Node.js & mongoDB

29 October 2018

Jonathan Lenchner,
Chief Scientist, IBM Research Africa

JS
node
Run Anywhere
High Reliability
Versatile
Fast Processing Times
Easy Integration With JS Stacks

Node.js– Short History

- Open source JavaScript engine developed by Google
- Now in V8, though officially still in Beta
- The engine behind Google's Chrome browser
- Created by Ryan Dahl in 2009
- Very popular with many well known tech companies (used in some of their key services):
  - Google
  - Netflix
  - LinkedIn
  - Trello
  - Uber
  - PayPal
  - Medium
  - eBay
  - Microsoft
  - Github
  - Groupon

Node.js– What is it? (Basic Answer)

- Server-side JavaScript
- Highly optimized for concurrent access
- Also a command line tool
- One of its main goals is to provide an easy way to build scalable web applications

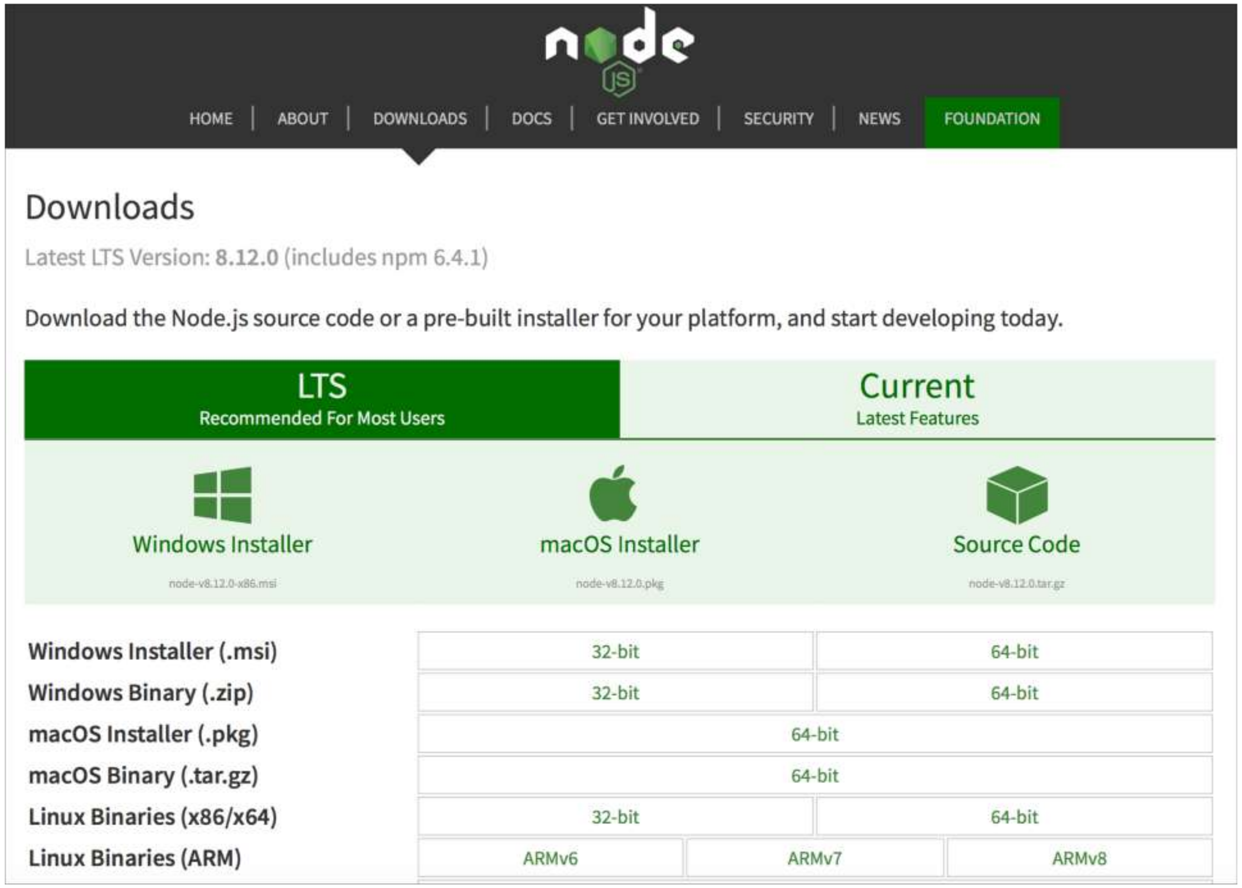
Node.js – What is it? (Advanced Answer)

