

Node.js

Node.js allows developers to create both front-end and back-end applications using JavaScript. It was released in 2009 by Ryan Dahl.

What is Node.js?

- ✓ Node.js is an open-source and cross-platform JavaScript runtime environment.

Open-source:

- ✓ This means that the source code for Node.js is publicly available. And it's maintained by contributors from all over the world.

Cross-platform:

- ✓ Node.js is not dependent on any operating system software. It can work on Linux, macOS, or Windows.

JavaScript runtime environment:

- ✓ When you write JavaScript code in your text editor, that code cannot perform any task unless you execute (or run) it. To run your code, you need a runtime environment.
- ✓ Node.js provides a runtime environment outside of the browser. It's also built on the Chrome V8 JavaScript engine. This makes it possible to build back-end applications using the same JavaScript programming language you may be familiar with.

Differences Between the Browser and Node.js Runtime Environments

- ✓ Both the browser and Node.js are capable of executing JavaScript programs. But there are some key differences that you need to know.

Access to the DOM APIs

- ✓ With the browser runtime, you can access the Document Object Model (DOM). And you can perform all the DOM operations. But Node.js does not have access to the DOM.
- ✓ Node.js exposes almost all the system resources to your programs. This means you can interact with the operating system, access the file systems, and read and write to the files. But, you cannot access operating systems and file systems from the browser.

Why Node.js?

- ✓ 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.

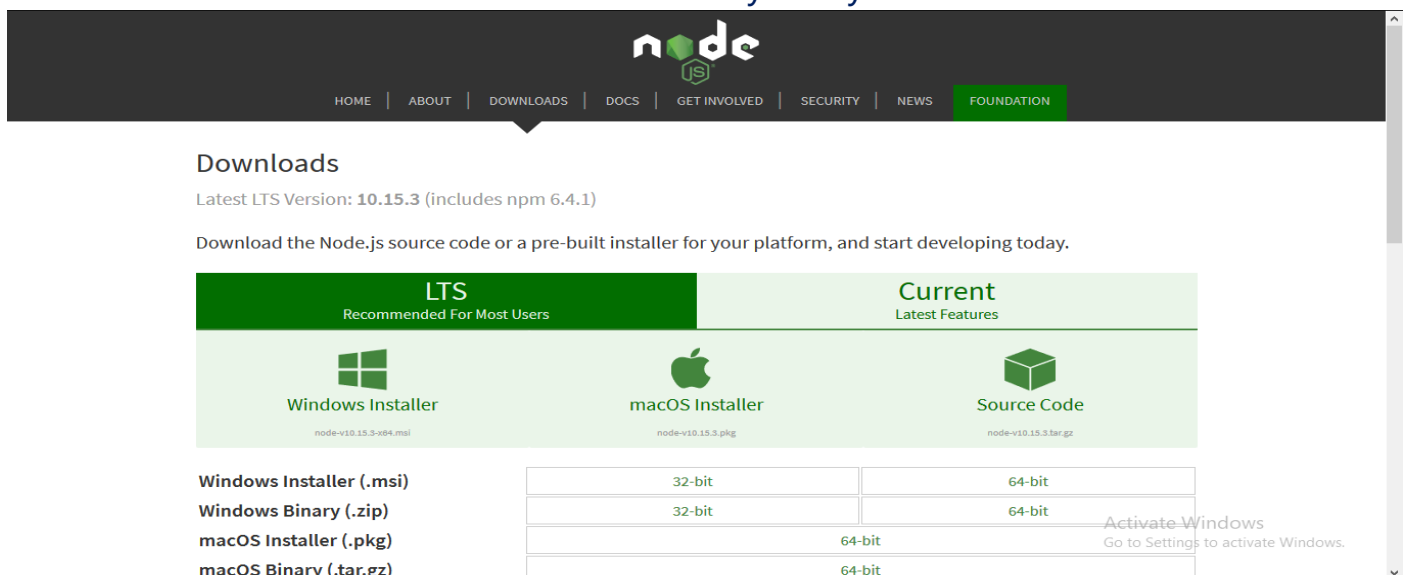
Here is how Node.js handles a file request:

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, asynchronous programming, which is very memory efficient.

Node.js Installation on Windows:

Step-1: Download the Node.js '.msi' installer.

- ✓ The first step to installing Node.js on Windows is to download the installer.
- ✓ Visit the official Node.js website i.e., <https://nodejs.org/en/download>, and download the '.msi file' according to your system environment (32-bit & 64-bit). An MSI installer will be downloaded to your system.

