* **Node.js** is a cross-platform environment and library for running JavaScript applications which is used to create networking and server-side applications.

Node.js is a cross-platform runtime environment and library for running JavaScript applications outside the browser. It is used for creating server-side and networking web applications. It is open source and free to use. It can be downloaded from this link <https://nodejs.org/en/>

Many of the basic modules of Node.js are written in JavaScript. Node.js is mostly used to run real-time server applications.

The definition given by its official documentation is as follows:

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.?

Node.js also provides a rich library of various JavaScript modules to simplify the development of web applications.

Node.js = Runtime Environment + JavaScript Library

**Node.js** is an [open-source](https://en.wikipedia.org/wiki/Open-source_software), [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) [JavaScript](https://en.wikipedia.org/wiki/JavaScript) [run-time environment](https://en.wikipedia.org/wiki/Runtime_system) that executes JavaScript code outside of a browser. Historically, JavaScript was used primarily for [client-side scripting](https://en.wikipedia.org/wiki/Client-side_scripting), in which scripts written in JavaScript are embedded in a webpage's HTML and run client-side by a JavaScript engine in the user's web browser. Node.js lets developers use JavaScript to write Command Line tools and for [server-side scripting](https://en.wikipedia.org/wiki/Server-side_scripting)—running scripts server-side to produce [dynamic web page](https://en.wikipedia.org/wiki/Dynamic_web_page) content *before* the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying [web application](https://en.wikipedia.org/wiki/Web_application) development around a single programming language, rather than different languages for server side and client side scripts.

Though .js is the conventional [filename extension](https://en.wikipedia.org/wiki/Filename_extension) for JavaScript code, the name "Node.js" does not refer to a particular file in this context and is merely the name of the product. Node.js has an [event-driven architecture](https://en.wikipedia.org/wiki/Event-driven_architecture) capable of [asynchronous I/O](https://en.wikipedia.org/wiki/Asynchronous_I/O). These design choices aim to optimize [throughput](https://en.wikipedia.org/wiki/Throughput) and [scalability](https://en.wikipedia.org/wiki/Scalability) in web applications with many input/output operations, as well as for [real-time Web](https://en.wikipedia.org/wiki/Real-time_Web) applications (e.g., [real-time communication](https://en.wikipedia.org/wiki/Real-time_communication) programs and [browser games](https://en.wikipedia.org/wiki/Browser_game)).

Node.js was originally written by [Ryan Dahl](https://en.wikipedia.org/wiki/Ryan_Dahl) in 2009,[[23]](https://en.wikipedia.org/wiki/Node.js#cite_note-training.com-23) about thirteen years after the introduction of the first server-side JavaScript environment, Dahl was inspired to create Node.js after seeing a file upload [progress bar](https://en.wikipedia.org/wiki/Progress_bar) on [Flickr](https://en.wikipedia.org/wiki/Flickr). The browser did not know how much of the file had been uploaded and had to query the Web server. Dahl desired an easier way.

In January 2010, a [package manager](https://en.wikipedia.org/wiki/Package_manager) was introduced for the Node.js environment called [*npm*](https://en.wikipedia.org/wiki/Npm_(software)). The package manager makes it easier for programmers to publish and share source code of Node.js libraries and is designed to simplify installation, updating, and uninstallation of libraries.

In June 2011, Microsoft and Joyent implemented a native [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows) version of Node.js. The first Node.js build supporting Windows was released in July 2011. In December 2014, Fedor Indutny started io.js, a [fork](https://en.wikipedia.org/wiki/Fork_(software_development)) of Node.js. Due to the internal conflict over Joyent's governance, io.js was created as an [open governance](https://en.wikipedia.org/wiki/Open_governance) alternative with a separate technical committee.[[36]](https://en.wikipedia.org/wiki/Node.js#cite_note-Krill,_Paul-36)[[37]](https://en.wikipedia.org/wiki/Node.js#cite_note-iwtw-37) Unlike Node.js,[[38]](https://en.wikipedia.org/wiki/Node.js#cite_note-38) the authors planned to keep io.js up-to-date with the latest releases of the [Google V8](https://en.wikipedia.org/wiki/V8_(JavaScript_engine)) JavaScript engine.

In February 2015, the intent to form a neutral Node.js Foundation was announced. By June 2015, the Node.js and io.js communities voted to work together under the Node.js Foundation.In September 2015, Node.js v0.12 and io.js v3.3 were merged back together into Node v4.0.[[41]](https://en.wikipedia.org/wiki/Node.js#cite_note-41) This brought V8 [ES6](https://en.wikipedia.org/wiki/ECMAScript#ES6) features into Node.js, and a long-term support release cycle.[[42]](https://en.wikipedia.org/wiki/Node.js#cite_note-42) As of 2016, the io.js website recommends that developers switch back to Node.js and that no further releases of io.js are planned due to the merger.

Node.js allows the creation of [Web servers](https://en.wikipedia.org/wiki/Web_server) and networking tools using [JavaScript](https://en.wikipedia.org/wiki/JavaScript) and a collection of "modules" that handle various core functionality. Modules are provided for [file system](https://en.wikipedia.org/wiki/File_system) I/O, networking ([DNS](https://en.wikipedia.org/wiki/Domain_Name_System), [HTTP](https://en.wikipedia.org/wiki/HTTP), [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol), [TLS/SSL](https://en.wikipedia.org/wiki/Transport_Layer_Security), or [UDP](https://en.wikipedia.org/wiki/User_Datagram_Protocol)), [binary](https://en.wikipedia.org/wiki/Binary_file) data (buffers), [cryptography](https://en.wikipedia.org/wiki/Cryptography) functions, [data streams](https://en.wikipedia.org/wiki/Stream_(computing)), and other core functions. Node.js's modules use an API designed to reduce the complexity of writing server applications.

