Node.js in a container

In this guide you will learn how to:

* Create a Dockerfile file for an [Express](https://expressjs.com/) Node.js service container
* Build, run, and verify the functionality of the service
* Debug the service running within a container

Prerequisites[#](https://code.visualstudio.com/docs/containers/quickstart-node#_prerequisites)

* Both Docker and the VS Code Docker extension must be installed as described in the [overview](https://code.visualstudio.com/docs/containers/overview#_installation)
* [Node.js](https://nodejs.org/) version 10 or later

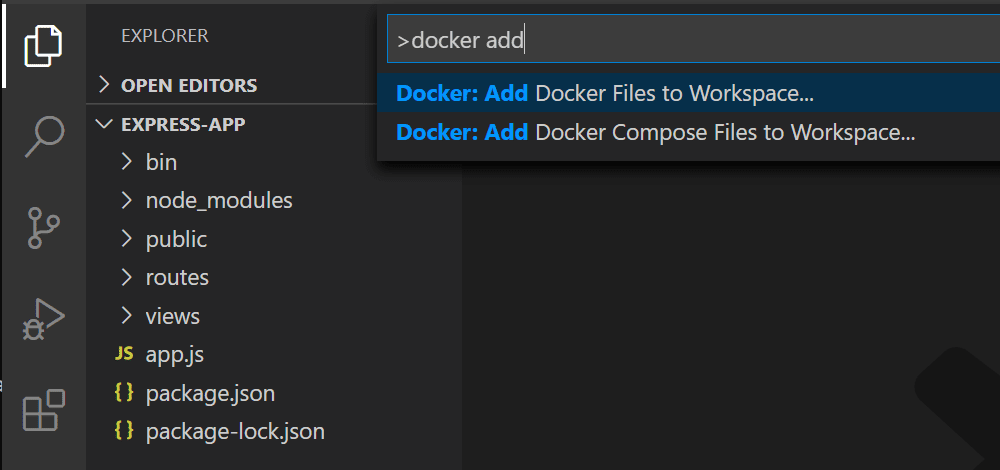
Create an Express Node.js application[#](https://code.visualstudio.com/docs/containers/quickstart-node#_create-an-express-nodejs-application)

1. Create a folder for the project.
2. Open a development command prompt in the project folder and create the project:
3. npx express-generator

npm install

Add Docker files to the project[#](https://code.visualstudio.com/docs/containers/quickstart-node#_add-docker-files-to-the-project)

1. Open the project folder in VS Code.
2. Open the Command Palette (Ctrl+Shift+P) and use **Docker: Add Docker Files to Workspace...** command:



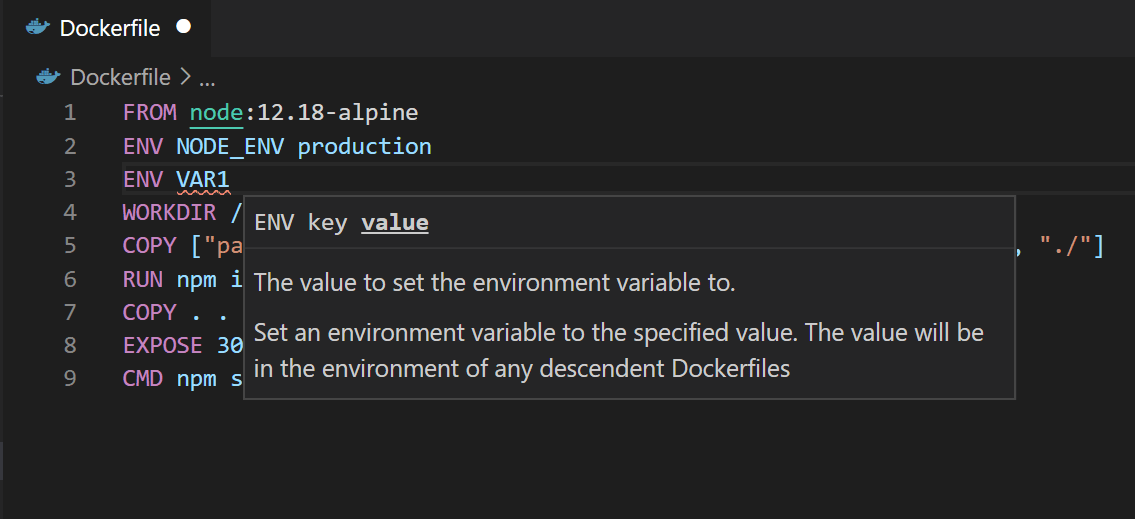
1. Select **Node.js** when prompted for the application platform.
2. Select either **Yes** or **No** when prompted to include Docker Compose files. Compose is typically used when running multiple containers at once.
3. Enter 3000 when prompted for the application port.