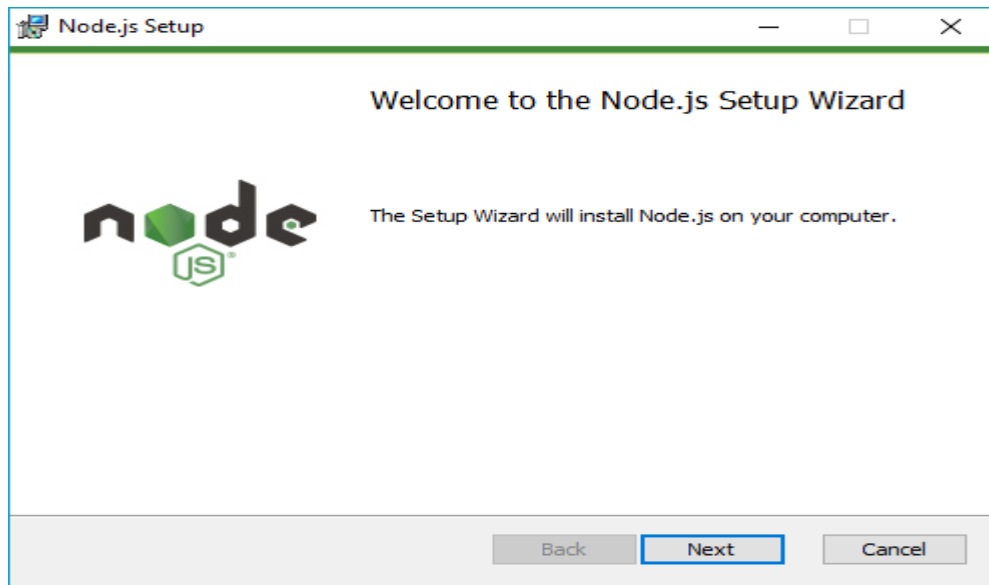


The screenshot shows the Node.js Downloads page. At the top, there's a navigation bar with links: HOME, ABOUT, DOWNLOADS, DOCS, GET INVOLVED, SECURITY, NEWS, and FOUNDATION. Below the navigation bar, the page title is "Downloads" with the subtitle "Latest LTS Version: 10.15.3 (includes npm 6.4.1)". A paragraph states: "Download the Node.js source code or a pre-built installer for your platform, and start developing today." Below this, there are two main sections: "LTS Recommended For Most Users" and "Current Latest Features". Under the "LTS" section, there are three options: "Windows Installer" (node-v10.15.3-x64.msi), "macOS Installer" (node-v10.15.3.pkg), and "Source Code" (node-v10.15.3.tar.gz). Under the "Current" section, there are three options: "Windows Installer" (node-v10.15.3-x64.msi), "macOS Installer" (node-v10.15.3.pkg), and "Source Code" (node-v10.15.3.tar.gz). Below these options, there is a table with columns for "32-bit" and "64-bit" for each platform. The table shows that for Windows, there are 32-bit and 64-bit installers. For macOS, there are 32-bit and 64-bit binaries. For Source Code, there are 64-bit binaries. An "Activate Windows" watermark is visible on the right side of the page.

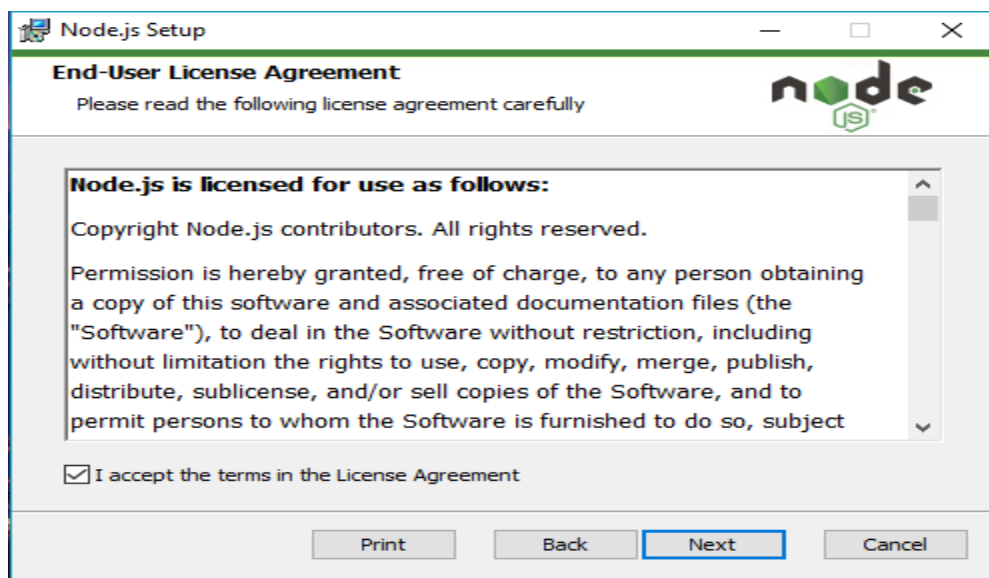
	32-bit	64-bit
Windows Installer (.msi)		
Windows Binary (.zip)		
macOS Installer (.pkg)		
macOS Binary (.tar.gz)		

Step-2: Running the Node.js installer.

