Unit – 1 Node JS

What is Node.js?

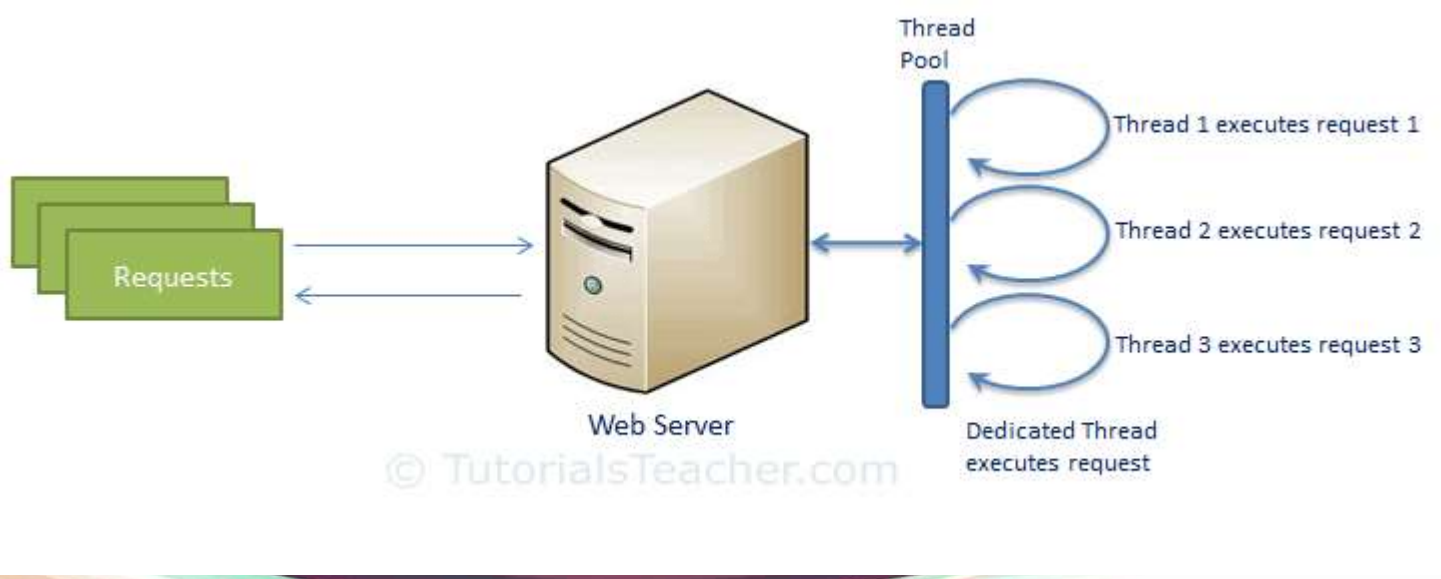
- Node.js is an open-source server side runtime environment built on Chrome's V8 JavaScript engine.
- It provides an event driven, non-blocking (asynchronous) I/O and cross-platform runtime environment for building highly scalable server-side application using JavaScript.
- Node.js can be used to build different types of applications such as command line application, web application, real-time chat application, REST API server etc.
- it is mainly used to build network programs like web servers, similar to PHP, Java, or ASP.NET.
- Node.js was written and introduced by Ryan Dahl in 2009.

Advantages of Node.js

- Node.js is an open-source framework under MIT license.
- Uses JavaScript to build entire server side application, so easy to learn & use for those who knows javascript.
- It is lightweight framework that includes minimum modules. However other modules can be included as per the need of an application.
- It is asynchronous by default. So it faster than other frameworks.
- Cross-platform framework that runs on Windows, MAC or Linux