Node.js is officially supported on [Linux](https://en.wikipedia.org/wiki/Linux), [macOS](https://en.wikipedia.org/wiki/MacOS), [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows), [SmartOS](https://en.wikipedia.org/wiki/SmartOS), [FreeBSD](https://en.wikipedia.org/wiki/FreeBSD), and [IBM AIX](https://en.wikipedia.org/wiki/IBM_AIX). The provided source code may also be built on similar operating systems or be modified by third parties to support others such as [NonStop](https://en.wikipedia.org/wiki/NonStop) and [Unix](https://en.wikipedia.org/wiki/Unix) servers. Alternatively, they can be written with [CoffeeScript](https://en.wikipedia.org/wiki/CoffeeScript) (a JavaScript alternative), [Dart](https://en.wikipedia.org/wiki/Dart_(programming_language)) or [TypeScript](https://en.wikipedia.org/wiki/TypeScript) ([strongly typed](https://en.wikipedia.org/wiki/Strongly_typed) forms of JavaScript), or any other language that [can compile to JavaScript](https://en.wikipedia.org/wiki/Source-to-source_compiler).

Node.js is primarily used to build network programs such as Web servers. The biggest difference between Node.js and [PHP](https://en.wikipedia.org/wiki/PHP) is that most functions in PHP [block](https://en.wikipedia.org/wiki/Asynchronous_I/O) until completion (commands execute only after previous commands finish), while Node.js functions are [non-blocking](https://en.wikipedia.org/wiki/Asynchronous_I/O) (commands execute [concurrently](https://en.wikipedia.org/wiki/Concurrent_computing) or even in [parallel](https://en.wikipedia.org/wiki/Parallel_computing),[[52]](https://en.wikipedia.org/wiki/Node.js#cite_note-node_cluster_module_doc-52)[[53]](https://en.wikipedia.org/wiki/Node.js#cite_note-53) and use [callbacks](https://en.wikipedia.org/wiki/Callback_(computer_programming)) to signal completion or failure).

Node.js brings [event-driven programming](https://en.wikipedia.org/wiki/Event-driven_programming) to [web servers](https://en.wikipedia.org/wiki/Web_server), enabling development of fast web servers in JavaScript.[[31]](https://en.wikipedia.org/wiki/Node.js#cite_note-b1-31) Developers can create highly scalable servers without using [threading](https://en.wikipedia.org/wiki/Thread_(computing)), by using a simplified model of [event-driven programming](https://en.wikipedia.org/wiki/Event-driven_programming) that uses callbacks to signal the completion of a task.[[31]](https://en.wikipedia.org/wiki/Node.js#cite_note-b1-31) Node.js connects the ease of a scripting language (JavaScript) with the power of Unix network programming.

Node.js was built on the [Google](https://en.wikipedia.org/wiki/Google) [V8 JavaScript engine](https://en.wikipedia.org/wiki/V8_JavaScript_engine) since it was open-sourced under the [BSD license](https://en.wikipedia.org/wiki/BSD_license). It is extremely fast and proficient with internet fundamentals such as [HTTP](https://en.wikipedia.org/wiki/HTTP), [DNS](https://en.wikipedia.org/wiki/DNS), [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol).[[28]](https://en.wikipedia.org/wiki/Node.js#cite_note-b3-28) Also, [JavaScript](https://en.wikipedia.org/wiki/JavaScript) was a well-known language, making Node.js immediately accessible to the entire [web development community](https://en.wikipedia.org/wiki/Web_developer).

The core functionality of Node.js resides in a JavaScript library. The Node.js bindings, written in C++, connect these technologies to each other and to the operating system.

* **Node JS Architecture – Single Threaded Event Loop**

Node JS applications uses “Single Threaded Event Loop Model” architecture to handle multiple concurrent clients. There are many web application technologies like JSP, Spring MVC, ASP.NET, HTML, Ajax, jQuery etc. But all these technologies follow “Multi-Threaded Request-Response” architecture to handle multiple concurrent clients.

Node.js operates on a [single thread](https://en.wikipedia.org/wiki/Single_threading), using [non-blocking I/O](https://en.wikipedia.org/wiki/Non-blocking_I/O) calls, allowing it to support tens of thousands of concurrent connections without incurring the cost of thread [context switching](https://en.wikipedia.org/wiki/Context_switch).[[68]](https://en.wikipedia.org/wiki/Node.js#cite_note-68) The design of sharing a single thread among all the requests that use the [observer pattern](https://en.wikipedia.org/wiki/Observer_pattern) is intended for building highly concurrent applications, where any function performing I/O must use a [callback](https://en.wikipedia.org/wiki/Callback_(computer_programming)). To accommodate the single-threaded event loop, Node.js uses the [libuv](https://en.wikipedia.org/wiki/Libuv) library—which, in turn, uses a fixed-sized thread pool that handles some of the non-blocking asynchronous I/O operations.

A downside of this single-threaded approach is that Node.js doesn't allow [vertical scaling](https://en.wikipedia.org/wiki/Vertical_scaling) by increasing the number of [CPU cores](https://en.wikipedia.org/wiki/CPU_core) of the machine it is running on without using an additional module, such as cluster,[[52]](https://en.wikipedia.org/wiki/Node.js#cite_note-node_cluster_module_doc-52) StrongLoop Process Manager, or pm2. However, developers can increase the default number of threads in the libuv thread pool. The server [operating system (OS)](https://en.wikipedia.org/wiki/Operating_system) is likely to distribute these threads across multiple cores.

A thread pool handles execution of parallel tasks in Node.js. The main thread call functions post tasks to the shared task queue, which threads in the thread pool pull and execute. Inherently non-blocking system functions such as networking translate to kernel-side non-blocking sockets, while inherently blocking system functions such as file I/O run in a blocking way on their own threads. When a thread in the thread pool completes a task, it informs the main thread of this, which in turn, wakes up and executes the registered callback. Since Node.js handles callbacks in serial on the main thread, long lasting computations and other CPU-bound tasks freeze the entire event-loop until completion.