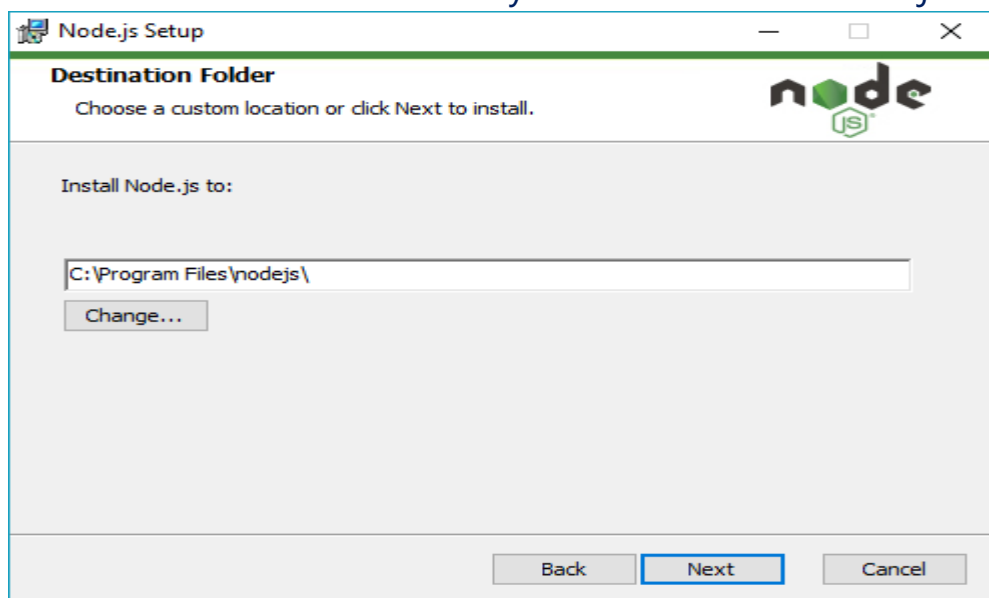
- ✓ Now you need to install the node.js installer on your PC. You need to follow the following steps for the Node.js to be installed:-
- ✓ Double-click on the .msi installer.
- ✓ The Node.js Setup wizard will open. Welcome To Node.js Setup Wizard.
- ✓ **Select "Next"**



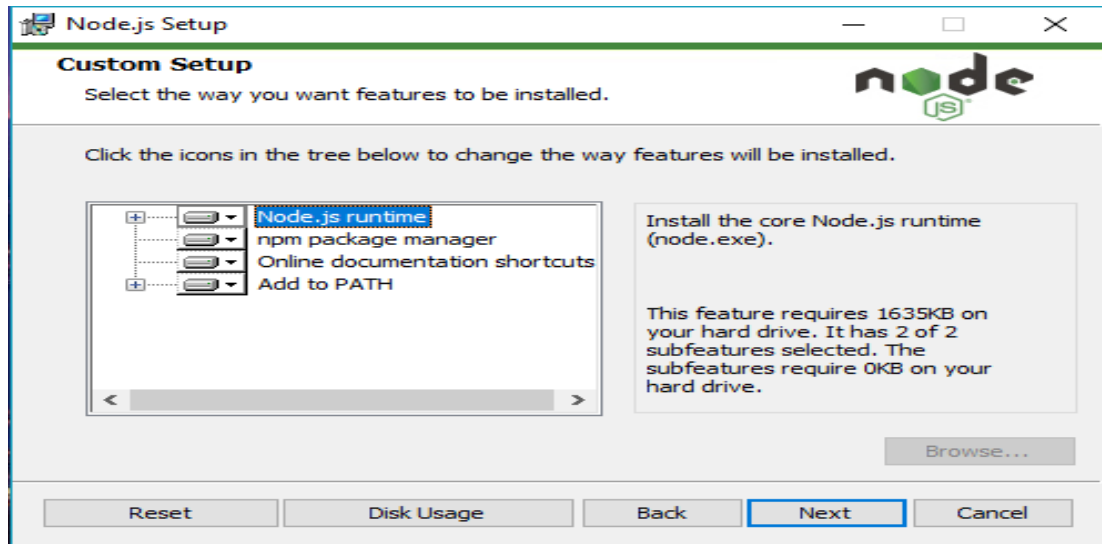
- ✓ After clicking "Next", End-User License Agreement (EULA) will open.
- ✓ Check "I accept the terms in the License Agreement"
- ✓ Select "Next"



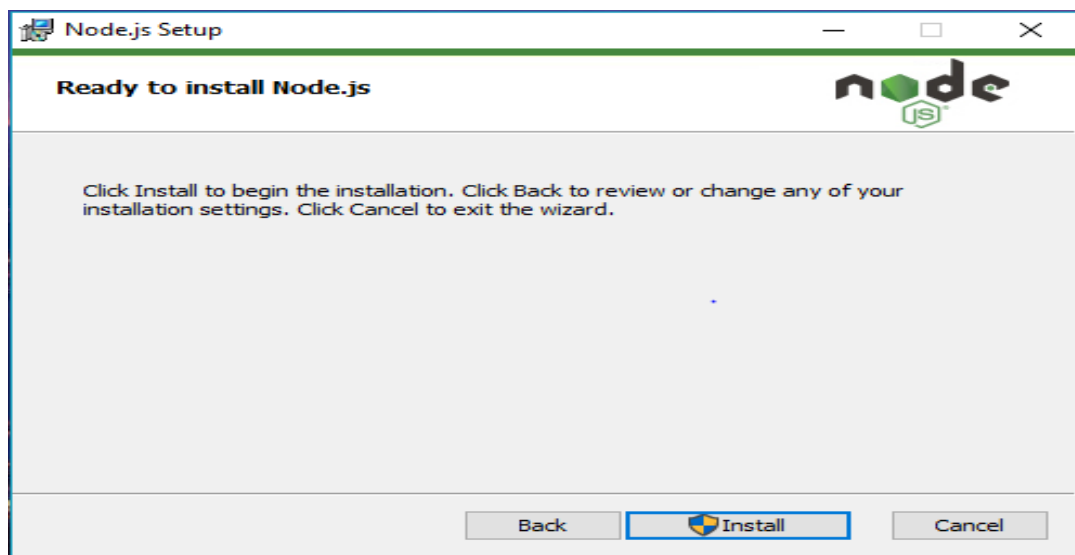
- ✓ Destination Folder
- ✓ Set the Destination Folder where you want to install Node.js & Select "Next"



