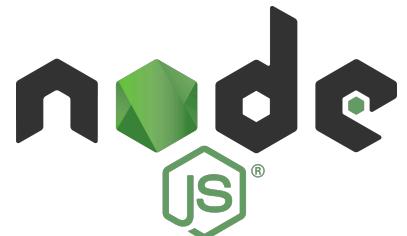




Getting Started with Node.js



Who is this nerd?

Presenter



Justin Reock
Chief Architect
Rogue Wave Software

Justin has over 20 years' experience working in various software roles and is an outspoken free software evangelist, delivering enterprise solutions and community education on databases, integration work, architecture, and technical leadership.

He is currently the Chief Architect at Rogue Wave Software.

What are we going to solve?



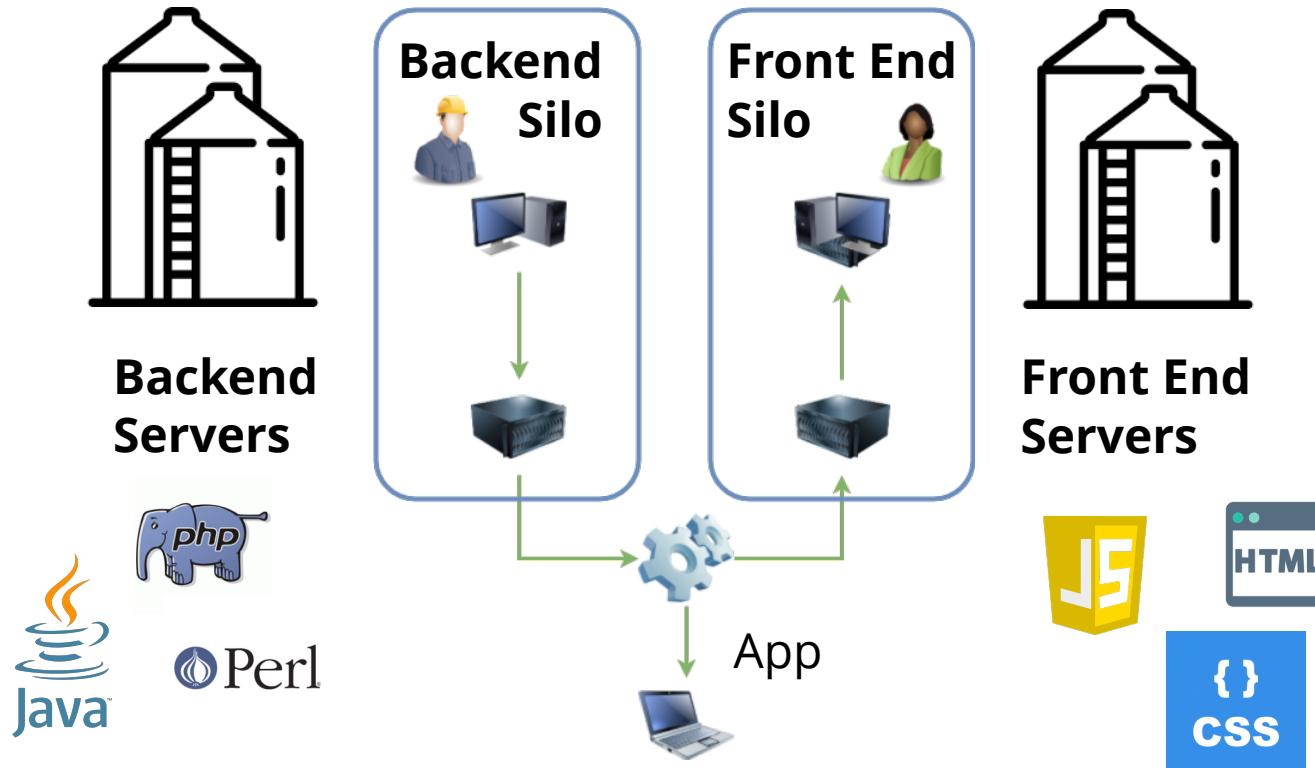
Asynchronous processing

- No one likes the spinning wheel of doom hope!
- Node.js is **inherently asynchronous**, blocking actions only when it absolutely needs to
- This is ideal for our **new(ish) world of parallel** compute, distributed compute, microservices, etc...
- Built on **industry standards** and vetted against **thousands of production systems**

