

NODE.JS

SERVER SIDE JAVASCRIPT



Introduction Node.js

- Node.js was created by **Ryan Dahl** starting in 2009, and its growth is sponsored by Joyent, his employer.
- Node JS is a JavaScript framework invented to execute the code by interacting with the input and output.
- Node's goal is to provide an easy way to build scalable network programs.

What is node.js ?

- 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 Runs on Google's V8 JavaScript Engine
 - ❖ Node.js programs are executed by V8 JavaScript engine the same used by Google chrome browser.
- Node.js is used to build server side applications.
- The Node.js platform provides abstractions that make it easier to create event-driven applications.
- Node.js works with I/O events, for example: connect from a database, open a file or read data from a stream, send messages, etc.

Why to use Node?

- Firstly, for performance and scalability.
- Node can handle thousands of requests concurrently where PHP would just collapse.
- Today it is used to power websites for LinkedIn, New York Times, PayPal, eBay, Walmart, Yahoo!, Intuit, Voxer and Uber, among other companies.

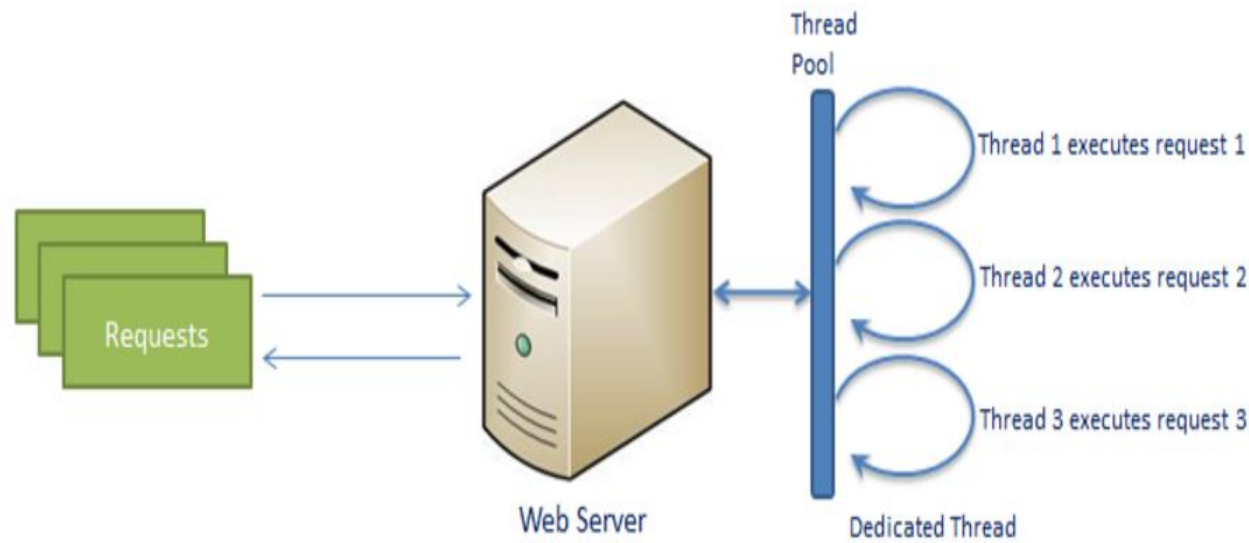


What is unique about Node.js?

1. JavaScript used in client-side but node.js puts the JavaScript on server-side thus making communication between client and server will happen in same language.
2. Servers are normally thread based but Node.JS is "Event" based. Node.JS serves each request in a Evented loop that can able to handle simultaneous requests.
3. Major security implementations in Node.js are:
 - Authentications
 - Error Handling

