What is Node? and Why to use it

Node.js is JavaScript run time framework built on top of Google V8 engine. One of the key reason to use Node is to perform asynchronous I/O operation.

What is the area of problem Node addresses

- Key areas of problem Node addresses:
- Slow web servers due to synchronous I/O.
- Multithreaded programming bottlenecks.
- Scaling.
- Package management and deployment.
- Why Node uses Google V8 Engine
- Google V8 is Chrome runtime engine which converts JavaScript code into native machine code which in turn provides a fast running application.

Can we use other engines than V8

Yes. Microsoft Chakra is another JavaScript engine which can be used with Node.js. It’s not officially declared yet.

If Node is single threaded then how it handles concurrency

Node provides a single thread to programmers so that code can be written easily and without bottleneck. Node internally uses multiple POSIX threads for various I/O operations such as File, DNS, Network calls etc.

When Node gets I/O request it creates or uses a thread to perform that I/O operation and once the operation is done, it pushes the result to the event queue. On each such event, event loop runs and checks the queue and if the execution stack of Node is empty then it adds the queue result to execution stack.

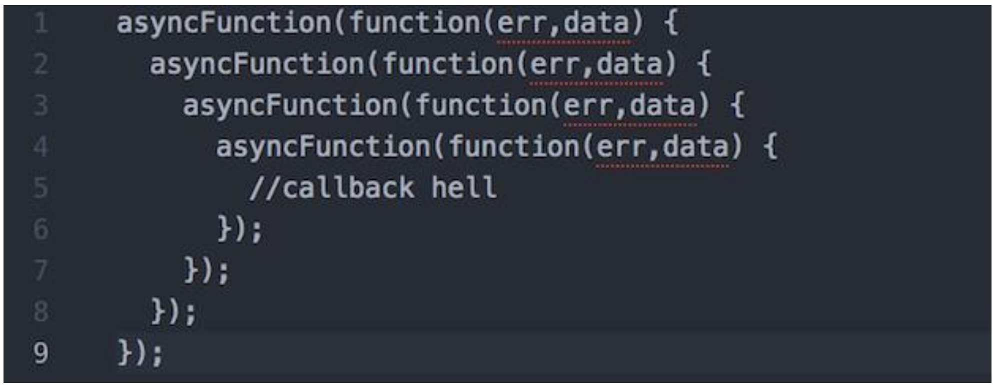
This is how Node manages concurrency.

Explain event loop

As said, Node.js is single threaded but it supports concurrency using events and callbacks. The event loop is a component which checks event queue on a periodic basis and pushes the result in execution stack if the stack is empty.

What is callback hell

The asynchronous function requires callbacks as a return parameter. When multiple asynchronous functions are chained together then callback hell situation comes up. Consider the example code below.



```
asyncFunction(function(err,data) {
  asyncFunction(function(err,data) {
    asyncFunction(function(err,data) {
      asyncFunction(function(err,data) {
        //callback hell
      });
    });
  });
});
```

This situation is referred to as “callback hell” situation.

Which is your preferred way to write asynchronous code in Node

Here is a list of methods I generally use to avoid callback hell and write asynchronous code.

- Using Async module.
- Using promises.
- Using function decomposition.
- Using async/await.

I personally prefer Async npm module.

What is stream and explain types of streams

Streams are a collection of data that might not be available all at once and don't have to fit in memory. Streams provide chunks of data in a continuous manner. It is useful to read a large set of data and process it.

There is 4 fundamental type of streams:

- Readable.
- Writeable.
- Duplex.
- Transform.

Readable streams as the name suggest used in reading a large chunk of data from a source. Writable streams are used in writing a large chunk of data to the destination.

Duplex streams are both readable and writable (Eg socket). Transform stream is the duplex stream which is used in modifying the data (eg zip creation).

Explain child processes in Node

Child process module enables us to access operating system functionaries. Scalability is baked into Node and child processes are the key factors to scale our application. You can use child process to run system commands, read large files without blocking event loop, decompose the application into various "nodes" (That's why it's called Node).

Can we send/receive messages between child processes

Yes, we can.

We can use send() function to send message to workers and receive the response on process.on('message')event.

Explain file system module of Node

Node fs module provides simple file system module to perform files related operation. This module comprises of synchronous and asynchronous functions to read/write files. For example, readFile() function is asynchronous function to read file content from specified path and readFileSync() is synchronous function to read files.

How to scale Node application

We can scale Node application in following ways:

cloning using cluster module.

Decomposing the application into smaller services – i.e micro services.

How to deploy Node application

You should know how to deploy Node application on various cloud providers. If you know the basics, such as SSH access, git cloning and running the application in process manager then more or less the steps are same in various cloud providers.

What is Node.js?

Node.js is a web application framework built on Google Chrome's JavaScript Engine (V8 Engine).

Node.js comes with runtime environment on which a Javascript based script can be interpreted and executed (It is analogus to JVM to JAVA byte code). This runtime allows to execute a JavaScript code on any machine outside a browser. Because of this runtime of Node.js, JavaScript is now can be executed on server as well.

Node.js = Runtime Environment + JavaScript Library

What is global installation of dependencies?

Globally installed packages/dependencies are stored in /npm directory. Such dependencies can be used in CLI (Command Line Interface) function of any node.js but can not be imported using require() in Node application directly. To install a Node project globally use -g flag.

What is an error-first callback?

Error-first callbacks are used to pass errors and data. The first argument is always an error object that the programmer has to check if something went wrong. Additional arguments are used to pass data.

```
fs.readFile(filePath, function(err, data) {
  if (err) {
    //handle the error
  }
  // use the data object
});
```

What are the benefits of using Node.js?

Following are main benefits of using Node.js

Aynchronous and Event DrivenAll APIs of Node.js library are aynchronous that is non-blocking. It essentially means a Node.js based server never waits for a API to return data. Server moves to next API after calling it and a notification mechanism of Events of Node.js helps server to get 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 server to respond in a non-bloking ways and makes server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and same program can services much larger number of requests than traditional server like Apache HTTP Server.

