**Working With Environment Variables in Node.js**



With just a few lines of code, you can run a Node.js application straight from the terminal and use Twilio to send and receive text messages with Twilio Programmable SMS.

**The basics of environment variables in Node.js**

Node.js is one of the most talked about frameworks in the web development community since Rails. While still very young, Node.js has proven to be fast, highly scalable, and efficient. The reason for this is due to its event-driven, nonblocking input/output structure.

Combining it with Twilio helps you build telephony apps in few steps. However, if you wanted to send an SMS, for example, you’ll need to install Node.js and the Twilio Node.js module.

Another great feature of the [Node.js](https://nodejs.org/) language (and its modules) is that it has a way of working with [environment variables](https://www.twilio.com/blog/how-to-set-environment-variables.html), which many cloud hosts (such as Heroku, Azure, AWS, Now.sh, etc.) use. This opens up an array of ways to configure different aspects of your Node.js application. So while hosts will set a PORT variable that specifies on which port the server should listen to, modules might have different behaviors (like logging) depending on the value of the NODE\_ENV variable.

We’ll review a few tricks and tools when working with environment variables in Node.js in this blog.

First things first, we need to ensure that we have a Node.js environment. We’ll then take you through the basics of setting up Node.js for use with Twilio on a Mac. This will include installing npm on a Mac and getting the environment ready for a Twilio project.

**How to check if Node.js is installed**

You can check if you already have Node.js installed on your machine by opening up a terminal and running the following:

node -v

If you don't have Node.js installed, the following steps will show you how to install Node.js:

1. Go to the [Node.js downloads page](https://nodejs.org/en/download/).
2. Download Node.js for macOS by clicking the "Macintosh Installer" option.
3. Run the downloaded Node.js .pkg installer.
4. Run the installer, including accepting the license, selecting the destination, and authenticating for the install.

Once the installer is complete, you can then ensure Node.js will run “node -v” in your terminal—you should get the current version number as shown above.

If you don't see the installed Node.js version, you may need to relaunch your terminal.

To ensure the installation is correct and running “npm -v,” look for a confirmation of the install via its version.

Now your Mac environment is ready to install the Twilio module so that Node.js scripts in the current directory can use it.

**Explicity loading varibles from .env files**

You can access environment variables in Node.js right out of the box. When your Node.js process boots up, it’ll automatically provide access to all existing environment variables by creating an env object within the process global object.

If you want to take a peek at the object, run the the Node.js REPL with “node” in your command line and type:

console.log(process.env);

This code should output all environment variables that this Node.js process can pick up. To access one specific variable, access it like you would any property of an object:

console.log('The value of PORT is:', process.env.PORT);

Here, you’ll notice that the value of PORT is undefined on your computer. Cloud hosts like Heroku or Azure, however, use the PORT variable to tell you which port your server should listen to for the routing to work properly. So the next time you set up a web server, you can determine the port to listen to by checking the PORT first and giving it a default value after:

const app = require('http').createServer((req, res) => res.send('Ahoy!'));

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Server is listening on port ${PORT}`);

});

The highlighted line will take the value of the PORT if it’s available or default to listening to 3000 as a fallback port. Try running the code by saving it in a file like server.js and run:

node server.js

The output should be a message saying, "Server is listening on port 3000." Stop the server with Ctrl+C and restart it with the following command:

PORT=9999 node server.js

The message should now say, “Server is listening on port 9999.” This means the PORT variable has been temporarily set in front of the node for this execution by PORT=9999.

Since process.env is a normal object, we can set/override the values:

process.env.MY\_VARIABLE = 'ahoy';

The code above will set or override the value of MY\_VARIABLE. However, keep in mind that this value is only set during the execution of this Node.js process and only available in child processes spawned by this process. Overall, you should avoid overriding environment variables as much as possible and just initialize a configuration variable, as shown in the PORT example.

If you develop multiple Node.js projects on one computer, you might have overlapping environment variable names. For example, different messaging apps might need different Twilio Messaging Service SIDs, but both would be called TWILIO\_MESSAGE\_SERVICE\_SID. A great way to achieve project-specific configuration is by using .env files. These files allow you to specify various environment variables and associated values.

Typically, you don’t want to check these files into source control. So when you create one, just add .env to your .gitignore file. With a lot of [Twilio demo applications](https://www.twilio.com/) .env.example files, you can then copy them to .env files and set the values yourself. Having an .env.example or similar file is a common practice if you want to share a template file with other people in the project.

How do we load the values from this file? The quickest way is by using an [npm module called dotenv](http://npm.im/dotenv). Install the module via npm:

npm install dotenv --save

Afterward, add the following line to the top of your entry file:

require('dotenv').config();

This code will automatically load the .env file in the root of your project and initialize the values, skipping any variables already preset. Be careful not to use .env files in your production environment, though. Instead, set the values directly on the respective host. So you might want to wrap your load statement in an if statement:

if (process.env.NODE\_ENV !== 'production') {

require('dotenv').config();

}

With this code, we’ll only load the .env file if the server isn’t already in production mode.

Let’s see this in action. Install dotenv in a directory, as shown above. Then, create an dotenv-example.js file in the same directory and place the following lines into it:

console.log('No value for FOO yet:', process.env.FOO);

if (process.env.NODE\_ENV !== 'production') {

require('dotenv').config();

}

console.log('Now the value for FOO is:', process.env.FOO);

Afterward, create a file called .env in the same directory with this content:

FOO=bar

Run the script:

node dotenv-example.js

The output should look like:

No value for FOO yet: undefined

Now the value for FOO is: bar

As you can see, the value was loaded and defined using dotenv. If you rerun the same command with NODE\_ENV set to production, you’ll see that it’ll stay undefined.

