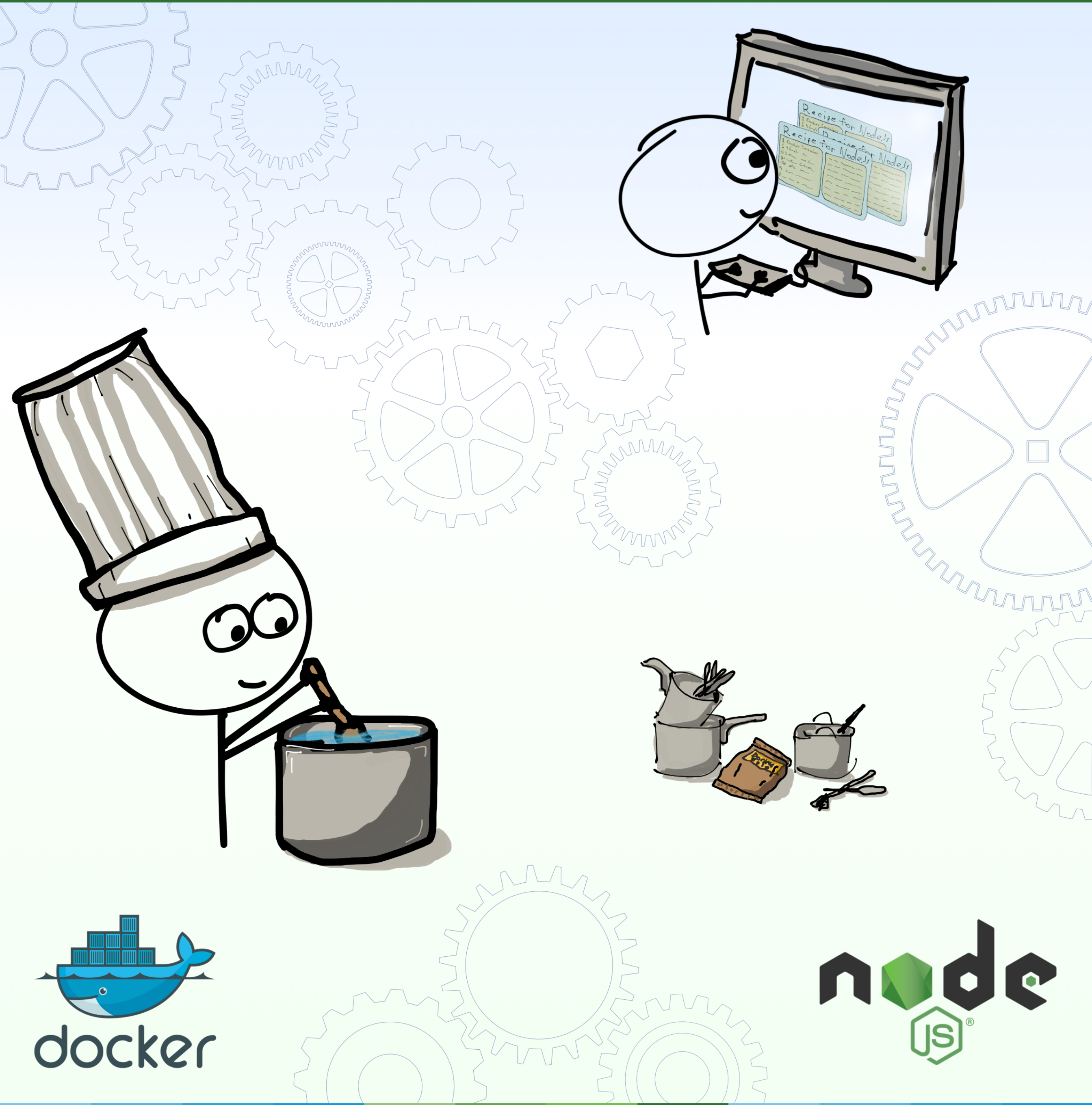
Docker Recipes for Node.js Development



by Derick Bailey

Docker Recipes for Node.js Development

Solve common problems with simple solutions for Node.js in Docker!