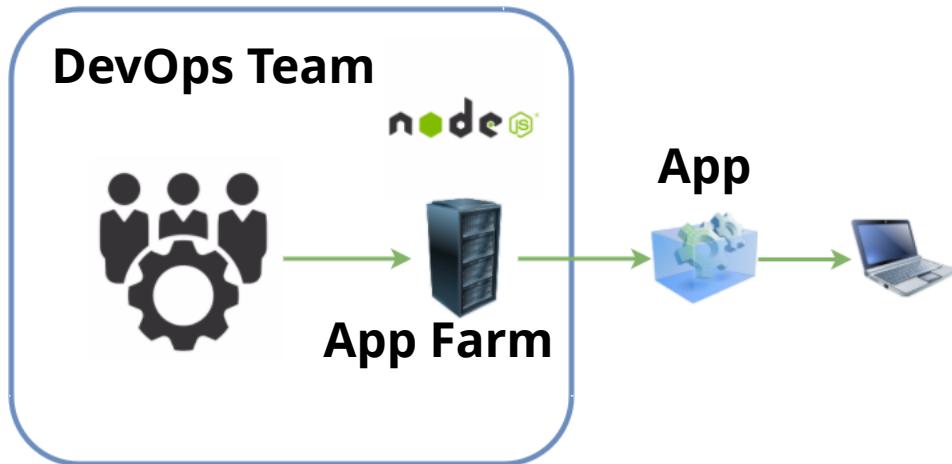


What else?

A Long Time Ago...



Behold, The Future!



Node allows us to code our front end and our backend using a single language

Node.js Overview

- Created by **Ryan Dahl** in **2009** to address concurrency issues.
- Dahl famously “built it over the weekend...”
- Built on Chrome’s V8 JavaScript engine. 
- **Single-threaded asynchronous event driven** JavaScript framework. In-progress tasks can easily be checked for completion.
- **Vertical scaling can be achieved** by adding CPU cores and adding a worker to each core. This means resources can be shared (via the cluster module and child_process.fork(...)).
- Using JavaScript on the browser as well as the server **minimizes the mismatch** between the environments. Example: Form validation code does not have to be duplicated on both client and server.
- Node.js objects maintain a **list of registered subscribers** and notifies them when their state changes ('Observer' design pattern).