The extension creates Dockerfile and .dockerignore files. If you elected to include Docker Compose files, docker-compose.yml and docker-compose.debug.yml will be generated as well. Finally, the extension will create a set of **VS Code tasks** in .vscode/tasks.json for building and running the container (in both debug- and release-configurations) and a **launch debug configuration** in .vscode/launch.json for debugging the service within the container.

Add an environment variable to the image[#](https://code.visualstudio.com/docs/containers/quickstart-node#_add-an-environment-variable-to-the-image)

The Docker extension helps you author Dockerfiles by using [IntelliSense](https://code.visualstudio.com/docs/editor/intellisense) to provide auto-completions and contextual help. To see this feature in action, add an environment variable to your service image by following these steps:

1. Open the Dockerfile file.
2. Use ENV instruction to add an environment variable to the service container image.



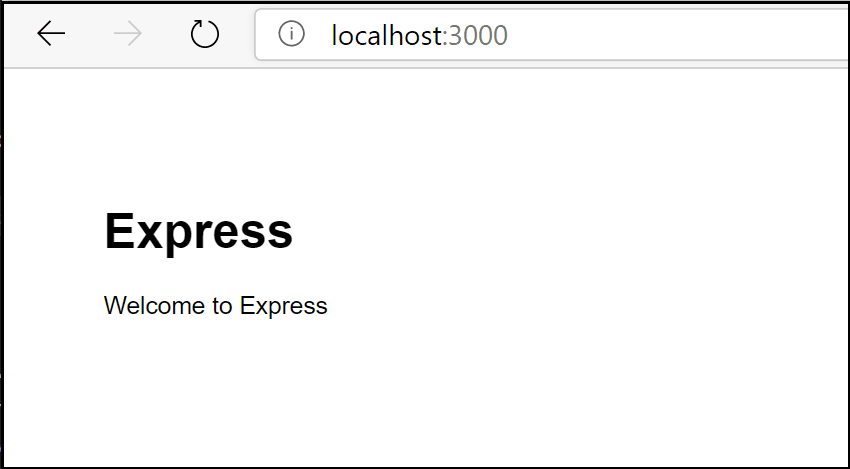
Note how the Docker extension lists all available Dockerfile instructions and describes the syntax.

The Docker extension uses the base stage of the Dockerfile to create a debug version of the container image for your service. Put the environment variable definition in the base stage to have this variable available in both debug and release versions of the container image.

1. Save the Dockerfile file.

Run the service locally[#](https://code.visualstudio.com/docs/containers/quickstart-node#_run-the-service-locally)

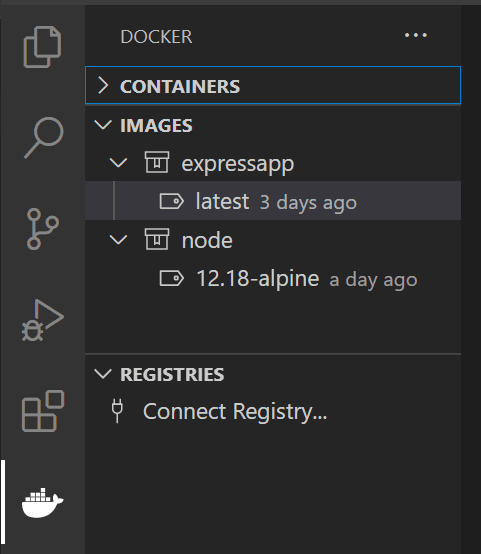
1. Open a terminal (Ctrl+`).
2. Enter npm run start to start the application:
3. > express-app@0.0.0 start /Users/user/code/scratch/express-app
4. > node ./bin/www
5. Open the web browser and navigate to [http://localhost:3000](http://localhost:3000/). You should see a page similar to the following:



1. When done testing, type Ctrl+C in the terminal.

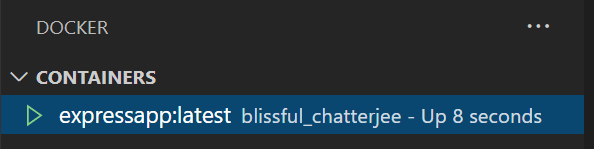
Build the service image[#](https://code.visualstudio.com/docs/containers/quickstart-node#_build-the-service-image)

1. Open the Command Palette (Ctrl+Shift+P) and select the **Docker Images: Build Image...** command.
2. Open the Docker Explorer and verify that the new image is visible in the Images tree:

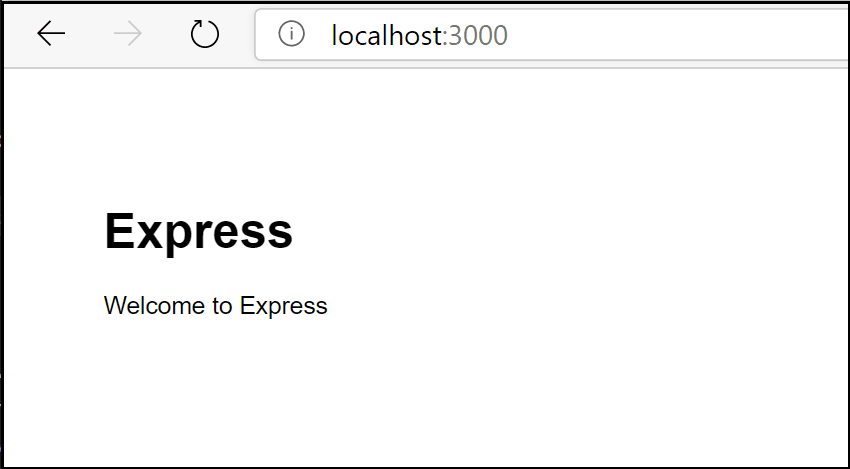


Run the service container[#](https://code.visualstudio.com/docs/containers/quickstart-node#_run-the-service-container)

1. Right-click on the image built in the previous section and select **Run** or **Run Interactive**. The container should start and you should be able to see it in the Docker Containers tree:



1. Open the web browser and navigate to [http://localhost:3000](http://localhost:3000/). You should see a page similar to the following:

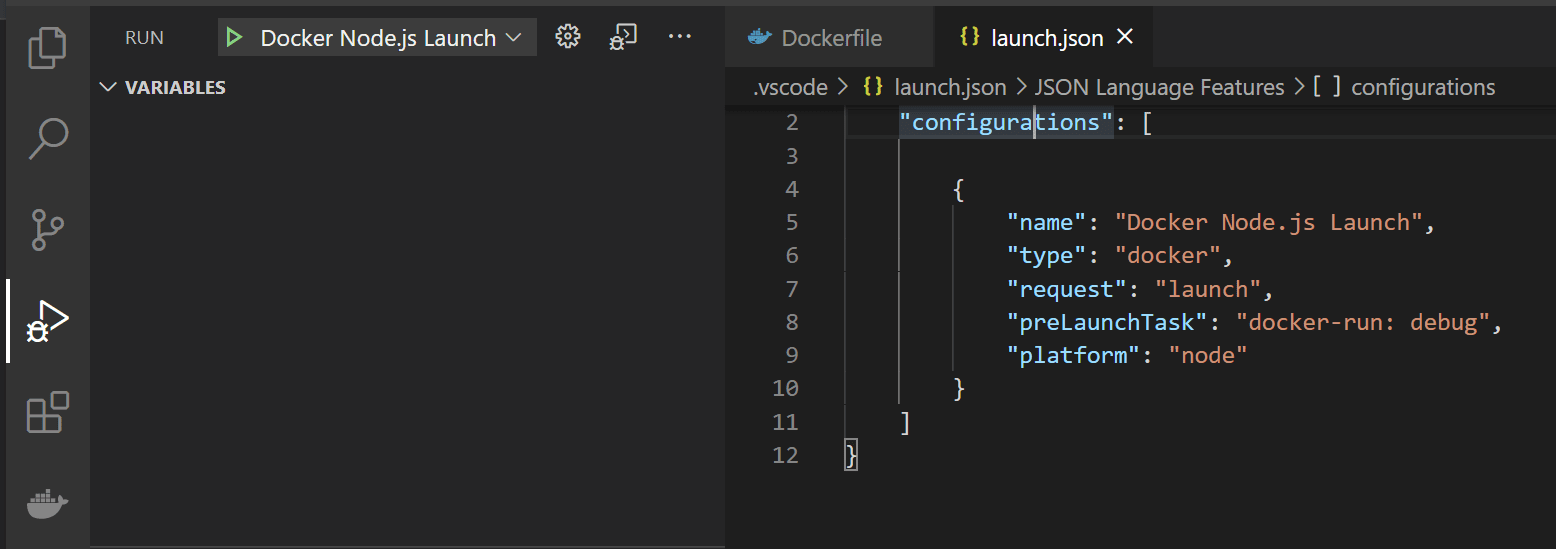


1. When done testing, right-click the container in the Containers tree and select **Stop**.

Debug in the service container[#](https://code.visualstudio.com/docs/containers/quickstart-node#_debug-in-the-service-container)

When the Docker extension adds files to the application, it also adds a **VS Code debugger configuration** in .vscode/launch.json for debugging the service when running inside a container. The extension detects the protocol and port used by the service and points the browser to the service.

1. Set a breakpoint in the get() handler for the '/' route in routes/index.js.
2. Make sure the **Docker Node.js Launch** debugger configuration is selected.



1. Start debugging (use the F5 key).
   * The Docker image for the service builds.
   * The Docker container for the service runs.
   * The browser opens to the (random) port mapped to the service container.
   * The debugger stops at the breakpoint in index.js.

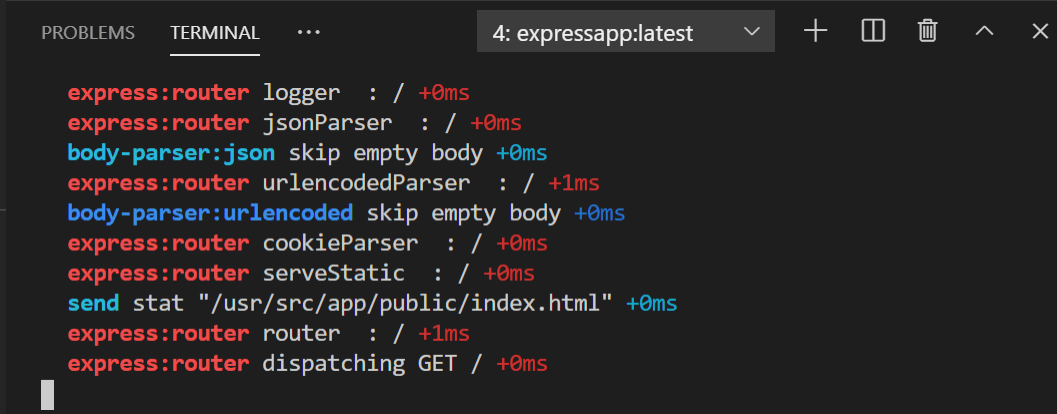
Note that, because the debugger attaches *after* the application starts, the breakpoint may missed the first time around; you might have to refresh the browser to see the debugger break on the second try.

You can configure the application to wait for the debugger to attach before starting execution by setting the [inspectMode](https://code.visualstudio.com/docs/containers/reference" \l "_node-object-properties-dockerrun-task) property to break in the docker-run: debug task in tasks.json under the node object.

View the application logs[#](https://code.visualstudio.com/docs/containers/quickstart-node#_view-the-application-logs)

You can view the logs in VS Code by using the **View Logs** command on the container:

1. Navigate to the Docker Explorer.
2. In the **Containers** tab, right-click on your container and choose **View Logs**.



1. The output will be displayed in the terminal.