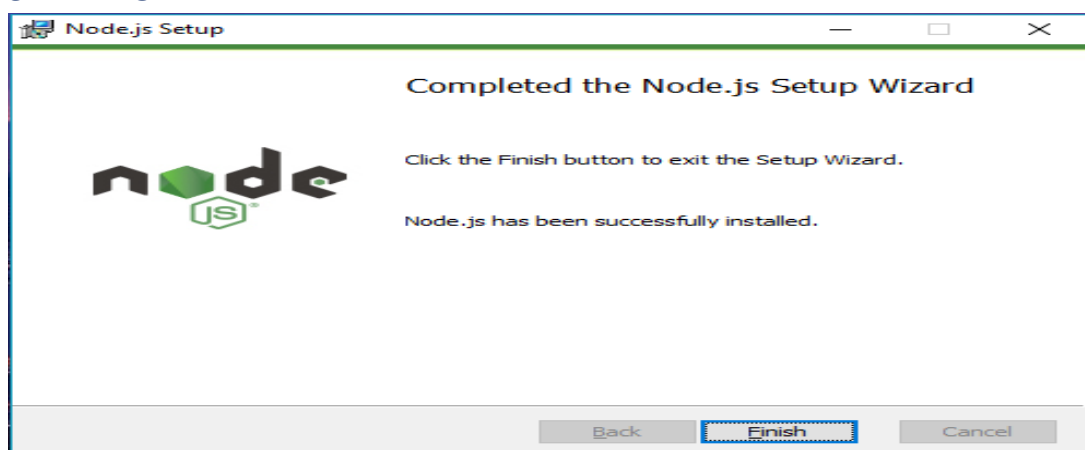
- ✓ Custom Setup
- ✓ **Select “Next”**



- ✓ Ready to Install Node.js.
- ✓ The installer may prompt you to “install tools for native modules”.
- ✓ **Select “Install”**

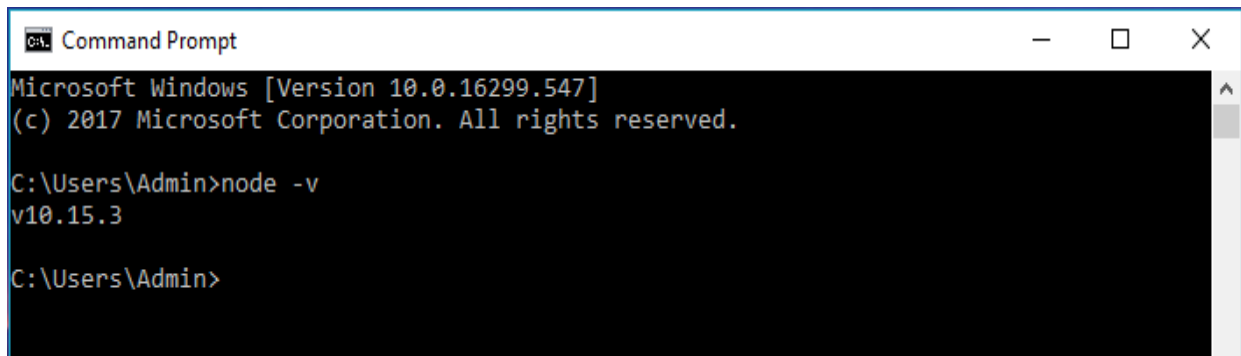


- ✓ Installing Node.js.
- ✓ Do not close or cancel the installer until the installation is complete
- ✓ Complete the Node.js Setup Wizard.
- ✓ **Click “Finish”**



Step 3: Verify whether Node.js was properly installed or not.

- ✓ To check that node.js was completely installed on your system or not, you can run the following command in your command prompt or Windows Powershell and test it:-
- ✓ **C:\Users\Admin> node -v**



```
Command Prompt
Microsoft Windows [Version 10.0.16299.547]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Admin>node -v
v10.15.3

C:\Users\Admin>
```

- ✓ If node.js was completely installed on your system, the command prompt will print the version of the node.js installed.