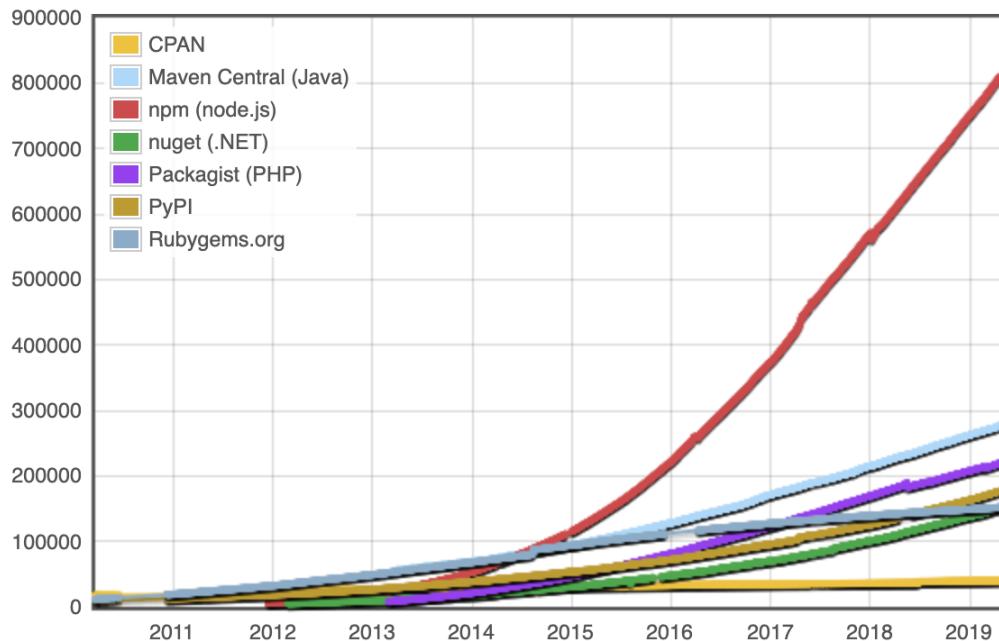


Why reinvent the wheel?

- npm is the node (sic) package manager for Node.js.
 - npm enables JavaScript developers to share code to solve common problems promoting reuse. This allows developers to take advantage of the expertise of a large community.
 - Very active and vibrant community: As of February 2017, there were nearly 400,000 total packages on npm.

- NPM is the **fastest growing** and **most prolific** module repository out there
- Over **800,000** modules and growing

Module Counts



Source: www.modulecounts.com – April 2019

Benefits of Node.js

Pros

- Lightweight: Low memory footprint
- Cross-platform environment. Server and client-side can be written in JavaScript.
- Fast: Asynchronous and event driven
- Popularity
- Concurrency implementation is abstract (handled behind the scenes).
- Full Stack Web Development: Front-end developers can code on the backend.
- Extensibility

Cons

- 1.7 GB memory limit (64-bit)
- Vertical scaling challenge (but workarounds exist)
- Module versions can get confusing
- Open source governance is not flawless (See LeftPad)
- Learning curve for developers not used to dealing with an inherently asynchronous language



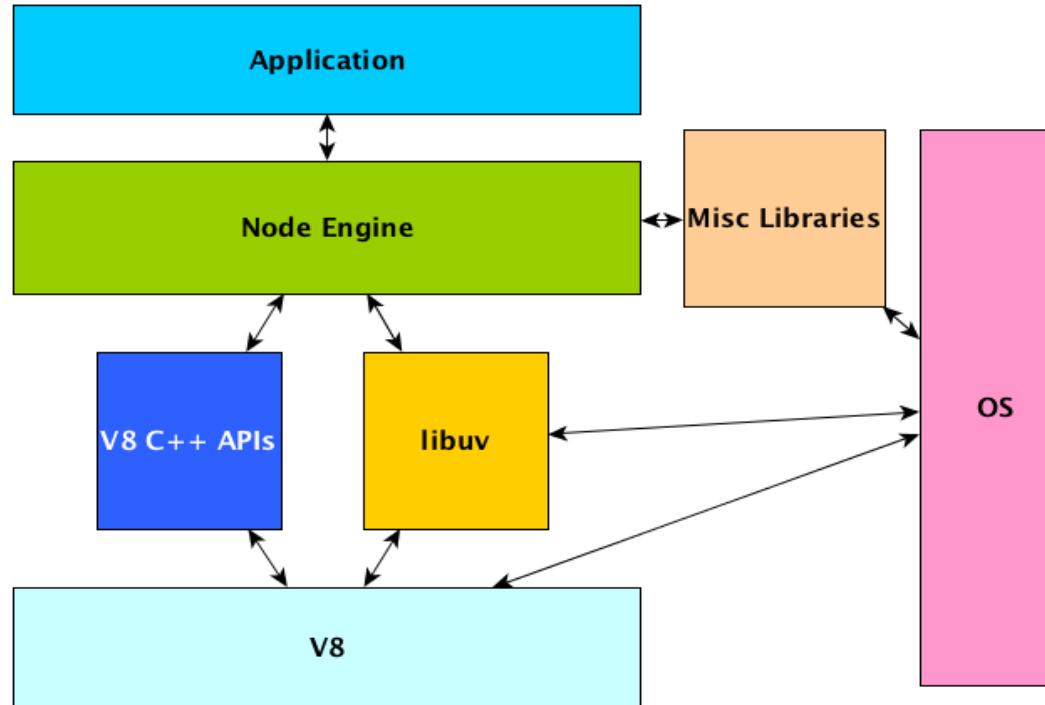
Node.js Web-Enabled Hello World

```
var http = require('http');
http.createServer(function (request, response) {
  response.writeHead(200, { 'Content-Type': 'text/html'});
  response.end('Hello World');
}).listen(8081);
```