Derick Bailey

This book is for sale at http://leanpub.com/docker-recipes

This version was published on 2017-05-25

Tweet This Book!

Please help Derick Bailey by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just bought the Docker Recipes for Node.js ebook from @derickbailey - check it out and be productive!

The suggested hashtag for this book is ##DockerRecipes4NodeJS.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=##DockerRecipes4NodeJS

Also By Derick Bailey

Contents

Preface

When I first started working with Docker to build Node.js applications, I ran into a series of rather difficult situations.

Some of the questions and situations included:

- How do I edit code in my container?
- What about debugging?
- Can I use Grunt, run my test and build processes in the container?
- What if I need to restart the app when code changes?

These, and other questions, drove me to understand the relationship between a Docker image, container and my local computer as a host.

The result was a series of experiments that eventually turned into what I consider good practices within specific situation and circumstances.

As with most of my development knowledge and experience, I began to record these lessons as screencasts for WatchMeCode.

Over time, however, I began to realize that the tutorial approach to the screencasts was only delivering half of the solution that a developer needs, when using Docker.

Learning how to use the individual parts of Docker and it's related tools is absolutely necessary, and the WatchMeCode screencasts cover this in great detail.

This book, however, aims to take the individual tools, bits of configuration and options, and bring them all together into solutions for specific problems.

Who Should Read This Book

If you're new to Docker, this book is not for you.

I would recommend you start with the WatchMeCode Guide to Learning Docker if you need to learn Docker from the ground, up.

If you're already using Docker to run services like MongoDB, RabbitMQ, Oracle, etc, and you're looking for a resource to show you the basics of configuring Docker to run a Node.js application, this book is not for you, either.

The WatchMeCode Guide to Building Node.js Apps in Docker will show you how to configure a Docker container, run Node.js, get Express.js installed with `npm install` and more.

If you're already using Docker, already know how to configure a basic Node.js and Express.js application in a container, but you find yourself running into difficult situations with debugging, reloading code, and running your tool set within Docker, *this book is for you.*

Within these recipes (chapters), you'll find answers to the problems that plague developers beyond the basics of standing up a Docker container.

You'll learn how to combine tools, command-line options, Dockerfile configuration items and more, into solutions for the unique problems that Node.js and JavaScript developers face when building applications within a Docker container.

How This Book Is Written

Each of the recipes (chapters) of this book will address a specific circumstance and problem, that requires a specific set of tools for a solution.

But, the goal of this book is to provide more than just text for you to read, to give you an idea of what to do.

These problems and solutions are taken directly from real-world experiences and problems, with solutions that can be applied to many different applications.

Many of these solutions are handled with configuration and code that can be re-used, as well. And whenever possible, these re-usable elements will be provided to you as additional resources outside of the book.

Where To Get The Code and Configuration

This is a work-in-progress, at the moment. It’s possible that a private Github repository will be created, or a simple .zip file will be included as a part of this book’s bundle.

Recipe 4: Editing Code with Volume Mounts

With your core Docker images created and your code up and running, it's time to think about editing within the container.

However, a Docker image is immutable - meaning once the image is created, it can't be changed. Additionally, changes made directly inside of a Docker container (an instance created from an image), will be reverted back to the image version, when the container is shut down.

So, how do you edit code in a Docker container if you can't change or save anything?

Host mounted volumes.

These allow you to specify a folder from your host operating system and mount it directly inside of a container, at a path you specify. By mounting your project's code folder into your docker container, you can edit the code directly on your host operating system (with your favorite editor!) and the changes will be immediately reflected in the Docker container.

The Recipe

This recipe will modify the way in which you `docker run` a container instance, adding a `-v` flag to specify a host mounted volume.