- An event-driven, **non-blocking I/O** model of programming
- Makes use of event-loops via JavaScript's **callback functionality** to implement the non-blocking I/O
- Programs for Node.js are written in JavaScript but not in the same JavaScript you may be use to.
- There is no DOM implementation provided by Node.js, i.e. you cannot do this

```
var element = document.getElementById("elementId");
```

Installing Node

- Go to https://nodejs.org/en/download/ and install either the Windows or Mac versions (there is also a Linux version and version for ARM processors)



HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | NEWS | FOUNDATION

Downloads

Latest LTS Version: 8.12.0 (includes npm 6.4.1)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS
Recommended For Most Users

Current
Latest Features

Windows Installer
node-v8.12.0-x86.msi

macOS Installer
node-v8.12.0.pkg

Source Code
node-v8.12.0.tar.gz

Windows Installer (.msi)	32-bit		64-bit
Windows Binary (.zip)	32-bit		64-bit
macOS Installer (.pkg)	64-bit		
macOS Binary (.tar.gz)	64-bit		
Linux Binaries (x86/x64)	32-bit		64-bit
Linux Binaries (ARM)	ARMv6	ARMv7	ARMv8

Hello World! in Node

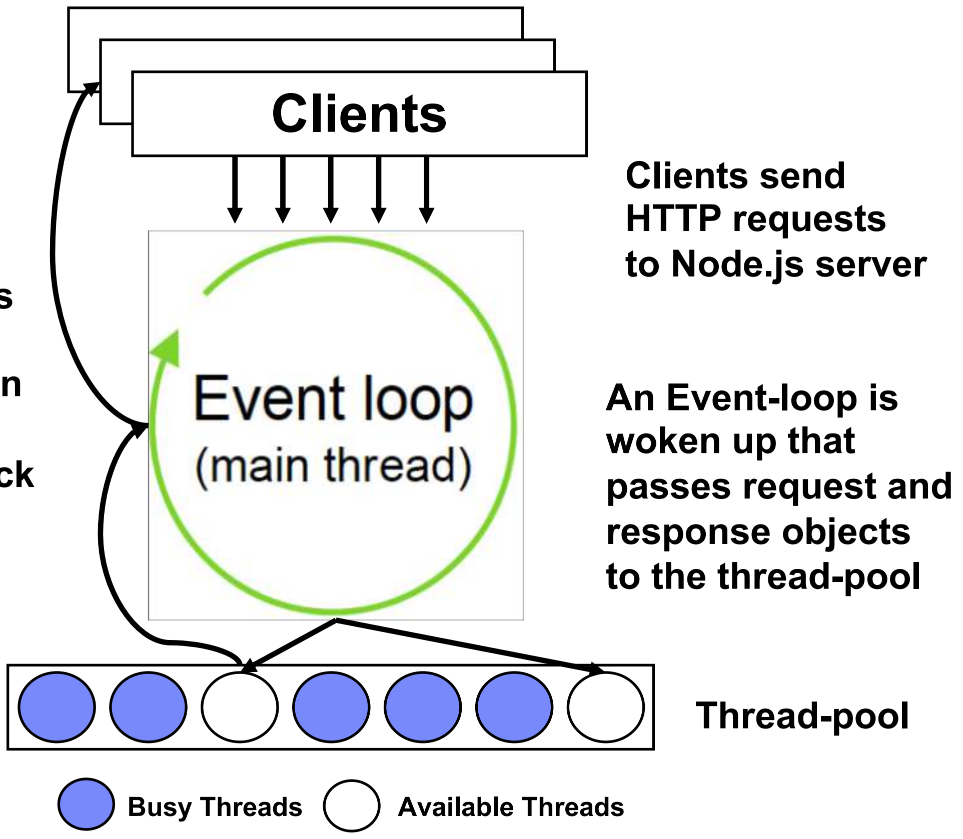
- Before creating an actual "Hello, World!" application using Node.js, useful to learn about the components of a Node.js application.
- A Node.js application consists of the following 3 important components:
  - **Import required modules**: One uses the **require** directive to load Node.js modules. Modules provide various node.js services.
  - **Create server** − A server which will listen to client requests.
  - **Read request and return response** − The server, created in the prior step will read the HTTP request made by the client, typically a, and return the response.