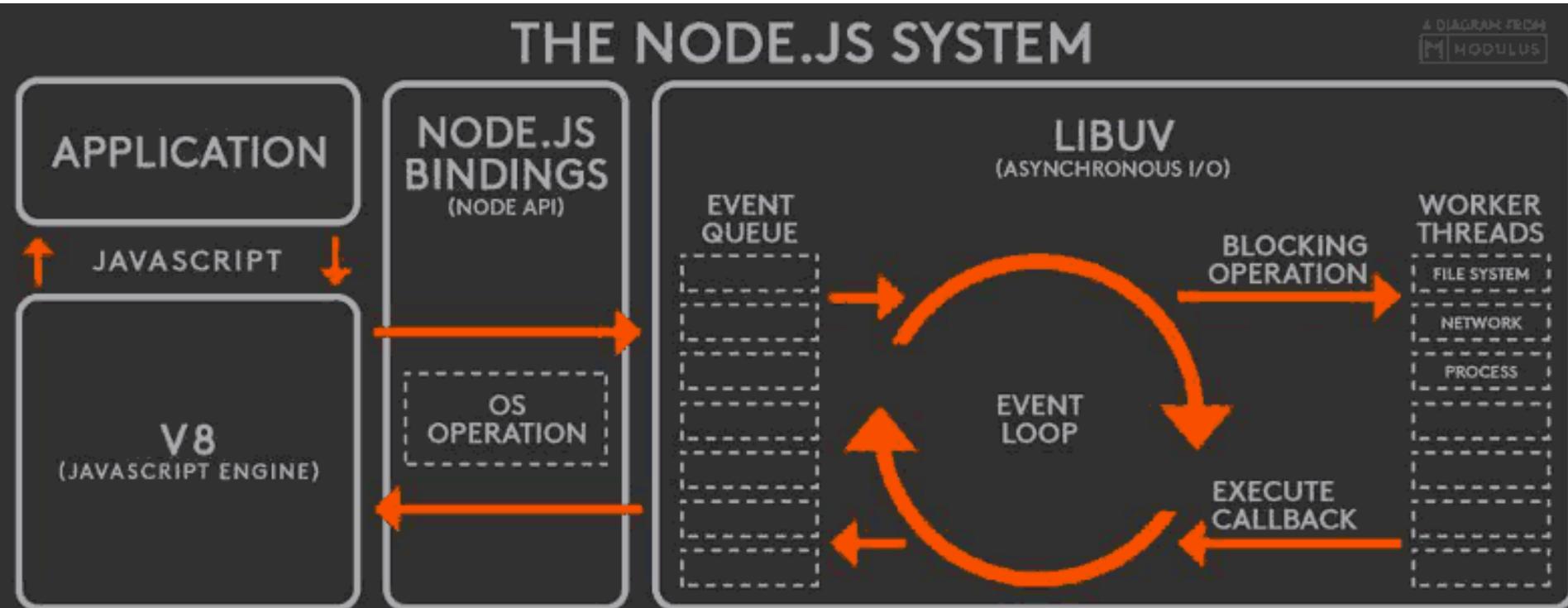


<http://www.haneycodes.net/to-node-js-or-not-to-node-js/>

Node Architecture

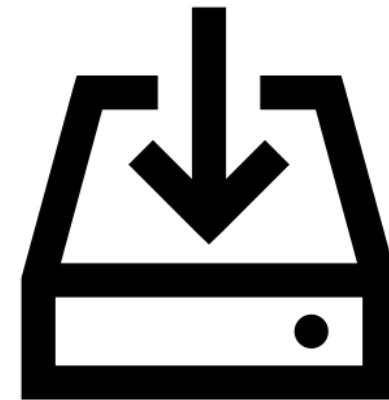


Node's Asynchronous Event Loop



Installing Node

- This will **vary by environment**, but Node can be installed in several different ways:
 - **Linux Package Manager** (like yum, apt-get, pkg, emerge)
 - **Standalone installer**
(<https://nodejs.org/en/download/>)
 - **Official Docker Image**
 - **Standalone Shell Script**