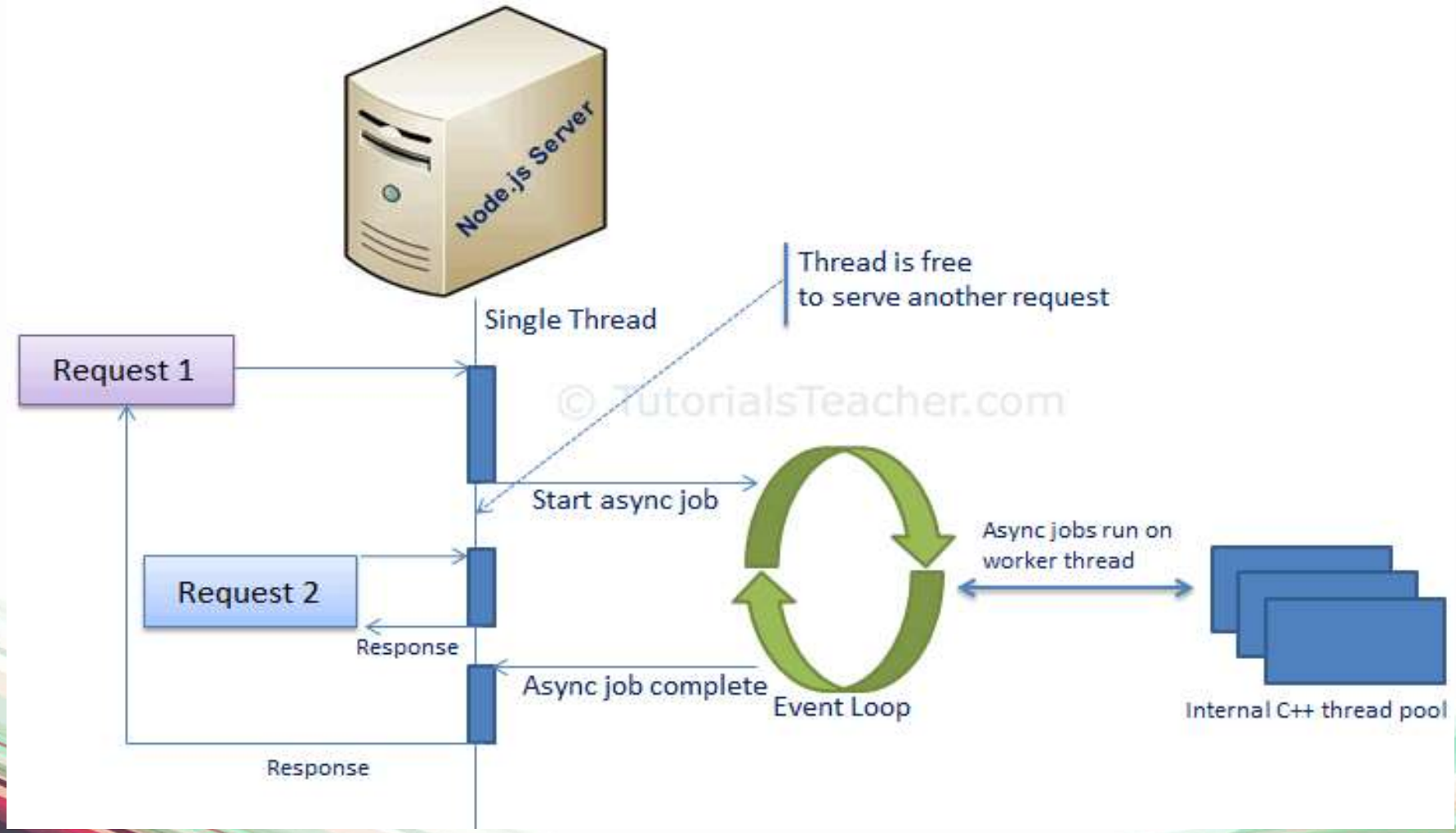
Let us first understand traditional web server model

- 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.



Now let us understand Node.js Process 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 asynchronously for a particular request.
- So, this single thread doesn't have to wait for the request to complete and is free to handle the next request. When asynchronous I/O work completes then it processes the request further and sends the response.
- An event loop is constantly watching for the events to be raised for an asynchronous job and executing callback function when the job completes.
- Internally, Node.js uses libev for the event loop which in turn uses internal C++ thread pool to provide asynchronous I/O.
- **Let us see the diagram**



Node js process model

- Node.js process model increases the performance and scalability with a few warning.
- Node.js is not fit for an application which performs CPU-intensive operations like image processing or other heavy computation work because it takes time to process a request and thereby blocks the single thread.

Node.js Console/REPL

- Node.js comes with virtual environment called REPL (aka 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 (in Windows) type *node*. It will change the prompt to > in Windows.
- You can now test any Node.js/JavaScript expression in REPL. For example 10 + 20 will display 30 immediately in new line.
- **You can execute an external JavaScript file by executing the node fileName command.**
- To exit from the REPL terminal, press Ctrl + C twice or write .exit and press Enter.