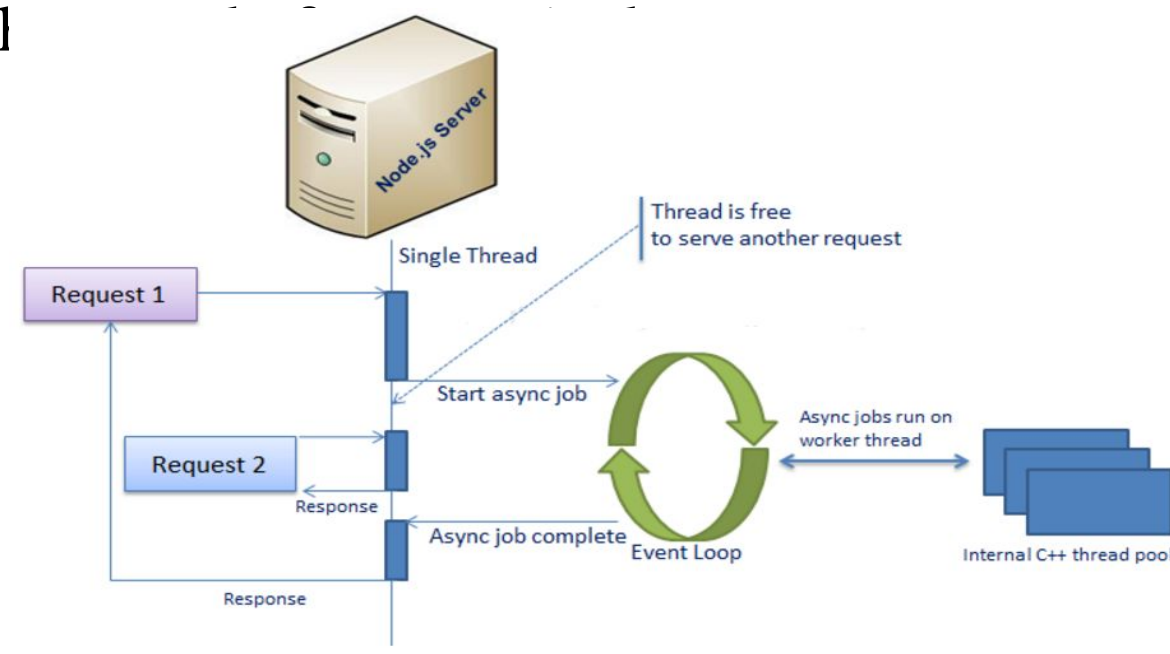
Traditional Web Server Model

- In the traditional web server model, each request is handled by a dedicated thread from the thread pool. If no thread is available in the thread pool at any point of time then the request waits till the next available thread. Dedicated thread executes a particular request and does not return to thread pool until it completes the execution and returns a response.



Node.js Process Model

- Node.js processes user requests differently when compared to a traditional web server model. Node.js runs in a single process and the application code runs in a single thread and thereby needs less resources than other platforms. All the user requests to your web application will be handled by a single thread and all the I/O work or long running job is performed async



What can you do with Node ?

- It is a command line tool. You download a tarball, compile and install the source.
- The JS executed by the V8 javascript engine (the thing that makes Google Chrome so fast)
- Node provides a JavaScript API to access the network and file system.
- Single Page Applications (SPAs) are used to create social networking applications. Node.js is a great fit for SPAs because it can handle asynchronous calls and heavy workloads of these applications.
- Node.js is the most widely used solution for developing microservices due to its repository features and highly flexible modules that can be used for specific parts of an application.

What can't do with Node?

- Node is a platform for writing JavaScript applications outside web browsers. This is not the JavaScript we are familiar with in web browsers. There is no DOM built into Node, nor any other browser capability.
- Node can't run on GUI, but run on terminal

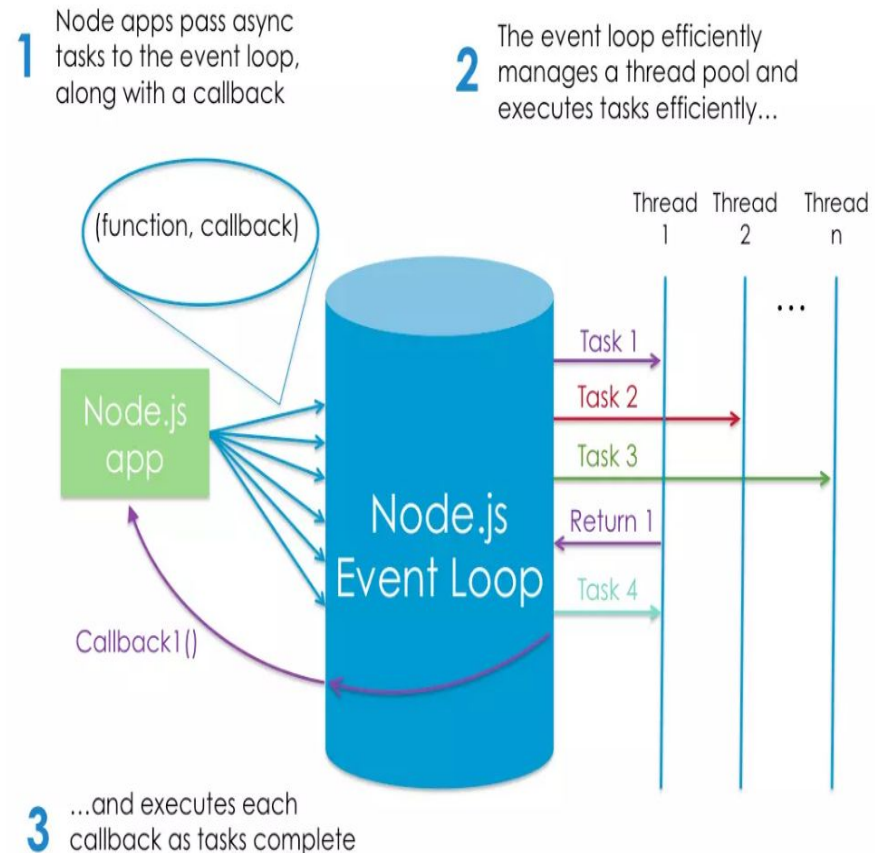
Why node.js use event-based?

