**Node.js**

*What is node js?*

* (server side technology stacking non blocking event driven io that can handle a lot of network traffic.)
* It is not a programming language nor a framework , but it’s a runtime environment for executing js code .
* Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine) for developing server side and networking application.
* It gives us the capabilities like work with the file system and network

*Back-end-services*

* Back end services are also known as API(Application Programming Interface)
* The app has to talk to some services in the server or cloud to store data, send emails or push notifications, kick off work flows etc.
* Node used to for building highly scalable, data intensive and real time back end services that powers the client app.

*Why not asp.net, rails, jango and so on?*

* Node is easy to get started and can be used for prototyping and agile development.
* It can also be used for building super fast and highly scalable services .
* It uses js , so it is easy for front end developer to reuse js and transition to a full stack developer.
* Since js is used at both front end and back end , the source code will be cleaner and more consistent so we can use same naming conventions, same tools and same best practices.
* There are free open source library for adding any features or building blocks to your code rather than writing code from scratch.

**Features of Node.js**

*Following are some of the important features that make Node.js the first choice of software architects.*

* **Asynchronous and Event Driven** − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
* **Very Fast** − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
* **Single Threaded but Highly Scalable** − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
* **No Buffering** − Node.js applications never buffer any data. These applications simply output the data in chunks.



*Where to Use Node.js?*

Following are the areas where Node.js is proving itself as a perfect technology partner.

* I/O bound Applications
* Data Streaming Applications
* Data Intensive Real-time Applications (DIRT)
* JSON APIs based Applications
* Single Page Applications

## *Where Not to Use Node.js?*

It is not advisable to use Node.js for CPU intensive applications.

*A Node.js application consists of the following three important components −*

* **Import required modules** − We use the **require** directive to load Node.js modules.
* **Create server** − A server which will listen to client's requests similar to Apache HTTP Server.
* **Read request and return response** − The server created in an earlier step will read the HTTP request made by the client which can be a browser or a console and return the response.

## *Creating Node.js Application*

### Step 1 - Import Required Module

* We use the **require** directive to load the http module and store the returned HTTP instance into an http variable as follows −
* var http = require("http");

### Step 2 - Create Server

We use the created http instance and call **http.createServer()** method to create a server instance and then we bind it at port 8081 using the **listen** method associated with the server instance. Pass it a function with parameters request and response. Write the sample implementation to always return "Hello World".

http.createServer(function (request, response) {

// Send the HTTP header

// HTTP Status: 200 : OK

// Content Type: text/plain

response.writeHead(200, {'Content-Type': 'text/plain'});

// Send the response body as "Hello World"

response.end('Hello World\n');

}).listen(8081);

// Console will print the message

console.log('Server running at http://127.0.0.1:8081/');

The above code is enough to create an HTTP server which listens, i.e., waits for a request over 8081 port on the local machine.

### Step 3 - Testing Request & Response

Let's put step 1 and 2 together in a file called **main.js** and start our HTTP server as shown below –

var http = require("http");

http.createServer(function (request, response) {

// Send the HTTP header

// HTTP Status: 200 : OK

// Content Type: text/plain

response.writeHead(200, {'Content-Type': 'text/plain'});

// Send the response body as "Hello World"

response.end('Hello World\n');

}).listen(8081);

// Console will print the message

console.log('Server running at http://127.0.0.1:8081/');

Now execute the main.js to start the server as follows −

$ node main.js

Verify the Output. Server has started.

Server running at http://127.0.0.1:8081/

## Make a Request to the Node.js Server

Open http://127.0.0.1:8081/ in any browser and observe the following result.



Congratulations, you have your first HTTP server up and running which is responding to all the HTTP requests at port 8081.

*Steps to run the program: (in visual studio code)*

* Debugger -> add configuration ->node.js launch program
* Go to the file that you want to launch(rename to app.js [default the name is app.js otherwise under add configuration -> program : mention the name you want])
* Then click on debugger and press play button (start debugging icon).

*Imports and exports modules:*

*Modules:*

myFirstModule.js

var consoleLog = (value) =>{

console.log(value);

}

module.exports= {

consoleLog //consoleLog : consoleLog

}

app.js

var myModule = require(‘./myFirstModule’);

myModule.consoleLog(“ragavi”);

* it has a built in package manager named npm (it’s a repository or server that delivers bundled js code on request
* npm install express
* express is a web framework
* it generates package-lock.json

*Building a website*

to build the website using node.js it’s best to use the express js framework

app.js

const express = require(‘express’);

const app = express();

var port = 3000;

app.get(‘/’,(req,res) =>res.send(‘<a href=”<https://www.google.com> “ >click me </a>’));

app.listen(port, () => console.log(‘we are listening on port ${port} ));

*Interacting with the file system:*

fs: file system

app.js

const express = require(‘express’);

const app = express();

const fs = require(‘fs’);

var port = 3000;

app.get(‘/’,(req,res) =>{

fs.readFile(‘./template/index,html’ , utf-8 , (err,data) => {

if(err){

console.log(“oops!”);

}

res.send(data);

});

});

app.listen(port, () => console.log(‘we are listening on port ${port} ));

templates folder -> index.html

index.html

<a href=”<https://www.google.com> “ >click me </a>

Why node but not c# (sharp) or ruby or any other programming languages?

Why node but not asp.net,rails, django or any other framework ?

Because they are all synchronous and blocking whereas node is non blocking and asynchronous

Single thread :

When a request arrives that single thread is used to handle the request ( if we need to query a db , the thread doesn’t have to wait for the db to return the data . while the db is executing the query , that thread will be used to handle/serve other request . when the db prepares the result it puts a message in the event queue .the single thread is continuously monitoring the queue in the background. When it find events in this queue (data ready) , it will take it out and process it.) so this makes node js useful in building application that include lot if disks and network access. So we can serve more clients without need to throw in more hardware. (highly scalable and intensive i/o applications).

CPU intensive applications like video encoding or image manipulation service because these involve a lot of cpu calculation rather than working with files and network .when a thread performs the calculations to serve one client , the other client should wait.