How to use Node.js

Step 1: Create a '.js' file (index.js)

Step 2: write an 'index.js' code

```
// Variable store number data type
var a = 35;
console.log(typeof a);

// Variable store string data type
a = "Codetantra";
console.log(typeof a);

// Variable store Boolean data type
a = true;
console.log(typeof a);

// Variable store undefined (no value) data type
a = undefined;
console.log(typeof a);
```

Step-3: run the index.js file

```
>node index.js
```

⇒ Verify how to execute different functions successfully in the Node.js platform

Example-1:

```
//multiplication function
function mul(a,b){
    return a*b;
}
console.log("multiply:"+mul(2,3))

//division with arrow function
const div = (a,b) => a/b
console.log("divide:" + div(10,5))
```

Example-2: (string functions)

```
var x = "Welcome to Codetantra";
var y = 'Node.js Tutorials';
var z = ['Node.js', 'Express.js', 'TypeScript'];

console.log(x);
console.log(y);

console.log("Concat Using (+) :", (x + y));
console.log("Concat Using Function :", (x.concat(y)));
console.log("Split string: ", x.split(' '));
console.log("Join string: ", z.join(', '));
console.log("Char At Index 5: ", x.charAt(5) );
```

Node Package Manager:

What is NPM?

- ✓ NPM is a package manager for Node.js packages, or modules if you like.
- ✓ <https://www.npmjs.com/> hosts thousands of free packages to download and use.
- ✓ The NPM program is installed on your computer when you install Node.js

What is a Package?

- ✓ A package in Node.js contains all the files you need for a module.
- ✓ Modules are JavaScript libraries you can include in your project.

Create a node.js project

- ✓ Create a new project directory and navigate into it using the terminal. Then, run the following command to initialize a new Node.js project

```
npm init -y
```

Download a Package

- ✓ Open the command line interface and tell NPM to download your desired package.
- ✓ I want to download a package called "to-case":

Download "to-upper":

```
C:\Users\Your Name>npm install to-case
```

- ✓ Now you have downloaded and installed your first package!
- ✓ NPM creates a folder named "node_modules", where the package will be placed. All packages you install in the future will be placed in this folder.
- ✓ My project now has a folder structure like this:

```
C:\Users\My Name\node_modules\to-case
```

Using a Package

- ✓ Once the package is installed, it is ready to use.
- ✓ Include the "to-case" package the same way you include any other module:

```
var to = require('to-case');
```

- ✓ Create a Node.js file that will convert the output "Hello World!"

```
var to = require('to-case');  
console.log(to.upper('Hello World!'));  
console.log(to.lower('Hello World!'));  
console.log(to.title("hello World!"));
```

- ✓ Save the code above in a file called "demo.js", and initiate the file:

Initiate demo_uppercase:

```
C:\Users\Your Name>node demo.js
```

Create a web server in node.js:

Node.js makes it easy to create a simple web server that processes incoming requests asynchronously.

Step1: import HTTP Module

Import the node.js module HTTP.

```
var http = require('http');
```

Step2: create a variable for the host and port

Define two variables host and port hold the information of the http server.

```
const host = "127.0.0.1"  
const port = 3000
```

step3: create a server

```
var server = http.createServer(function(){
    //Write a code here
});
server.listen(port,host,function(){
console.log('Server run on http://'+host+": "+port)
});
```

Step 4: Implement the function within http.createServer

```
var server = http.createServer(function(req,res){
    res.writeHead(200);
    res.end('Hello world');
});
```

- ✓ The function is expecting to get two arguments: **req** and **res**. With **req**, we have access to the request object and with **res**, we have access to the corresponding response object. Inside the method, we're using the **res** argument to first set the HTTP header status of the response to value 200 (okay) by using the method **writeHead**.

Step 5: Test the Http Server

```
> Node server.js
```

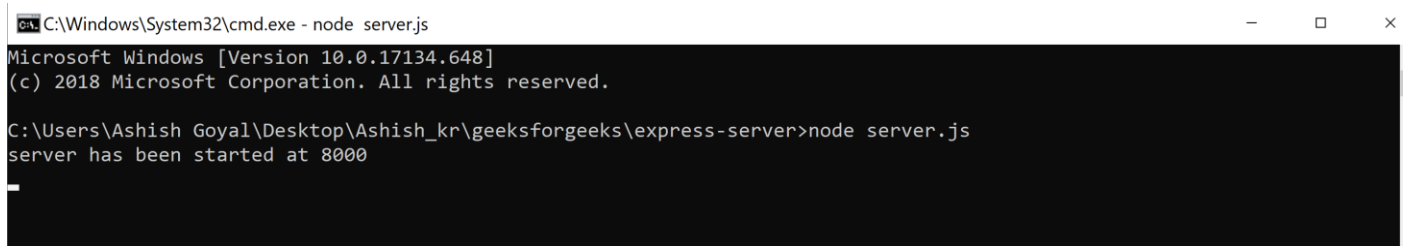
Example

```
var http = require('http');
const host = "127.0.0.1";
const port = 3000;
var server = http.createServer(function(req,res){
    if(req.url == '/'){
        res.writeHead(200)
        res.write("welcome page");
        res.end();
    }
    else if(req.url == "/about"){
        res.writeHead(200);
        res.write('about page.....');
        res.end();
    }
    else{
        res.writeHead(404);
        res.write('404 not found');
        res.end();
    }
});
server.listen(port,host,function(){
console.log('Server run on http://'+host+": "+port)
});
```


Restarting Node Application

- ✓ Node.js Automatic restart Node.js server with nodemon
- ✓ We generally type the following command for starting the Node.js server:

```
>node server.js
```



```
C:\Windows\System32\cmd.exe - node server.js
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Ashish Goyal\Desktop\Ashish_kr\geeksforgeeks\express-server>node server.js
server has been started at 8000
```