NODE\_ENV=production node dotenv-example.js

Now the output should be:

No value for FOO yet: undefined

Now the value for FOO is: undefined

If you don’t want to modify your actual code, you can also use Node’s -r argument to load dotenv when executing the script. To do so, change your dotenv-example.js file to:

console.log('The value for FOO is:', process.env.FOO);

Now execute the file as you normally would:

node dotenv-example.js

The script should output that the current value for FOO is undefined. Now, execute it with the appropriate flag to require dotenv:

node -r dotenv/config dotenv-example.js

The result is that FOO is now set to bar since the .env file has been loaded. If you want to learn more about dotenv, check out its [documentation.](https://www.npmjs.com/package/dotenv)

**An alternative way to load .env files**

There’s also an alternative module based on dotenv to make loading environment variables more convenient. The result is [node-env-run or nodenv](http://npm.im/node-env-run). This command-line tool will load a .env file, initialize the values using dotenv, and execute your script.

You can install it globally, but this is recommended only for development purposes and locally to the project. Install it by running:

npm install node-env-run --save-dev

Afterward, create a file nodenv-example.js and place this code in it:

console.log('The value for FOO is:', process.env.FOO);

As you can see, we don’t need to require anything here because it’s just the [application logic](https://www.twilio.com/frontline). To try running it, use node:

node nodenv-example.js

This executed code should output “The value for FOO is: undefined.” Now, try using node-env-run by running:

node\_modules/.bin/nodenv nodenv-example.js

The result should be “The value for FOO is: bar” since it loaded the .env file.

Using node-env-run can override these existing values. To see it in action, first run the following command:

FOO=foo node\_modules/.bin/nodenv nodenv-example.js

The command-line output should say, “The value for FOO is: foo.” If we now enable force mode, we can override existing values:

FOO=foo node\_modules/.bin/nodenv --force nodenv.js

As a result, we should be back to the original “The value for FOO is: bar.”

If you want to use this tool regularly, I recommend that you wrap it into an npm script by adding it in the package.json like so:

{

"name": "twilio-blog",

"version": "1.0.0",

"description": "",

"main": "nodenv-example.js",

"scripts": {

"start": "node .",

"start:dev": "nodenv -f ."

},

"author": "",

"license": "ISC",

"devDependencies": {

"node-env-run": "^2.0.1"

}

}

This way, you can run:

npm run start:dev

If you want to learn more about node-env-run, check out its [documentation](https://www.npmjs.com/package/node-env-run).

**Environment variables & npm scripts**

There are also scenarios where it’s useful to check the value of an environment variable before entering the Node.js application in npm scripts. For example, if you want to use node-env-run when you’re in a development environment but use node when you’re in production mode. Use the [if-env](http://npm.im/if-env) tool for this. Install it by running:

npm install if-env --save

However, make sure to not install it as a “dev dependency” since we'll require this in production as well.

Now, modify your npm scripts in your package.json:

"scripts": {

"start": "if-env NODE\_ENV=production ?? npm run start:prod || npm run start:dev",

"start:dev": "nodenv -f .",

"start:prod": "node ."

}

This script will now execute npm run start:prod and subsequently node. If NODE\_ENV has the value production, it will then be able to execute npm run start:dev and subsequently nodenv -f. You can do this with any other environment variable as well.

Try it out by running:

# should output "The value of FOO is: bar"

npm start

# should output "The value of FOO is: undefined"

NODE\_ENV=production npm start

If you want to learn more about if-env, check out its [documentation](https://npm.im/if-env).

**How to install npm on a Mac**

Usually, this will automatically install as part of the above installation. You can check this by running the npm command in the terminal. If it did install, you’ll see a usage list. If not, to install npm, run the following:

sudo apt install npm

**Debugging in Node.js**

Next up is debugging time. One strategy that helped me out a lot is using the DEBUG environment variable to receive more verbose logs for numerous modules. Take, for example, a basic express server like this:

const app = require('express')();

const bodyParser = require('body-parser');

app.use(bodyParser.json()); // for parsing application/json

app.use(bodyParser.urlencoded({ extended: true })); // for parsing application/x-www-form-urlencoded

app.post('/', (req, res, next) => {

console.log(req);

});

app.listen(3000);

Start it up with the DEBUG variable set to \*:

DEBUG=\* node server.js

You’ll receive extensive logging that looks something like this:

The magic behind it is a [lightweight module called debug](http://npm.im/debug). When you want to use it, all you have to do is initialize it with a namespace. Afterward, you can log it to that namespace. If someone wants to see the output, all they have to do is enable the namespace in the DEBUG variable. In this case, express uses a bunch of sub-namespaces. So if you would want everything from the express router, all you have to do is set the DEBUG with the appropriate wild card:

DEBUG=express:router\* node server.js

If you want to use debug in your module, first you have to install it:

npm install debug --save

Afterward, apply it in the following way:

const debug = require('debug')('myApp:someComponent');

debug('Here is a pretty object %o', { someObject: true });

If you want to learn more about debug, check out its [documentation](https://www.npmjs.com/package/debug).

**Use all the Node.js environment variables in Node.js with Twilio**

These are just some of things that you can do with environment variables and all the tools that exist. Now that you’ve learned a bit more about environment variables in Node.js, try out Twilio’s [Node.js quickstarts](https://www.twilio.com/docs/quickstart/node), including our [SMS quickstart](https://www.twilio.com/docs/sms/quickstart/node#install-the-twilio-cli). [Get started for free!](https://www.twilio.com/try-twilio)