Hello World! in Node

```
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'});

       //alternatively could send 'text/html'

       // 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/');
```

Event Loops

- Event-loops are the core of event-driven programming
- Almost all UI programs use event-loops to track and respond to user events, such as mouse clicks, keyboard entry, etc.


Clients
Clients send HTTP requests to Node.js server
Response is sent back to main thread via a callback and then returned to the client
Event loop
(main thread)
An Event-loop is woken up that passes request and response objects to the thread-pool
Thread-pool
● Busy Threads
○ Available Threads

Long-running jobs run on worker threads in the thread-pool

Threads

- In computing, a **thread** of execution is a sequence of programming steps that are managed independently by a scheduler – an integral part of the computer's operating system.
- A fundamental distinction in programming and operating systems is that of **processes** versus **threads**.
- An example of a process is an application, or a background task
  - You know about apps
  - Examples of background tasks on your phone:
    - GPS service
    - Bluetooth service
    - Accelerometer service
- Processes can contain multiple threads that execute concurrently, sharing resources such as memory and access to common variables

Threads

- Systems with a single processor implement multiple threads by time slicing.
  - The context switching happens very often and rapidly so users perceive the threads as running in parallel.
- On a multi processor system multiple threads can execute in parallel, each processor (also called a **core**) executing a single thread.
  - However, as request for new threads increase, the individual cores end up doing their own time slicing.

Block I/O vs. Non-Blocking I/O

- Traiditional I/O

```
var result = db.query(“select x from table_Y”);
doSomethingWith(result); //wait for result!
doSomethingWithoutResult(); //execution is blocked!
   //no way to do something while you wait for the results
```

- Non-traditional, Non-blocking I/O

```
db.query(“select x from table_Y”,function (result){
   doSomethingWith(result); //wait for result!
});
doSomethingWithOutResult(); //executes without delay!
```

Block I/O vs. Non-Blocking I/O

- Traiditional I/O

```
var result = db.query(“select x from table_Y”);
doSomethingWith(result); //wait for result!
doSomethingWithoutResult(); //execution is blocked!
   //no way to do something while you wait for the results
```

- Non-traditional, Non-blocking I/O

```
db.query(“select x from table_Y”,function (result){
   doSomethingWith(result); //wait for result!
});
doSomethingWithOutResult(); //executes without delay!
```

Call-back function: Calls you back when the operation is done!

What can you do with Node.js?

- Can create an HTTP server and write arbitrarily complex web applications
- Can create a TCP (socket-based) server for direct communication and arbitrary data exchange
- Can create a File server
- Can create online games, and many more things…. The limit is just your imagination.

Difference between an HTTP server and a TCP server

- An **HTTP server** is used for the usual web paradigm
  - Server sits waiting for requests
  - Client briefly opens a connection to the server, makes a request, waits for a response, and then closes the connection.
- With a **TCP server**, also known as a **socket server**, clients open <u>longer lived connections</u>.
  - Both clients and servers can send each other any sort of data at any time.
  - Anyone can terminate a connection at any time.
  - Messaging apps often use this protocol.