Verifying Node Install

- Once Node has been installed in your environment, check to **make sure NPM has been installed** as well
- You can **validate the installation** by simply checking the version of each binary from the command line:

```
jreock@JUSTINREOCKMBP15:~$ node --version  
v10.15.0  
jreock@JUSTINREOCKMBP15:~$ npm --version  
6.7.0
```

Demo: A Quick Node App

- Node applications can be run using the “node” command line binary
- We’ll write a simple console hello world app from scratch
- We’ll then execute it on the command line



Picking an IDE

- I'll cut to the chase, I **really like** **VSCode** for Node.js development
- **Terminal integration is essential** when building your Node apps
- And the **built-in debugger** works flawlessly with Node.js
- However, you **have other options**
- In the end, go with what is **familiar** and **available** (and is **not** Notepad)

A screenshot of the Visual Studio Code (VSCode) interface. The Explorer sidebar on the left shows a file tree with folders like 'OPEN EDITORS', '.vscode', 'etc', 'node_modules', 'util', 'dbutils', and 'MCCLANE'. The 'dbops.js' file under 'util/dbutils' is selected. The main editor pane displays the code for 'dbops.js', which interacts with 'util/pg' and 'util/logger'. The terminal at the bottom shows command-line output related to the code execution.

```
const { Pool, Client } = require('pg');
const logMe = require("../logger");
const pool = new Pool();
pool.connect();

var self = {
    cleanCVE : function(id, lastmodified) {
        var statement = "DELETE FROM vulnerabilities WHERE id = ? AND lastmodified < ?";
        return self.runQuery(statement, [id, lastmodified]);
    },
    closeConnections : function() {
        setTimeout(function () {
            return process.exit(0);
        }, 10000);
    },
    Complexity is 3 Everything is cool!
    closeConnections : function() {
        setTimeout(function () {
            return process.exit(0);
        }, 10000);
    },
    Complexity is 5 Everything is cool!
    runQuery : async function(statement, params) {
        var isFound = false;
        Complexity is 3 Everything is cool!
        await pool.query(statement, params);
        logMe.logger(statement);
        if (err != null) {
            logMe.logger(err + " " + isFound);
        }
    }
}
```

Picking an IDE

- We've all got our preferences, but make sure whatever you choose meets at least the following requirements:
 - ✓ Application Debugger
 - ✓ Syntax Highlighting
 - ✓ ESLint
 - ✓ Terminal Integration
 - ✓ NPM Integration
 - ✓ IntelliSense (Code Completion)
 - ✓ Code Beautifier
 - ✓ Docker Integration
 - ✓ Node_modules Integration
 - ✓ Source Control Integration

Picking an IDE



- **Cloud 9:** <https://c9.io/>
 - Online Code Editor
 - Debugging
 - Docker Integration: <https://github.com/kdelfour/cloud9-docker>
 - Live Preview