* **Traditional Web Application Processing Model**

Any Web Application developed without Node JS, typically follows “Multi-Threaded Request-Response” model. Simply we can call this model as Request/Response Model. Client sends request to the server, then server do some processing based on clients request, prepare response and send it back to the client. This model uses HTTP protocol. As HTTP is a Stateless Protocol, this Request/Response model is also Stateless Model. So we can call this as Request/Response Stateless Model. However, this model uses Multiple Threads to handle concurrent client requests.

**Request/Response Model Processing Steps**:

* Clients Send request to Web Server.
* Web Server internally maintains a Limited Thread pool to provide services to the Client Requests.
* Web Server is in infinite Loop and waiting for Client Incoming Requests
* Web Server receives those requests.
  + Web Server pickup one Client Request
  + Pickup one Thread from Thread pool
  + Assign this Thread to Client Request
  + This Thread will take care of reading Client request, processing Client request, performing any Blocking IO Operations (if required) and preparing Response
  + This Thread sends prepared response back to the Web Server
  + Web Server in-turn sends this response to the respective Client.

Server waits in Infinite loop and performs all sub-steps as mentioned above for all n clients. That means this model creates one Thread per Client request. If more clients requests require Blocking IO Operations, then almost all threads are busy in preparing their responses. Then remaining clients Requests should wait for longer time.

[Request Response Model, Multithreaded request response architecture](https://cdn.journaldev.com/wp-content/uploads/2015/04/Request-Response-Model.png)

**Diagram Description**:

* Here “n” number of Clients Send request to Web Server. Let us assume they are accessing our Web Application concurrently.
* Let us assume, our Clients are Client-1, Client-2… and Client-n.
* Web Server internally maintains a Limited Thread pool. Let us assume “m” number of Threads in Thread pool.
* Web Server receives those requests one by one.
  + Web Server pickup Client-1 Request-1, Pickup one Thread T-1 from Thread pool and assign this request to Thread T-1
    - Thread T-1 reads Client-1 Request-1 and process it
    - Client-1 Request-1 does not require any Blocking IO Operations
    - Thread T-1 does necessary steps and prepares Response-1 and send it back to the Server
    - Web Server in-turn send this Response-1 to the Client-1
  + Web Server pickup another Client-2 Request-2, Pickup one Thread T-2 from Thread pool and assign this request to Thread T-2
    - Thread T-2 reads Client-2 Request-2 and process it
    - Client-2 Request-2 does not require any Blocking IO Operations
    - Thread T-2 does necessary steps and prepares Response-2 and send it back to the Server
    - Web Server in-turn send this Response-2 to the Client-2
  + Web Server pickup another Client-n Request-n, Pickup one Thread T-n from Thread pool and assign this request to Thread T-n
    - Thread T-n reads Client-n Request-n and process it
    - Client-n Request-n require heavy Blocking IO and computation Operations
    - Thread T-n takes more time to interact with external systems, does necessary steps and prepares Response-n and send it back to the Server
    - Web Server in-turn send this Response-n to the Client-n

If “n” is greater than “m” (Most of the times, its true), then server assigns Threads to Client Requests up to available Threads. After all m Threads are utilized, then remaining Client’s Request should wait in the Queue until some of the busy Threads finish their Request-Processing Job and free to pick up next Request.

If those threads are busy with Blocking IO Tasks (For example, interacting with Database, file system, JMS Queue, external services etc.) for longer time, then remaining clients should wait longer time. Once Threads are free in Thread Pool and available for next tasks, Server pickup those threads and assign them to remaining Client Requests. Each Thread utilizes many resources like memory etc. So before going those Threads from busy state to waiting state, they should release all acquired resources.

**Drawbacks of Request/Response Stateless Model**:

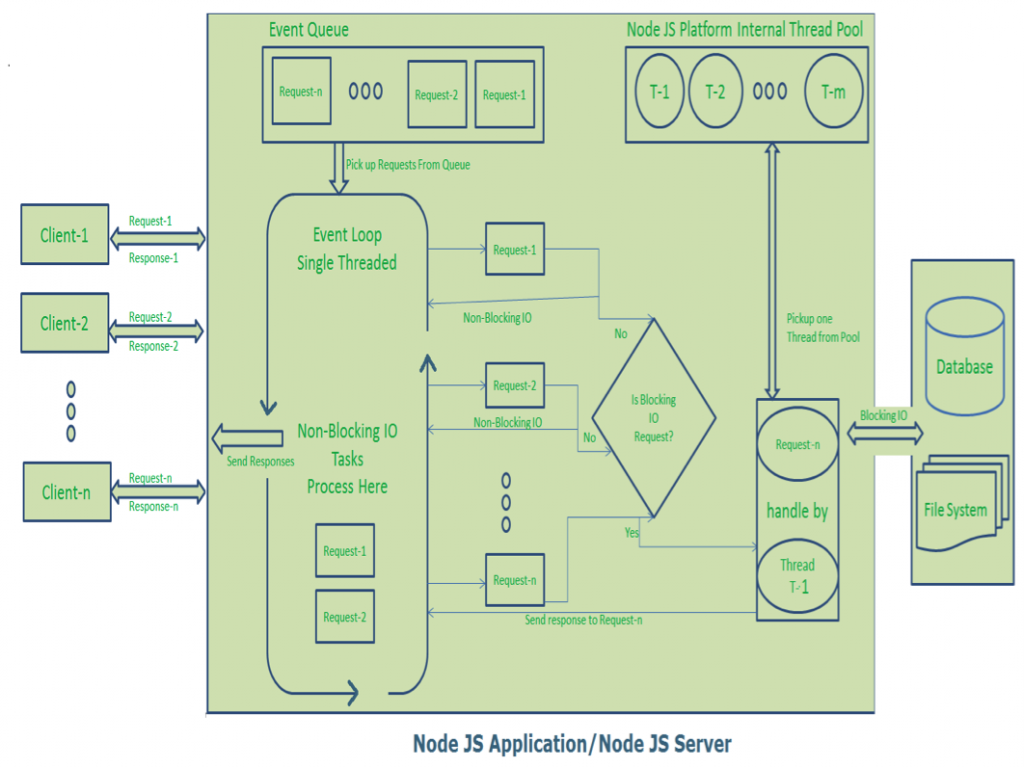
* Handling more and more concurrent client’s request is bit tough.
* When Concurrent client requests increases, then it should use more and more threads, finally they eat up more memory.
* Sometimes, Client’s Request should wait for available threads to process their requests.
* Wastes time in processing Blocking IO Tasks.
* **Node JS Architecture – Single Threaded Event Loop**