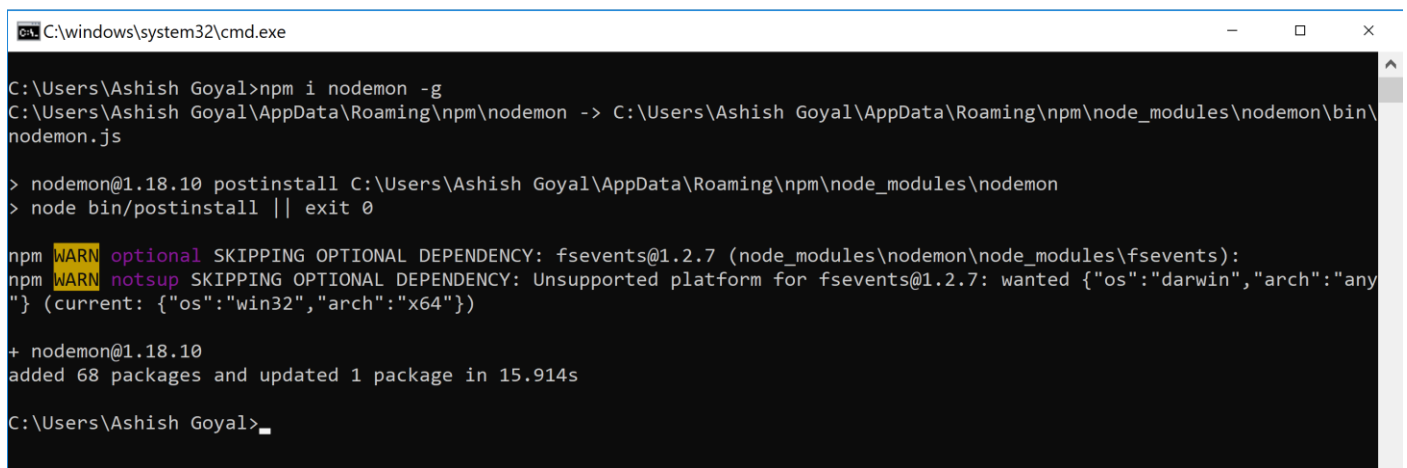
- ✓ In this case, if we make any changes to the project then we will have to restart the server by killing it using CTRL+C and then typing the same command again.

```
>node server.js
```

- ✓ It is a very hectic task for the development process.
- ✓ Nodemon is a package for handling this restart process automatically when changes occur in the project file.
- ✓ **Installing nodemon:** nodemon should be installed globally in our system:

Windows system:

```
>npm i nodemon -g
```



```
C:\windows\system32\cmd.exe
C:\Users\Ashish Goyal>npm i nodemon -g
C:\Users\Ashish Goyal\AppData\Roaming\npm\nodemon -> C:\Users\Ashish Goyal\AppData\Roaming\npm\node_modules\nodemon\bin\nodemon.js

> nodemon@1.18.10 postinstall C:\Users\Ashish Goyal\AppData\Roaming\npm\node_modules\nodemon
> node bin/postinstall || exit 0

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\nodemon\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.7: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

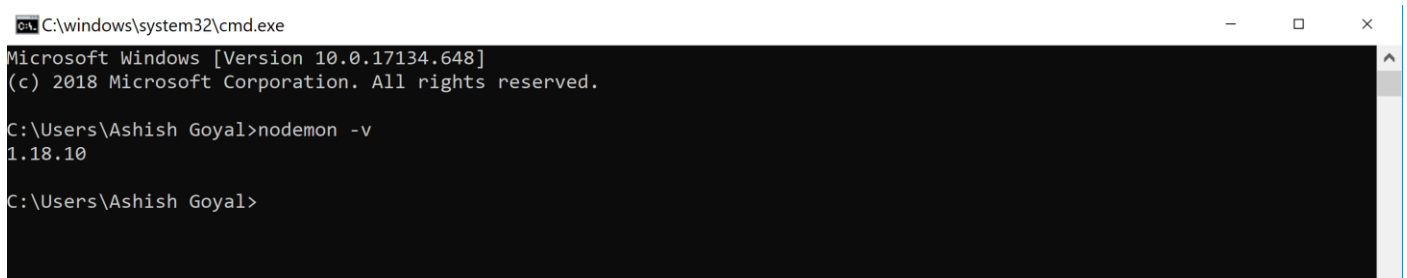
+ nodemon@1.18.10
added 68 packages and updated 1 package in 15.914s

C:\Users\Ashish Goyal>
```

- ✓ Now, let's check that nodemon has been installed properly in the system by typing the following command in terminal or command prompt:

```
nodemon -v
```

- ✓ It will show the version of nodemon as shown in the below screenshot.



```
C:\windows\system32\cmd.exe
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Ashish Goyal>nodemon -v
1.18.10

C:\Users\Ashish Goyal>
```

- ✓ Starting node server with nodemon:

nodemon [Your node application]

```
C:\Windows\System32\cmd.exe - nodemon server.js
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Ashish Goyal\Desktop\Ashish_kr\geeksforgeeks\express-server>nodemon server.js
[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
server has been started at 8000
```

- ✓ Now, when we make changes to our nodejs application, the server automatically restarts by nodemon as shown in the below screenshot.

```
C:\Windows\System32\cmd.exe - nodemon server.js
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Ashish Goyal\Desktop\Ashish_kr\geeksforgeeks\express-server>nodemon server.js
[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
server has been started at 8000
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
server has been started at 8000
```

- ✓ In this way with nodemon server automatically restarts.