No Buffering - Node.js applications never buffer any data. These applications simply output the data in chunks.

If Node.js is single threaded then how it handles concurrency?

Node provides a single thread to programmers so that code can be written easily and without bottleneck. Node internally uses multiple POSIX threads for various I/O operations such as File, DNS, Network calls etc.

When Node gets I/O request it creates or uses a thread to perform that I/O operation and once the operation is done, it pushes the result to the event queue. On each such event, event loop runs and checks the queue and if the execution stack of Node is empty then it adds the queue result to execution stack.

This is how Node manages concurrency.

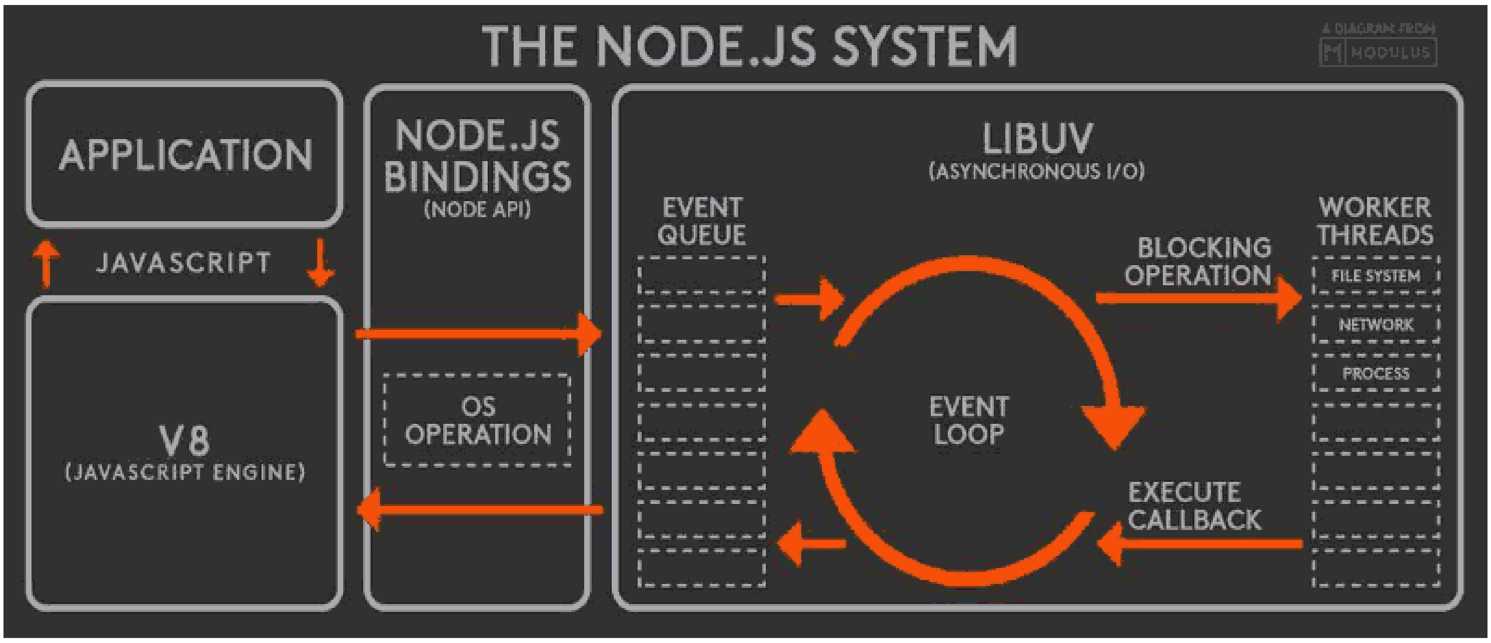
How can you avoid callback hells?

To do so you have more options:

- modularization: break callbacks into independent functions
- use *Promises*
- use yield with *Generators* and/or *Promises*

What's the event loop?

The event loop is what allows Node.js to perform non-blocking I/O operations — despite the fact that JavaScript is single-threaded — by offloading operations to the system kernel whenever possible.



Every I/O requires a callback - once they are done they are pushed onto the event loop for execution. Since most modern kernels are multi-threaded, they can handle multiple operations executing in the background. When one of these operations completes, the kernel tells Node.js so that the appropriate callback may be added to the poll queue to eventually be executed.

How Node prevents blocking code?

By providing callback function. Callback function gets called when

What is Event Emmitter?

All objects that emit events are members of EventEmitter class. These objects expose an eventEmitter.on() function that allows one or more functions to be attached to named events emitted by the object.

When the EventEmitter object emits an event, all of the functions attached to that specific event are called synchronously.