Recipe Listing

```
docker run -v /my/local/folder:/var/app # ...
```

Cooking Instructions

When using the `-v` option with `docker run`, you need to specify a full path to the current folder on the left-hand side of the `local:remote` value for the flag.

The `local` path must start with a `/`. If it does not begin with a `/`, then Docker treats this as a named volume and not a host mounted volume (see Recipe 6 for more info on that).

The `remote` path should be the correct folder where you code lives. In this case, it will be the `/var/app` folder that was created in the Dockerfile. This is where the code for the application is copied during the image build.

Specifying a `-v` flag with a remote destination of the `/var/app` folder will tell Docker to use your local host folder instead of the folder contained in the Docker image.

Example Local Folder Specifications

On a macOS host, you can specify a local folder such as `/Users/derick/dev/docker-recipes/recipe-4/` on the left-hand side of the `-v` flag value. Note that you cannot use the ~`/` shortcut as the path must start with `/`.

Linux machines will be similar to macOS, though the `Users` folder will differ slightly in name. For example, on an Ubuntu Linux machine, the folder would be `/home/derick/...`

On Windows machines, you have to specify the drive letter using the `//c/...` format. For example, if you have a folder at `c:\dev\docker-recipes\recipe-4`, you would specify `//c/dev/docker-recipes/recipe-4` for the volume mount.

Expanded Environment Variables

It’s often difficult to get the path to your current project correct, when creating host mounted volumes. With long path names, the chance for mistakes increases greatly.

Fortunately, you can use environment variable to expand the current folder (or “process working directory”) into the `-v` value.

On a macOs or Linux machine, use the `$PWD` variable in place of the volume location.

```
docker run -v $PWD:/var/app/ # ...
```

In a Windows command shell, use `%CD%`.

```
docker run -v %CD%:/var/app/ # ...
```

Or in a Windows PowerShell environment, use `${PWD}`

```
docker run -v ${PWD}:/var/app/ # ...
```

Bring Your Own Editor

With the `-v` flag set correctly and your code mounted from the host environment into the Docker container, you can now edit your code with your own editor.

This is great news! You will not be limited to using text-based editors within a Docker container. Instead, you can use any editor installed on your host computer.

Whether that is Visual Studio, VS Code, vim, emacs, Webstorm, Eclipse, or anything else, you will have complete control and freedom in choosing an editor.

What's Next?

Editing code is only part of the picture for development. You'll find, over time, that you need to install new and updated npm modules and other runtime dependencies. This can be problematic with Docker and host mounted volumes, as they tend to be rather slow in performance on Windows and macOS.

Coming up next, you'll see how to correct for this using Docker's volume instructions.

Recipe 8: Debug with VS Code

You have a container that is configured for development, and you need to debug some code. The command-line debugger may be an option, but you don't have time to learn a new tool and get comfortable with it before your deadline for the next delivery.

Fortunately, the editor you are using - Visual Studio Code - has a debugger built into it, and you're already familiar with it. In the past, you've had it configured to launch your Node.js application with the debugger. Now, however, you need to launch the debugger and have it attach to the running code inside of your Docker container.

Can I Use My Favorite Debugger?

YES!

While this recipe does provide configuration for VS Code, the concepts and other configuration items are applicable to any editor that allows remote debugging via a URL.

The Recipe

This recipe is split into three parts, requiring changes to your Docker image, configuration for VS Code, and command-line tools to allow a remote debugger to attach.

Part 1 sets the NODE_ENV and exposes port 5858, which is the port on which the Node.js debugger will listen.

Recipe, Pt 1: dev.dockerfile

```
FROM node-recipes:prod

# ensure development configuration is used
ENV NODE_ENV=development

# allow a debugger to attach
EXPOSE 5858
```

Part 2 adds configuration to VS Code, in the `launch.json` configuration file. The specific settings and highlights will be explained, below.