First program in node js

first program in node

```
function GreetingMessage(name)
{
    console.log("Hello Good Morning Mr/MRS/MISS " + name);
}
GreetingMessage('Ankit Patel');
```

Scope of variables

- Node's JavaScript is different from browser's JavaScript in case of global scope.
- In the browser's JavaScript, variables declared without var keyword become global.
- **In Node.js, everything is local by default.**

Modules in node

- Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.
- Each module in Node.js has its own context, so it cannot interfere with other modules.
- module is a separate .js file with any valid name.
- **Node.js Module Types**
- Node.js has 3 types of modules:
  1. Core Modules (built in modules)
  2. Local Modules (user defined modules)
  3. Third Party Modules (downloadable modules)

Node.js Core Modules

- Core modules provide only basic functionalities of Node.js.
- These core modules are compiled into its binary distribution and load automatically when Node.js process starts.
- But you need to import the core module first in order to use it in your application.
- Let us see few of the core modules.

Core Module	Description
http	http module includes classes, methods and events to create Node.js http server.
url	url module includes methods for URL resolution and parsing.
querystring	querystring module includes methods to deal with query string.
path	path module includes methods to deal with file paths.
fs	fs module includes classes, methods, and events to work with file I/O.
util	util module includes utility functions useful for programmers.

Loading Core Modules

- In order to use node.js core or NPM modules, you first need to import it using require() function as shown below.

```
var modulename = require('module_name');
```

example

```
var http = require(‘http’)
```

- As per above syntax, specify the module name in the require() function.
- The require() function will return an object, function, property or any other JavaScript type, depending on what the specified module returns.
- Let us see example

How to use Node.js as Web Server

- To access web pages of any web application, you need a web server.
- The web server will handle all the http requests for the web application also return response to client.
- We use IIS is a web server for ASP.NET web applications and apache web server for PHP or Java web applications.
- **While node.js has its own web server which will handle HTTP requests asynchronously.**
- it is recommended to use Node.js web server.
- Let us learn how to create web server.

example of loading standard module

```
var http = require('http');
var server = http.createServer(function(req, res){
    console.log("Server is ready....");
});
server.listen(5000);
```

- In the above example, require() function returns an object because http module returns its functionality as an object, you can then use its properties and methods using dot notation e.g. http.createServer().
- In this way, you can load and use Node.js core modules in your application.

1. In the above example, we import the http module using require() function.
2. The http module is a core module of Node.js, so no need to install it using NPM.
3. The next step is to call createServer() method of http and specify callback function with request and response parameter.
4. Finally, call listen() method of server object which was returned from createServer() method with port number, to start listening to incoming requests on port 5000.
5. You can specify any unused port here.
6. The http.createServer() method includes request and response parameters which is supplied by Node.js.
7. The request object can be used to get information about the current HTTP request e.g., url, request header, and data.
8. The response object can be used to send a response for a current HTTP request.

Node.js Local Module

- Local modules are modules created locally in your Node.js application by developer.
- These modules has different functionalities of your application.
- Such modules are stored in separate files and folders.
- You can also package it and distribute it via NPM, so that Node.js community can use it.
- For example, if you need to connect to MongoDB and fetch data then you can create a module for it, which can be reused in your application.

Example of local module (filename **mymodule.js**)

example of local module

```
function logger(message)
{
    console.log(message);
}
function anotherloger(message)
{
    console.log(message)
}
//this line is require to export module
module.exports.logger = logger;
module.exports.logger2 = anotherloger
```

How to use local module

example of using local module

```
// using module
const mymodule = require("./mymodule")
mymodule.logger("Good morning brother and sisters...")
mymodule.logger2("Good evening to all ");
```

example of creating local module (2nd way)

```
//another way to create module
var log = {
    info: function (info) {
        console.log('Info: ' + info);
    },
    warning:function (warning) {
        console.log('Warning: ' + warning);
    },
    error:function (error) {
        console.log('Error: ' + error);
    }
};
module.exports = log
```

example of calling module

```
var mymodule = require('./mymodule2.js');
mymodule.info('Node.js started');
mymodule.warning('this is warning messsage');
mymodule.error('this is error message');
```

Export Module in Node.js …

- The module.exports is a special object which is included in every JavaScript file in the Node.js application by default.
- The module is a variable that represents the current module, and exports is an object that will be exposed as a module. So, whatever you assign to module.exports will be exposed as a module.
- Let's see how to expose different types as a module using module.exports.
- **Export Literals**
- if you assign a string literal then it will expose that string literal as a module.

Message.js

```
module.exports = 'Hello world';
```

app.js

```
var msg = require('./Messages.js');

console.log(msg);
```

Export simple Object & function

- The exports is an object. So, you can attach properties or methods to it.

