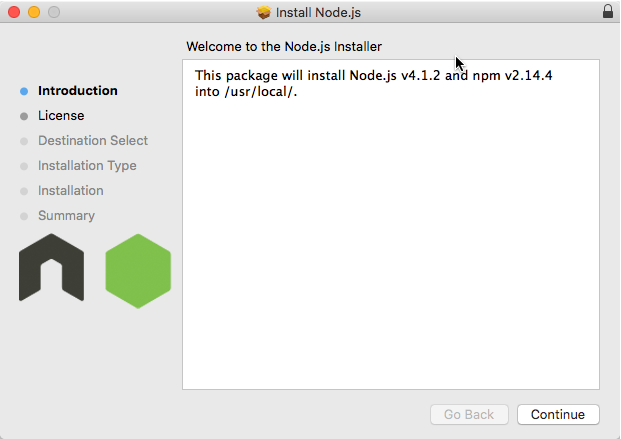
# How to use node.js

Once we know how to work with JavaScript and TypeScript, node.js should not take much time. Traditionally, JavaScript is associated with browsers and web. But thanks to node.js runtime you can run JavaScript applications anywhere including your Mac, cloud platforms, IoT devices and so on. In order to start experiments with node.js you need to install it on your Mac visiting <https://nodejs.org/> site:



Traditionally, node.js installation package contains not just node runtime but a package manager (npm) that allows to download and install lots of different packages/modules and use them once they are installed. You can see information about node.js and the package manager once you start installer on your machine:



Running your applications under node.js you can continue to use JavaScript or TypeScript languages. So, first of all you need to create jsconfig.json or tsconfig.json to get the best experience. For example, if you want to use JavaScript the jsconfig.json can look like:

{

"compilerOptions": {

"target": "ES5",

"module": "commonjs"

}

}

But node.js is not just a tool that allows to run JavaScript; it also has an API with a huge number of modules that are part of node.js runtime. So, the question there is how to use these modules and other packages using VS Code experience. Looking in the node.js tutorial you can find the following JavaScript code:

var http = require('http');

http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/plain'});

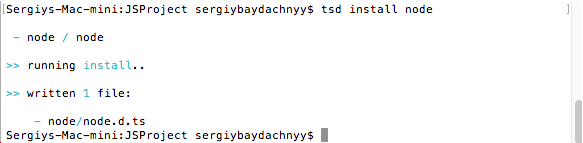
res.end('Hello World\n');

}).listen(1337, "127.0.0.1");

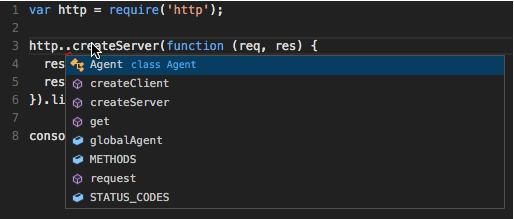
console.log('Server running at http://127.0.0.1:1337/');

Just copy this code to JavaScript file and try to play with it for some time. You can see that IntelliSense system doesn’t know anything about http module, parameters and IntelliSense system doesn’t work. In order to fix the problem you can use TypeScript package manager to install **.d.ts** files for Node.js. Yes, we are not using TypeScript but it’s not important for us because we simply want to use TypeScript definitions to fix the problem with IntelliSense, providing information to VS Code rather than using TypeScript itself. So, run the following command from the Terminal:

**tsd install node**



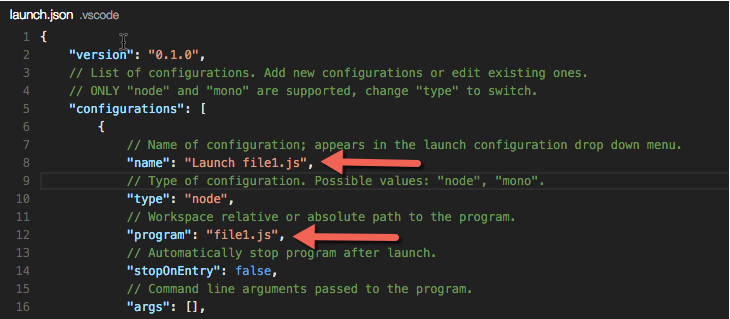
And you will see that TypeScript package manager create node subfolder and node.d.ts inside. The last one Code recognizes automatically and you can use IntelliSense system right away:



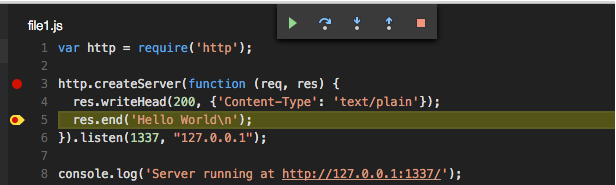
The second important feature is Debugging. With Code it’s easy to run your Node.js application in debug mode. Just create a couple of breakpoints clicking the place near line numbers:



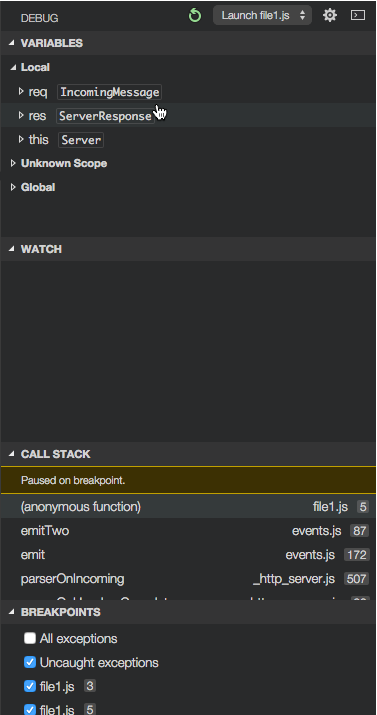
To use Debugger we need to create launch.json file. In order to create this file you need to open Debug view and click Run button. Visual Studio Code offers to create launch.json for you, and you need to change the default JavaScript file to your own:



Once you save all changes, you can run the Debugger once again. If everything is OK, the first breakpoint will fire once you start the process and the second one once you open your browser and navigate to your service:



At the same time you can review lots of information that is associated with a breakpoint and the process:



So, debugging in Visual Studio Code is easy task.