Node.js Module

- ✓ 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 or pollute the global scope. Also, each module can be placed in a separate .js file under a separate folder.

Node.js Module Types

Node.js includes three types of modules:

1. Core Modules
2. Local Modules
3. Third Party Modules

1. Core Modules:

- ✓ To use Node.js core or NPM modules, you first need to import it using `require()` function as shown below.

Syntax

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

Example

```
var http = require('http');
```

- ✓ As per the 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.

2. Local Modules:

- ✓ Local modules are created by us locally in our Node.js application.
- ✓ These modules are included in our program in the same way as we include the built in module.
- ✓ Let's build a module with the name as a sum to add two numbers and include them in our index.js file to use them.

Code for creating local modules and exporting:

```
exports.add = function (a,b){  
  return a+b;  
}
```

- ✓ 'exports' keyword is used to make properties and methods available outside the file.
- ✓ In order to include the add function in our index.js file we use the require function.

Code for including local modules:

```
var mylog = require('./local.js')  
console.log(mylog.add(10,20));
```

- ✓ Add the above code to a server.js file
- ✓ To run this file, open a terminal in the project directory, type node index.js, and press enter. You can see the result of the addition of 10 and 20. This addition has been performed by the add function in the sum module.

3. Third-party modules:

- ✓ Third-party modules are modules that are available online using the Node Package Manager(NPM).
- ✓ These modules can be installed in the project folder or globally.
- ✓ Some of the popular third-party modules are Mongoose, express, angular, and React.

Example:

- npm install express
- npm install mongoose
- npm install -g @angular/cli

url module

Node.js provides the url module, which allows you to parse, format, and manipulate URLs. To use the url module, you need to require it at the beginning of your code:

```
const url = require('url');
```

Parsing a URL:

To parse a URL and extract its components (such as protocol, hostname, pathname, etc.), you can use the url.parse() method.

Example:

```
const url = require('url')
const urlString = 'https://www.example.com/users?id=1234';
const parsedUrl = url.parse(urlString, true);
console.log(parsedUrl.protocol); // Output: 'https:'
console.log(parsedUrl.hostname); // Output: 'www.example.com'
console.log(parsedUrl.pathname); // Output: '/users'
console.log(parsedUrl.query.id); // Output: '1234'
```

Creating a URL:

To create a URL from its components, you can use the url.format() method. Here's an

Example:

```
const url = require('url')
const urlObject = {
  protocol: 'https:',
  hostname: 'www.example.com',
  pathname: '/users',
  query: { id: '1234' }
};

const formattedUrl = url.format(urlObject);
console.log(formattedUrl); // Output: 'https://www.example.com/users?id=1234'
```

Resolving a URL:

The url.resolve() method is useful for resolving a relative URL against a base URL.

Example:

```
const url = require('url')
const baseUrl = 'https://www.example.com';
const relativeUrl = '/users';
```

```
const resolvedUrl = url.resolve(baseUrl, relativeUrl);  
console.log(resolvedUrl); // Output: 'https://www.example.com/users'
```

Parsing Query Parameters:

If you have a URL with query parameters and want to parse them into an object, you can use the `url.parse()` method with the `query` option set to `true`.

Example:

```
const url = require('url')  
const urlString = 'https://www.example.com/users?id=1234&name=John';  
const parsedUrl = url.parse(urlString, true);  
console.log(parsedUrl.query); // Output: { id: '1234', name: 'John' }
```

Query String Module

Parsing a Query String:

To parse a query string into an object, you can use the `querystring.parse()` method. It takes a query string as input and returns an object representing the key-value pairs.

Example:

```
const querystring = require('querystring');  
const queryString = 'id=1234&name=John&age=25';  
const parsedQuery = querystring.parse(queryString);  
console.log(parsedQuery);  
// Output: { id: '1234', name: 'John', age: '25' }
```

Formatting an Object into a Query String:

If you have an object representing key-value pairs and want to format it into a query string, you can use the `querystring.stringify()` method. It takes an object as input and returns a formatted query string.

Example:

```
const querystring = require('querystring');  
const obj = { id: '1234', name: 'John', age: '25' };  
const formattedQuery = querystring.stringify(obj);  
console.log(formattedQuery);  
// Output: 'id=1234&name=John&age=25'
```

Encoding and Decoding Query Strings:

The `querystring` module provides methods for encoding and decoding query strings to handle special characters. The `querystring.escape()` method can be used to encode a string, and the `querystring.unescape()` method can be used to decode a string.

Example:

```
const querystring = require('querystring');
const str = 'Hello World$';
const encodedStr = querystring.escape(str);
console.log(encodedStr); // Output: 'Hello%20World%24'
const decodedStr = querystring.unescape(encodedStr);
console.log(decodedStr); // Output: 'Hello World$ '
```

Creating a Custom Module

Create a new JavaScript file:

Start by creating a new JavaScript file that will serve as your custom module. For example, let's create a file named `mathUtils.js`.