- **JetBrains WebStorm:** <https://www.jetbrains.com/webstorm/>

- Code Completion
 - Debugger

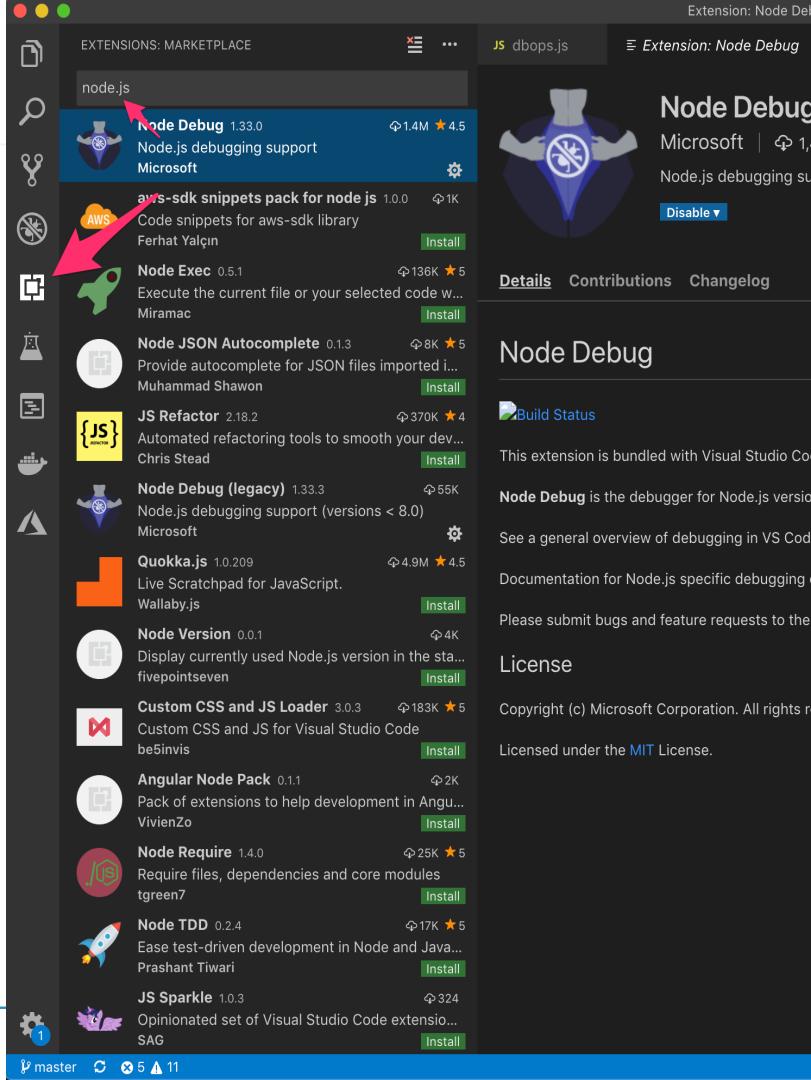
- **Komodo IDE:** <http://www.activestate.com/komodo-ide>

- **Nodeclipse:** <http://www.nodeclipse.org/>
 - Eclipse with Node.js Plugins



Setting up VSCode

- Countless VSCode extensions exist
- Start With at least:
 - Node Debug
 - Node.js Extension Pack
- Take time to explore what is out there