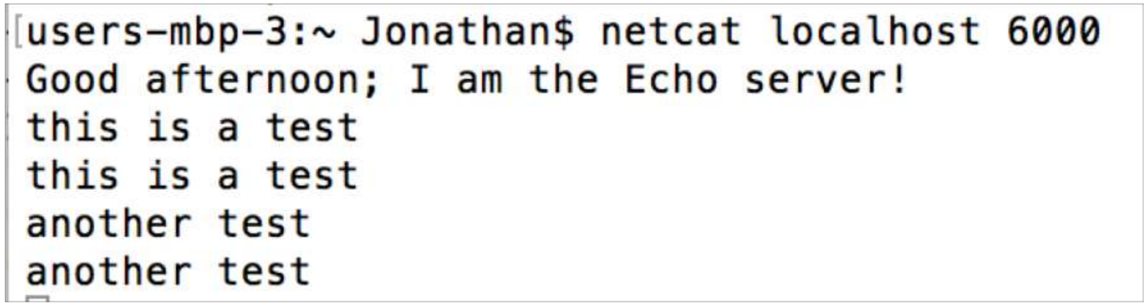
Simple TCP Server App

```
var net = require('net');
net.createServer(function (socket) {
   socket.write("Good afternoon; I am the Echo
server!\r\n");
socket.pipe(socket); }).listen(6000, "127.0.0.1");
```

- The last line of this code reads an input stream ending with a carriage return and redirects ("pipes") it to output.
- Thus, this little app starts up saying "Good afternoon; I am the Echo server!" and then echoes anything a client sends to it.
- To test this app, we use a freely available command line TCP client called netcat.
  - Google "netcat installation" to see how to install it on your system

Sample TCP Server App

- From a command line type:



```
users-mbp-3:~ Jonathan$ netcat localhost 6000
Good afternoon; I am the Echo server!
this is a test
this is a test
another test
another test
```

Use of Modules in Node.js

- Node.js packages a lot of useful code into so-called **modules**
- In order to utilize a module in your code you use the **require** keyword
- For example,

```
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);
```

enabled us to create an HTTP server

Use of Modules in Node.js

- And

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

net.createServer(function (socket) {

   socket.write(”Good afternoon; I am the Echo
server!\r\n");

socket.pipe(socket); }).listen(6000, "127.0.0.1");
```

enabled us to create an TCP (socket) server.

- To discover more of the functionality of a given module, Google (node [module_name]

Use of Modules in Node.js

- You can also create a module of your own code
- Put your JavaScript code in a separate .js file and include it in your code by using the **require** keyword, e.g.

```
var my_module = require('./my_module');
```

- There are also special libraries in Node.js with additional function that can be installed using the Node Package Manager (NPM) – NPM comes as part of the Node.js installation
  - Syntax: npm install "package_name";
  - For a list of available node packages goto https://www.npmjs.com
  - Discover packages for mobile development, IoT, and more

A Couple of Little Node.js apps: A Cumulative Adder

```
var http = require("http");
var url = require("url");
var sum = 0;

http.createServer(function (request, response) {
        // Send the HTTP header
        // HTTP Status: 200 : OK
        // Content Type: text/plain
        response.writeHead(200,{'Content-Type': 'text/plain'});
        var query = url.parse(request.url, true).query;
        console.log(query.num);
        console.log(sum);
        if(!isNaN(query.num)) {
                sum = sum + parseInt(query.num);
        }


        // Send a response giving the sum
response.end('Adder sum = ' + sum + '\n'); }).listen(8082);


// Console will print the message
console.log('Server running at http://127.0.0.1:8082/');
```

*Note how we are using a new module here – the URL module, which allows us to parse out the parameters passed in with the HTTP request.

A Couple of Little Node.js apps: Computing the Factorial of a Number

- Given N, compute N! = N * N-1 * … * 1

```
var http = require("http");
var url = require("url");


http.createServer(function (request, response) {
    // Send the HTTP header
    // HTTP Status: 200 : OK
    // Content Type: text/plain
    var query = url.parse(request.url, true).query;
    var n =  parseInt(query.num);
    // Send a response giving the factorial: n!
    response.writeHead(200,{'Content-Type': 'text/plain'});
response.end(n + '! = ' + fact(n) + '\n'); }).listen(8083);


