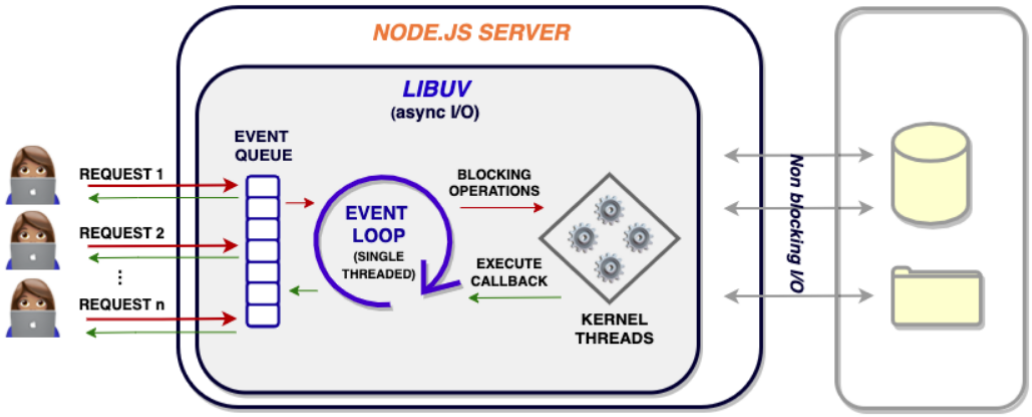
NodeJS Architecture and Best Practices

How Things Work

> *Every time there's a client request, it is handled by a single main thread.*
>
> *The **event loop** is the primary component that allows Node.js to run (otherwise) blocking I/O operations in a non-blocking way. It constantly keeps track of the status of your asynchronous tasks (eg. the code in your callback functions) and moves them back to the execution queue when they're completed. It operates in the same main thread we have been talking about.*
>
> *The interesting thing to note here is that even though there's just one main thread on the surface, there are a bunch of auxiliary threads in the system kernel that Node.js can utilize for extensive disk and network-based async operations. This group of threads constitutes (what is known as) the worker pool.*



> *The event loop can take care of basic processing itself, but for async I/O operations, involving modules such asfs(I/O-heavy) and cry pto(CPU-heavy), it can offload the processing to the worker pool in the system kernel. The worker pool is implemented in libuv and can spawn and manage multiple threads as per the requirement. These threads can individually run their respective assigned tasks in a synchronous manner and return their response to the event loop whenever ready. While these threads work on their assigned operations, the event loop can continue operating as usual, concurrently catering to other requests. When the threads are done with their tasks, they can return their output to the event loop, which can then place this back on the execution queue to be executed or returned back to the client.*
>
> *The thought process behind adopting such an architecture can be attributed to the fact that under typical web loads, a single main thread can perform and scale much better as compared to conventional "one thread per request" architectures. As a result, Node.js is the go-to option for many because of it's advantages in terms of speed and scalability. The caveat here however, is that performance can suffer for upfront complex, memory intensive operations like matrix multiplications for image processing, data science and machine learning applications. These can block the one and only main thread, making the server unresponsive. However, for such cases, Node.js has also introduced worker threads which developers can leverage to create efficient multi-threaded Node.js applications.*

Common Folder Structure

```
src
       app.js                        app entry point
       /api                               controller layer: api routes
       /config                         config settings, env variables
       /services                         service layer: business logic
       /models                         data access layer: database models
       /scripts                         miscellaneous NPM scripts
       /subscribers                async event handlers
       /test                test suites
```

References

Reference	URL
Node.js Architecture and 12 Best Practices for Node.js Development	https://scoutapm.com/blog/nodejs-architecture-and-12-best-practices-for-nodejs-development