Understanding Launch.json in VSCode

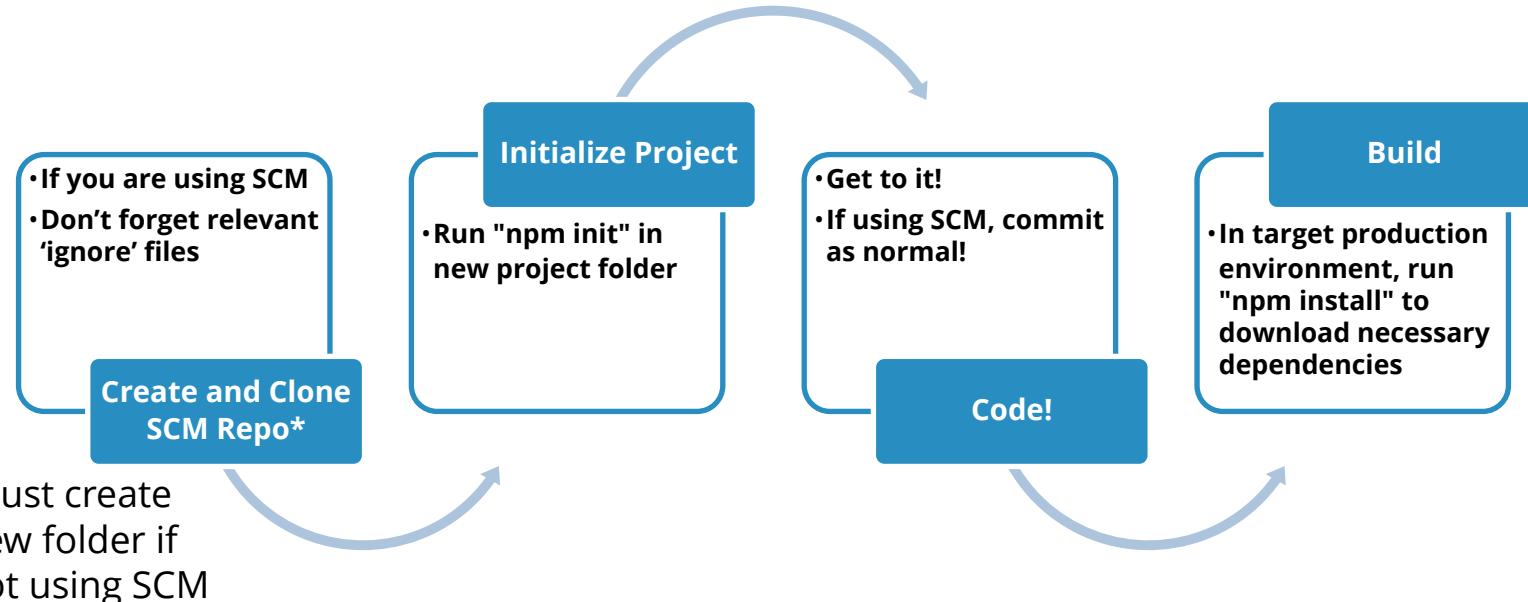
- In VSCode, application **launch configuration** is controlled through a JSON file called “**“launch.json”**”
- This file will be **referenced before VSCode launches** the app when debugging
- It **describes the environment** the application will run in including environment variables, **bootstrap information**, etc
- It is **essential for debugging**, and convenient for launching **unit and functional tests**

Demo: Debugging an App in VSCode

- We'll modify our HelloWorld example to store our message in a variable
- Then we'll create a launch configuration for our project
- Finally, we'll launch our project and inspect our message variable



Node App Basic Lifecycle



Node App Basic Lifecycle

```
jreock@JUSTINREOCKMBP15:~/Development/NodeJS/Getting-Started-With-Node/HelloWorld$ npm init
```

This utility will walk you through creating a package.json file.

It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.

```
package name: (helloworld)
version: (1.0.0)
description: Hello World!
entry point: (index.js) hello.js
test command:
git repository:
keywords:
author: Justin Reock
license: (ISC) GPL-3.0-or-later
```

About to write to /Users/jreock/Development/NodeJS/Getting-Started-With-Node/HelloWorld/package.json:

```
{
  "name": "helloworld",
  "version": "1.0.0",
  "description": "Hello World!",
  "main": "hello.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "Justin Reock",
  "license": "GPL-3.0-or-later"
}
```

Is this OK? (yes) yes

```
jreock@JUSTINREOCKMBP15:~/Development/NodeJS/Getting-Started-With-Node/HelloWorld$
```

Build

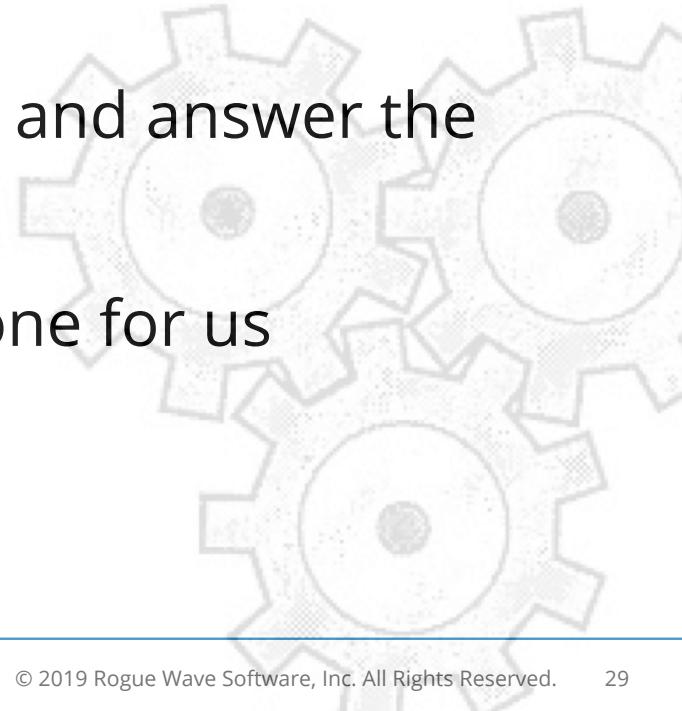
In target production environment, run "npm install" to download necessary dependencies