Node JS Platform does not follow Request/Response Multi-Threaded Stateless Model. It follows Single Threaded with Event Loop Model. Node JS Processing model mainly based on Javascript Event based model with Javascript callback mechanism. As Node JS follows this architecture, it can handle more and more concurrent client requests very easily.

The main heart of Node JS Processing model is “Event Loop”. If we understand this, then it is very easy to understand the Node JS Internals.

**Single Threaded Event Loop Model Processing Steps**:

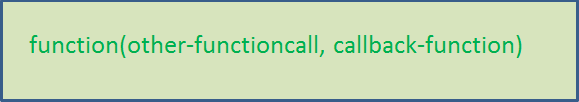
* Clients Send request to Web Server.
* Node JS Web Server internally maintains a Limited Thread pool to provide services to the Client Requests.
* Node JS Web Server receives those requests and places them into a Queue. It is known as “Event Queue”.
* Node JS Web Server internally has a Component, known as “Event Loop”. Why it got this name is that it uses indefinite loop to receive requests and process them. (See some Java Pseudo code to understand this below).
* Event Loop uses Single Thread only. It is main heart of Node JS Platform Processing Model.
* Even Loop checks any Client Request is placed in Event Queue. If no, then wait for incoming requests for indefinitely.
* If yes, then pick up one Client Request from Event Queue
  + Starts process that Client Request
  + If that Client Request Does Not requires any Blocking IO Operations, then process everything, prepare response and send it back to client.
  + If that Client Request requires some Blocking IO Operations like interacting with Database, File System, External Services then it will follow different approach
    - Checks Threads availability from Internal Thread Pool
    - Picks up one Thread and assign this Client Request to that thread.
    - That Thread is responsible for taking that request, process it, perform Blocking IO operations, prepare response and send it back to the Event Loop
    - Event Loop in turn, sends that Response to the respective Client.

[](https://cdn.journaldev.com/wp-content/uploads/2015/04/NodeJS-Single-Thread-Event-Model.png)

**Diagram Description**:

* Here “n” number of Clients Send request to Web Server. Let us assume they are accessing our Web Application concurrently.
* Let us assume, our Clients are Client-1, Client-2… and Client-n.
* Web Server internally maintains a Limited Thread pool. Let us assume “m” number of Threads in Thread pool.
* Node JS Web Server receives Client-1, Client-2… and Client-n Requests and places them in the Event Queue.
* Node JS Even Loop Picks up those requests one by one.
  + Even Loop pickups Client-1 Request-1
    - Checks whether Client-1 Request-1 does require any Blocking IO Operations or takes more time for complex computation tasks.
    - As this request is simple computation and Non-Blocking IO task, it does not require separate Thread to process it.
    - Event Loop process all steps provided in that Client-1 Request-1 Operation (Here Operations means Java Script’s functions) and prepares Response-1
    - Event Loop sends Response-1 to Client-1
  + Even Loop pickups Client-2 Request-2
    - Checks whether Client-2 Request-2does require any Blocking IO Operations or takes more time for complex computation tasks.
    - As this request is simple computation and Non-Blocking IO task, it does not require separate Thread to process it.
    - Event Loop process all steps provided in that Client-2 Request-2 Operation and prepares Response-2
    - Event Loop sends Response-2 to Client-2
  + Even Loop pickups Client-n Request-n
    - Checks whether Client-n Request-n does require any Blocking IO Operations or takes more time for complex computation tasks.
    - As this request is very complex computation or Blocking IO task, Even Loop does not process this request.
    - Event Loop picks up Thread T-1 from Internal Thread pool and assigns this Client-n Request-n to Thread T-1
    - Thread T-1 reads and process Request-n, perform necessary Blocking IO or Computation task, and finally prepares Response-n
    - Thread T-1 sends this Response-n to Event Loop
    - Event Loop in turn, sends this Response-n to Client-n

Here Client Request is a call to one or more Java Script Functions. Java Script Functions may call other functions or may utilize its Callback functions nature. So Each Client Request looks like as shown below:

[](https://cdn.journaldev.com/wp-content/uploads/2015/04/javascript-callback-mechanism.png)

For Example:

**Copy**

function1(function2,callback1);

function2(function3,callback2);

function3(input-params);

**Node JS Architecture – Single Threaded Event Loop Advantages**

1. Handling more and more concurrent client’s request is very easy.
2. Even though our Node JS Application receives more and more Concurrent client requests, there is no need of creating more and more threads, because of Event loop.
3. Node JS application uses less Threads so that it can utilize only less resources or memory

**Event Loop Pseudo Code**

As I’m a Java Developer, I will try to explain “How Event Loop works” in Java terminology. It is not in pure Java code, I guess everyone can understand this. If you face any issues in understanding this, please drop me a comment.

