****

**Node.js Start-Up Guide**

By: Siddharth Kumar

**Table of Content**

[**Background** 3](#_Toc521507690)

[**What is Node.js?** 3](#_Toc521507691)

[**Features of Node.js** 4](#_Toc521507692)

[**Node.js Installation** 11](#_Toc521507693)

[**First Node.js Application** 16](#_Toc521507694)

[**Node JS Platform Components** 17](#_Toc521507695)

[Node CLI 18](#_Toc521507696)

[NPM (Node Package Manager) 18](#_Toc521507697)

[package.json file 20](#_Toc521507698)

[Node Modules 24](#_Toc521507699)

[Development Tools and Frameworks 27](#_Toc521507700)

[**Node.js Project Structure** 27](#_Toc521507701)

[**Node.js REPL** 29](#_Toc521507702)

[**Node.js Architecture** 30](#_Toc521507703)

# **Background**

Companies operation backbone is application.

**What is an Application:**

An applications are typically designed to interface or integrate with different system and software used within the organization to fulfill the business goal.

**Types of an Application:**

* **Web Application**
* **Enterprise Application**

**What is a Web Application? -** A web application or web app is a client–server computer program that utilizes web browsers and web technology to perform tasks over the Internet, which the client runs in a web browser. Web application runs on Web Server.

**What is an Enterprise Application? -** An enterprise application is a large software system platform designed to operate in a corporate environment that a business would use to assist the organization in solving enterprise goal. Enterprise Application runs on App Server.

**Tools & Technology used:**

**Web Application (Client Side Application) –** bootstrap, Angular.js, etc

**Web Application (Server Side Applications) -** PHP, Node.js, etc

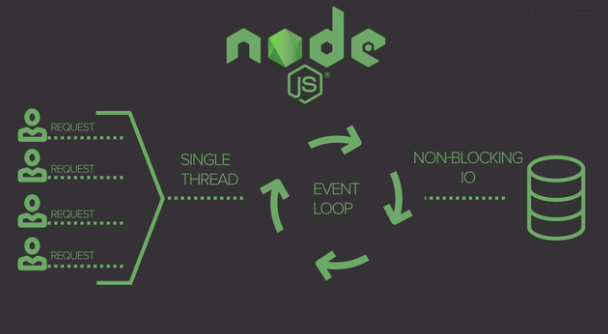
# **What is Node.js?**

Node.js is an open-source, cross-platform runtime environment used for development of server-side web and networking applications. Node.js applications are written in JavaScript and can be run on a wide variety of operating systems.

Node.js is based on an event-driven single threaded architecture and a non-blocking Input/Output API that is designed to optimize an application's throughput and scalability for real-time web applications. Node.js uses Google Java Script V8 engine to execute code.

Over a long period of time, the framework available for web development were all based on a stateless model. A stateless model is where the data generated in one session (such as information about user settings and events that occurred) is not maintained for usage in the next session with that user.

A lot of work had to be done to maintain the session information between requests for a user. But with Node.js there is finally a way for web applications to have a real-time, two-way connections, where both the client and server can initiate communication, allowing them to exchange data freely.



**Why use Node.js?**

Over the years, most of the applications were based on a stateless request-response framework. In these sort of applications, it is up to the developer to ensure the right code was put in place to ensure the state of web session was maintained while the user was working with the system.

But with Node.js web applications, you can now work in real-time and have a 2-way communication. The state is maintained, and the either the client or server can start the communication.

# **Features of Node.js**

**The key features of Node.js**

1. **Modularity**

Major advantage of Node JS Platform is that it’s modularity. Each and every functionality is divided and implemented as a separate module or package. When we install Node JS Platform, by default it installs only few modules. If our application requires other modules, then we can easily install and configure then at any point of our application development phases.

Node JS has thousands of thousands modules. Some modules were developed by Node JS Community and some were by Third-party Clients. Some of the Node JS Modules are written in Java Script and some are in C/C++ Language. It’s 40% modules are written in JS and 60% modules are written C/C++.

1. **Express JS**

Node JS is used to develop Server-side Java Script. It also contains a separate module for Web Application Framework i.e. Express JS.

1. **Non-blocking or Asynchronous IO**

Node JS supports Non-blocking IO i.e. it uses Asynchronous IO Model to interact with File system or to do Socket communication or network communication.

Asynchronous IO Model means if IO processing is taking more time, then it permits other processing to continue before the transmission has finished.

Java Developers are very much familiar about Blocking IO and Non-Blocking IO. Java IO package follows Blocking IO or Synchronous IO Architecture whereas Java NIO 2 follows Non-Blocking IO or Asynchronous IO Architecture.

1. **Event-Driven Asynchronous Platform**

Node JS Platform follows Even-Driven Loop architecture to interact or handle requests. We will discuss in details about “Even-Drive Loop” in coming posts.

1. **MongoDB Wrappers API**

Node JS platform contains a separate module to integrate MongoDB No SQL database with applications. It provides a MongoDB wrapper API.

We can use this API to write JavaScript easily to interact with MongoDB database. Node JS-mongodb module uses Mongoose to interact with MongoDB database.

1. **Redis Client Library API**

Node JS platform contains a separate module to integrate Redis No SQL database with applications. It provides a Redis wrapper API. We can use this API to write JavaScript easily to interact with Redis database.

1. **Jade Template Engine**

Node JS platform supports many template engines to write HTML. Default template engine supported by Node JS is “Jade”.  
Jade is a whitespace-sensitive template engine for developing HTML applications very easily.

We can write template pages once and reuse them very easily to reduce development time. We will discuss how to install and write Jade templates in a separate post.

1. **Web Server**

We can develop and use HTTP Web Server within no time. We need to use http package to implement Web Server. We will discuss one post on how to install http package and develop a Web Server.

1. **Better Socket API**

Node JS Platform provides very good Socket Module API to develop Real-time, Multi-User Chat and Multi-Player Gaming Applications very easily. It supports Unix Socket programming like pipe().

**Advantages of Node JS**

1. **One language and One Data Format**

We need to use one and only one language to write whole applications i.e. Java Script. We need to code all layers from UI to Controller to Business Layer to Persistence Layer (From Front-End to Back-End) in Java Script only.

All layers from Front-End to Back-End support same Data Format i.e. JSON (Java Script Object Notation). UI Layer Angular JS Supports JSON Format, Server-Side Scripts written in Express JS supports JSON Format and Back-end MongoDB NO SQL supports same JSON Format.

1. **Open Source**

Node JS is Open Source Module. It provides many modules for free to use in our applications.

1. **Highly Scalable**

To improve our application performance, we will scale our infrastructure. There are two ways to scale any application:

* Horizontal Scaling
* Vertical Scaling

Vertical Scalability means adding more resources to a single node. Horizontal Scalability means adding more nodes to a system.

Node JS uses Horizontal Scaling instead of Vertical Scaling to our applications. It’s consistent with today’s Cloud Technology trend.

1. **Better Performance and Low Latency**

I/O operations often blocks our application because it can take a long time to read a big file, or make a large query against a database, or wait for any stream of data to fully transmit.

As Node JS follows Non-Blocking IO architecture and also uses only JavaScript to develop entire application, we can achieve better performance and low latency.

If we develop our UI and Data Intensive applications by using **MEAN Stack**, we can achieve very high performance. **MEAN Stands for MongoDB, Express JS, Angular JS and Node JS**.

1. **Caching Modules**

Node JS Modules are cached once they are loaded into application for first time. Multiple calls to load a module may not cause the module code to be executed multiple times.

1. **Less Problems with**[**Concurrency**](https://www.journaldev.com/1162/java-multithreading-concurrency-interview-questions-answers)

Node JS does not follow Multi-Thread architecture. It follows Single-Thread with Event Loop Architecture. It serves any number of requests without any issues.  
Node JS does NOT need to handle pooling of requests to a set of threads. That’s why it has less overhead to handle multiple requests concurrently.

1. **Easy to Extend and Lightweight**

We can install only required modules and use them to develop applications. Whenever we need to add new feature or module, we can easily extend it.  
By following this approach, we can also achieve very light-weight nature into our applications.

1. **Faster Development and Easy to Maintain**

As Node JS Platform had already provided support for many modules, we can use them and develop wide variety of applications within no time. We can develop and maintain Node JS applications very easily.

1. **REST API**

Node JS Platform supports developing RESTful Web Services API very easily.

1. **Many Development Frameworks and Tools**

We can get many development tools like IDE and UI Frameworks to develop Node JS applications very easily.

1. **Active Development Community**

Node JS Development Community is very active in adding new modules or packages to existing modules to support wide variety of applications development.

1. **Unit Testing**

In Node JS applications, we need to write whole code in Java Script. Today we have many Java Script Unit Testing frameworks and tools like Jasmin Framework.

1. **Streaming Data**

We can easily develop Streaming-based data applications by using Node JS.

1. **Creating Servers**

Node JS Platform has some built-in API to develop or create HTTP Server, DNS Server, TCP Server etc very easily or quickly.

1. **It can handle thousands of concurrent connections with minimal overhead (CPU/Memory) on a single process**
2. **Easy Module Loading process**

It is very easy to load required Node JS Modules into Node JS Applications. We just need to use require() statement in our application. It is similar to “import” statement in Java Programming language. We will discuss about require() in detail with some examples in coming posts.

**Limitations of Node JS**

* It does NOT support Multi-threading programming.
* It does support for Computational Intensive Tasks. Node JS struggles in handling of very high computational intensive tasks, because whenever it does something long running task, it will queue all remaining incoming requests, because it follows Single-Thread Architecture with Event Loop.
* Don’t use Node JS for Blocking/Synchronous and CPU-intensive tasks.
* Unstable API – Node JS is still in Beta stage and most of its modules are in unstable state. It’s not ready for Production or Live Systems.

**Who uses Node.js**

**Node.js is used by a variety of large companies. Below is a list of a few of them.**

* Paypal – A lot of sites within Paypal have also started the transition onto Node.js.
* LinkedIn - LinkedIn is using Node.js to power their[Mobile](https://www.guru99.com/mobile-testing.html)Servers, which powers the iPhone, Android, and Mobile Web products.
* Mozilla has implemented Node.js to support browser APIs which has half a billion installs.
* Ebay hosts their HTTP API service in Node.js
* Netflix uses Node.js for streaming service on variety of devices.

**When to Use Node.js**

**Node.js is best for usage in streaming or event-based real-time applications like**

1. Chat applications
2. Game servers – Fast and high-performance servers that need to processes thousands of requests at a time, then this is an ideal framework.
3. Good for collaborative environment – This is good for environments which manage document. In document management environment you will have multiple people who post their documents and do constant changes by checking out and checking in documents. So Node.js is good for these environments because the event loop in Node.js can be triggered whenever documents are changed in a document managed environment.
4. Advertisement servers – Again here you could have thousands of request to pull advertisements from the central server and Node.js can be an ideal framework to handle this.
5. Streaming servers – Another ideal scenario to use Node is for multimedia streaming servers wherein clients have requests to pull different multimedia contents from this server.

Node.js is good when you need high levels of concurrency but less amount of dedicated CPU time.

Best of all, since Node.js is built on JavaScript, it's best suited when you build client-side applications which are based on the same JavaScript framework.

**When to not use Node.js**

Node.js can be used for a lot of applications with various purpose, the only scenario where it should not be used is if there are long processing times which is required by the application.

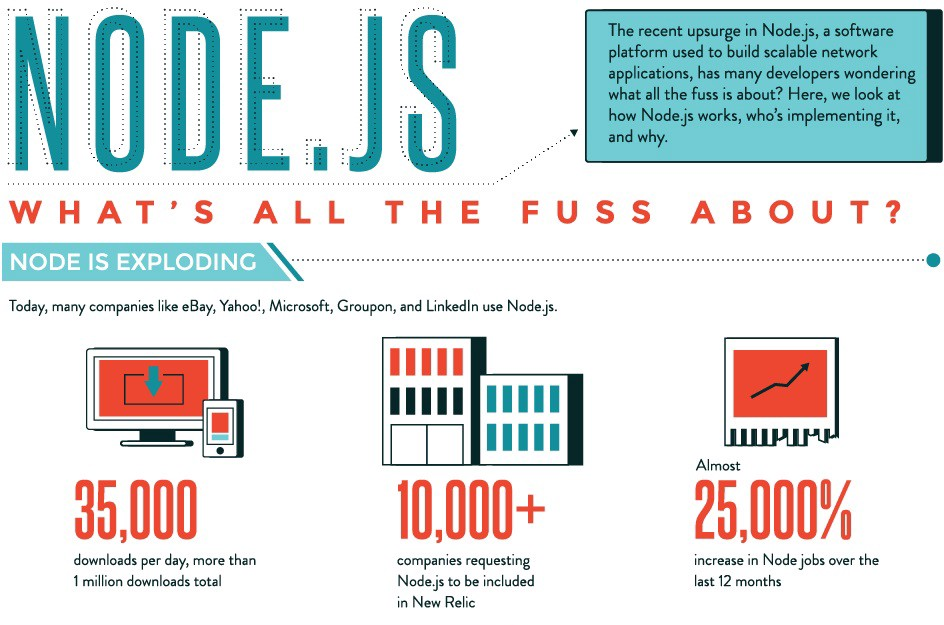
Node is structured to be single threaded. If any application is required to carry out some long running calculations in the background. So if the server is doing some calculation, it won't be able to process any other requests. So, Node.js is best when processing needs less dedicated CPU time.

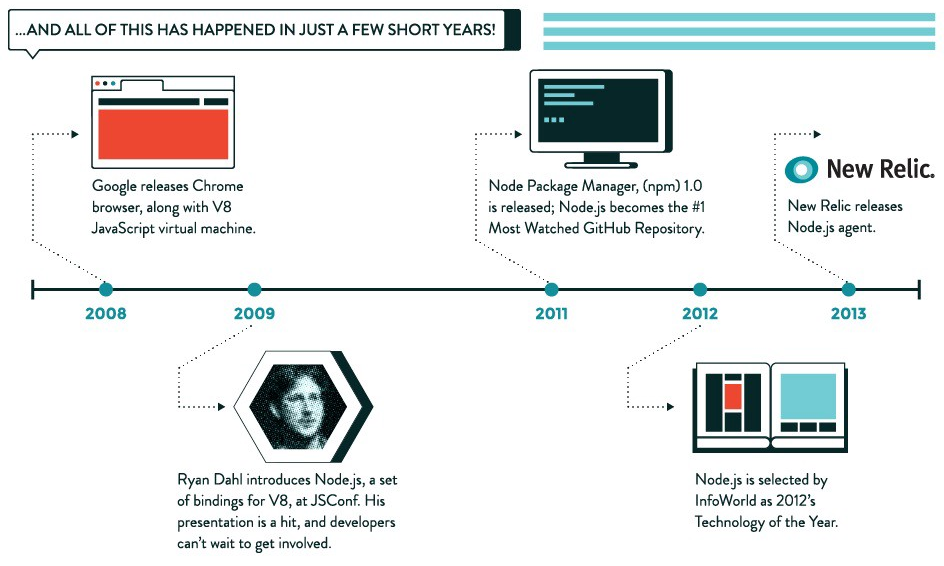
**Node JS Applications**

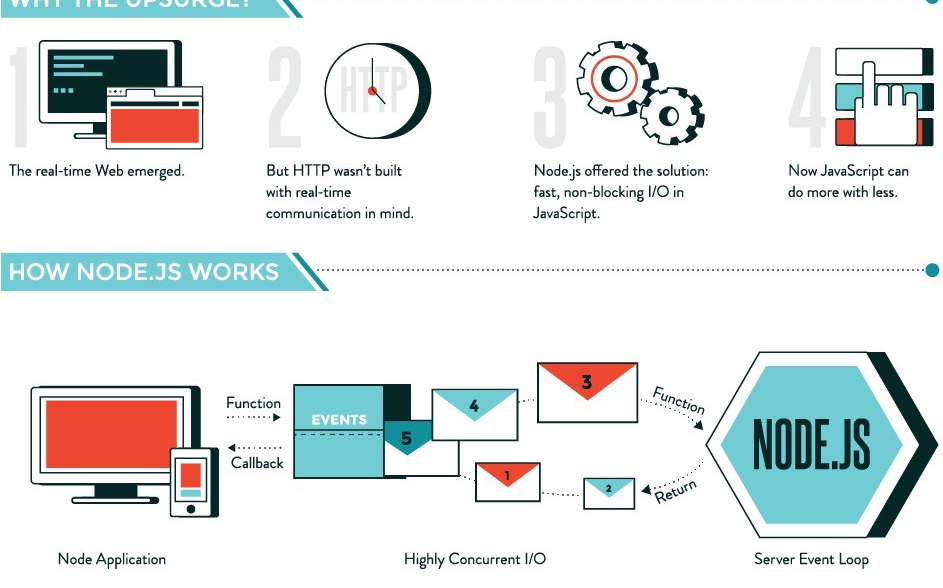
We can use Node JS Platform to develop the following kind of applications.

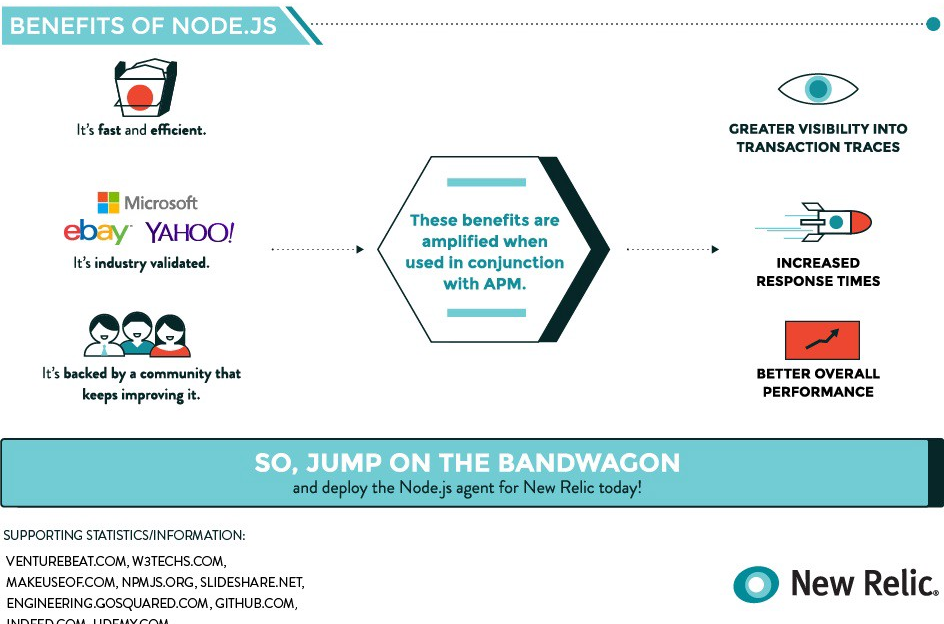
* Network applications
* Asynchronous Events
* Intelligent Networking Proxies
* Proxy Server
* Multiplayer Game Applications
* Data Intensive Real-time applications
* Web Applications
* High Definition(HD) Voice and Video Communications
* File Uploading Tools
* Process Monitoring Tools
* Many-To-Many Instant Chatting Applications
* Many-To-Many Instant Messing Applications
* HTTP Web Server
* Real-time Logistics Systems
* Streaming Server
* High Concurrency Applications
* Communication Hubs
* Coordinators
* DNS Server
* Static File Server
* TCP Server
* Stock-Trading Dashboard

**Node.js Facts by New Relic :**

****

****

****

****

# **Node.js Installation**

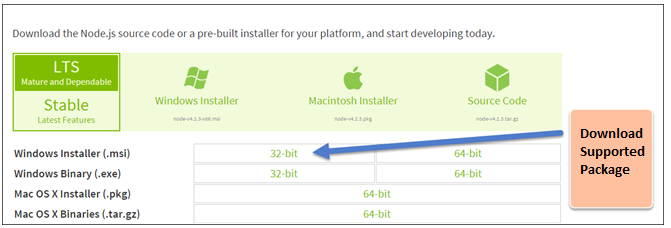
To start building your Node.js applications, the first step is the installation of the node.js framework. The Node.js framework is available for a variety of operating systems right from Windows to Ubuntu and OS X. Once the Node.js framework is installed you can start building your first Node.js applications.

Node.js also has the ability to embedded external functionality or extended functionality by making use of custom modules. An example of a module is [MongoDB](https://www.guru99.com/mongodb-tutorials.html)module which allows you to work with MongoDB databases from your Node.js application. These modules need to be installed separately using NPM (Node Package Manager).

**Installing Node.js on Windows**

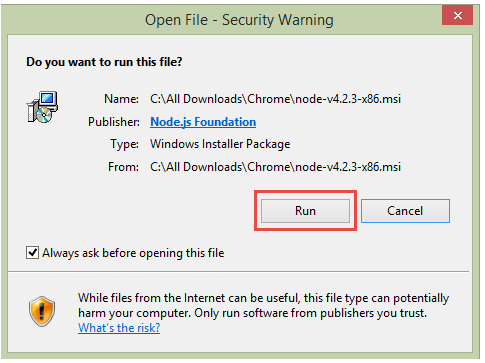
The first steps in using Node.js is the installation of the Node.js libraries on the client system. To perform the installation of Node.js, perform the below steps;

**Step 1)** Go to the site <https://nodejs.org/en/download/> and download the necessary binary files.



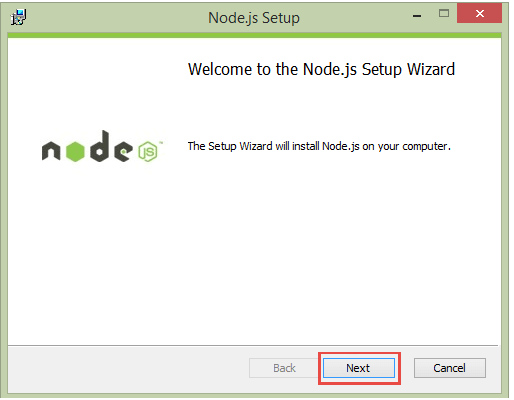
**Step 2)**

Double click on the downloaded .msi file to start the installation. Click the Run button in the first screen to begin the installation.



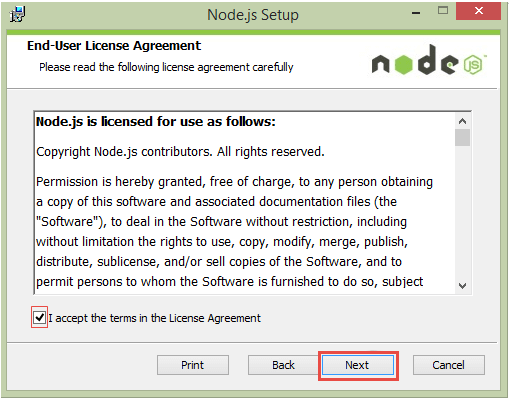
**Step 3)**

In the next screen, click the "Next" button to continue with the installation



**Step 4)**

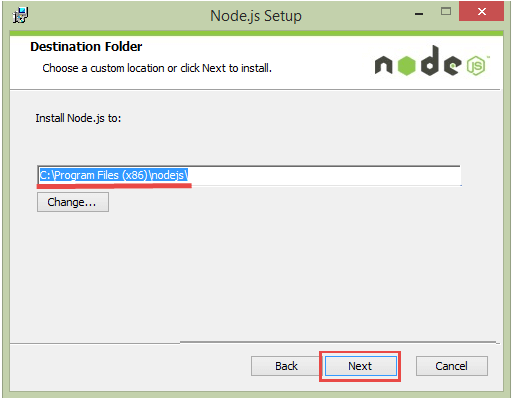
**In** the next screen Accept the license agreement and click on the Next button.



**Step 5)**

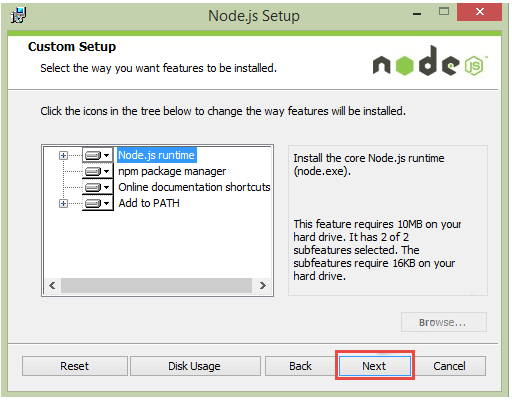
In the next screen, choose the location where Node.js needs to be installed and then click on the Next button.

1. First enter the file location for the installation of Node.js. This is where the files for Node.js will be stored after the installation.
2. Click on the Next button to proceed ahead with the installation.



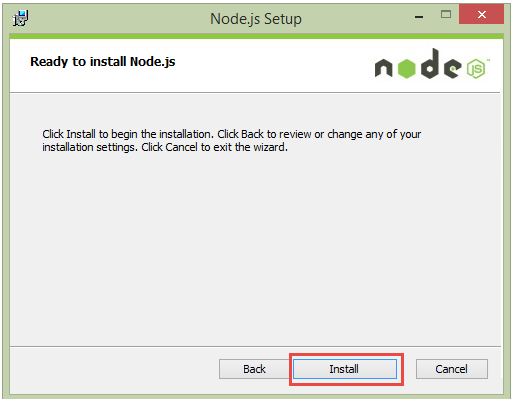
**Step 6)**

Accept the default components and click on the next button.



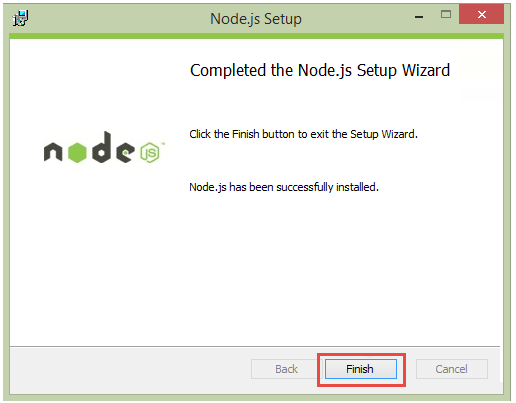
**Step 7)**

In the next screen, click the Install button to start the installation.



**Step 8)**

Click the Finish button to complete the installation.



**Note: The NPM comes default with Node.js installation. In order to install any module below command need to be executed on the command prompt or Node.js terminal.**

**npm install <package>**

**Ex: npm install mongodb**

**Note: It requires an open Internet access to install the modules. In case of any proxy, below error might occur. If so then run the following command before running install <module>.**



npm config set registry http://registry.npmjs.org/

# **First Node.js Application**

Let’s try to create an application to display "Hello World" in a web browser.

Create a new Node.js file with file name helloworld.js

Type the following code.



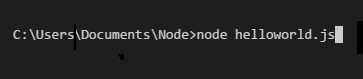
**Code Explanation:**

1. The basic functionality of the "require" function is that it reads a[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)file, executes the file, and then proceeds to return an object. Using this object, one can then use the various functionalities available in the module called by the require function. So in our case, since we want to use the functionality of http and we are using the require(http) command.
2. In this 3rd line of code, we are creating a server application which is based on a simple function. This function is called, whenever a request is made to our server application.
3. When a request is received, we are asking our function to return a "Hello World" response to the client. The writeHead function is used to send header data to the client and while the end function will close the connection to the client.
4. We are then using the server.listen function to make our server application listen to client requests on port no 8080. You can specify any available port over here.

**Executing the code**

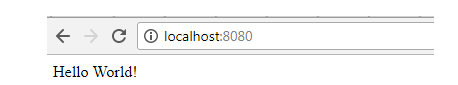
1. Save the file on your computer: C:\Users\Your Name\ helloword.js
2. In the command prompt, navigate to the folder where the file is stored. Enter the command

Node helloworld.js



1. Now, your computer works as a server! If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return!
2. Start your internet browser, and type in the address: http://localhost:8080

**Output**

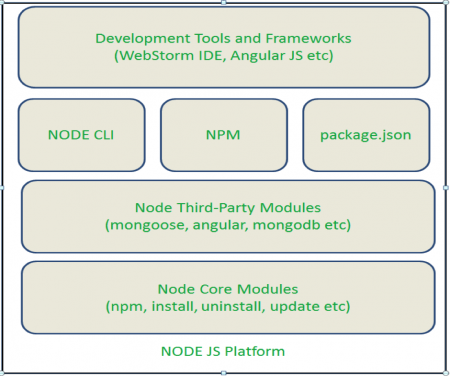


# **Node JS Platform Components**

Node JS is Platform, not a Framework. It contains many components to develop, test and deploy Enterprise applications.

**Node JS Components List:**

* [Node CLI](https://www.journaldev.com/7423/node-js-components-modules-npm-install-update-uninstall-example#node-cli)
* [NPM](https://www.journaldev.com/7423/node-js-components-modules-npm-install-update-uninstall-example#npm)
* [package.json](https://www.journaldev.com/7423/node-js-components-modules-npm-install-update-uninstall-example#package.json)
* [Node Modules](https://www.journaldev.com/7423/node-js-components-modules-npm-install-update-uninstall-example#node-modules)
* [Development Tools and Frameworks](https://www.journaldev.com/7423/node-js-components-modules-npm-install-update-uninstall-example#node.js-development-tools)



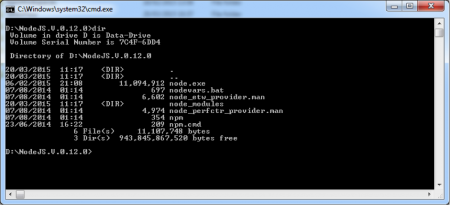
## Node CLI

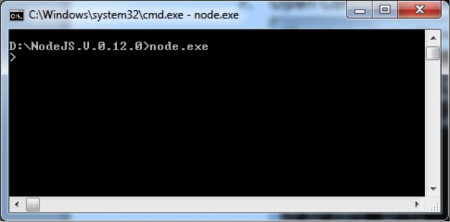
Node JS Platform has a CLI (Command Line Interface) to run basic commands and also script files.

When we install Node JS Platform, by default we will get this component. We do not need to any extra configurations for this component.

We can observe “**node.exe**” file at NODSJS\_HOME. It is also known as Node Platform CLI component.

**How to access Node CLI**:

1. Open Command prompt at NODSJS\_HOME  
   [](https://cdn.journaldev.com/wp-content/uploads/2015/03/Node.js-CLI-1.png)
2. Run “**node.exe**” command

[](https://cdn.journaldev.com/wp-content/uploads/2015/03/Node.js-CLI-2.png)

Now we are able to see NODE CLI i.e. “>”, that means our Node JS Setup is working fine.

Here we can run basic Java Script commands one by one or entire Java Script files (\*.js files) to test the basic functionality.

## NPM (Node Package Manager)

The "Node package manager" has the ability to manage modules, which are required by Node.js applications.

Let's look at some of the functions available in the node package manager for managing modules.

1. Installing packages in global mode – Modules can be installed at the global level, which just basically means that these modules would be available for all Node.js projects on a local machine. The example below shows how to install the "express module" with the global option.

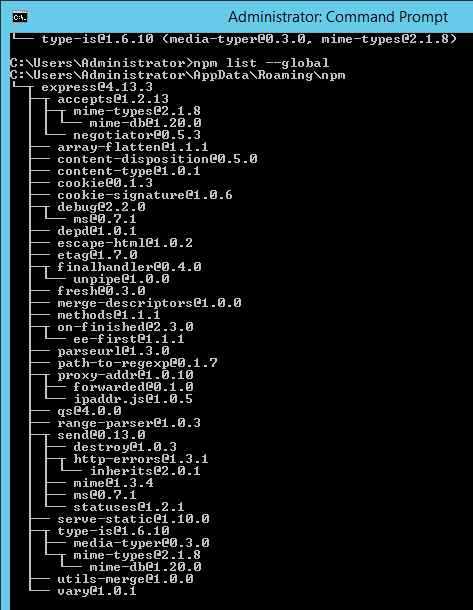
**npm install express –global**

1. The global option in the above statement is what allows the modules to be installed at a global level.
2. Listing all of the global packages installed on a local machine. This can be done by executing the below command in the command prompt

**npm list –global**

Below is the output which will be shown, if you have previously installed the "express module" on your system.

Here you can see the different modules installed on the local machine.

[](https://cdn.guru99.com/images/NodeJS/010716_0523_NodejsModul8.png)

1. Installing a specific version of a package – Sometimes there may be a requirement to install just the specific version of a package. Once you know what is the package and the relevant version that needs to be installed, you can use the npm install command to install that specific version.
2. The example below shows how to install the module called underscore with a specific version of 1.7.0

**npm install underscore@1.7.0**

1. Updating a package version – Sometimes you may have an older version of a package in a system, and you may want to update to the latest one available in the market. To do this one can use the npm update command. The example below shows how to update the underscore package to the latest version

**npm update underscore**

1. Searching for a particular package – To search whether a particular version is available on the local system or not, you can use the search command of npm. The example below will check if the express module is installed on the local machine or not.

**npm search express**

1. Un-installing a package – The same in which you can install a package, you can also un-install a package. The uninstallation of a package is done with the uninstallation command of npm. The example below shows how to uninstall the express module

**npm uninstall express**

## package.json file

The "package.json" file is used to hold the **metadata about a particular project**. This information provides the Node package manager the necessary information to understand how the project should be handled along with its dependencies.

The package.json files contains information such as the project description, the version of the project in a particular distribution, license information, and configuration data.

The package.json file is normally located at the root directory of a Node.js project. The best way to manage locally installed npm packages is to create a package.json file.

A package.json file:

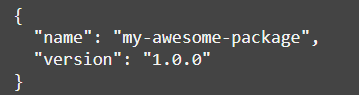
* lists the packages that your project depends on.
* allows you to specify the versions of a package that your project can use using [semantic versioning rules](https://docs.npmjs.com/getting-started/semantic-versioning).
* makes your build reproducible, and therefore much easier to share with other developers.

**Requirements**

A package.json must have:

* "name"
  + all lowercase
  + one word, no spaces
  + hyphens and underscores allowed
* "version"
  + in the form of x.x.x
  + follows [semver spec](https://docs.npmjs.com/getting-started/semantic-versioning)

For example:



**Creating a package.json**

There are two basic ways to create a package.json file.

1. Run a CLI questionnaire

To create a package.json with values that you supply, run:



This will initiate a command line questionnaire that will conclude with the creation of a package.json in the directory in which you initiated the command.

1. Create a default package.json

To get a default package.json, run npm init with the --yes or -y flag:



This method will generate a default package.json using information extracted from the current directory.



* **name:** the current directory name
* **version:** always 1.0.0
* **description:** info from the readme, or an empty string ""
* **main:** always index.js
* **scripts:** by default, creates an empty test script
* **keywords:** empty
* **author:** empty
* **license:** [ISC](https://opensource.org/licenses/ISC)
* **bugs:** info from the current directory, if present
* **homepage:** info from the current directory, if present

**Attributes of Package.json**

* name − name of the package
* version − version of the package
* description − description of the package
* homepage − homepage of the package
* author − author of the package
* contributors − name of the contributors to the package
* dependencies − list of dependencies. NPM automatically installs all the dependencies mentioned here in the node\_module folder of the package.
* repository − repository type and URL of the package
* main − entry point of the package
* keywords − keywords

**Specifying Dependencies**

Dependencies are specified in a simple object that maps a package name to a version range. The version range is a string which has one or more space-separated descriptors.

To specify the packages your project depends on, you need to list the packages you'd like to use in your package.json file. There are 2 types of packages you can list:

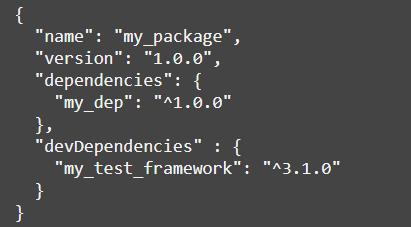
* **"dependencies"**: These packages are required by your application in production.
* **"devDependencies"**: These packages are only needed for development and testing.

**Manually editing your package.json**

You can manually edit your package.json. You'll need to create an attribute in the package object called dependencies that points to an object. This object will hold attributes that name the packages you'd like to use. It will point to a [semver](https://docs.npmjs.com/getting-started/semantic-versioning) expression that specifies the versions of that project that are compatible with your project.

If you have dependencies you only need to use during local development, follow the same instructions as above but use the attribute called devDependencies.

For example, the project below uses any version of the package my\_dep that matches major version 1 in production and requires any version of the package my\_test\_framework that matches major version 3, but only for development:



**The --save-prod and --save-dev install flags**

The easier (and more awesome) way to add dependencies to your package.json is to do so from the command line, flagging the npm install command with either --save-prod (assumed by default) or --save-dev, depending on how you'd like to use that dependency.

To add an entry to your package.json's dependencies:



To add an entry to your package.json's devDependencies:



**Managing dependency versions**

npm uses Semantic Versioning, or, as we often refer to it, SemVer, to manage versions and ranges of versions of packages.

If you have a package.json file in your directory and you run npm install, npm will look at the dependencies that are listed in that file and download the latest versions, using semantic versioning.

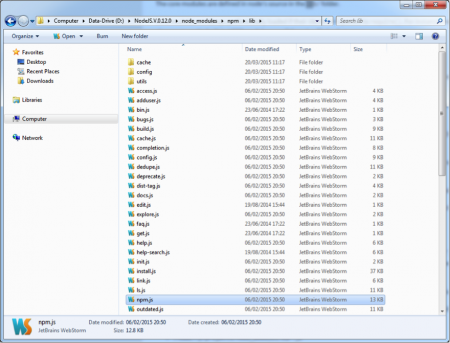
## Node Modules

**What are modules in Node.js?**

Modules in Node js are a way of encapsulating code in a separate logical unit. There are many readymade modules available in the market which can be used within Node js.

When we install Node JS Platform, by default only one module is installed i.e. npm module. We need to use “npm” command to install required modules one by one.

All Core or Default modules are installed at /lib folder as \*.js files as show below. In our case this path refers to “D:\NodeJS.V.0.12.0\node\_modules\npm\lib”.

[](https://cdn.journaldev.com/wp-content/uploads/2015/03/Node.js-npm-lib.png)

Node JS has thousands of modules, but below are some of the popular modules.

1. **Express framework** – Express is a minimal and flexible Node js web application framework that provides a robust set of features for the web and[Mobile](https://www.guru99.com/mobile-testing.html)applications.
2. **Socket.io** - Socket.IO enables real-time bidirectional event-based communication. This module is good for creation of chatting based applications.
3. **Jade** - Jade is a high-performance template engine and implemented with[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)for node and browsers.
4. **MongoDB**- The[MongoDB](https://www.guru99.com/mongodb-tutorials.html)Node.js driver is the officially supported node.js driver for MongoDB.
5. **Restify** - restify is a lightweight framework, similar to express for building REST APIs.
6. **PM2:** This tool helps you to manage Node apps processes in production

**Node.js built in modules**

Node.js has a set of built-in modules which you can use without any further installation.

Here is a list of the built-in modules of Node.js

|  |  |
| --- | --- |
| **Module** | **Description** |
| assert | Provides a set of assertion tests |
| buffer | To handle binary data |
| child\_process | To run a child process |
| cluster | To split a single Node process into multiple processes |
| crypto | To handle OpenSSL cryptographic functions |
| dgram | Provides implementation of UDP datagram sockets |
| dns | To do DNS lookups and name resolution functions |
| domain | Deprecated. To handle unhandled errors |
| events | To handle events |
| fs | To handle the file system |
| http | To make Node.js act as an HTTP server |
| https | To make Node.js act as an HTTPS server. |
| net | To create servers and clients |
| os | Provides information about the operation system |
| path | To handle file paths |
| punycode | Deprecated. A character encoding scheme |
| querystring | To handle URL query strings |
| readline | To handle readable streams one line at the time |
| stream | To handle streaming data |
| string\_decoder | To decode buffer objects into strings |
| timers | To execute a function after a given number of milliseconds |
| tls | To implement TLS and SSL protocols |
| tty | Provides classes used by a text terminal |
| url | To parse URL strings |
| util | To access utility functions |
| v8 | To access information about V8 (the JavaScript engine) |
| vm | To compile JavaScript code in a virtual machine |
| zlib | To compress or decompress files |

**Using modules in Node.js**

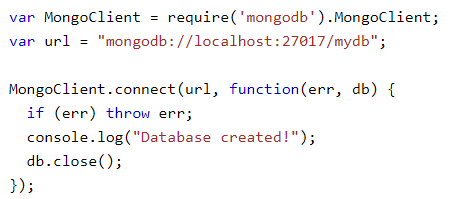
In order to use modules in a Node.js application, they first need to be installed using the Node package manager.

The below command line shows how a module "mongodb" can be installed.

**npm install mongodb**

* The above command will download the necessary files which contain the "mongodb modules" and take care of the installation as well
* Once the module has been installed, in order to use a module in a Node.js application you need to use the 'require' keyword. This keyword is a way that Node.js uses to incorporate the functionality of a module in an application.

Let's look at an example how we can use the "require" keyword. The below code example shows how to use the require function



1. In the first statement itself, we are using the "require" keyword to include the mongodb module.
2. After the module is included, in order to use the functionality within the module, an object needs to be created. Here an object of the mongodb module is created.
3. Once the module is included using the "require" command and an "object" is created, the required methods of the mongodb module can be invoked.
4. The above example creates a database in mongodb.

* Node.js has the ability to create custom modules and allows you to include those custom modules in your Node.js application.
* Also when creating modules, it is also possible to extend or inherit one module from another. This is called Extending Modules.
* One can publish their own module to their own Github repository. By publishing your module to a central location, you are then not burdened with having to install yourself on every machine that requires it.

## Development Tools and Frameworks

As Node JS Platform became very popular to develop Data-Sensitive Real-time and Network applications, many companies have developed some tools and framework to ease and reduce the overhead of Node JS applications.

|  |  |
| --- | --- |
| **CATEGORY** | **FRAMEWORK/TOOLS** |
| IDE | Eclipse with node.js plugins, Jetbrains Webstorm, Cloud9 IDE,Visual Studio Node JS Toolkit |
| Database | Mongoose for MongoDB |
| UI Build Tools | Grunt, Yeoman, Gulp |
| CLI | Node CLI, grunt-cli |
| Authentication | Passport.js |
| UI Library Management | Bower |
| UI Frameworks | Backbone.js, Angular.js, Ember.js |
| Layout Frameworks | Twitter Bootstrap Framework |
| Template Engine | Jade, EJS, Hogan.JS |
| CSS Engine | Stylus, LESS, Compass |
| Unit Testing Frameworks | Jasmin, Node Unit |

# **Node.js Project Structure**

NodeJS is not a framework like Rails or .Net. The project structure varies from developer’s adoption.

One sample project structure is as below

.  
├── config # App configuration files  
│ ├── sequalize.json # Sequalize config  
│ ├── serviceOne.json # ServiceOne config  
│ └── ... # Other configurations  
├── routes   
│ ├── controllers # Request managers  
│ ├── middlewares # Request middlewares  
│ └── routes.js # Define routes and middlewares here  
├── services # External services implementation   
│ ├── serviceOne  
│ └── serviceTwo  
│ └── ... # Other services  
├── db # Data access stuff (Sequalize mostly)  
│ ├── models # Models  
│ ├── migrations # Migrations  
│ ├── seeds # Seeds  
│ └── index.js # Sequalize instantiation  
├── core # Business logic implementation  
│ ├── accounts.js   
│ ├── sales.js   
│ ├── comments.js   
│ └── ... # Other business logic implementations  
├── utils # Util libs (formats, validation, etc)  
├── tests # Testing  
├── scripts # Standalone scripts for dev uses  
├── pm2.js # pm2 init  
├── shipitfile.js # deployment automation file  
├── package.json # project metadata   
├── README.md   
└── app.js # App starting point

**Another Project Structure**

project/

config/

routes.js

settings.js

index.js

controllers/

index.js

users.js

models/

index.js

user.js

repositories/

index.js

user.js

views/

index.jade

users/

index.jade

tests/

controllers/

index.js

models/

index.js

repositories/

index.js

index.js

package.json

# **Node.js REPL**

REPL stands for Read Eval Print Loop and it represents a computer environment like a Windows console or Unix/Linux shell where a command is entered and the system responds with an output in an interactive mode. Node.js or Node comes bundled with a REPL environment. It performs the following tasks −

* Read − Reads user's input, parses the input into JavaScript data-structure, and stores in memory.
* Eval − Takes and evaluates the data structure.
* Print − Prints the result.
* Loop − Loops the above command until the user presses ctrl-c twice.

The REPL feature of Node is very useful in experimenting with Node.js codes and to debug JavaScript codes.

**Starting REPL**

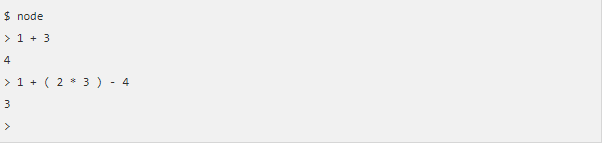
REPL can be started by simply running node on shell/console without any arguments as follows.

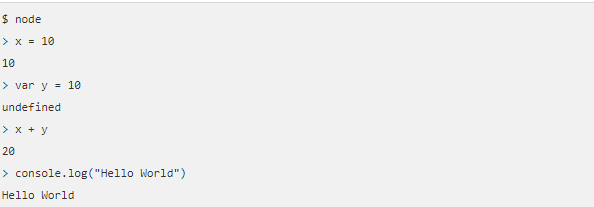


You will see the REPL Command prompt > where you can type any Node.js command –



You can write simple arithmetic operation or a use a variable.





# **Node.js Architecture**

**Node JS Single Threaded Event Loop Model**

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.

We are already familiar with “Multi-Threaded Request-Response” architecture because it’s used by most of the web application frameworks. But why Node JS Platform has chosen different architecture to develop web applications. What is the major differences between multithreaded and single threaded event loop architecture?

Any web developer can learn Node JS and develop applications very easily. However, without understanding Node JS Internals, we cannot design and develop Node JS Applications very well. So before starting developing Node JS Applications, first we will learn Node JS Platform internals.

**Node JS Platform**

Node JS Platform uses “Single Threaded Event Loop” architecture to handle multiple concurrent clients. Then how it really handles concurrent client requests without using multiple threads. What is Event Loop model? We will discuss these concepts one by one.

Before discussing “Single Threaded Event Loop” architecture, first we will go through famous “Multi-Threaded Request-Response” architecture.

**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.

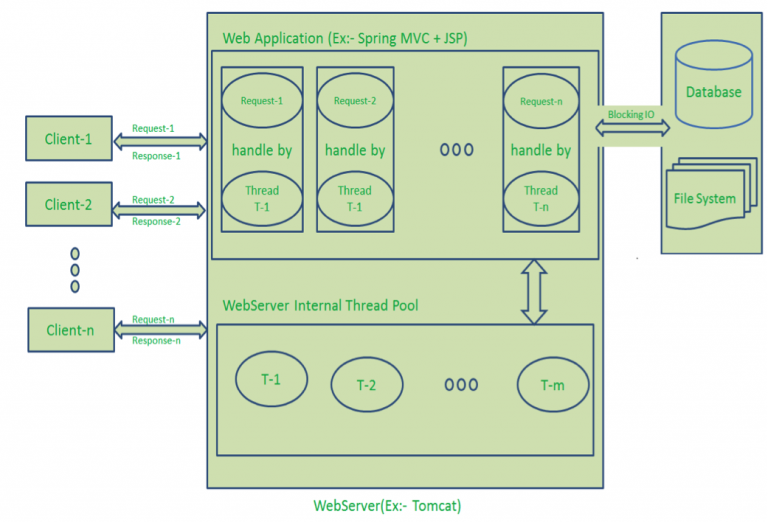
However, this model uses Multiple Threads to handle concurrent client requests. Before discussing this model internals, first go through the diagram below.

**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 client’s requests require Blocking IO Operations, then almost all threads are busy in preparing their responses. Then remaining clients Requests should wait for longer time.



**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-1 Request-2 and process it
    - Client-1 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.

You should have some good knowledge about how Javascript events and callback mechanism works. If you don’t know, Please go through those posts or tutorials first and get some idea before moving to the next step in this post.

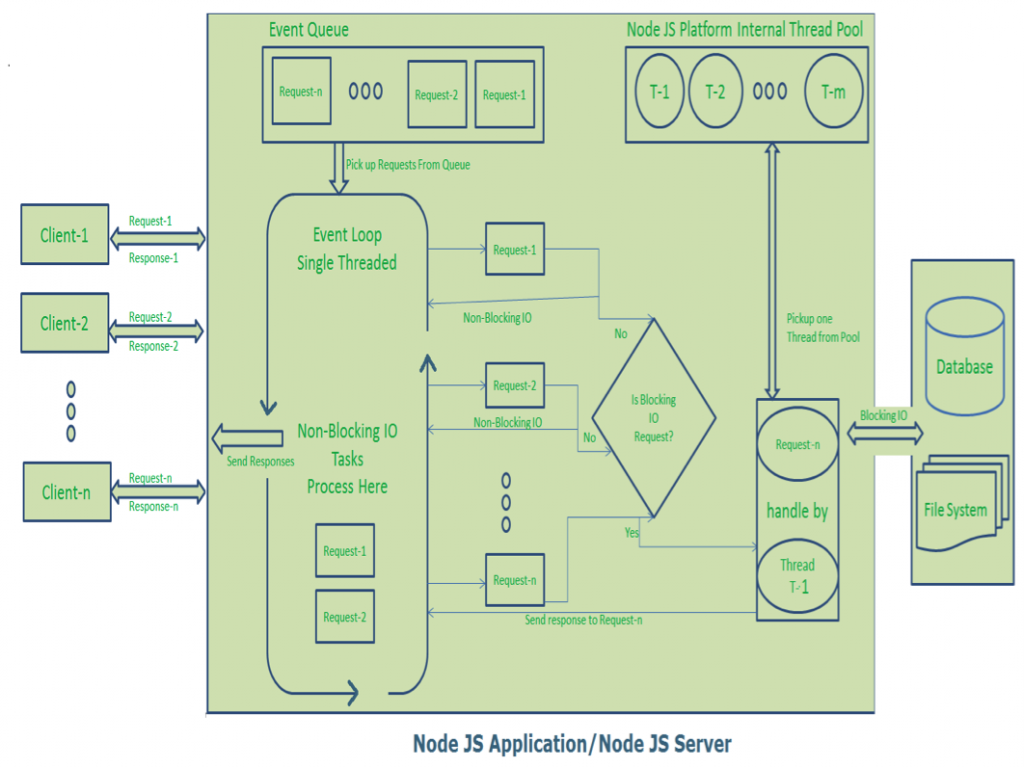
As Node JS follows this architecture, it can handle more and more concurrent client requests very easily. Before discussing this model internals, first go through the diagram below.

I tried to design this diagram to explain each and every point of Node JS Internals.

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.

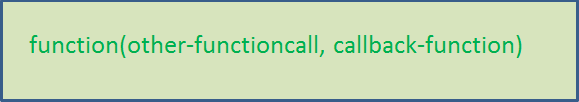
[](https://cdn.journaldev.com/wp-content/uploads/2015/04/NodeJS-Single-Thread-Event-Model.png)

**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:

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