

# 15CSE337

## Cloud Computing and Services

Node.js

**Dr Ganesh Neelakanta Iyer**

Associate Professor, Dept of Computer Science and Engg

Amrita Vishwa Vidyapeetham, Coimbatore

# MEAN STACK

- The Friendly & Fun Fullstack JavaScript framework



MongoDB is the leading NoSQL database, empowering businesses to be more agile and scalable

Express is a minimal and flexible node.js web application framework, providing a robust set of features for building single and multi-page, and hybrid web applications

express



React is a JavaScript library for building user interfaces

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications



# Node.js

- Node.js is an open-source, cross-platform runtime environment used for development of server-side web 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 architecture and a non-blocking Input/Output API that is designed to optimize an application's throughput and scalability for real-time web applications

# Features

## **Asynchronous event driven IO helps concurrent request handling**

- This is probably the biggest selling points of Node.js
- This feature basically means that if a request is received by Node for some Input/Output operation, it will execute the operation in the background and continue with processing other requests

# Asynchronous event driven IO helps concurrent request handling

- This code looks at reading a file called Sample.txt. In other languages, the next line of processing would only happen once the entire file is read
- But in Node.js the important fraction of code to notice is the declaration of the function ('function(error,data)'). This is known as a callback function
- What happens? → file reading operation will start in the background
- And other processing can happen simultaneously while the file is being read. Once the file read operation is completed, this anonymous function will be called and the text "Reading Data completed" will be written to the console log

```
var fs = require('fs');  
fs.readFile("Sample.txt",function(error,data)  
{  
    console.log("Reading Data completed");  
});
```

# Features (cntd)

---

Node uses the V8 JavaScript Runtime engine

The one which is used by Google Chrome. Node has a wrapper over the JavaScript engine which makes the runtime engine much faster and hence processing of requests within Node also become faster.

---

Handling of concurrent requests

Another key functionality of Node is the ability to handle concurrent connections with a very minimal overhead on a single process.

---

The Node.js library used JavaScript

This is another important aspect of development in Node.js. A major part of the development community are already well versed in javascript, and hence, development in Node.js becomes easier for a developer who knows javascript.

---

There are an Active and vibrant community for the Node.js framework

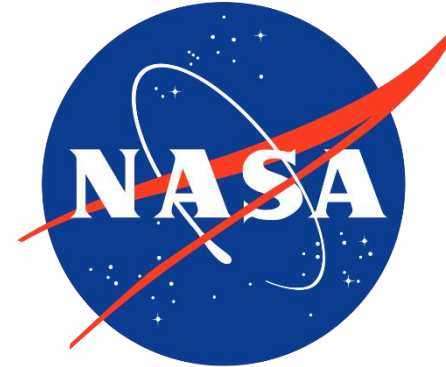
Because of the active community, there are always keys updates made available to the framework. This helps to keep the framework always up-to-date with the latest trends in web development.

---



# Words to describe node.js







# When to use Node.js

Best for usage in streaming or event-based real-time applications

- Chat applications
- Game servers – Fast and high-performance servers that need to process thousands of requests at a time, then this is an ideal framework.
- Good for collaborative environment – This is good for environments which manage documents. 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.
- Advertisement servers – Again here you could have thousands of requests to pull advertisements from the central server and Node.js can be an ideal framework to handle this.
- 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

# Installation and setup

- Installation must have been done as a prerequisite for React
- If not, visit <http://node.js.org> to install node.js
- You can check the installation by running the following command
- `node --version` in the command prompt

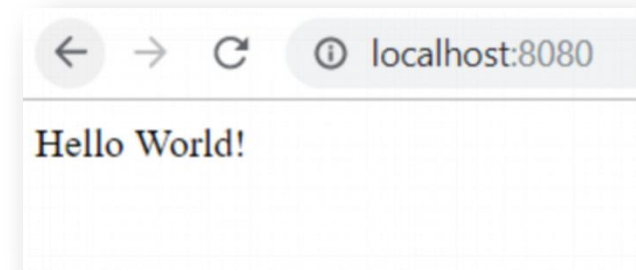
```
Giri@Giri-Iyer MINGW64 /d/Dropbox/  
Ganesh_CC_Materials_2019/Node.js  
$ node --version  
v10.15.3
```