**Copy**

**public class EventLoop {**

**while(true){**

**if(Event Queue receives a JavaScript Function Call){**

**ClientRequest request = EventQueue.getClientRequest();**

**If(request requires BlokingIO or takes more computation time)**

**Assign request to Thread T1**

**Else**

**Process and Prepare response**

**}**

**}**

**}**

That’s all for Node JS Architecture and Node JS single threaded event loop.

Node.js has mainly two types of components – core components and node.js API (modules). The core components are written in C and C++, and node.js API are written in JavaScript. Diagram for Node.js architecture is given below:

## Node.js API

These are written in JavaScript and directly exposed to outer world to interact with Node.js internal components. Node.js Binding – These are Core API, which bind the JavaScript with C / C++ libraries.

## C/C++ Add-ons

You can also develop your Node.js Add-ons using C/C++ to work with Node.js.

## V8

It is Google’s open source JavaScript engine, written in C++. Actually, it is a JavaScript VM which compile the JavaScript code into native machine code instead interpretation. It is the fastest JIT (Just-In-Time) compiler for JavaScript. [It](https://en.wikipedia.org/wiki/Chrome_V8) is the JavaScript execution engine which was initially built for [Google Chrome](https://en.wikipedia.org/wiki/Google_Chrome). It was then open-sourced by Google in 2008. Written in [C++](https://en.wikipedia.org/wiki/C%2B%2B), V8 compiles JavaScript source code to native [machine code](https://en.wikipedia.org/wiki/Machine_code) instead of interpreting it in real time.[[6]](https://en.wikipedia.org/wiki/Node.js#cite_note-readwrite-6)

## Libuv

It is a multi-platform support C++ library which is responsible for handling thread pool, event loop and async I/O operations in Node.js. In Node.js, blocking I/O operations are delegated to Libuv modules which has a fixed size C++ thread pool to handle these operations. When these operations are completed, they are notified to Event loop. Node.js uses [libuv](https://en.wikipedia.org/wiki/Libuv) to handle asynchronous events. Libuv is an abstraction layer for network and file system functionality on both Windows and [POSIX](https://en.wikipedia.org/wiki/POSIX)-based systems such as Linux, macOS, OSS on NonStop, and Unix.

## C-ares

It is a C library for handling async DNS request, name resolves and multiple DNS queries in parallel.

## http\_parser

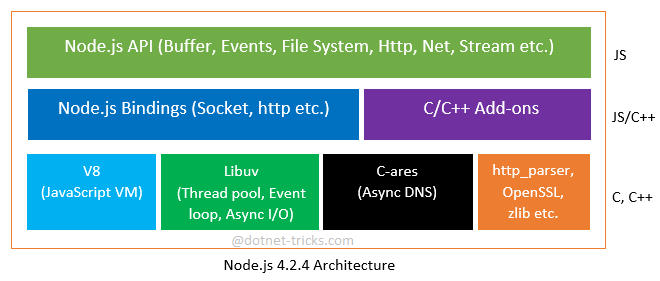
It is a C library for parsing HTTP request and response.

## OpenSSL

It is a C library for the implementation of Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols. It also provides all the necessary cryptography methods like hash, cipher, decipher, sign and verify etc.

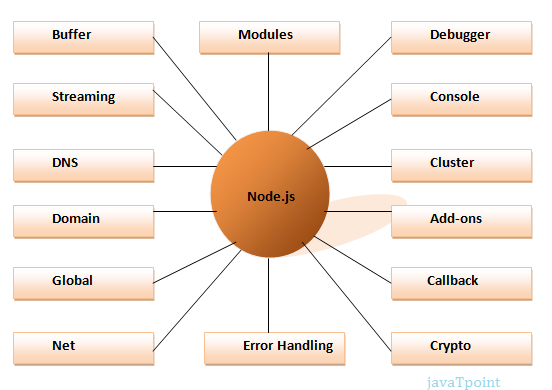
## Zlib

It is a C library for data compression and decompression.



* **Different parts of Node.js**

The following diagram specifies some important parts of Node.js:



**Features of Node.js**

Following is a list of some important features of Node.js that makes it the first choice of software architects.

1. **Extremely fast:** Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.
2. **I/O is Asynchronous and Event Driven:** All APIs of Node.js library are asynchronous i.e. non-blocking. So a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call. It is also a reason that it is very fast.
3. **Single threaded:** Node.js follows a single threaded model with event looping.
4. **Highly Scalable:** Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.
5. **No buffering:** Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.
6. **Open source:** Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.
7. **License:** Node.js is released under the MIT license.

* **The Node.js Runtime:**

The source code written in source file is simply JavaScript. It is interpreted and executed by the Node.js interpreter.

**There can be console-based and web-based node.js applications.**

## Node.js console-based Example

## ckk01.js

console.log("God is great, this is my first node j")

Open Node.js command prompt and run the following code:

node ckk01.js

Here, console.log() function displays message on console.

## Node.js web-based Example

A node.js web application contains the following three parts:

1. **Import required modules:** The "require" directive is used to load a Node.js module. The first step is to use ?require? directive to load http module and store returned HTTP instance into http variable. For example: var http = require("http");
2. **Create server:** You have to establish a server which will listen to client's request similar to Apache HTTP Server. In the second step, you have to use created http instance and call http.createServer() method to create server instance and then bind it at port 8081 using listen method associated with server instance. Pass it a function with request and response parameters
3. **Read request and return response:** Server created in the second step will read HTTP request made by client which can be a browser or console and return the response.

**ckk02.js**

//

// Program to check the server and client

//

var http = require("http");

http.createServer(function (request, response) {

// Send the HTTP header

// HTTP Status: 200 : OK

// Content Type: text/plain

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

// Send the response body as "God is great"

response.end('God is great\n');

}).listen(8081);

// Console will print the message

console.log('Server running at http://127.0.0.1:8081/');

**How to start the server:**

Go to start menu and click on the Node.js command prompt.

