A function is allowed to use external Node.js modules as well as local data. Dependencies in Node.js are managed with npm (https://docs.npmjs.com/) and expressed in a metadata file called `package.json` (https://docs.npmjs.com/files/package.json). The Cloud Functions Node.js runtimes generally support installing using npm (https://docs.npmjs.com/) or yarn (https://yarnpkg.com/en/).

To specify a dependency for your function, add it to your `package.json` file.

In this example, a dependency is listed in the `package.json` file:

The dependency is then imported in the function:

functions/helloworld/index.js (https://github.com/GoogleCloudPlatform/nodejs-docs-samples/blob/master/functions/helloworld/index.js)

Hub (https://github.com/GoogleCloudPlatform/nodejs-docs-samples/blob/master/functions/helloworld/index.js)

When you deploy your function, Cloud Functions installs dependencies declared in the `package.json` file using the `npm install` (https://docs.npmjs.com/cli/install) command:

In the Node.js 8 runtime and higher, if a `yarn.lock` file exists, Cloud Functions instead uses the `yarn install` (https://yarnpkg.com/lang/en/docs/cli/install/) command:

During deployment, before starting your application, you can perform a custom build step by adding a `gcp-build` script (/appengine/docs/standard/nodejs/running-custom-build-step#example) in your `package.json` file.

When this script is executed, the dependencies in the `dependencies` and `devDependencies` fields of your `package.json` file are available. After executing your custom build step, Cloud Functions removes and regenerates the `node_modules` folder by only installing the production dependencies declared in the `dependencies` field of your `package.json` file.

If there is no `gcp-build` script in `package.json`, Cloud Functions just installs production dependencies.

The Node.js runtime also includes a number of system packages (/functions/docs/reference/nodejs-system-packages) in the execution environment. If your function uses a dependency that requires a package that is not listed, you can request a package (/functions/docs/reference/nodejs-system-packages#requesting_a_package).

you can also deploy containerized versions of your Node.js Cloud Functions to Cloud Run (/run/docs) using the Func work (/functions/docs/functions-framework).

You can also include local Node.js modules as part of your function. You can achieve this by declaring your module in `package.json` using the `file:` prefix (https://docs.npmjs.com/files/package.json#local-paths). In the following example, `mymodule` refers to your module name and `mymoduledir` is the directory containing your module:

The code for this local module should not be in your `node_modules` folder. You can simply store it at the root of your function's directory.

f you deploy using the `gcloud` command-line tool, `gcloud` automatically creates a `.gcloudignore` ://cloud.google.com/sdk/gcloud/reference/topic/gcloudignore) file in your function's directory. By default, the `modules` folder is added to this `.gcloudignore` and is not uploaded as part of your deployment.

The easiest way to install a Node.js module locally is to use the `npm install` command in the folder containing your Cloud Function. For instance, the following command adds the `uuid` module:

This combines two steps:

1. It marks the latest version of the module as a dependency in your `package.json` file. This is very important: Cloud Functions only installs modules that are declared in your `package.json` file.
2. It downloads the module into your `node_modules` directory. This lets you use the module when developing locally.

If you don't have npm installed on your machine, get npm (https://www.npmjs.com/get-npm).

Use the Node.js `require()` (https://nodejs.org/api/modules.html#modules_require) function to load any Node.js module you have installed. You can also use the `require()` function to import local files you deploy alongside your function.

In order to use a private npm module (https://docs.npmjs.com/private-modules/intro), you must provide credentials (auth token) for the npm registry in a `.npmrc` file located in the function's directory. The npm documentation (https://docs.npmjs.com/getting-started/working_with_tokens) explains how to create custom read-only access tokens. We discourage using the `.npmrc` file created in the home directory because it contains a read-write token. Write permissions are not required during deployment, and could pose a security risk.

Do not include the `.npmrc` file if you're not using private repositories, as it can increase the deployment time for your functions.