*Nodejs*

**Node.js** is a JavaScript runtime built on Chrome’s V8 JavaScript engine.

**V8 is the engine that compiles JavaScript code into C++** and then (when seeing native modules like “fs”, “crypto”, “http”, …) utilizing a library called **libuv** which gives easy access to the underlying operating system, including its multi threading capabilities.

**Note:** **libuv is actually also the library that implements the infamous EVENT LOOP**.

This library handles Node’s asynchronous I/O operation and main event loop. There are thread pool reserve in Libuv which handles the thread allocation to individual I/O operations.

**libuv** is managing something that is called a “thread pool”.  
Each of those threads in this pool can get assigned (by the event loop) a specific task from your ***Event Queue*** and work on it concurrently.

 

*Multi-threaded node js single threaded env*

**It is 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.

**Super Fast :**

Being built on Google Chrome’s V8 JavaScript Engine, Node.js is super efficient and quick in code execution.

**Is node single threaded ?**

Yes NodeJS is single threaded, but this is a half truth, actually it is event-driven and single-threaded with background workers. The main event loop is single-threaded but most of the I/O works run on separate threads, because the I/O APIs in Node.js are asynchronous/non-blocking by design, in order to accommodate the event loop.

*setImmediate() === settimeout(()=>{},0)*

**EVENT LOOP**



**16. What are “streams” in Node.js? Explain the different types of streams present in Node.js.**

Streams are objects that allow reading of data from the source and writing of data to the destination as a continuous process.

By default the data you read from a stream is a Buffer object. streams are EventEmitters, they emit several events at various points. We will use these events to work with the streams. (e.g. 'error','end','data')

There are four types of streams.

* to facilitate the reading operation
* to facilitate the writing operation
* to facilitate both read and write operations
* is a form of Duplex stream that performs computations based on the available input



Here we use fs and zlib to read a zipped file, unzip it and write unzipped contents somewhere.

**4 JS designm Patterns**

1. Module Design Pattern
2. Prototype Design Pattern
3. Observer Design Pattern
4. Singleton

**Wrapper func of node modules**



All Node modules are wrapped inside a IIFE function

**Node.js Globals Object**

* process
* global
* module.exports and exports

**mongoDB is Document Oriented**

**Q13: Find objects between two dates MongoDB**



**Pros(Mongo)**

* schema-less. If you have a flexible schema, this is ideal for a document store like MongoDB. This is difficult to implement in a performant manner in RDBMS
* ease of scale-out. Scale reads by using replica sets. Scale writes by using sharding (auto balancing). Just fire up another machine and away you go. Adding more machines = adding more RAM over which to distribute your working set.
* cost. Depends on which RDBMS of course, but MongoDB is free and can run on Linux, ideal for running on cheaper commodity kit.
* you can choose what level of consistency you want depending on the value of the data (e.g. faster performance = fire and forget inserts to MongoDB, slower performance = wait til insert has been replicated to multiple nodes before returning)

**Cons(Mongo)**

* Data size in MongoDB is typically higher due to e.g. each document has field names stored it
* less flexibity with querying (e.g. no JOINs)
* no support for transactions - certain atomic operations are supported, at a single document level
* at the moment Map/Reduce (e.g. to do aggregations/data analysis) is OK, but not blisteringly fast. So if that's required, something like Hadoop may need to be added into the mix
* less up to date information available/fast evolving product

**$elemMatch in mongo**



**aggregation with $group**