// Console will print the message
console.log('Factorial server running at http://127.0.0.1:8083/');


function fact(n) {
    if(n > 1) {
        return n*fact(n-1);
    }
    else {
        return 1;
    }
}
```

Some Useful Frameworks for Node.js

- **Express** – extremely popular framework for creating web and mobile applications in node

  npm install express

- **Meteor** – another extremely popular full-stack framework for creating web and mobile applications in node. Especially popular for multi-user real time game creationg

  npm install meteor

- **React** – a JavaScript library for building user interfaces that works especially well with node

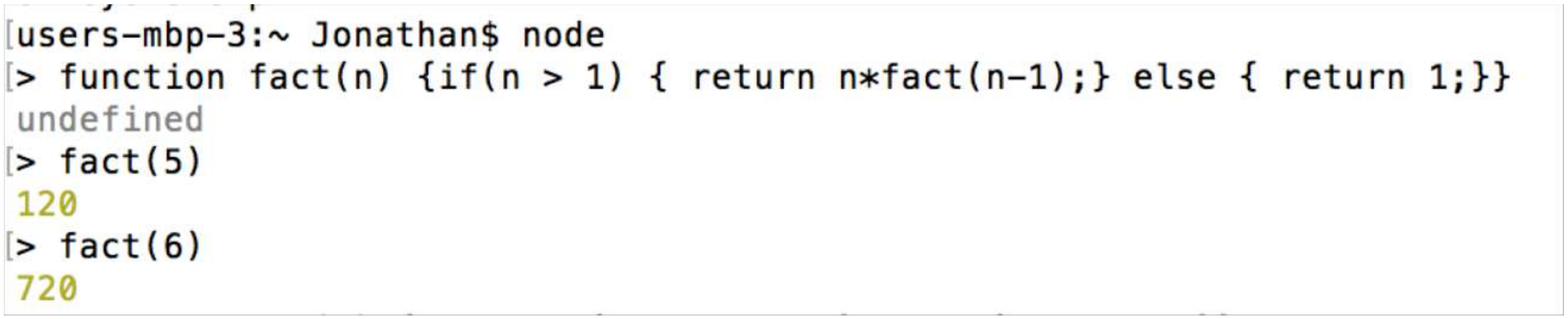
  npm install create-react-app

- **Mongoose** – mongoDB object modelling framework for node.js
  - mongoDB is a JSON repository that we'll cover next time

  npm install mongoose

Command Line Node.js

- Node.js can be used from the command line:



```
[users-mbp-3:~ Jonathan$ node
[> function fact(n) {if(n > 1) { return n*fact(n-1);} else { return 1;}}
 undefined
[> fact(5)
 120
[> fact(6)
 720
```

When to use Node.js?

- Node.js is good for creating streaming applications and apps or services that require real-time response such as chat applications, games (especially multi-player games) and file servers
- If you need a high level of concurrency and are not worried about CPU-cycles (i.e. you either have very simple calculations to do each time you are contact, or you have very powerful servers)
- If you really like JavaScript – you will now be using the same language on the server side as on the client side
- More recommendations for when to use node can be found at http://stackoverflow.com/questions/5062614/howto-decide-when-to-use-nodejs

Integrated Development Environments for Node.js

- If you just use a text editor to write you code and the command line runtime for Node.js it is very hard to debug
  - Typically use console.log() to output the values of variables
  - However, there are tools, called IDEs that allow you to interactively debug Node.js applications:
    - IntelliJ
    - VisualStudio Code
    - Eclipse Che
    - Many others; see https://www.slant.co/topics/46/~best-ides-for-node-js (will demo one of these next time)

Resources to get started with Node.js

- Watch the Youtube: video http://www.youtube.com/watch?v=jo_B4LTHi3I
- Read the free O'Reilly Book 'Up and Running with Node.js' at http://ofps.oreilly.com/titles/9781449398583/
- Watch Node.js tutorials at http://nodetuts.com/
- Visit www.nodejs.org for Info/News about Node.js
- For Info about MongoDB, the noSQL JSON database that is almost always used with node.js: http://www.mongodb.org/display/DOCS/Home

Homework

- Install node.js on your system
- Create a node.js app to
  - Print out the average of numbers submitted
  - Extra Credit: Use HTML and make the output look a bit nicer, showing the numbers that have been entered and the average in a table.
  - Extra Extra Credit: Print out the number of distinct clients (by IP address) who have submitted numbers. [You will have to do some Google searching to figure out how to do this.]
  - Submit a printout of your code and evidence that it works correctly

Thank-you!