Recipe, Pt 2: launch.json

```
{
    "configurations": [
        {
            "type": "node",
            "request": "attach",
            "name": "Attach to Process",
            "address": "localhost",
            "port": 5858,
            "localRoot": "${workspaceRoot}/",
            "remoteRoot": "/var/app"
        }
    ]
}
```

Part 3 tells the Node.js runtime, inside of your container, to start the debugger.

Recipe, Pt 3. Start The Node Debugger

```
docker exec <container> ps -a
docker exec <container> kill -s SIGUSR1 <pid>
```

Cooking Instructions

Part 1 of the recipe modifies your `dev.dockerfile` to expose port 5858 along with port 3000.

Build this image as you normally would, and run it with both ports mapped to your local machine.

Build and Run, Mapping Port 5858

```
docker build -t node-recipes:dev -f dev.dockerfile .
docker run -d                  \
  -p 3000:3000                 \
  -p 5858:5858                 \
  -v <local/folder/>:/var/app \
  --name node-recipes-dev      \
  node-recipes:dev
```

Once you have the application up and running, you will need to force the debugger to start, from within the container. This is done with the `kill -s SIGUSR1 <pid>` command.

Node.js reserves the `SIGUSR1` signal for starting the debugger. While you can trap this signal and run other code in response to it, you cannot prevent the Node runtime from starting the debugger.

Get the PID for the running application, by executing `ps -a` and then `kill` that PID, as shown.

After the node debugger has been started in the container, you can attach the VS Code debugger.

Recipe 13: Encapsulate Command-Line Options

For all of Docker's power and capabilities, there is one major flaw that consistently trips up even the most seasoned of Docker experts: command-line options.

It's not secret that Docker comes with a large amount of command-line tools, an incredibly lengthy command-line option list, and often requires command-line values that are extremely verbose.

This just makes it easy to make mistakes. Constantly.

Like any good developer, however, there are simple solutions and automation tools that we can use to solve the command-line complexity problem.

By using any of a number of command build tools or command-line / shell scripting tools, you can easily create a set of very simple scripts that will take care of 90 to 95 percent of your Docker command-line needs.

But which tool should you choose for this?

If you're on Windows, you could write .cmd scripts or PowerShell scripts. If you're on Linux or macOS, you could write .sh shell scripts. But writing individual scripts inside of individual files clutters your folder structure. And if you have a cross-platform team, you'll end up duplicating these scripts for each platform.

Instead, you should look to the myriad of build tools available. These tend to be cross-platform, and will encapsulate multiple commands and scripts into a single file.

The most commonly available build tools include Gulp, Grunt, Make and npm scripts. Each of these has their own benefit, but not all of them fit well with the goal of executing command-line tooling. You can throw Gulp and Grunt out the window almost immediately, for this deficiency.

This leaves you with Make and npm scripts (among many others not mentioned here). B since you're a Node.js developer, it makes sense to choose npm scripts. Use the tools that you already have, to your advantage.

The Recipe

The `npm` command-line tool may seem like a bad choice off-hand. Aren't you just trading one command-line tool for another? Technically, this is true. The benefit that `npm` offers, though, is encapsulating many scripts and command-line parameters into a single command.

This recipe will show you how to encapsulate some common commands from the rest of this book, into simple one-liners with no additional parameters required.

Recipe Listing: package.json

```
{
...

  "scripts": {
    "docker:build": "docker build -t docker-recipes .",
    "docker:build:dev": "docker build -t docker-recipes:dev -f dev.d\
ockerfile .",
    "docker:run": "docker run -d -p 3000:3000 --name recipes-prod do\
cker-recipes",
    "docker:run:dev": "docker run -d -p 3000:3000 -p 5858:5858 -p 35\
729:35729 -v $PWD:/var/app --name recipes-dev docker-recipes:dev",
    "docker:shell:dev": "docker exec -it recipes-dev /bin/sh; exit 0\
",
    "docker:shell:dev:root": "docker exec -it --user root recipes-de\
v /bin/sh; exit 0",
    "docker:clean": "docker stop recipes-dev recipes-prod; docker rm\
 recipes-dev rcipes-prod; docker rmi docker-recipes docker-recipes:d\
ev; exit 0"
  },
...
}
```

The content from the above listing will go into your `package.json` file, in the `scripts` section. Each command can be executed with `npm run <command name>` in a terminal window or command prompt.

Cooking Instructions

Each of the scripts does essentially one thing - even if that "one thing" involves multiple commands. This is the nature of encapsulation - hiding the details and giving the higher concept a common name.

You'll notice the use of `:` to separate sub-commands and grouping of commands. This is not any official structure or anything - it's simply a way of distinguishing commands that are related to each other.

docker:build

```
npm run docker:build
```

This command will build your production Docker image from the `Dockerfile`, applying a tag of `docker-recipes` and using a build context of the current folder.

The result is a Docker image that can be used for creating containers or other images.

docker:build:dev

```
npm run docker:build:dev
```

This command will build your development image from the `dev.dockerfile`. It applies a tag of `docker-recipes:dev` and uses the current folder as the build context.

Since the dev.dockerfile builds `FROM docker-recipes`, you will need to have a build of the production image before running this command.

docker:run

```
npm run docker:run
```

The name seems a little redundant, but it also fits the effect: running a Docker container from the production image that you previously built.

This command maps TCIP/IP port 3000, and names the new container `recipes-prod`.

docker:run:dev

```
npm run docker:run:dev
```

Similar to the `docker:run` command, this one creates a new Docker container from the development image you had previously created. The new container is named `recipes-dev`.

In addition to TCP/IP port 3000, this command also maps port 5858 for debugging purposes.

docker:shell:dev

```
npm run docker:shell:dev
```

This command opens a shell into the existing `recipes-dev` container. Since the container has a `USER app` instruction in the Dockerfile, the shell will run as the `app` user.

docker:shell:dev:root

```
npm run docker:shell:dev:root
```

This command also opens a shell into the existing `recipes-dev` container. However, the `--user root` parameter is added, to ensure the shell is executed as the `root` user in the container.

docker:clean

```
npm run docker:clean
```

The nuclear option - completely stop and remove the Docker Recipes containers and images. This will delete everything created from the previous commands.

What's Next?

As you build your Docker infrastructure for your project, you should encapsulate the commands that you find yourself repeating. The above list is a good place to start, but will certainly not be the end result for your project. You'll want to add new commands as you need them, modify the existing commands to suit your needs, and remove commands that don't fit your environment.

Additionally, the above commands are an extremely important part of a good Docker development process. But they only represent a part of the process.

For more information on how to perfect your Docker builds and development process, see the Debugging Docker Images video and resources (included in the Webinar bundle of this book, or sold separately at the above link).

About Derick Bailey

Hello, my name is Derick Bailey.

I’m an entrepreneur and software developer, a consultant, screencaster, blogger, speaker, and so more. I’ve been working professionally in software development since the late 90’s and have been writing code since the late 80’s.

You can find me writing about these and many other subjects at my blog. I also produce screencasts, provide information about my consulting services, and more, through the following websites:

- My Blog: DerickBailey.com
- Screencasts (free and paid): WatchMeCode.net
- Open Source Projects: GitHub.com/DerickBailey
- Entrepreneurial Podcast: The Entreprogrammers

If you have any questions, comments or concerns, you can contact me using the following:

- Email: derick@mutedsolutions.com
- Twitter: @derickbailey

Thanks For Reading

I hope you've enjoyed …

If you need anything else, have any questions or want to know more about how I work with Docker and Node.js, let me know! I'll be happy to answer any question you have, and I love getting feedback (good and bad).

Thanks for reading, and happy Dockering. :)

Derick