Define functions or variables:

Inside the `mathUtils.js` file, define the functions, variables, or other entities that you want to export and make available to other parts of your application.

Here's an example where we define two basic math functions:

```
// mathUtils.js
function add(a, b) {
  return a + b;
}

function multiply(a, b) {
  return a * b;
}
```

Export the functions or variables:

To make the functions or variables accessible outside the module, you need to export them. In Node.js, you can use the `module.exports` object to export the desired entities.

Here's how you can export the add and multiply functions:

```
// mathUtils.js
function add(a, b) {
  return a + b;
}

function multiply(a, b) {
  return a * b;
}

module.exports = {
  add,
  multiply
};
```

Use the custom module in your application:

Now that you have created and exported your custom module, you can use it in other parts of your Node.js application. To use the module, you need to require it using the require function.

Example:

```
// main.js
const mathUtils = require('./mathUtils');

const sum = mathUtils.add(2, 3);
console.log(sum); // Output: 5

const product = mathUtils.multiply(4, 5);
console.log(product); // Output: 20
```

node.js Events:

- ✓ The EventEmitter class is a core module in Node.js that provides an event-driven architecture.
- ✓ It allows you to define, emit, and handle events in your Node.js applications.
- ✓ The class is an implementation of the Observer pattern, where objects can subscribe to and receive notifications when certain events occur.

Methods and Events of EventEmitter Class:

`on(event, listener)`: Adds a listener function to the specified event. Multiple listeners can be added for the same event.

Example:

```
const EventEmitter = require('events');
const myEmitter = new EventEmitter();

myEmitter.on('greet', () => {
  console.log('Hello!');
});
```

`emit(event, [args])`: Emits (triggers) the specified event, invoking all listeners attached to that event. Additional arguments can be passed to the listeners.

Example:

```
myEmitter.emit('greet');
// Output:
// Hello!
```

`once(event, listener)`: Adds a one-time listener function for the specified event. The listener will be removed after it is called once.

Example:

```
myEmitter.once('log', (message) => {
  console.log(`Logged: ${message}`);
});

myEmitter.emit('log', 'Hello, World!');
// Output:
// Logged: Hello, World!

myEmitter.emit('log', 'Again, World!');
// No output, as the listener has been removed after the first emission.
```

`removeListener(event, listener)`: Removes a specific listener function from the specified event.

Example:

```
const listener = () => {  
  console.log('Listener is called.');
```



```
};  
myEmitter.on('event', listener);  
myEmitter.emit('event');
```



```
// Output:  
// Listener is called.  
myEmitter.removeListener('event', listener);  
myEmitter.emit('event');
```



```
// No output, as the listener has been removed.
```

`removeAllListeners([event])`: Removes all listeners for the specified event or all listeners for all events.

Example:

```
myEmitter.removeAllListeners('greet');
```



```
// Removes all listeners for the 'greet' event.
```



```
myEmitter.removeAllListeners();  
// Removes all listeners for all events.
```

`eventNames()`: Returns an array of the event names for which listeners are registered.

Example:

```
const events = myEmitter.eventNames();  
console.log(events);
```



```
// Output:  
// ['greet', 'log']
```

`listenerCount(event)`: Returns the number of listeners for the specified event.

Example:

```
const count = myEmitter.listenerCount('greet');
```



```
console.log(count);
```



```
// Output:  
// 1
```

Connect to DB:

Step 1: initialize the node project.

1. Open your terminal or command prompt.
2. Navigate to the directory where you want to create your Node.js project.
3. Run the following command:
npm init -y
4. Once you have initialized your project, you can install the necessary dependencies such as the MongoDB Node.js driver (mongodb) using npm.
npm install mongodb
5. Connect the node.js to the mongoDB atlas connection.

```
const { MongoClient } = require('mongodb');

const uri = "mongodb+srv://username:password@your-cluster.mongodb.net/your-
database-name";

const client = new MongoClient(uri);
client.connect();
console.log("Connected to MongoDB successfully!");
```

6. Create a Database, collection, and insert a document

```
const { MongoClient } = require('mongodb');

async function connectToMongoDB() {
const uri = "mongodb+srv://username:password@your-cluster.mongodb.net/";
const client = new MongoClient(uri);

try {
  await client.connect();
  console.log("Connected to MongoDB successfully!");

  // Specify the name of the database
  const dbName = "mydatabase";

  // Get the reference to the database
  const database = client.db(dbName);

  // Specify the name of the collection
  const collectionName = "mycollection";

  // Get the reference to the collection
  const collection = database.collection(collectionName);

  // Define the document you want to insert
  const document = { name: "John", age: 30, email: "john@example.com" };
```

```
// Insert the document into the collection
const result = await collection.insertOne(document);
console.log(`Document inserted with _id: ${result.insertedId}`);
} catch (error) {
  console.error("Error connecting to MongoDB:", error);
} finally {
  // Ensure that the client will close when you finish/error
  await client.close();
  console.log("MongoDB connection closed.");
}
}
connectToMongoDB();
```