

28/Nov/2016



C o m m u n i t y E x p e r i e n c e D i s t i l l e d

Learning Bootstrap 4

Second Edition

Unearth the potential of Bootstrap 4 to create highly responsive and beautiful websites using modern web techniques

Matt Lambert

[PACKT] open source*
PUBLISHING community experience distilled

www.allitebooks.com

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows:

"Bootstrap has taken `normalize.css` and extended it with a new module. "

A block of code is set as follows:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Required meta tags always come first -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
```

Any command-line input or output is written as follows:



Then, when something changes, you only have to update a few files instead of 500. Once your website is complete, you then generate a version that is plain HTML, CSS, and JavaScript, and deploy it to your server. This is what I would call creating your own frontend web development environment. This is also how most people work on larger projects nowadays to keep them manageable.

Converting the base template to a generator

Why don't we integrate the basic template into a generator so I can show you what I'm talking about? My generator of choice is called **Harp.js** and you can install it over at <http://harpjs.com/>.

Before we get too far ahead of ourselves, we need to install **Node.js**. Harp runs off Node.js so it's a dependency you'll need to use. If this feels too advanced for you, feel free to skip ahead to *Chapter 2, Using Bootstrap Build Tools*. This section is totally optional. Head to the following URL to install Node.js if you don't already have it on your computer: <https://nodejs.org/download/>.

Follow the instructions on the Node.js website and, once you've finished installing it, run the following command in a command-line application such as Terminal or Cygwin:

After the installation completes, run the following command to get the Harp version number, which will also confirm that the installation was successful:

Compiling your project

Now that the template files are ready, we need to compile the project before we can preview it in the browser. Head back to your command-line app and make sure you are in the root of your project directory. Once there, run the following command to compile the project:

Installing Surge

To install Surge, you'll need to open up your Terminal again. Once you have done this, run the following command:

Installing Node.js

If you skipped installing Node.js in Chapter 1, *Introducing Bootstrap 4* then now is the time to follow along and install and configure all your build tools. Let's start by heading to its website, <https://nodejs.org>, and downloading Node.js:

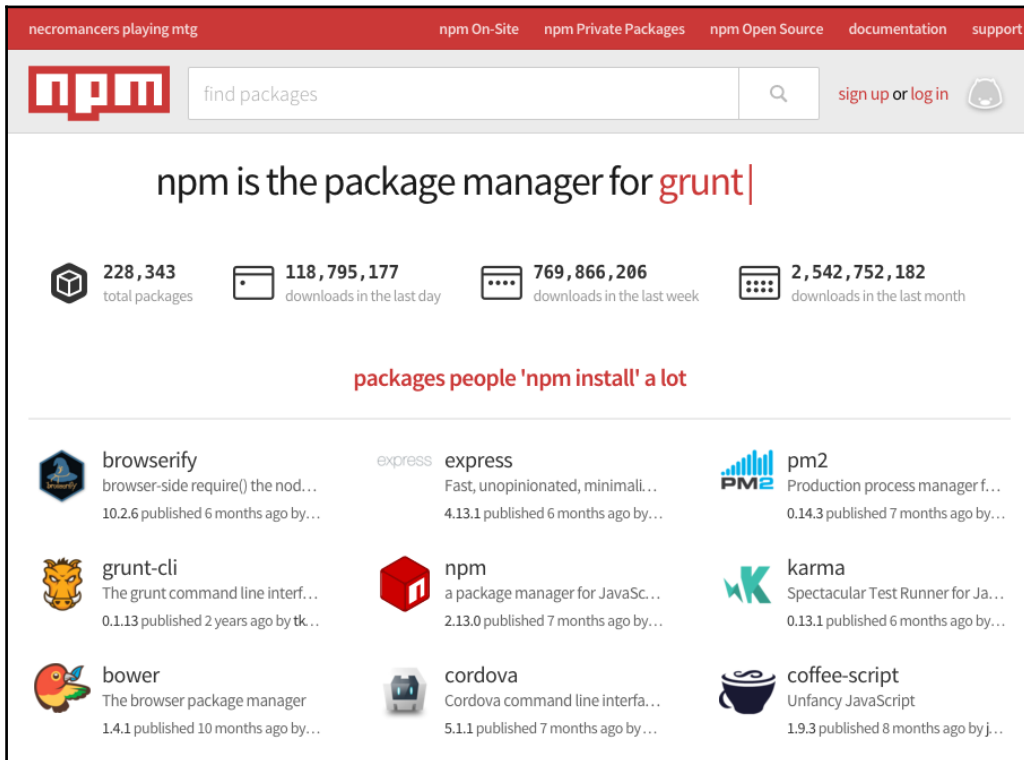


Node is a JavaScript runtime that uses Google Chrome's V8 JavaScript engine. What that means is that Node is a JavaScript-based web server that you can run locally or in production. It includes an event-driven, non-blocking I/O model which is easy to use and lightweight. Node comes with a built-in package manager called `npm` which includes the largest ecosystem of open source libraries on the Web.

Follow the installation instructions on the Node.js website and once you're done, open up a command-line application such as Terminal or Cygwin. Run the following command:

Updating npm

Now that Node is installed, let's ensure that the latest version of npm is also installed. npm is a package manager for Node and allows you to install useful tools such as Grunt, which we'll do in our next step.



The screenshot shows the npm website homepage. At the top, there's a navigation bar with links: "necromancers playing mtg", "npm On-Site", "npm Private Packages", "npm Open Source", "documentation", and "support". Below this is a search bar with the npm logo and the text "find packages". To the right of the search bar are links for "sign up or log in" and a GitHub icon. The main heading reads "npm is the package manager for **grunt**". Below this, there are four statistics: "228,343 total packages", "118,795,177 downloads in the last day", "769,866,206 downloads in the last week", and "2,542,752,182 downloads in the last month". A section titled "packages people 'npm install' a lot" follows, displaying a grid of popular packages: browserify, express, pm2, grunt-cli, npm, karma, bower, cordova, and coffee-script. Each package entry includes its icon, name, a brief description, and its version and publication date.

To make sure the latest version of npm is installed, run the following command in the Terminal:



Once the update is complete, we can safely start to install the other packages we'll need for our Bootstrap development environment.

Installing Grunt

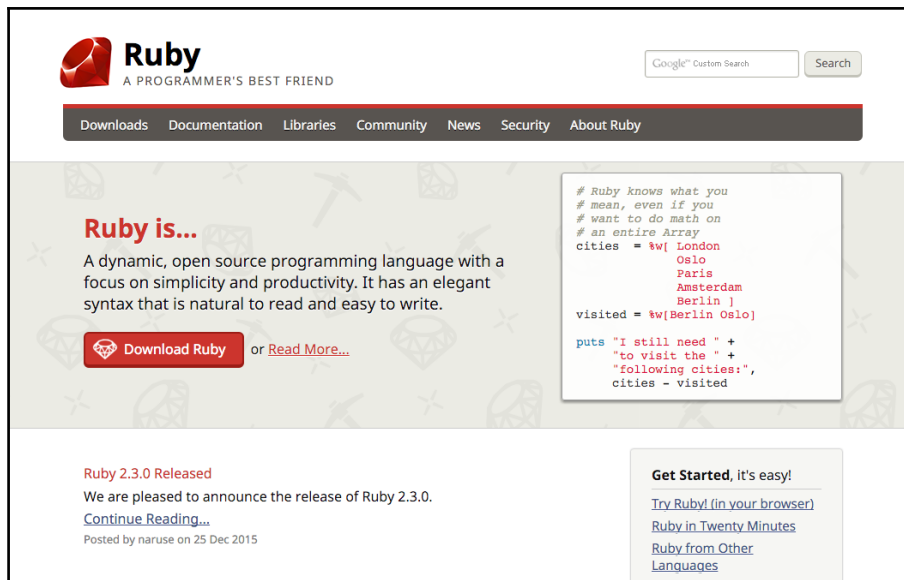
Grunt is a JavaScript task runner and it's the tool that will do the actual compiling and building of the development Bootstrap files into the production versions.



Grunt provides automation and allows you to chain together repetitive tasks such as compiling, minification, linting, and unit testing. Therefore, it's commonly used in frameworks such as Bootstrap to build the source files into production. To install Grunt, run the following command in the Terminal:

You should expect to see something like this printed out in the Terminal:





In Bootstrap, Ruby is used to run the documentation website and to compile the core Sass files into regular CSS. For the Bootstrap documentation, you can always visit <http://getbootstrap.com/>. However, in some cases, you may find yourself offline, so you might want to install a local version of the docs that you can use. Let's first start by installing Ruby before we get to the documentation.

Good news! If you're on a Mac, Ruby comes pre-installed with OS X. Run the following command to check the Ruby version number and verify that it's available:

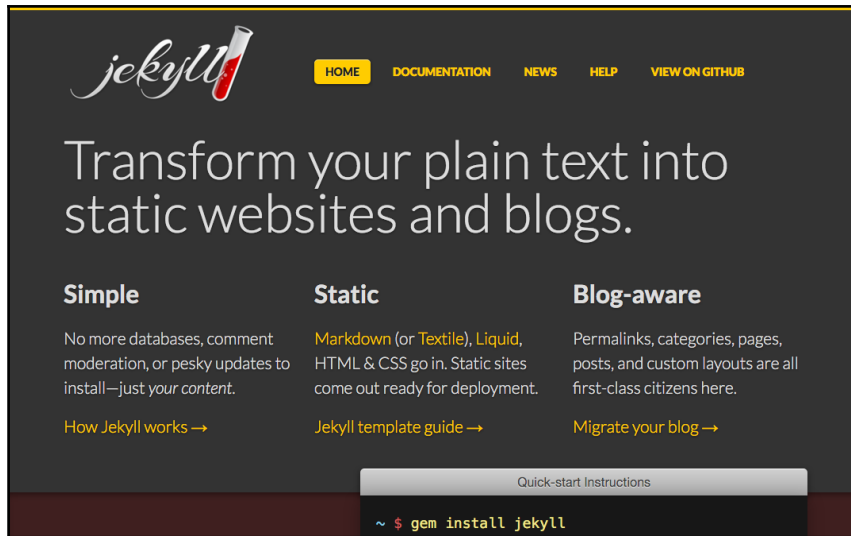
Installing the Bundler gem

After Ruby is ready to roll, you need to install a Ruby gem called Bundler. In the words of the developers of Bundler: *Bundler provides a consistent environment for Ruby projects by tracking and installing the exact gems and versions that are needed.* For more info on Bundler, please visit <http://bundler.io/>.

Don't worry too much about what Bundler does. The important thing is to just install it and move on.

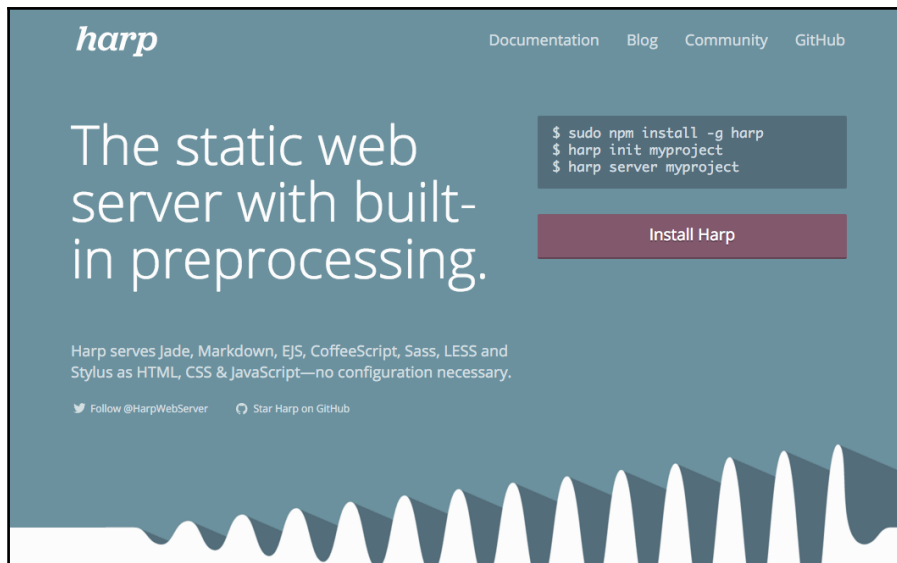
1. To do this, we need to run the following command in the Terminal in your Bootstrap source file root directory:





Running the documentation

Getting the documentation running locally is actually pretty easy. From the root of the Bootstrap source file directory, run the below command in the terminal:



Why use Harp.js

There are a number of great arguments for using a static site generator such as Harp.js: cleaner code, modern best practices, and more. However, the best reason is that it will just make your life simple. Instead of having to update a header on all 50 pages of the website, you can simply update the header partially and then have that compiled into all your templates. You can also take advantage of using variables to insert content and configuration.

Installing Harp.js

Harp is another project that runs on Node.js so we can use `npm` to install it with the following command:



To confirm that Harp was successfully installed, let's use our version-checking trick by entering the following command into the Terminal:

Creating our first page template

For our first page template, we're going to create our `Home` or `index` page. In the root of the blog project, create a new file called `index.ejs`. Note this file is not prepended with an underscore like the previous files. With Harp, any file that has the underscore will be compiled into another and ignored when the files are copied into the production directory. For example, you don't want the compiler to spit out `layout.html` because it's fairly useless with the content of the `Home` page. You only want to get `index.html`, which you can deploy to your web server. The basic thing you need to remember is to *not* include an underscore at the beginning of your page template files. Once you've created the file, insert the following code:

```
<div class="container">
  <div class="row">
    <div class="col-lg-12">
      <h1>hello world!</h1>
    </div>
  </div>
</div>
```

To get us started, I'm going to keep this really simple. Here's a quick breakdown of what is happening:

- I've created another `.container` which will hold the content for the `Home` page
- Within the container, there is a full-width column. In that column, I've inserted an `<h1>` with a `hello world!` message

That will be it for now. Later on, we'll build this page out further. Save the file and close it. We've now completed setting up all the basic files for our Harp development environment. The last step is to compile the project and test it out.

Compiling your project

When we compile a project in Harp, it will find all the different partial, layout, and template files and combine them into regular HTML, CSS, and JavaScript files. We haven't used any Sass yet but, as with the template files, you can have multiple Sass files that are compiled into a single CSS file that can be used on a production web server. To compile your project, navigate to the root of the blog project in the Terminal. Once you are there, run the following command:

If everything worked, a new blank line in the terminal will appear. This is good! If the compiler spits out an error, read what it has to say and make the appropriate changes to your template files. A couple of common errors that you might run into are the following:

- Syntax errors in `_harp.json` or `_data.json`
- Syntax errors for variable or partial names in `_layout.ejs`
- If you have created additional page templates in the root of your project, and *not* included them in `_data.json`, the compile will fail

Once your compile is successful, head back to the root of the blog project and notice that there is a new `www` directory. This directory holds all the compiled HTML, CSS, and JavaScript files for your project. When you are ready to deploy your project to a production web server, you would copy these files up with FTP or using another means of file transfer. Every time you run the harp compile command in your project, these files will be updated with any new or edited code.

Running your project

Harp has a built-in web server that is backed by `Node.js`. This means you don't need a web hosting account or web server to actually test your project. With a simple command, you can fire up the built-in server and view your project locally. This is also really great if you are working on a project somewhere with no Internet connection. It will allow you to continue building your projects Internet-free. To run the server, head back to the Terminal and make sure you are still in the root directory of your blog project. From there, enter the following command:

This is just a taste of the properties you can use to customize the Flexbox grid. As I mentioned previously, I just wanted to give you a quick introduction to using Flexbox and some of the terminology that is needed. Let's take what we've learned and build on that by building a Flexbox grid in Bootstrap.

Setting up the Bootstrap Flexbox layout grid

Whether you are using Flexbox or not, the grid is based on Bootstrap's regular row and column classes. If you are familiar with the Bootstrap grid, this will work exactly as you expect it to. Before you start any Bootstrap project, you need to decide if you want to use a Flexbox or regular grid. Unfortunately, you can't use both at the same time in a Bootstrap project. Since the focus of this chapter is on Flexbox, we'll be using the appropriate grid configuration. By default Bootstrap is set up to use the regular grid. Therefore, we are going to need to edit the source files to activate the Flexbox grid. Let's start by downloading the source files again from <http://v4-alpha.getbootstrap.com/>.

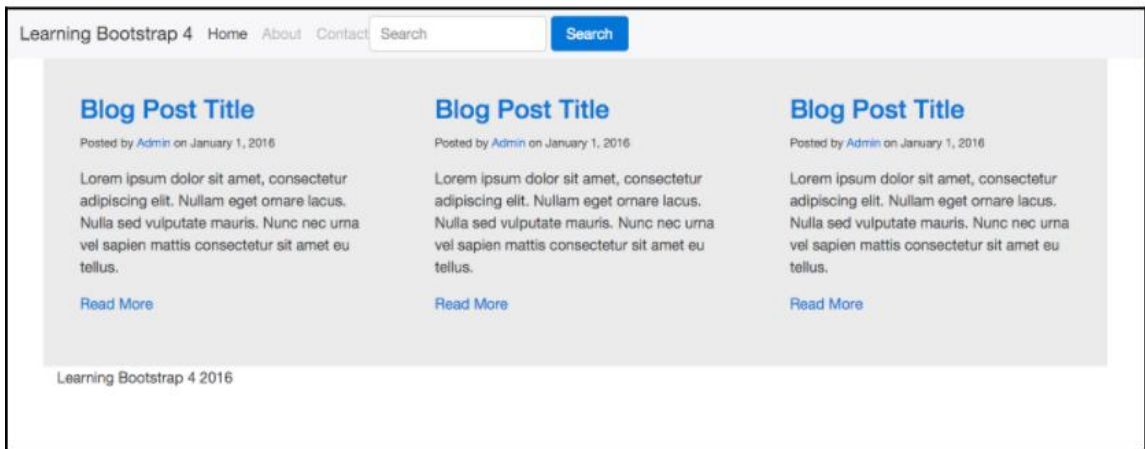
Once you've downloaded the ZIP file, expand it and rename it so you don't get confused. Call it something like `Flexbox Bootstrap`. Next we'll need to edit a file and recompile the source files to apply the changes.

Updating the Sass variable

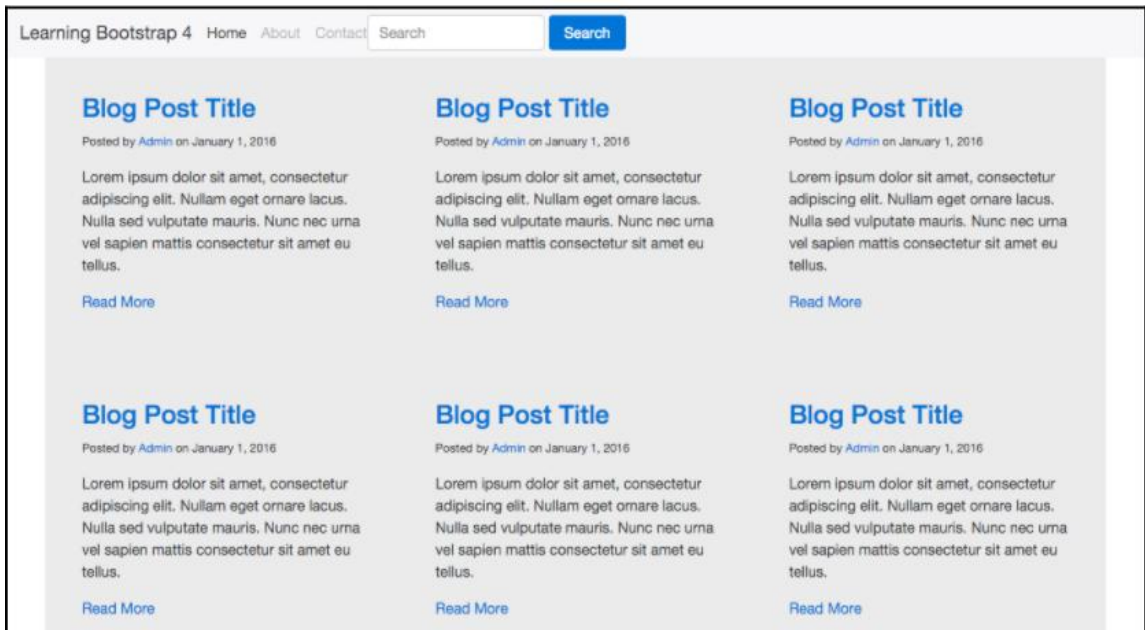
To use the Flexbox grid, we need to edit a Sass variable in the `_variables.scss` file. The way Sass variables work is that you set a single value in the `_variables.scss` file. When you run the built-in compiler, that value is written into every component of the Bootstrap framework where it is needed. You can then grab the compiled `bootstrap.min.css` file and it will have all the required code you need to use the Flexbox grid:

1. In your new source file directory, using the Terminal, navigate to:

4. Save the file and close it. Before this change is applied, we need to recompile the source files. Since we downloaded a new version of the source files, we'll need to reinstall the project dependencies. Navigate to the root of the new Flexbox source files in the Terminal and run the following command:



Great, now we have a decent-looking blog homepage. However, we need to add more posts to fill this out. Let's go ahead and add more column `<div>`s inside the same row. Since this is Flexbox, we don't need to start a new `<div>` with a row class for each row of posts. Let's add three more posts in then see what it looks like:



Testing out the blog home page layout

Let's test it out in the browser to make sure it's looking the way we want. Before we can do that we'll need to compile our code with Harp. Open the Terminal back up and navigate to the project directory for this chapter's code that we created. Run the `harp compile` command, here it is again in case you forgot:

If all went as planned, your blog post page should look like this:

