What and Why Node.js?

Node JS is an Open Source Java Script Platform used to develop fast and scalable server-side and networking applications very easily.

Node.js is an open source server framework, which uses JavaScript on the server. **Node.js uses asynchronous programming!**

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

### Features of Node JS

* **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.

* **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.

* **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.

* **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.
* **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.

* **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.

* **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.

* **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.

* **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

* **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.

* **Open Source**

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

* **Highly Scalable**

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

* 1. Horizontal Scaling
  2. 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.

* **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**.

* **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.

* **Less Problems with Concurrency**  
  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.
* **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.
* **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.
* **REST API**  
  Node JS Platform supports developing RESTful Web Services API very easily.
* **Many Development Frameworks and Tools**  
  We can get many development tools like IDE and UI Frameworks to develop Node JS applications very easily.
* **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.
* **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.
* **Streaming Data**  
  We can easily develop Streaming-based data applications by using Node JS.
* **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.
* **It can handle thousands of concurrent connections with minimal overhead (CPU/Memory) on a single process**
* **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.

## Node JS Architecture

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.

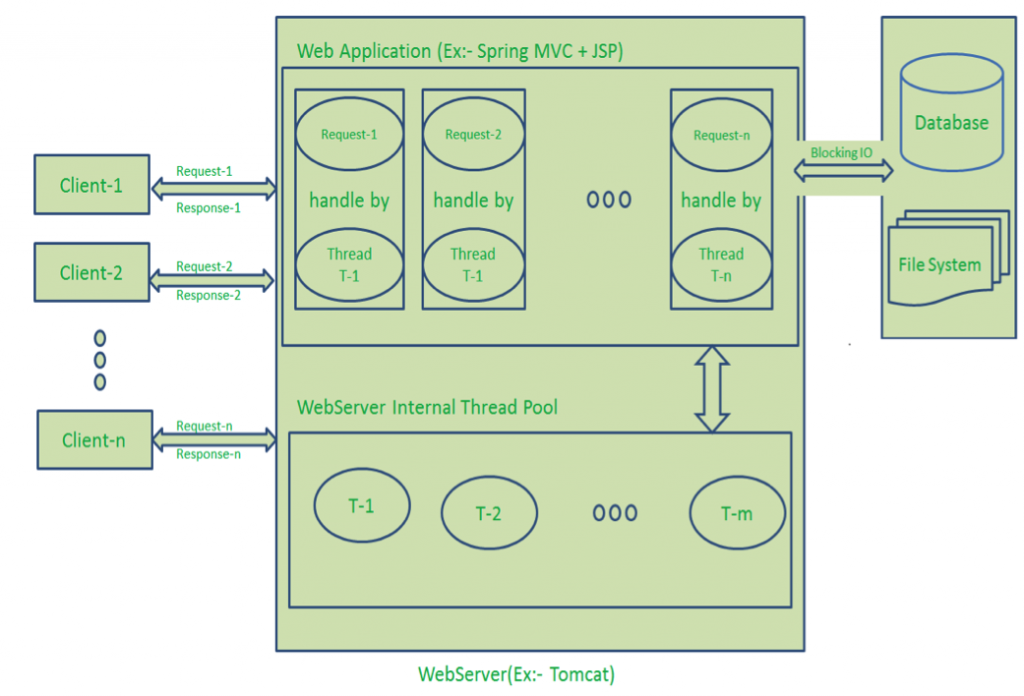
However, this model uses Multiple Threads to handle concurrent client requests. Before discussing this model internals, first go through the diagram below.

**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 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.

[](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-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.

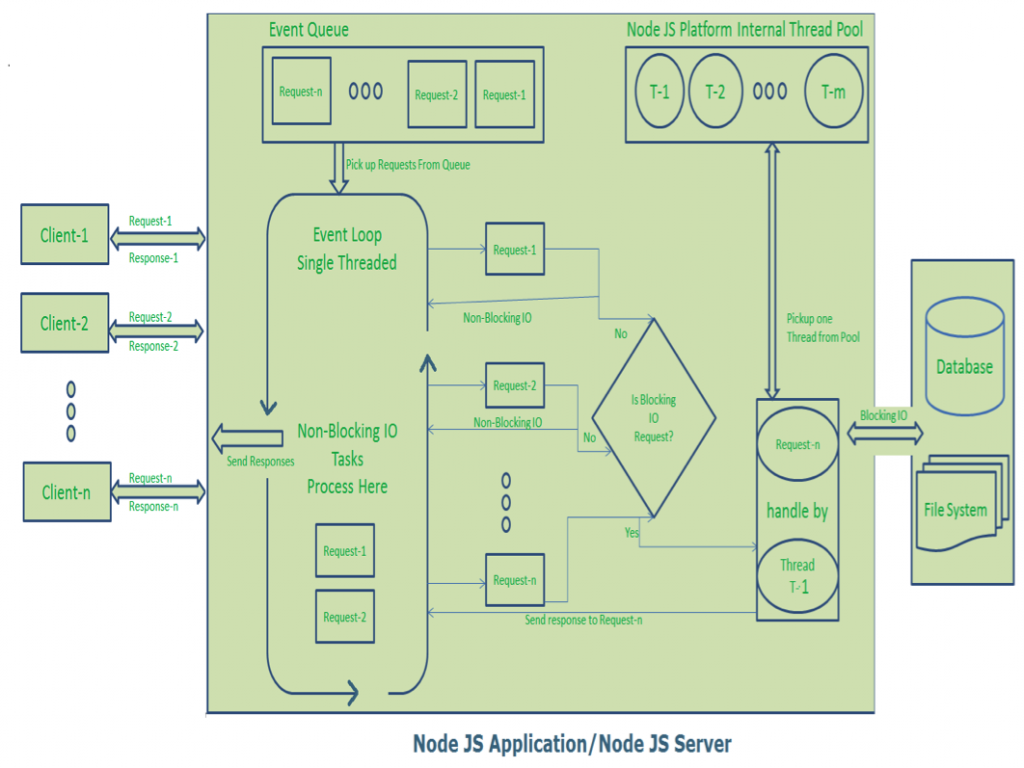
As Node JS follows this architecture, it can handle more and more concurrent client requests very easily. Before discussing this model internals, first go through the diagram below.

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 Event Loop Picks up those requests one by one.
  + Event 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
  + Event 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
  + Event 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