Now command prompt is open:

Change the directory where the ckk02.js stored, and type the following:

node ckk02.js

we will get the message as Server running at <http://127.0.0.1:8081/>

Now server is started.

**Make a request to Node.js server:**

Open http://127.0.0.1:8081/ in any browser. We will get the message “God is great” in the browser window.

# Node.js Console

The Node.js console module provides a simple debugging console similar to JavaScript console mechanism provided by web browsers.

There are three console methods that are used to write any node.js stream:

1. console.log() : **(ckk01.js)**function is used to display simple message on console.
2. console.error() : **(ckk03.js)** function is used to render error message on console.
3. console.warn() : **(ckk04.js)** function is used to display warning message on console.

# Node.js REPL

The term REPL stands for **Read Eval Print** and **Loop**. It specifies a computer environment like a window console or a Unix/Linux shell where you can enter the commands and the system responds with an output in an interactive mode.

## REPL Environment

The Node.js or node come bundled with REPL environment. Each part of the REPL environment has a specific work.

**Read:** It reads user's input; parse the input into JavaScript data-structure and stores in memory.

**Eval:** It takes and evaluates the data structure.

**Print:** It prints the result.

**Loop:** It loops the above command until user press ctrl-c twice.

## How to start REPL

You can start REPL by simply running "node" on the command prompt.

You can execute various mathematical operations on REPL Node.js command prompt:

## Node.js Simple expressions

After starting REPL node command prompt put any mathematical expression:

Example: >10+20-5

25

Example2: >10+12 + (5\*4)/7

## Using variable

Variables are used to store values and print later. If you don't use **var** keyword then value is stored in the variable and printed whereas if **var** keyword is used then value is stored but not printed. You can print variables using console.log().

## Node.js Multiline expressions

Node REPL supports multiline expressions like JavaScript. See the following do-while loop example:

var x = 0

undefined

> do {

... x++;

... console.log("x: " + x);

... } while ( x < 10 );

## Node.js Underscore Variable

You can also use underscore \_ to get the last result.

## Node.js REPL Commands

|  |  |
| --- | --- |
| **Commands** | **Description** |
| ctrl + c | It is used to terminate the current command. |
| ctrl + c twice | It terminates the node repl. |
| ctrl + d | It terminates the node repl. |
| up/down keys | It is used to see command history and modify previous commands. |
| tab keys | It specifies the list of current command. |
| .help | It specifies the list of all commands. |
| .break | It is used to exit from multi-line expressions. |
| .clear | It is used to exit from multi-line expressions. |
| .save filename | It saves current node repl session to a file. |
| .load filename | It is used to load file content in current node repl session. |

## Node.js Exit REPL

Use ctrl + c command twice to come out of Node.js REPL.

# Node.js Package Manager

Node Package Manager provides two main functionalities:

1. It provides online repositories for node.js packages/modules which are searchable on search.nodejs.org
2. It also provides command line utility to install Node.js packages, do version management and dependency management of Node.js packages.

The npm comes bundled with Node.js installables in versions after that v0.6.3. You can check the version by opening Node.js command prompt and typing the following command:

npm  version

## Installing Modules using npm

Following is the syntax to install any Node.js module:

npm install <Module Name>

Open the Node.js command prompt and execute the following command:

npm install express

## Global vs Local Installation

By default, npm installs dependency in local mode. Here local mode specifies the folder where Node application is present. For example if you installed express module, it created node\_modules directory in the current directory where it installed express module.

You can use npm ls command to list down all the locally installed modules. Open the Node.js command prompt and execute "npm ls":

Globally installed packages/dependencies are stored in system directory. Let's install express module using global installation. Although it will also produce the same result but modules will be installed globally. Open Node.js command prompt and execute the following code:

npm install express -g

## Uninstalling a Module

To uninstall a Node.js module, use the following command:

npm uninstall express

## Searching a Module

"npm search express" command is used to search express or module.

# Node.js Global Objects

Node.js global objects are global in nature and available in all modules. You don't need to include these objects in your application; rather they can be used directly. These objects are modules, functions, strings and object etc. Some of these objects aren't actually in the global scope but in the module scope. A list of Node.js global objects are given below:

1. \_\_dirname: It is a string. It specifies the name of the directory that currently contains the code. (**ckk06.js**)
2. \_\_filename: It specifies the filename of the code being executed. This is the resolved absolute path of this code file.
3. Console
4. Process
5. Buffer
6. setImmediate(callback[, arg][, ...])
7. setInterval(callback, delay[, arg][, ...])
8. setTimeout(callback, delay[, arg][, ...])
9. clearImmediate(immediateObject)
10. clearInterval(intervalObject)
11. clearTimeout(timeoutObject)

# Express.js

Express.js is a web framework for Node.js. It is a fast, robust and asynchronous in nature. Express is a fast, assertive, essential and moderate web framework of Node.js. You can assume express as a layer built on the top of the Node.js that helps manage a server and routes. It provides a robust set of features to develop web and mobile applications.

**Features of Express framework:**

1. It can be used to design single-page, multi-page and hybrid web applications.
2. It allows to setup middlewares to respond to HTTP Requests.
3. It defines a routing table which is used to perform different actions based on HTTP method and URL.
4. It allows to dynamically render HTML Pages based on passing arguments to templates.