# Running your first Hello world application in Node.js

- Create a folder (e.g. node) in your computer and copy this code into a file called firstprogram.js

```
1  var http = require('http');
2
3  http.createServer(function (req, res) {
4      res.writeHead(200, {'Content-Type': 'text/html'});
5      res.end('Hello World!');
6  }).listen(8080);
```

- In command prompt, run `node firstprogram.js`
- Open <https://localhost:8080>



# Modules in node.js

- Consider modules to be the same as JavaScript libraries.
- A set of functions you want to include in your application.
- Node.js has a set of built-in modules which you can use without any further installation.
- To include a module, use the `require()` function with the name of the module:

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

- Now your application has access to the HTTP module, and is able to create a server:

# createServer method

- HTTP server is created using the http module's createServer() method
- Like most Node.js functions, createServer() takes a callback function as an argument
- This callback function is executed each time the server receives a new request
- 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



# createServer method

- The callback function begins by calling the `res.writeHead()` method
- This method sends an HTTP status code and a collection of response headers back to the client
- The status code is used to indicate the result of the request
- For example, everyone has encountered a 404 error before, indicating that a page could not be found
- The example server returns the code 200, which indicates success

```
res.writeHead(200, { 'Content-Type': 'text/html' });
```

# createServer method

- Along with the status code, the server returns a number of HTTP headers which define the parameters of the response
- If you do not specify headers, Node.js will implicitly send them for you
- The example server specifies only the Content-Type header
- This particular header defines the MIME type of the response
- In the case of an HTML response, the MIME type is “text/html”.

```
res.writeHead(200, { 'Content-Type': 'text/html' });
```

# Lets Modify the file a bit

```
1 var http = require('http');
2
3 http.createServer(function (req, res) {
4     res.writeHead(200, {'Content-Type': 'text/html'});
5     res.write('Welcome'+ '<br/>');
6     res.end('Hello World!');
7 }) .listen(8080);
8
9 console.log('Node.js web server at port 8080 is
running.. Type https://localhost:8080 to verify the
same ')
```

# res.write

- Re-run the command `node firstprogram.js` and check the browser again
- Look at the line, the server `res.write()`
- This call is used to write the HTML page
- ‘<br>’ is used inside to break line after “Welcome”

# res.write

## Another sample

- Look at the lines, the server res.write()
- These calls are used to write the HTML page

```
var http = require("http");
var server = http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/html"});
  response.write("<!DOCTYPE html>");
  response.write("<html>");
  response.write("<head>");
  response.write("<title>Hello World Page</title>");
  response.write("</head>");
  response.write("<body>");
  response.write("Hello World!");
  response.write("</body>");
  response.write("</html>");
  response.end();
});

server.listen(80);
console.log("Server is listening");
```



# res.end

- After the HTML page has been written, the res.end() method is called
- By calling this method, we are telling the server that the response headers and body have been sent, and that the request has been fulfilled
- The example server call end() can be used with no parameters or like write(), assuming only one call is needed

# listen

- The call to `listen()` causes the server to bind to a port and listen for incoming connections
- Computers have thousands of ports, which act as communication end points
- In order to connect to the server, clients must know exactly which port the server is listening on
- Ports are identified by port numbers, with HTTP servers typically listening to port 80
- In this slide deck, did you notice two ways of calling listen?

# Thank you!

Dr Ganesh Neelakanta Iyer

ni\_amrita@cb.amrita.edu

ganesh.vigneswara@gmail.com



Office Hours

– Thursday 1350-1440  
@ My office