Initializing a Node App

- It's development best practice to **use npm to initialize** a new Node.js app
- This **process will prompt the user** for several standard pieces of metainformation about the application
- Things like the **application name, version, license, and bootstrap class** will be collected
- NPM will **generate a build configuration** file for the app called **package.json**
- Package.json will **describe to NPM everything needed** to build your Node.js app

Demo: Initializing a Node app

- Let's turn our HelloWorld demo into a proper Node application
- We'll run npm init in the root folder and answer the prompts
- Then we'll look at what NPM has done for us



Understanding package.json

- After running “npm init”, this **file will be generated** in the project root folder
- Our project currently has no dependencies, but **dependency information is held here too**

```
{  
  "name": "helloworld",  
  "version": "1.0.0",  
  "description": "Hello World!",  
  "main": "hello.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test  
specified\\\" && exit 1"  
  },  
  "author": "Justin Reock",  
  "license": "GPL-3.0-or-later"  
}
```

Working with dependencies

- Our final topic will introduce dependencies, or **modules**
- Most modern open-source languages have **a set of open-source modules** that can be added to a project
- You can also **create your own modules**, whether **private** or **freely available**
- Recall that the **NPM repository provides these modules**
- NPM can also be **used to add dependencies** to a project, in both development and production environments, using the **“npm install”** command
- **Once NPM has installed** a module into a project, a developer can begin using it with the **“require” directive** inside the Node.js app

Example: Let's Add Some Color

- The “**colors**” module for **Node** adds easy support for generating console colors through escape commands
- You want your **console output to be readable**, and even fun!
- The **colors module makes that easy**, and we can add it with npm install
- Note that the **--save directive** will ensure that the **module is saved to package.json**

```
npm install --save colors
var colors = require('colors');
console.log(colors.red.underline('This is red underlined text.'));
or
console.log('This is red underlined text.'.underline.red)
```

Demo: Adding Project Modules from NPM

- So, let's add some color to our HelloWorld app!
- We'll use npm to install the "colors" module and make sure it's contained in package.json
- Then we'll "require" that module, and use it in our code
- BONUS: Remove the node_modules and rebuild with npm install

What did we learn?

- Node.js is a language that focuses on **concurrency and unified development**
- Dependencies (modules) and **builds are assisted through npm**
- Node is **asynchronous** by default
- **Many IDEs** exist for Node.js, and we have focused on VSCode
- Node **projects should be initialized** using “npm init”
- **Modules can be installed** with “npm install –save [module]”
- Projects are **built from package.json** in other environments with “npm install”
- **Modules are included as variables** in a project using the “require” directive

What Next?

Still So Much to Learn!!

- Node is a huge subject, but, I'd focus on the following from here:
 - Understand Asynchronous Coding
 - Get to Know Events and Streams
 - Learn about Functional Programming
 - Master your IDE's Debugger
 - Start Exploring Other Modules!

I Like People!!

Feel Free to Reach Out – I Get Lonely...



LinkedIn – Only Justin Reock in the world apparently!



Twitter – @jreock - But I do get a little political on there....



Blog - <http://blog.klocwork.com/author/justin-reock/>



Email – justin.reock@roguewave.com

Questions?