## Why use Express

1. Ultra fast I/O
2. Asynchronous and single threaded
3. MVC like structure
4. Robust API makes routing easy

Firstly, you have to install the express framework globally to create web application using Node terminal. Use the following command to install express framework globally.

npm install -g express  or

npm install express --save

The above command install express in node\_module directory and create a directory named express inside the node\_module. You should install some other important modules along with express. Following is the list:

1. **body-parser:** This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.

npm install body-parser --save

1. **cookie-parser:** It is used to parse Cookie header and populate req.cookies with an object keyed by the cookie names.

npm install cookie-parser --save

1. **multer:** This is a node.js middleware for handling multipart/form-data.

npm install multer --save

**ckk05.js**

var express = require('express');

var app = express();

app.get('/',function(req,res)

{

res.send('God is great, this is my first nodejs express program');

});

var server=app.listen(8000,function()

{

var host=server.address().address;

var port=server.address().port;

console.log('server is listing at http://%s:%s',host,port);

});

Run on Node.js command prompt as “node ckk05.js” then

Open http://127.0.0.1:8000/ in your browser to see the result.

# Express.js Request Object

Express.js Request and Response objects are the parameters of the callback function which is used in Express applications. The express.js request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.

**Syntax:**

app.get('/', function (req, res) {

   // --

})

## Express.js Request Object Properties

The following table specifies some of the properties associated with request object.

|  |  |  |
| --- | --- | --- |
| **Index** | **Properties** | **Description** |
| 1. | req.app | This is used to hold a reference to the instance of the express application that is using the middleware. |
| 2. | req.hostname | It contains the hostname from the "host" http header. |
| 3. | req.ip | It specifies the remote IP address of the request. |
| 4. | req.ips | When the trust proxy setting is true, this property contains an array of IP addresses specified in the ?x-forwarded-for? request header. |
| 5. | req.params | An object containing properties mapped to the named route ?parameters?. For example, if you have the route /user/:name, then the "name" property is available as req.params.name. This object defaults to {}. |
| 6. | req.path | It contains the path part of the request URL. |
| 7. | req.protocol | The request protocol string, "http" or "https" when requested with TLS. |
| 8. | req.query | An object containing a property for each query string parameter in the route. |

## Request Object Methods

Following is a list of some generally used request object methods:

## req.accepts (types)

This method is used to check whether the specified content types are acceptable, based on the request's Accept HTTP header field.

**Examples:**

req.accepts('html');

//=>?html?

req.accepts('text/html');

// => ?text/html?

## req.get(field)

This method returns the specified HTTP request header field.

**Examples:**

req.get('Content-Type');

// => "text/plain"

req.get('content-type');

// => "text/plain"

req.get('Something');

// => undefined

## req.is(type)

This method returns true if the incoming request's "Content-Type" HTTP header field matches the MIME type specified by the type parameter.

**Examples:**

// With Content-Type: text/html; charset=utf-8

req.is('html');

req.is('text/html');

req.is('text/\*');

// => true

## req.param(name [, defaultValue])

This method is used to fetch the value of param name when present.

**Examples:**

// ?name=sasha

req.param('name')

// => "sasha"

// POST name=sasha

req.param('name')

// => "sasha"

// /user/sasha for /user/:name

req.param('name')

// => "sasha"

# Express.js Response Object

The Response object (res) specifies the HTTP response which is sent by an Express app when it gets an HTTP request.

1. It sends response back to the client browser.
2. It facilitates you to put new cookies value and that will write to the client browser (under cross domain rule).
3. Once you res.send() or res.redirect() or res.render(), you cannot do it again, otherwise, there will be uncaught error.

## Response Object Properties

Let's see some properties of response object.

|  |  |  |
| --- | --- | --- |
| **Index** | **Properties** | **Description** |
| 1. | res.app | It holds a reference to the instance of the express application that is using the middleware. |
| 2. | res.headersSent | It is a Boolean property that indicates if the app sent HTTP headers for the response. |
| 3. | res.locals | It specifies an object that contains response local variables scoped to the request |

## Response Object Methods

Following are some methods:

**res.append(field [, value])**

This method appends the specified value to the HTTP response header field. That means if the specified value is not appropriate then this method redress that.

**Examples:**

res.append('Link', ['<http://localhost/>', '<http://localhost:3000/>']);

res.append('Warning', '199 Miscellaneous warning');

**res.attachment([filename])**

This method facilitates you to send a file as an attachment in the HTTP response.

**Examples:**

res.attachment('path/to/js\_pic.png');

**res.cookie(name, value [, options])**

This method is used to set a cookie name to value. The value can be a string or object converted to JSON.

**Examples:**

res.cookie('name', 'Aryan', { domain: '.xyz.com', path: '/admin', secure: true });

res.cookie('Section', { Names: [Aryan,Sushil,Priyanka] });

res.cookie('Cart', { items: [1,2,3] }, { maxAge: 900000 });

**res.clearCookie(name [, options])**

As the name specifies, the clearCookie method is used to clear the cookie specified by name.

**Examples:**

**To set a cookie**

res.cookie('name', 'Aryan', { path: '/admin' });

**To clear a cookie:**

res.clearCookie('name', { path: '/admin' });

**res.download(path [, filename] [, fn])**

This method transfers the file at path as an "attachment" and enforces the browser to prompt user for download.

**Example:**

res.download('/report-12345.pdf');

**res.end([data] [, encoding])**

This method is used to end the response process.

**Example:**

res.end();

res.status(404).end();

**res.format(object)**

This method performs content negotiation on the Accept HTTP header on the request object, when present.

**res.get(field)**

This method provides HTTP response header specified by field.

**Example:**

res.get('Content-Type');

**res.json([body])**

This method returns the response in JSON format.

**Example:**

res.json(null)

res.json({ name: 'ajeet' })

**res.links(links)**

This method populates the response?s Link HTTP header field by joining the links provided as properties of the parameter.

**Examples:**

res.links({

  next: 'http://api.rnd.com/users?page=5',

  last: 'http://api.rnd.com/users?page=10'

});

**res.location(path)**

This method is used to set the response location HTTP header field based on the specified path parameter.