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

class MyEmitter extends EventEmitter {}

const myEmitter = new MyEmitter();
myEmitter.on('event', () => {
  console.log('an event occurred!');
});
myEmitter.emit('event');
```

What is Node.js?

Wikipedia defines Node.js as “an open-source, cross-platform runtime environment for developing server-side Web applications.” It is essentially server side scripting which is used to build scalable programs.

Features of Node JS

Features	Description
Fast	Node.js is built on Google Chrome’s V8 JavaScript Engine which makes its library very fast in code execution
Asynchronous	Node.js based server never waits for an API to return data thus making it asynchronous
Scalable	It is highly scalable because of its event mechanism which helps the server to respond in a non-blocking way
Open Source	Node.js has an extensive open source community which has contributed in producing some excellent modules to add additional capabilities to Node.js applications
No Buffering	Node.js applications simply output the data in chunks and never buffer any data.

What is the relation of Node.js with JavaScript?

Though Node.js is not a JavaScript framework, many of its modules are written in JavaScript.

It allows the developers to create new modules in JavaScript.

Node.js is a virtual machine that leverages JavaScript as its scripting language to achieve high output.

What is the fundamental difference between Node.js and Ajax?

While Ajax is a client-side technology, Node.js is a server-side JavaScript environment.

Explain the term I/O in the context of Node.js.

I/O stands for input and output. It is used to access anything outside of the application. I/O gets loaded on to the machine memory in order to run programs after the application is fired up.

State where Node.js can be used.

- Web apps
- Network applications
- Distributed systems
- General purpose applications

Which are the two types of API functions in Node.js?

Asynchronous, non-blocking functions and Synchronous, blocking functions.

List the tools and IDEs that are used for Node.js.

- Atom
- Nodeclipse Enide Studio
- JetBrains WebStorm
- JetBrains InteliJ IDEA

- MS Visual Studio
- NoFLo

Explain the role of the Callback function in Node.js.

In Node.js, the Callback function is used to cater to multiple requests made to the server. If there is a large file that is expected to take the server a long time to process, the Callback function can be invoked to ensure that other requests to the server are uninterrupted.

What is an error-first callback?

Error-first callbacks are essentially used to pass errors and data. By default, the first argument is always an error object, where the user needs to check if something is wrong.

In the context of Node.js, differentiate between operational and programmer errors.

Operational errors are not real errors. They are system errors, for example “request timeout” or “hardware error”. Programmer errors are actual bugs in the code.

Which is the framework that is most commonly used in Node.js?

The most commonly used Node.js framework is “Express”.

Define “event-driven programming.

It is essentially a programming paradigm where the program flow is characterized by events such as messages from other programs.

What are the two sections of event-driven programming?

Event Selection and Event Handling are two sections of event-driven programming.

List some of the big advantages of using Node.js.

- Ability to build scalable programs
- Increased concurrency
- Asynchronous capabilities

Explain the Control Flow function.

It is a generic piece of code that runs concurrently between several asynchronous function calls.

List the steps involved in the Control Flow function.

Control the order of execution à Collect data à Limit concurrency à Call the next program step

Can a user access DOM in a Node?

No, you cannot access DOM.

In Node.js, how do you access the last expression?

We have to use the underscore (_) character to access the last expression.

In Node.js, which command is used to import external libraries?

A command called “require” is used for importing external libraries.

What is the biggest drawback of Node.js?

The biggest drawback is the fact that it is challenging to have one process with a single thread to scale up on multi core servers.

What is event loop in JavaScript?

Event loop, as the name suggests is loop which runs in background listening for various event emitters.

What is a callback?

Callback is a piece of code which is passed as an argument to another function (say ***fn1***) to execute the callback function (say ***fn2***) after executing the function ***fn1***. This is used to handle the asynchronous behaviour of javascript.

The functionality of a callback can be imagined as somewhat **analogous to*Interfaces* in JAVA**.

What is callback hell and how can it be avoided?

Callback hell refers to a coding pattern where there is a lot of nesting of callback functions. The code forms a pyramid like structure and it becomes difficult to debug.

It can be avoided by:

- *Using promises*
- *Yield operator and Generator functions from ES6*
- *Modularizing code*
- *Using **async** library* (http://caolan.github.io/async/)

What is a Promise?

A promise is a method that eventually produces a value. Instead of passing a function as argument as with callbacks, "Once the result is received from an asynchronous operation **then** the required function is executed". The required functions are never passed as arguments to the asynchronous operation.

It has the form:

```
callAsyncFunction()
.then(firstFunction)
.then(secondFunction)
.then(thirdFunction)
.then(fourthFunction);
```

Same thing using callbacks

```
callAsyncFunction( function(err, result) {
  firstFunction(err1, result, function(result2){
    secondFunction(err2, result2, function(result3){
      thirdFunction(err3, result3, function(result4){
        fourthFunction(err4, result4){
          //Final code here
        });
      });
    });
  });
});
```

P.S. This pyramid like code structure depicts a Callback Hell. It can make debugging very difficult in large projects.

What are ACID properties wrt to a database?

ACID stands for **Atomicity**, **Consistency**, **Isolation** and **Durability.**

Atomicity

It means that either the record/document updates completely or does not update with respect to the operation.

Either everything occurs or nothing occurs.

Consistency

It implies that a transaction either creates a new and valid state of data, or, if any failure occurs, returns all data to its state before the transaction was started.

If successful then valid data else revert back to original data.

Isolation

Two or more simultaneous database queries should run independent of the other transactions.

Durability

Data once committed should be saved in memory so that even in case of system failure, data is not affected.

Where and why should you not use NodeJS?

Now this one is pretty interesting question. I have heard answers like:

"NodeJS should not be used because it is not secure as other low level language based web servers as it uses JavaScript which is a client side scripting language"

To everyone out there, this is NOT the reason. You can make NodeJS servers as secure as you want. It depends on your requirements and your code.

To answer this question, **NodeJS should NOT be used where computations are CPU intensive.** Eg: Data Analytics Server, Image Processing Servers, Video Processing Servers etc.

NodeJS is meant for highly I/O-bound operations which does not require heavy CPU intensive operation/tasks.

Some Additional facts which you may be asked:

MongoDB is not ACID compliant

This means that MongoDB does not guarantee any of the ACID properties.

NodeJS is a **Runtime Environment**. It is NOT a JavaScript framework.

Provide some example of config file separation for dev and prod environments

A perfect and flawless configuration setup should ensure:

keys can be read from file AND from environment variable

secrets are kept outside committed code

config is hierarchical for easier findability

Consider the following config file:

```
var config = {
  production: {
    mongo : {
      billing: '****'
    }
  },
  default: {
    mongo : {
      billing: '****'
    }
  }
}
```