Message.js

```
exports.SimpleMessage = 'Hello world';

//or

module.exports.SimpleMessage = 'Hello world';
```

app.js

```
var msg = require('./Messages.js');

console.log(msg.SimpleMessage);
```

Log.js

```
module.exports.log = function (msg) {
    console.log(msg);
};
```

app.js

```
var msg = require('./Log.js');

msg.log('Hello World');
```

Export object

export object (mymodule2.js)

```
module.exports = {
    name: 'Ankit',
    surname: 'Patel',
    age : 38
}
```

use of exported object

```
var person = require('./mymodule2.js');
console.log(person.name + ' ' + person.surname + ' ' + person.age);
```

Export class

export class example

```
module.exports = function (firstName, lastName) {
    this.name = firstName;
    this.surname = lastName;
    this.fullName = function () {
        return this.name + ' ' + this.surname;
    }
}
```

export class example 2

```
class person{
    constructor(firstName,lastName)
    {
        this.name = firstName;
        this.surname = lastName;
    }
    fullName()
    {
        return this.name + ' ' + this.surname;
    }
}
module.exports = person
```

export class example

```
var person = require('./mymodule2.js');
var p1 = new person('Ankit', 'Patel');
console.log(p1.fullName());
```

NPM - Node Package Manager …

1. Node Package Manager (NPM) is a command line tool that installs, updates or uninstalls Node.js packages in your application.
2. It is also an online repository for open-source Node.js packages.
3. It's Official website: https://www.npmjs.com
4. NPM is included with Node.js installation. After you install Node.js, verify NPM installation by writing the following command in terminal or command prompt.

   1. **npm version**

5. If you have an older version of NPM then you can update it to the latest version using the following command.

   npm install npm -g

1. To access NPM help, write npm help in the command prompt or terminal window.

   npm help

NPM - Node Package Manager

- NPM performs the operation in two modes: global and local.
- In the global mode, NPM performs operations which affect all the Node.js applications on the computer
- whereas in the local mode, NPM performs operations for the particular local directory which affects an application in that directory only.
- **Install Package Locally**
- Use the following command to install any third party module in your local Node.js project folder.
- npm install <package name>
- For example, the following command will install ExpressJS into MyNodeProj folder.
- npm install express
- All the modules installed using NPM are installed under **node_modules** folder. The above command will create Express folder under node_modules folder in the root folder of your project and install Express.js there. also adds dependency entry into the **package.json**.

Install Package Globally

- NPM can also install packages globally so that all the node.js application on that computer can import and use the installed packages.
- NPM installs global packages into */<User>/local/lib/node_modules* folder.
- Apply -g in the install command to install package globally.
- For example, the following command will install Express globally.
- npm install -g express

Update Package

- To update the package installed locally in your Node.js project, navigate the command prompt or terminal window path to the project folder and write the following update command.
- npm update <package name>
- npm update express

Uninstall Package

- Use the following command to remove a local package from your project.
- npm uninstall <package name>
- npm uninstall express

How to handle request and return response?

- The http.createServer() method includes request and response parameters which is supplied by Node.js.
- The request object can be used to get information about the current HTTP request e.g., url, request header, and data.
- The response object can be used to send a response for a current HTTP request.

Handling request and returning response

```
var http = require('http');
var server  = http.createServer(function(request,response){
    console.log(request.url); //will log url for which request is received.
    response.writeHead(200,{'content-type':'text/html'});
    if(request.url=="/")
    {
        response.write("<html><head></head><body><h1>Home</h1><hr/></body></html>");
    }
    else if(request.url=="/aboutus")
    {
        response.write("<html><head></head><body><h1>About us</h1><hr/></body></html>");
    }
    else if(request.url=="/contactus")
    {
        response.write("<html><head></head><body><h1>Contact us</h1><hr/></body></html>");
    }
    response.end();
});
server.listen(5000);
console.log("Ready to accept request....")
```

Now let us understand it

- In the above example, request.url is used to check the url of the current request and based on that it sends the response.
- To send a response, first it sets the response header using response.writeHead() method and then writes a string as a response body using response.write() method.
- Finally, Node.js web server sends the response using response.end() method.

Example of how to return output in JSON format?

Example of how to return output in JSON format?

```
var http = require('http');
var server  = http.createServer(function(request,response){
    console.log(request.url); //will log url for which request is received.
    response.writeHead(200,{'content-type':'application/json'});
    if(request.url=="/info")
    {
        var data = JSON.stringify({
            name: "the easylearn academy",
            course: "Node js",
            lecture: 60,
            trainer: "Ankit Patel"});
        response.write(data);
    }
    response.end();
});
server.listen(5000);
console.log("Ready to accept request....")
```