**Examples:**

res.location('http://xyz.com');

**res.redirect([status,] path)**

This method redirects to the URL derived from the specified path, with specified HTTP status

**Examples:**

res.redirect('http://example.com');

**res.render(view [, locals] [, callback])**

This method renders a view and sends the rendered HTML string to the client.

**Examples:**

// send the rendered view to the client

res.render('index');

// pass a local variable to the view

res.render('user', { name: 'aryan' }, function(err, html) {

  // ...

});

**res.send([body])**

This method is used to send HTTP response.

**Examples:**

res.send(new Buffer('whoop'));

res.send({ some: 'json' });

res.send('

.....some html

');

**res.sendFile(path [, options] [, fn])**

This method is used to transfer the file at the given path. It sets the Content-Type response HTTP header field based on the filename's extension.

**Examples:**

res.sendFile(fileName, options, function (err) {

  // ...

});

# Express.js GET Request

GET and POST both are two common HTTP requests used for building REST API's. GET requests are used to send only limited amount of data because data is sent into header while POST requests are used to send large amount of data because data is sent in the body. Express.js facilitates you to handle GET and POST requests using the instance of express.

## Express.js GET Method (**Fetch data in JSON format:)**

Get method facilitates you to send only limited amount of data because data is sent in the header. It is not secure because data is visible in URL bar.

Let's take an example to demonstrate GET method.

**File: 1.html**

**<html>**

**<body>**

**<form action="http://127.0.0.1:8000/process\_get" method="GET">**

**First Name: <input type="text" name="first\_name"> <br>**

**Last Name: <input type="text" name="last\_name">**

**<input type="submit" value="Submit">**

**</form>**

**</body>**

**</html>**

**File: ckk07.js**

var express = require('express');

var app = express();

app.use(express.static('public'));

//app.get('/1.html', function (req, res) {

//res.sendFile( \_\_dirname + "/" + "1.html" );

//})

app.get('/process\_get', function (req, res)

{

response = {

first\_name:req.query.first\_name,

last\_name:req.query.last\_name

};

console.log(response);

res.end(JSON.stringify(response));

})

var server = app.listen(8000, function ()

{

var host = server.address().address

var port = server.address().port

console.log("Example app listening at http://%s:%s", host, port)

})

Start the server with the above ckk07.js file and run 1.html in any browser. You will get the response on the node js command prompt and on the client browser.

**Fetch data in paragraph format**

**File: 2.html**

**<html>**

**<body>**

**<form action="http://127.0.0.1:8000/get\_example2" method="GET">**

**First Name: <input type="text" name="first\_name"/> <br/>**

**Last Name: <input type="text" name="last\_name"/><br/>**

**<input type="submit" value="Submit"/>**

**</form>**

**</body>**

**</html>**

**File: ckk08.js**

var express = require('express');

var app=express();

app.get('/get\_example2', function (req, res)

{

res.send('<p>Username: ' + req.query['first\_name']+'</p><p>Lastname: '+req.query['last\_name']+'</p>');

})

var server = app.listen(8000, function ()

{

var host = server.address().address

var port = server.address().port

console.log("Example app listening at http://%s:%s", host, port)

})

File: 3.html

<!DOCTYPE html>

<html>

<body>

<form action="http://127.0.0.1:8000/ get\_example3">

<table>

<tr><td>Enter First Name:</td><td><input type="text" name="firstname"/><td></tr>

<tr><td>Enter Last Name:</td><td><input type="text" name="lastname"/><td></tr>

## <tr><td>Enter Password:</td><td><input type="password" name="password"/></td></tr>

## <tr><td>Sex:</td><td>

## <input type="radio" name="sex" value="male"> Male

## <input type="radio" name="sex" value="female">Female

## </td></tr>

## <tr><td>About You :</td><td>

## <textarea rows="5" cols="40" name="aboutyou" placeholder="Write about yourself">

## </textarea>

## </td></tr>

## <tr><td colspan="2"><input type="submit" value="register"/></td></tr>

## </table>

## </form>

## </body>

## </html>

## File: ckk09.js

var express = require('express');

var app=express();

app.get('/get\_example3', function (req, res)

{

res.send('<p>Firstname: ' + req.query['firstname']+'</p><p>Lastname: '+req.query['lastname']+'</p><p>Password: '+req.query['password']+'</p><p>AboutYou: '+req.query['aboutyou']+'</p>');

})

var server = app.listen(8000, function ()

{

var host = server.address().address

var port = server.address().port

console.log("Example app listening at http://%s:%s", host, port)

})

# Express.js POST Request

GET and POST both are two common HTTP requests used for building REST API's. POST requests are used to send large amount of data. Express.js facilitates you to handle GET and POST requests using the instance of express.

## Express.js POST Method

Post method facilitates you to send large amount of data because data is send in the body. Post method is secure because data is not visible in URL bar but it is not used as popularly as GET method. On the other hand GET method is more efficient and used more than POST.

**File: 4.html**

<html>

<body>

<form action="http://127.0.0.1:8000/process\_post" method="POST">

First Name: <input type="text" name="first\_name"> <br>

Last Name: <input type="text" name="last\_name">

<input type="submit" value="Submit">

</form>

</body>

</html>

## File: ckk10.js

var express = require('express');

var app = express();

var bodyParser = require('body-parser');

// Create application/x-www-form-urlencoded parser

var urlencodedParser = bodyParser.urlencoded({ extended: false })

//app.use(express.static('public'));

//app.get('/4.html', function (req, res) {

// res.sendFile( \_\_dirname + "/" + "4.html " );

//})

app.post('/process\_post', urlencodedParser, function (req, res) {

// Prepare output in JSON format

response = {

first\_name:req.body.first\_name,

last\_name:req.body.last\_name

};

console.log(response);

res.end(JSON.stringify(response));

})

var server = app.listen(8000, function () {

var host = server.address().address

var port = server.address().port

console.log("Example app listening at http://%s:%s", host, port)

})