```
exports.get = function get(env) {
  return config[env] || config.default;
}

And it's usage:

const config = require('./config/config.js').get(process.env.NODE_ENV);
const dbconn = mongoose.createConnection(config.mongo.billing);
```

What are the timing features of Node.js?

The Timers module in Node.js contains functions that execute code after a set period of time.

- **setTimeout/clearTimeout** - can be used to schedule code execution after a designated amount of milliseconds
- **setInterval/clearInterval** - can be used to execute a block of code multiple times
- **setImmediate/clearImmediate** - will execute code at the end of the current event loop cycle
- **process.nextTick** - used to schedule a callback function to be invoked in the next iteration of the Event Loop

```
function cb(){
  console.log('Processed in next iteration');
}
process.nextTick(cb);
console.log('Processed in the first iteration');

Output:
```

Processed in the first iteration
Processed in next iteration

Explain what is Reactor Pattern in Node.js?

Reactor Pattern is an idea of non-blocking I/O operations in Node.js. This pattern provides a handler(in case of Node.js, a *callback function*) that is associated with each I/O operation. When an I/O request is generated, it is submitted to a *demultiplexer*.

This *demultiplexer* is a notification interface that is used to handle concurrency in non-blocking I/O mode and collects every request in form of an event and queues each event in a queue. Thus, the demultiplexer provides the *Event Queue*.

At the same time, there is an Event Loop which iterates over the items in the Event Queue. Every event has a callback function associated with it, and that callback function is invoked when the Event Loop iterates.

What is LTS releases of Node.js why should you care?

An **LTS(Long Term Support)** version of Node.js receives all the critical bug fixes, security updates and performance improvements.

LTS versions of Node.js are supported for at least 18 months and are indicated by even version numbers (e.g. 4, 6, 8). They're best for production since the LTS release line is focussed on stability and security, whereas the *Current* release line has a shorter lifespan and more frequent updates to the code. Changes to LTS versions are limited to bug fixes for stability, security updates, possible npm updates, documentation updates and certain performance improvements that can be demonstrated to not break existing applications.

Why should you separate Express 'app' and 'server'?

Keeping the API declaration separated from the network related configuration (port, protocol, etc) allows testing the API in-process, without performing network calls, with all the benefits that it brings to the table: fast testing execution and getting coverage metrics of the code. It also allows deploying the same API under flexible and different network conditions. Bonus: better separation of concerns and cleaner code.

API declaration, should reside in app.js:

```
var app = express();
app.use(bodyParser.json());
app.use("/api/events", events.API);
app.use("/api/forms", forms);
```

Server network declaration, should reside in /bin/www:

```
var app = require('../app');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);
```

What is the difference between process.nextTick() and setImmediate() ?

The difference between process.nextTick() and setImmediate() is that process.nextTick() defers the execution of an action till the next pass around the event loop or it simply calls the callback function once the ongoing execution of the event loop is finished whereas setImmediate()executes a callback on the next cycle of the event loop and it gives back to the event loop for executing any I/O operations.

Rewrite the code sample without try/catch block

Consider the code:

```
async function check(req, res) {
  try {
    const a = await someOtherFunction();
    const b = await somethingElseFunction();
    res.send("result")
  } catch (error) {
    res.send(error.stack);
  }
}
```

Rewrite the code sample without try/catch block.

Answer:

```
async function getData(){
  const a = await someFunction().catch((error)=>console.log(error));
  const b = await someOtherFunction().catch((error)=>console.log(error));
  if (a && b) console.log("some result")
}
```
or if you wish to know which specific function caused error:

```
async function loginController() {
  try {
    const a = await loginService().
    catch((error) => {
      throw new CustomErrorHandler({
        code: 101,
        message: "a failed",
        error: error
      })
    });
    const b = await someUtil().
    catch((error) => {
      throw new CustomErrorHandler({
        code: 102,
        message: "b failed",
        error: error
      })
    });
    //someoeeoe
    if (a && b) console.log("no one failed")
  } catch (error) {
    if (!(error instanceof CustomErrorHandler)) {
      console.log("gen error", error)
    }
  }
}
```

What is an error-first callback?

Error-first callbacks are used to pass errors and data. The first argument is always an error object that the programmer has to check if something went wrong. Additional arguments are used to pass data.

```
fs.readFile(filePath, function(err, data) {
  if (err) {
    //handle the error
  }
  // use the data object
});
```

How does this question help?

The answer for this question will get you some insight on whether the candidate has some basic knowledge on how async operations work in Node.

How can you avoid callback hells?

To do so you have more options:

- **modularization**: break callbacks into independent functions
- use *Promises*
- use `yield` with *Generators* and/or *Promises*

How does this question help?

The answer for this question may vary a lot, depending on how up-to-date one is, how closely is she following the latest developments, be it ES6, ES7 or just a new control flow library.

How can you listen on port 80 with Node?

Trick question! You should not try to listen with Node on port 80 *(in Unix-like systems)* - to do so you would need superuser rights, but it is not a good idea to run your application with it.

Still, if you want to have your Node.js application listen on port 80, here is what you can do. Run the application on any port above 1024, then put a reverse proxy like nginx in front of it.

How does this question help?

This question helps you to find out whether the one you are talking to has any experience operating Node applications.

What's the event loop?

TL;DR:

It is a magical place filled with unicorns and rainbows - Trevor Norris

Node.js runs using a single thread, at least from a Node.js developer's point of view. Under the hood Node.js uses many threads through libuv.

Every I/O requires a callback - once they are done they are pushed onto the event loop for execution. If you need a more detailed explanation, I suggest viewing this video:

How does this question help?

This will give you an insight on how deep someone's knowledge on Node is, if she/he knows what libuv is.

What tools can be used to assure consistent style?

You have plenty of options to do so:

JSLint by Douglas Crockford

JSHint

ESLint

JSCS

These tools are really helpful when developing code in teams, to enforce a given style guide and to catch common errors using static analysis.

What's the difference between operational and programmer errors?

Operation errors are not bugs, but problems with the system, like *request timeout*or *hardware failure*.

On the other hand programmer errors are actual bugs.

Why npm shrinkwrap is useful?

This command locks down the versions of a package's dependencies so that you can control exactly which versions of each dependency will be used when your package is installed. - npmjs.com

It is useful when you are deploying your Node.js applications - with it you can be sure which versions of your dependencies are going to be deployed.

What's a stub? Name a use case.

Stubs are functions/programs that simulate the behaviours of components/modules. Stubs provide canned answers to function calls made during test cases. Also, you can assert on with what these stubs were called.

A use-case can be a file read, when you do not want to read an actual file:

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

var readFileStub = sinon.stub(fs, 'readFile', function (path, cb) {
  return cb(null, 'filecontent');
});

expect(readFileStub).to.be.called;
readFileStub.restore();
```

What's a test pyramid? How can you implement it when talking about HTTP APIs?

A test pyramid describes that when writings test cases there should be a lot more low-level unit tests than high level end-to-end tests.

When talking about HTTP APIs, it may come down to this:

a lot of low-level unit tests for your models

less integration tests, where your test how your models interact with each other

a lot less acceptance tests, where you test the actual HTTP endpoints

What's your favourite HTTP framework and why?

There is no right answer for this. The goal here is to understand how deeply one knows the framework she/he uses, if can reason about it, knows the pros, cons.

Things that work better than these questions

As you may already guessed, we are not huge fans of these type of questions. Instead we do believe in **small, real-life problems**, solved together. During these you will get a **very good understanding of how one thinks**. But not just that. You will know if she/he **is a good fit for your team**, as you have to solve something together.

When we are hiring *(and we are always hiring)* we usually look for a combination of the following:

- cultural fit
- transparency
- self-improvement
- bias towards clarity
- do things smarter than harder
- skill and expertise

What Is Node.Js?

Node.js is a JavaScript runtime or platform which is built on Google Chrome's JavaScript v8 engine. This runtime allows executing the JavaScript code on any machine outside a browser (this means that it is the server that executes the Javascript and not the browser).

Node.js is single-threaded, that employs a concurrency model based on an event loop. It doesn't block the execution instead registers a callback which allows the application to continue. It means Node.js can handle concurrent operations without creating multiple threads of execution so can scale pretty well.

It uses JavaScript along with C/C++ for things like interacting with the filesystem, starting up HTTP or TCP servers and so on. Due to it's extensively fast growing community and NPM, Node.js has become a very popular, open source and cross-platform app. It allows developing very fast and scalable network app that can run on Microsoft Windows, Linux, or OS X.

Following are the areas where it’s perfect to use Node.js.

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

At the same time, it’s not suitable for heavy applications involving more of CPU usage.

What Are The Key Features Of Node.Js?

Let’s look at some of the key features of Node.js.

Asynchronous event driven IO helps concurrent request handling – All APIs of Node.js are asynchronous. This feature means that if a Node receives a request for some Input/Output operation, it will execute that operation in the background and continue with the processing of other requests. Thus it will not wait for the response from the previous requests.

Fast in Code execution – Node.js 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.js also become faster.

Single Threaded but Highly Scalable – Node.js uses a single thread model for event looping. The response from these events may or may not reach the server immediately. However, this does not block other operations. Thus making Node.js highly scalable. Traditional servers create limited threads to handle requests while Node.js creates a single thread that provides service to much larger numbers of such requests.

Node.js library uses JavaScript – This is another important aspect of Node.js from the developer’s point of view. The majority of developers are already well-versed in JavaScript. Hence, development in Node.js becomes easier for a developer who knows JavaScript.

There is an Active and vibrant community for the Node.js framework – The active community always keeps the framework updated with the latest trends in the web development.

No Buffering – Node.js applications never buffer any data. They simply output the data in chunks.

Explain How Do We Decide, When To Use Node.Js And When Not To Use It?

When Should We Use Node.Js?

It’s ideal to use Node.js for developing streaming or event-based real-time applications that require less CPU usage such as.

Chat applications.

Game servers.

Node.js is good for fast and high-performance servers, that face the need to handle thousands of user requests simultaneously.

Good For A Collaborative Environment.

It is suitable for environments where multiple people work together. For example, they post their documents, modify them by doing check-out and check-in of these documents.

Node.js supports such situations by creating an event loop for every change made to the document. The “Event loop” feature of Node.js enables it to handle multiple events simultaneously without getting blocked.

Advertisement Servers.

Here again, we have servers that handle thousands of request for downloading advertisements from a central host. And Node.js is an ideal solution to handle such tasks.

Streaming Servers.

Another ideal scenario to use Node.js is for multimedia streaming servers where clients fire request’s towards the server to download different multimedia contents from it.

To summarize, it’s good to use Node.js, when you need high levels of concurrency but less amount of dedicated CPU time.

Last but not the least, since Node.js uses JavaScript internally, so it fits best for building client-side applications that also use JavaScript.

When To Not Use Node.Js?

However, we can use Node.js for a variety of applications. But it is a single threaded framework, so we should not use it for cases where the application requires long processing time. If the server is doing some calculation, it won’t be able to process any other requests. Hence, Node.js is best when processing needs less dedicated CPU time.

What IDEs Can You Use For Node.Js Development?

Here is the list of most commonly used IDEs for developing node.js applications.

Cloud9.

It is a free, cloud-based IDE that supports, application development, using popular programming languages like Node.js, PHP, C++, Meteor and more. It provides a powerful online code editor that enables a developer to write, run and debug the app code.

JetBrains WebStorm.

WebStorm is a lightweight yet powerful JavaScript IDE, perfectly equipped for doing client-side and server-side development using Node.js. The IDE provides features like intelligent code completion, navigation, automated and safe refactorings. Additionally, we can use the debugger, VCS, terminal and other tools present in the IDE.

JetBrains InteliJ IDEA.

It is a robust IDE that supports web application development using mainstream technologies like Node.js, Angular.js, JavaScript, HTML5 and more. To enable the IDE that can do Node.js development we have to install a Node.js plugin. It provides features, including syntax highlighting, code assistance, code completion and more. We can even run and debug Node.js apps and see the results right in the IDE. It’s JavaScript debugger offers conditional breakpoints, expression evaluation, and other features.

Komodo IDE.

It is a cross-platform IDE that supports development in main programming languages, like Node.js, Ruby, PHP, JavaScript and more. It offers a variety of features, including syntax highlighting, keyboard shortcuts, collapsible Pane, workspace, auto indenting, code folding and code preview using built-in browser.

Eclipse.

It is a popular cloud-based IDE for web development using Java, PHP, C++ and more. You can easily avail the features of Eclipse IDE using the Node.js plug-in, which is <**nodeclipse**>.

Atom.

It is an open source application built with the integration of HTML, JavaScript, CSS, and Node.js. It works on top of Electron framework to develop cross-platform apps using web technologies. Atom comes pre-installed with four UI and eight syntax themes in both dark and light colors. We can also install themes created by the Atom community or create our own if required.

Explain How Does Node.Js Work?

A Node.js application creates a single thread on its invocation. Whenever Node.js receives a request, it first completes its processing before moving on to the next request.

Node.js works asynchronously by using the event loop and callback functions, to handle multiple requests coming in parallel. An Event Loop is a functionality which handles and processes all your external events and just converts them to a callback function. It invokes all the event handlers at a proper time. Thus, lots of work is done on the back-end, while processing a single request, so that the new incoming request doesn't have to wait if the processing is not complete.

While processing a request, Node.js attaches a callback function to it and moves it to the back-end. Now, whenever its response is ready, an event is called which triggers the associated callback function to send this response.

Let's Take An Example Of A Grocery Delivery.

Usually, the delivery boy goes to each and every house to deliver the packet. Node.js works in the same way and processes one request at a time. The problem arises when any one house is not open. The delivery boy can't stop at one house and wait till it gets opened up. What he will do next, is to call the owner and ask him to call when the house is open. Meanwhile, he is going to other places for delivery. Node.js works in the same way. It doesn't wait for the processing of the request to complete (house is open). Instead, it attaches a callback function (call from the owner of the house) to it. Whenever the processing of a request completes (the house is open), an event gets called, which triggers the associated callback function to send the response.

To summarize, Node.js does not process the requests in parallel. Instead, all the back-end processes like, I/O operations, heavy computation tasks, that take a lot of time to execute, run in parallel with other requests.

Explain REPL In Node.Js?

The REPL stands for "Read Eval Print Loop". It is a simple program that accepts the commands, evaluates them, and finally prints the results. REPL provides an environment similar to that of Unix/Linux shell or a window console, in which we can enter the command and the system, in turn, responds with the output. REPL performs the following tasks.

READ

It Reads the input from the user, parses it into JavaScript data structure and then stores it in the memory.

EVAL

It Executes the data structure.

PRINT

It Prints the result obtained after evaluating the command.

LOOP

It Loops the above command until the user presses Ctrl+C two times.

Is Node.Js Entirely Based On A Single-Thread?

Yes, it’s true that Node.js processes all requests on a single thread. But it’s just a part of the theory behind Node.js design. In fact, more than the single thread mechanism, it makes use of events and callbacks to handle a large no. of requests asynchronously.

Moreover, Node.js has an optimized design which utilizes both JavaScript and C++ to guarantee maximum performance. JavaScript executes at the server-side by Google Chrome v8 engine. And the C++ lib UV library takes care of the non-sequential I/O via background workers.

To explain it practically, let’s assume there are 100s of requests lined up in Node.js queue. As per design, the main thread of Node.js event loop will receive all of them and forwards to background workers for execution. Once the workers finish processing requests, the registered callbacks get notified on event loop thread to pass the result back to the user.

How To Get Post Data In Node.Js?

Following is the code snippet to fetch Post Data using Node.js.

```
app.use(express.bodyParser());
app.post('/', function(request, response){
console.log(request.body.user);
});
```

How To Make Post Request In Node.Js?

Following code snippet can be used to make a Post Request in Node.js.

```
var request = require('request');
request.post(
'http://www.example.com/action',
{ form: { key: 'value' } },
function (error, response, body) {
if (!error && response.statusCode == 200) {
console.log(body)
}
}
);
```

What Is Callback In Node.Js?

We may call “callback” as an asynchronous equivalent for a function. Node.js makes heavy use of callbacks and triggers it at the completion of a given task. All the APIs of Node.js are written in such a way that they support callbacks.

For example, suppose we have a function to read a file, as soon as it starts reading the file, Node.js return the control immediately to the execution environment so that the next instruction can be executed. Once file read operation is complete, it will call the callback function and pass the contents of the file as its arguments. Hence, there is no blocking or wait, due to File I/O. This functionality makes Node.js as highly scalable, using it processes a high number of requests without waiting for any function to return the expected result.

What Is Callback Hell?

Initially, you may praise Callback after learning about it. Callback hell is heavily nested callbacks which make the code unreadable and difficult to maintain.

Let’s see the following code example.

```
downloadPhoto('http://coolcats.com/cat.gif', displayPhoto)
function displayPhoto (error, photo) {
if (error) console.error('Download error!', error)
else console.log('Download finished', photo)
}
console.log('Download started')
```

In this scenario, Node.js first declares the “displayPhoto” function. After that, it calls the “downloadPhoto” function and pass the “displayPhoto” function as its callback. Finally, the code prints ‘Download started’ on the console. The “displayPhoto” will be executed only after “downloadPhoto” completes the execution of all its tasks.

How To Avoid Callback Hell In Node.Js?

Node.js internally uses a single-threaded event loop to process queued events. But this approach may lead to blocking the entire process if there is a task running longer than expected.

Node.js addresses this problem by incorporating callbacks also known as higher-order functions. So whenever a long-running process finishes its execution, it triggers the callback associated. With this approach, it can allow the code execution to continue past the long-running task.

