LOCOMOTIVECMS

LocomotiveCMS Crash Course

LocomotiveCMS will help you develop faster websites that your clients will love to use.
In this course, you'll learn how to create websites like never before.
Have more questions? Contact us at contact@locomotivehosting.com

1. Introduction to LocomotiveCMS

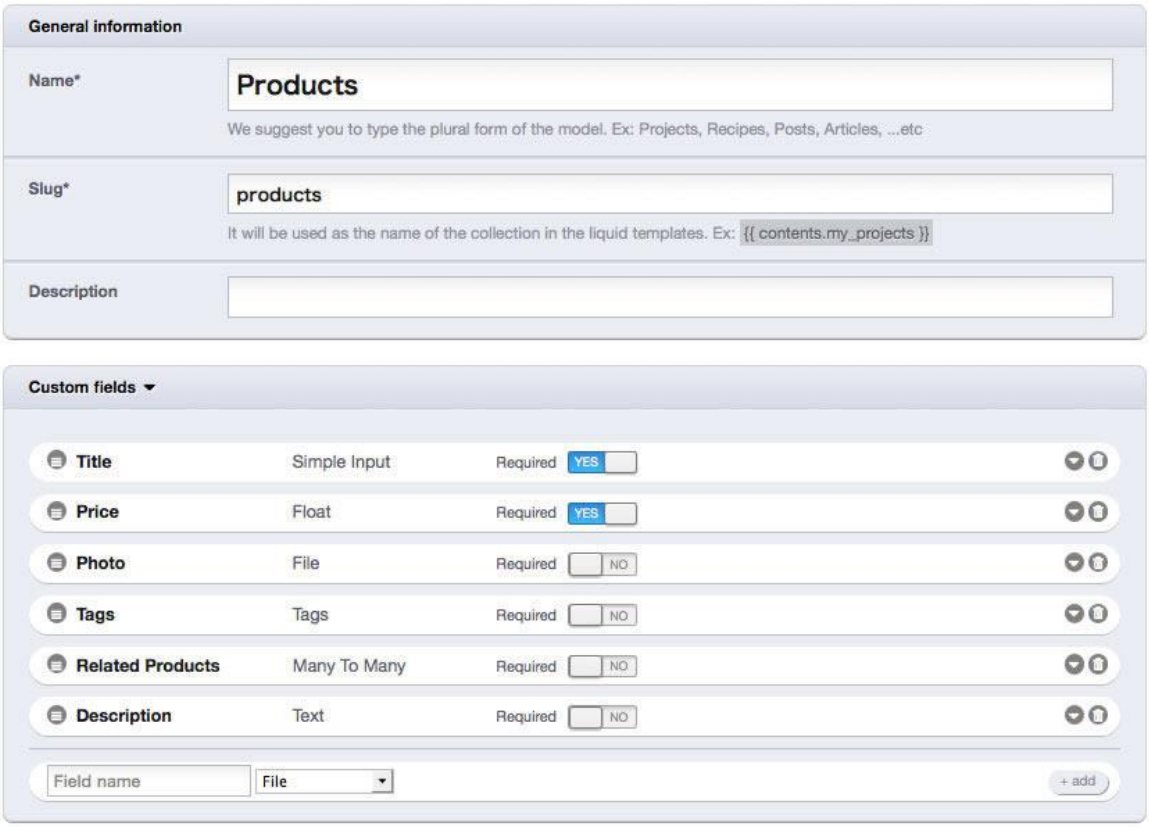
Hello and thanks for signing up for the LocomotiveCMS Crash Course! This first part of the seven part series is just going to tell you a bit about LocomotiveCMS and why it's so great.

LocomotiveCMS is a free and open source content management system, or CMS, for websites. LocomotiveCMS's slick back-office editor makes managing site content easy for users and Wagon development tool makes coding these sites faster and more fun than ever for developers.

Here are a few reasons you might want to use LocomotiveCMS.