In a normal process cycle, the web server while processing the request will have to wait for the IO operations and thus blocking the next request to be processed.

Node.JS process each request as events, The server doesn't wait for the IO operation to complete while it can handle other request at the same time.

When the IO operation of first request is completed it will call-back the server to complete the request.

The following image depicts Node's execution model:



Node.js VS Apache

1. It's fast
2. It can handle tons of concurrent requests
3. It's written in JavaScript (which means you can use the same code server side and client side)

Platform	Number of request per second
PHP (via Apache)	3187,27
Static (via Apache)	2966,51
Node.js	5569,30

Steps to install Node.js in Ubuntu

```
sudo apt-get install nodejs
```

```
sudo apt-get install npm
```

```
sudo npm cache clean -f
```

```
sudo npm install -g n
```

```
sudo n stable
```

- To get the node version, following command is used:

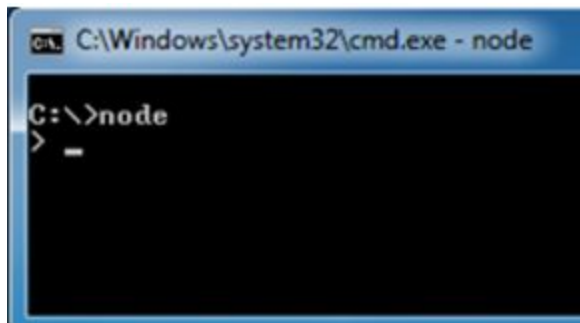
```
node -v
```

- To get the npm version, following command is used:

```
npm -v
```

Node.js Console

- Node.js comes with virtual environment called REPL (i.e., Node shell). REPL stands for Read-Eval-Print-Loop. It is a quick and easy way to test simple Node.js/JavaScript code.
- To launch the REPL (Node shell), open command prompt or terminal (in UNIX/Linux) and type *node* as shown below. It will change the prompt to `>`.
- Node.js Examples in REPL

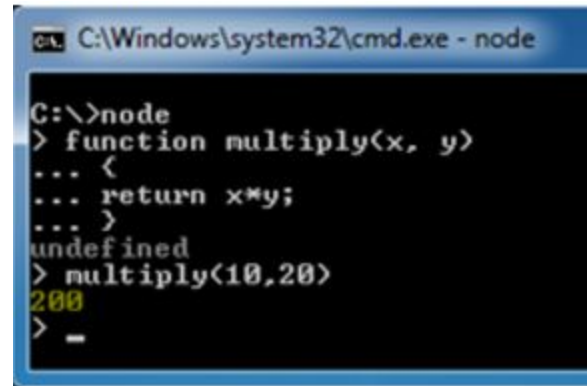


```
C:\Windows\system32\cmd.exe - node
C:\>node
>
```

```
> 10 + 20
30
```

```
> "Hello" + "World"
Hello World
```

```
> var x = 10, y = 20;
> x + y
30
```