However, the above solution looks extremely promising. But sometimes, it could lead to complex and unreadable code. More the no. of callbacks, longer the chain of returning callbacks would be. Just see the below example.

With such an unprecedented complexity, it’s hard to debug the code and can cause you a whole lot of time. There are four solutions which can address the callback hell problem.

1. Make Your Program Modular.

It proposes to split the logic into smaller modules. And then join them together from the main module to achieve the desired result.

2. Use Async Mechanism.

It is a widely used Node.js module which provides a sequential flow of execution.

The async module has <async.waterfall> API which passes data from one operation to other using the next callback.

Another async API <async.map> allows iterating over a list of items in parallel and calls back with another list of results.

With the async approach, the caller’s callback gets called only once. The caller here is the main method using the async module.

3. Use Promises Mechanism.

Promises give an alternate way to write async code. They either return the result of execution or the error/exception. Implementing promises requires the use of <.then()> function which waits for the promise object to return. It takes two optional arguments, both functions. Depending on the state of the promise only one of them will get called. The first function call proceeds if the promise gets fulfilled. However, if the promise gets rejected, then the second function will get called.

4. Use Generators.

Generators are lightweight routines, they make a function wait and resume via the yield keyword. Generator functions uses a special syntax <function* ()>. They can also suspend and resume asynchronous operations using constructs such as promises or <thunks> and turn a synchronous code into asynchronous.

Can You Create HTTP Server In Nodejs, Explain The Code Used For It?

Yes, we can create HTTP Server in Node.js. We can use the <**http-server**> command to do so.

Following is the sample code.

```
var http = require('http');
var requestListener = function (request, response) {
response.writeHead(200, {'Content-Type': 'text/plain'});
response.end('Welcome Viewers\n');
}
var server = http.createServer(requestListener);
server.listen(8080); // The port where you want to start with.
```

What Is The Difference Between Nodejs, AJAX, And JQuery?

The one common trait between Node.js, AJAX, and jQuery is that all of them are the advanced implementation of JavaScript. However, they serve completely different purposes.

Node.Js –

It is a server-side platform for developing client-server applications. For example, if we've to build an online employee management system, then we won't do it using client-side JS. But the Node.js can certainly do it as it runs on a server similar to Apache, Django not in a browser.

AJAX (Aka Asynchronous Javascript And XML) –

It is a client-side scripting technique, primarily designed for rendering the contents of a page without refreshing it. There are a no. of large companies utilizing AJAX such as Facebook and Stack Overflow to display dynamic content.

JQuery –

It is a famous JavaScript module which complements AJAX, DOM traversal, looping and so on. This library provides many useful functions to help in JavaScript development. However, it's not mandatory to use it but as it also manages cross-browser compatibility, so can help you produce highly maintainable web applications.

What Are Globals In Node.Js?

There are three keywords in Node.js which constitute as Globals. These are Global, Process, and Buffer.

Global.

The Global keyword represents the global namespace object. It acts as a container for all other <global> objects. If we type <console.log(global)>, it'll print out all of them.

An important point to note about the global objects is that not all of them are in the global scope, some of them fall in the module scope. So, it's wise to declare them without using the var keyword or add them to Global object.

Variables declared using the var keyword become local to the module whereas those declared without it get subscribed to the global object.

Process.

It is also one of the global objects but includes additional functionality to turn a synchronous function into an async callback. There is no boundation to access it from anywhere in the code. It is the instance of the EventEmitter class. And each node application object is an instance of the Process object.

It primarily gives back the information about the application or the environment.

<process.execPath> – to get the execution path of the Node app.

<process.Version> – to get the Node version currently running.

<process.platform> – to get the server platform.

Some of the other useful Process methods are as follows.

<process.memoryUsage> – To know the memory used by Node application.

<process.NextTick> – To attach a callback function that will get called during the next loop. It can cause a delay in executing a function.

Buffer.

The Buffer is a class in Node.js to handle binary data. It is similar to a list of integers but stores as a raw memory outside the V8 heap.

We can convert JavaScript string objects into Buffers. But it requires mentioning the encoding type explicitly.

<ascii> – Specifies 7-bit ASCII data.

<utf8> – Represents multibyte encoded Unicode char set.

<utf16le> – Indicates 2 or 4 bytes, little endian encoded Unicode chars.

<base64> – Used for Base64 string encoding.

<hex> – Encodes each byte as two hexadecimal chars.

Here is the syntax to use the Buffer class.

> var buffer = new Buffer(string, [encoding]);

The above command will allocate a new buffer holding the string with <utf8> as the default encoding. However, if you like to write a <string> to an existing buffer object, then use the following line of code.

> buffer.write(string)

This class also offers other methods like <readInt8> and <writeUInt8> that allows read/write from various types of data to the buffer.

How To Load HTML In Node.Js?

To load HTML in Node.js we have to change the "Content-type" in the HTML code from text/plain to text/html. Let's see an example where we have created a static file in web server.

```
fs.readFile(filename, "binary", function(err, file) {
if(err) {
response.writeHead(500, {"Content-Type": "text/plain"});
response.write(err + "\n");
response.end();
return;
}

response.writeHead(200);
response.write(file, "binary");
response.end();
});
```

Now we will modify this code to load an HTML page instead of plain text.

```
fs.readFile(filename, "binary", function(err, file) {
if(err) {
response.writeHead(500, {"Content-Type": "text/html"});
response.write(err + "\n");
response.end();
return;
}

response.writeHead(200, {"Content-Type": "text/html"});
response.write(file);
response.end();
});
```

What Is EventEmitter In Node.Js?

Events module in Node.js allows us to create and handle custom events. The Event module contains "EventEmitter" class which can be used to raise and handle custom events. It is accessible via the following code.

