**About Node.js®**

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. “Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.” As an asynchronous event driven JavaScript runtime, Node is designed to build scalable network applications. In the following "hello world" example, many connections can be handled concurrently. Upon each connection a callback is fired, but if there is no work to be done Node is sleeping.

const http = require('http');

const hostname = '127.0.0.1';

const port = 8000;

const server = http.createServer((req, res) => {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

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

});

server.listen(port, hostname, () => {

console.log(`Server running at http://${hostname}:${port}/`);

});

This is in contrast to today's more common concurrency models where OS threads are employed. Threading model vary depending upon the operating system in use. Thread-based networking is relatively inefficient and very difficult to use. Multi-threaded applications require specific programming expertise and can be difficult to develop. Furthermore, users of Node.js are free from worries of dead-locking the process, since there are no locks. Almost no function in Node.js directly performs I/O, so the process never blocks. Because nothing blocks, scalable systems are very reasonable to develop in Node.

Node is similar in design to, and influenced by, systems like Ruby's Event Machine or Python's Twisted. Node takes the event model a bit further. It presents an event loop as a runtime construct instead of as a library. In other systems there is always a blocking call to start the event-loop. Typically behavior is defined through callbacks at the beginning of a script and at the end starts a server through a blocking call like EventMachine::run(). In Node there is no such start-the-event-loop call. Node simply enters the event loop after executing the input script. Node exits the event loop when there are no more callbacks to perform. This behavior is like browser JavaScript — the event loop is hidden from the user. In Node HTTP is designed with streaming and low latency in mind. This makes Node well suited for the foundation of a web library or framework.

Even without threads, node can use multiple cores. Child processes can be spawned by using the child\_process.fork() API, and are designed to be easy to communicate with. Built upon that same interface is the cluster module, which allows you to share sockets between processes to enable load balancing over your cores.

Following are some of the important features that make Node.js the first choice of software architects (see [www.tutorialspoint.com](http://www.tutorialspoint.com))

**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.

**Fast Performance** − 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.

**License** − Node.js is released under the MIT license