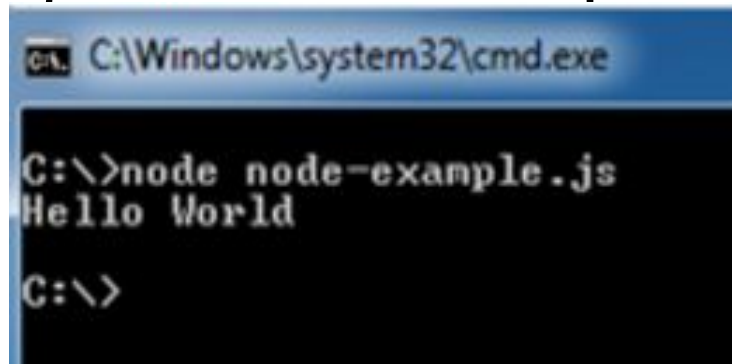
```
C:\Windows\system32\cmd.exe - node
C:\>node
> function multiply(x, y)
... <
... return x*y;
... >
undefined
> multiply(10,20)
200
>
```

Execute external java script

- You can execute an external JavaScript file by writing `node fileName` command. For example, assume that **node-example.js** is on your home directory with following code.

```
console.log("Hello World");
```

- Now, you can execute `node-example.js` from command prompt as shown below.

A screenshot of a Windows command prompt window. The title bar at the top reads "C:\Windows\system32\cmd.exe". The command prompt shows the following sequence of text: the prompt "C:\>" followed by the command "node node-example.js", the output "Hello World", and then the prompt "C:\>" again.

```
C:\Windows\system32\cmd.exe  
  
C:\>node node-example.js  
Hello World  
  
C:\>
```

Understanding NPM

- NPM is a package manager for Node JS. NPM is the pre-build libraries for Node JS. It was created in 2009 and it was introduced for Node JS in 2011. It is a public collection of packages of open source code for Node JS. NPM is command line client. Whenever you install Node JS, npm will automatically install in your system.
- NPM is basically a repository of libraries you can use in your project like ruby has gems.
- You can also use it just like npm install to install any packages and their dependencies in package.json

NPM: Introduction

- npm makes it easy for JavaScript developers to share the code that they've created to solve particular problems, and for other developers to reuse that code in their own applications.
- Once you're depending on this code from other developers, npm makes it really easy to check to see if they've made any updates to it, and to download those updates when they're made.
- These bits of reusable code are called packages, or sometimes modules. A package is just a directory with one or

Bootstrapping Application

- In Node JS, Everything works in JSON.(Java Script Object Notation)
- There is one main file `package.json` which gives information about the application in which `package.json` is a collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
- Create the folder for project using the command:
`mkdir mynodejs_app`
- In Node JS, each application work like library/package so anyone can install and start working on it. Next, go to the folder `mynodejs_app` and execute the following command:

`npm init`

package.json

□ The **package.json file** in Node.js is the heart of the entire application. It is basically the manifest file that contains the metadata of the project where we define the properties of a package.

□ **package.json**: Instructs the Node.js package manager (npm) what it needs to do; including which dependency packages should be installed.

1. **name : (mynodejs_app)** – It will take default name which is bracket if not entered.

2. **version :(version number)** – It will take default version if not entered. Better add version as it play vital role when you get bug.

3. **description** – Enter the description

entry point : (index.js) – This is the default entry file which is called default. you can override with any file.

4. **test command** – Whenever you type “npm test”, what all set of command should be fire to test application.

5. **git repository** – if you are using Git Repository, you can enter here. So, you can easily deploy or update code on server.