// Import events module

var events = require('events');

// Create an eventEmitter object

var eventEmitter = new events.EventEmitter();

When an EventEmitter instance encounters an error, it emits an "error" event. When a new listener gets added, it fires a "newListener" event and when a listener gets removed, it fires a "removeListener" event.

EventEmitter provides multiple properties like "on" and "emit". The "on" property is used to bind a function to the event and "emit" is used to fire an event.

How Many Types Of Streams Are Present In Node.Js?

Stream in Node.js are objects that allow reading data from a source or writing data to a specific destination in a continuous fashion. In Node.js, there are four types of streams.

<Readable> – This is the Stream to be used for reading operation.

<Writable> – It facilitates the write operation.

<Duplex> – This Stream can be used for both the read and write operations.

<Transform> – It is a form of a duplex Stream, which performs the computations based on the available input.

All the Streams, discussed above are an instance of an "EventEmitter" class. The event thrown by the Stream varies with time. Some of the commonly used events are as follows.

<data> – This event gets fired when there is data available for reading.

<end> – The Stream fires this event when there is no more data to read.

<error> – This event gets fired when there is any error in reading or writing data.

<finish> – It fires this event after it has flushed all the data to the underlying system.

List And Explain The Important REPL Commands?

Following is the list of some of the most commonly used REPL commands.

<.help> – It displays help for all the commands.

<tab Keys> – It displays the list of all the available commands.

<Up/Down Keys> – Its use is to determine what command was executed in REPL previously.

<.save filename> – Save the current REPL session to a file.

<.load filename> – To Load the specified file in the current REPL session.

<ctrl + c> – used to Terminate the current command.

<ctrl + c (twice)> – To Exit from the REPL.

<ctrl + d> – This command perfoms Exit from the REPL.

<.break> – It leads Exitting from multiline expression.

<.clear> – Exit from multiline expression.

What Is NPM In Node.Js?

NPM stands for Node Package Manager. It provides following two main functionalities.

It works as an Online repository for node.js packages/modules which are present at <nodejs.org>.

It works as Command line utility to install packages, do version management and dependency management of Node.js packages.

NPM comes bundled along with Node.js installable. We can verify its version using the following command-

```
$ npm --version
```

NPM helps to install any Node.js module using the following command.

```
$ npm install <Module Name>
```

For example, following is the command to install a famous Node.js web framework module called express-

```
$ npm install express
```

What Is The Global Installation Of Dependencies?

Globally installed packages/dependencies are stored in <user-directory>/npm directory. Such dependencies can be used in CLI (Command Line Interface) function of any node.js, but cannot be imported using require() in the Node application directly.

To install a Node project globally use -g flag as.

```
C:\Nodejs_WorkSpace>npm install express -g
```

What Is The Local Installation Of Dependencies?

By default, NPM installs any dependency in the local mode. It means that the package gets installed in "node_modules" directory which is present in the same folder, where Node application is placed. Locally deployed packages are accessible via require(). Following is the syntax to install a Node project locally.

C:\Nodejs_WorkSpace>npm install express

What Is <Package.Json>?

It is a plain JSON (JavaScript Object Notation) text file which contains all metadata information about Node.js Project or application.

This file should be present in the root directory of every Node.js Package or Module to describe its metadata in JSON format.

The file is named as "package" because Node.js platform treats every feature as a separate component. Node.js calls these as Package or Module.

Who Use It?

NPM (Node Package Manager) uses <package.json> file. It includes details of the Node.js application or package. This file contains a no. of different directives or elements. These directives guide NPM, about how to handle a module or package.

Does Node.Js Support Multi-Core Platforms? And Is It Capable Of Utilizing All The Cores?

Yes, Node.js would run on a multi-core system without any issue. But it is by default a single-threaded application, so it can't completely utilize the multi-core system.

However, Node.js can facilitate deployment on multi-core systems where it does use the additional hardware. It packages with a Cluster module which is capable of starting multiple Node.js worker processes that will share the same port.

Which Is The First Argument Usually Passed To A Node.Js Callback Handler?

Node.js core modules follow a standard signature for its callback handlers and usually the first argument is an optional error object. And if there is no error, then the argument defaults to null or undefined.

Here is a sample signature for the Node.js callback handler.

```
function callback(error, results) {
// Check for errors before handling results.
if ( error ) {
// Handle error and return.
}
// No error, continue with callback handling.
}
```

What Is Chaining Process In Node.Js?

It's an approach to connect the output of one stream to the input of another stream, thus creating a chain of multiple stream operations.

How To Create A Custom Directive In AngularJS?

To create a custom directive, we have to first register it with the application object by calling the <directive> function. While invoking the <register> method of <directive>, we need to give the name of the function implementing the logic for that directive.

For example, in the below code, we have created a copyright directive which returns a copyright text.

```
app.directive('myCopyRight', function ()
{
return
{
template: '@CopyRight MyDomain.com '
};
});
```

Note – A custom directive should follow the camel case format as shown above.

What Is A Child_process Module In Node.Js?

Node.js supports the creation of child processes to help in parallel processing along with the event-driven model.

The Child processes always have three streams <child.stdin>, child.stdout, and child.stderr. The <stdio> stream of the parent process shares the streams of the child process.

Node.js provides a <child_process> module which supports following three methods to create a child process.

exec – <child_process.exec> method runs a command in a shell/console and buffers the output.

spawn – <child_process.spawn> launches a new process with a given command.

fork – <child_process.fork> is a special case of the spawn() method to create child processes.

What Are The Different Custom Directive Types In AngularJS?

AngularJS supports a no. of different directives which also depend on the level we want to restrict them.

So in all, there are four different kinds of custom directives.

Element Directives (E)

Attribute Directives (A)

CSS Class Directives (C)

Comment Directives (M)

What Is A Control Flow Function? What Are The Steps Does It Execute?

It is a generic piece of code which runs in between several asynchronous function calls is known as control flow function.

It executes the following steps.

Control the order of execution.

Collect data.

Limit concurrency.

Call the next step in the program.