!');

// End the response

res.end();

});

// Start the server and listen on port 3000

server.listen(3000, () => {

console.log('Server is running on port 3000');

});