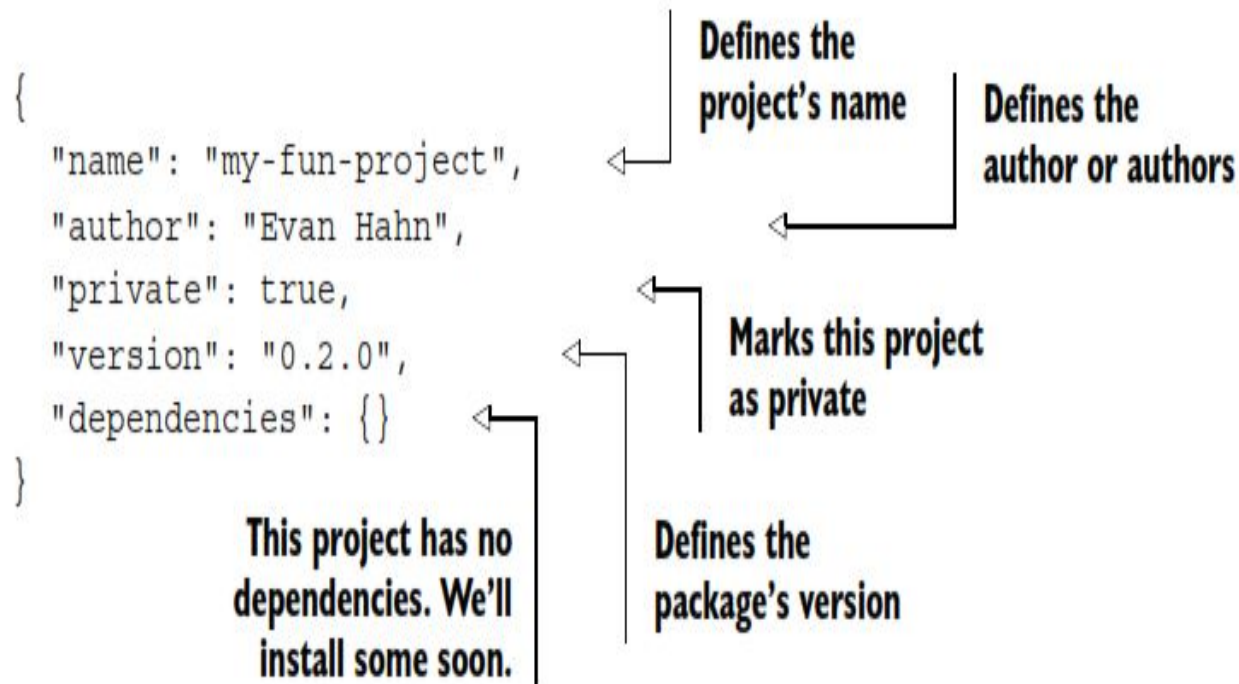
6. **keywords** – You can add keywords matched with package.

7. **author** – Enter the name of author.

8. **license :(ISC)** – If you creating public package, what kind of licence applicable, you can enter here.

Refer this URL for more info: <http://browsenpm.org/package.json#scripts>

A simple package.json file



- This will generate the JSON snippet and write it in the package.json file. It will ask for the confirmation. After that package.json file will be created in your application folder which contains following JSON information.

- package.json

```
{
  "name": "@sri/msrit",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "directories": {
    "test": "test"
  },
  "dependencies": {
    "express": "^4.15.1",
    "fs": "^0.0.1-security",
    "mysql": "^2.13.0"
  },
  "devDependencies": {},
  "scripts": {
    "test": "\"echo \\\"Error: no test specified\\\"\" && exit 1\"",
    "start": "node index.js"
  },
  "author": "",
  "license": "ISC"
}
```

Steps to create simple node application

- Create a file called **calc.js** in the folder **mynodejs_app**

```
// mynodejs_app/calc.js
function sum (arr) {
  return arr.reduce(function(a, b) {
    return a + b
  }, 0)
}

module.exports.sum = sum
```

- Create **index.js** in the same folder

```
// mynodejs_app/index.js
const calc = require('./calc')

const numbersToAdd = [3,4,10,2]

const result = calc.sum(numbersToAdd)
console.log("The result is: "+result)
```

- To start node application enter the following
npm start

Node.js Web Server

- Node.js provides capabilities to create your own web server which will handle HTTP requests asynchronously. You can use IIS or Apache to run Node.js web application but it is recommended to use Node.js web server.
- When you start building HTTP-based applications in Node.js, the built-in http/https modules are the ones you will interact with.
- We'll need to require the http module and bind our server to the port 5000 (user defined) to listen on.
- **requestHandler:** this function will be invoked every time a request hits the server. If you visit localhost:5000 from your browser, two log messages will appear: one for / and one

Node.js Web Server

- if (err): error handling - if the port is already taken, or for any other reason our server cannot start, we get notified here.
- The http module is very low-level - creating a complex web application using the snippet below is very time-consuming. This is the reason why we usually pick a framework to work with for our projects. There are a lot you can pick from, but these are the most popular ones:
 - express
 - hapi
 - koa

Web Server Application

myserver.js

```
var my_http = require("http");  
my_http.createServer(function(request,response){  
  response.writeHead(200, {"Content-Type": "text/plain"});  
  response.write("Hello MSRIT");  
  response.end();  
}).listen(5000);  
console.log("Server Running on 5000");
```

You can start it with:

```
E:\testnodejs>node myserver  
Server Running on 5000
```

Type Ctrl+c to Stop the server

Open in chrome browser