Structure content easily

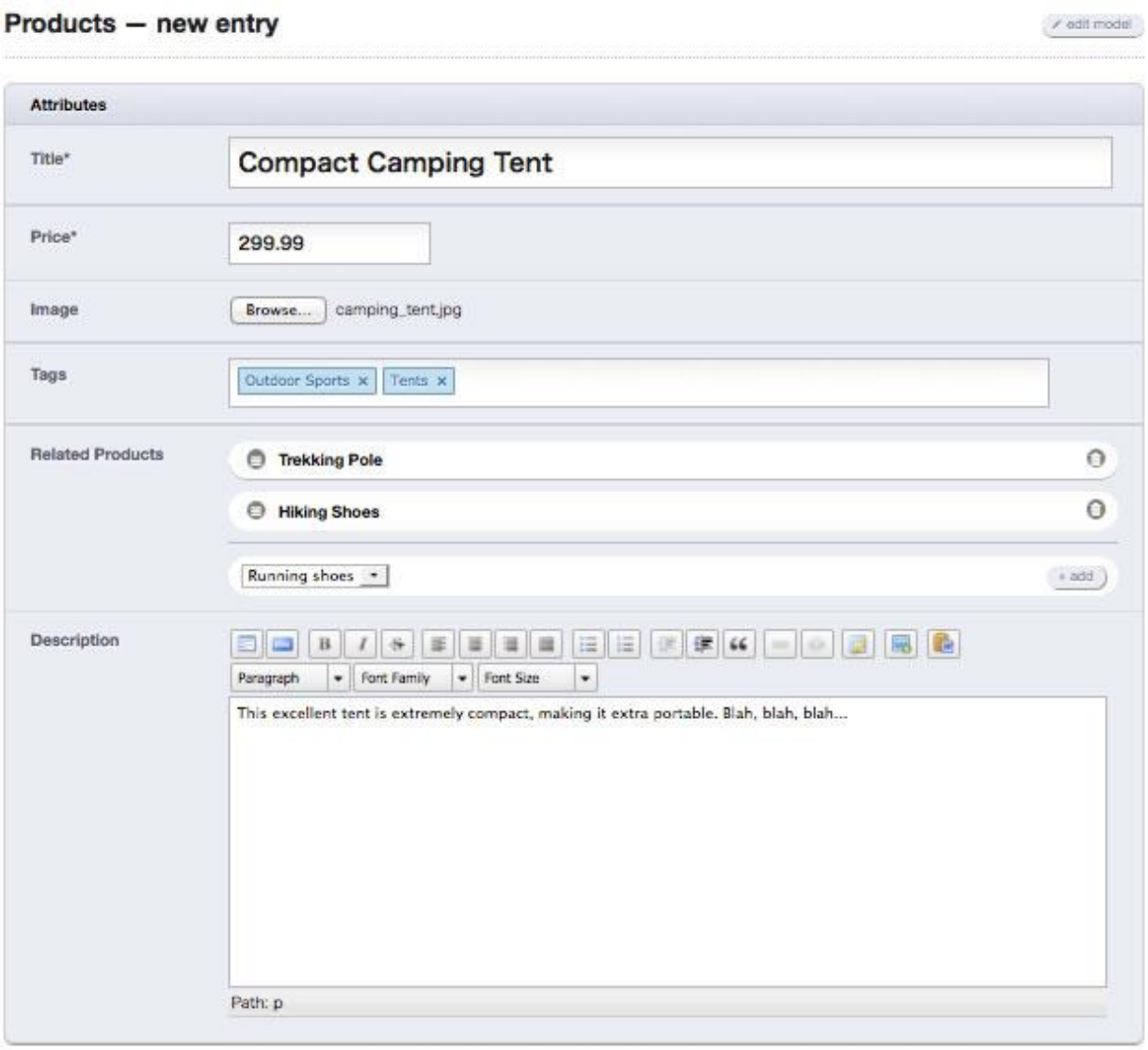
Creating custom content types that properly organize your data is a breeze. Let's say we have some *products* for a catalog with the following custom fields: *title*, *price*, *photo*, *tags*, *related products* (a list of other similar products), and *description*.
In LocomotiveCMS we can easily create this model using the back-office editor seen below.



Or if you prefer text to GUIs, we could define this content type by adding a single YAML file that reads like plain English to our site's wagon.

Stylish back-office

How about adding in some actual products? That's easy, too since the back-office automatically provides a CRUD interface for all content types.
Just login into the LocomotiveCMS's stylish back-office and add products using the intuitive editor pictured below.



Beginners welcome

To start making custom templates that allow you to control every aspect of your site's design and add functionality, you need basic knowledge of just three things: HTML, CSS, and Liquid Markup.

If you've never heard of Liquid, don't panic; most people can pick up the basics in less than a day. If you can understand the code below, you're most of the way there.

```
{% for product in contents.products %}
   <div class="product">
       <h3>{{ product.title }}</h3>
       <div class="description">{{ product.description }}</p>
```

```
{% for product in contents.products %}
   <div class="product">
       <h3>{{ product.title }}</h3>
       <div class="description">{{ product.description }}</p>
   </div>
{% endfor %}
```

Powerful

LocomotiveCMS has built-in (but optional) support for HAML, SASS, LESS, and CoffeeScript. You can also enhance LocomotiveCMS with custom Liquid tags, filters, and drops. And since LocomotiveCMS is just a Rails engine, you can extend your site using Ruby on Rails.

Thanks to the LocomotiveCMS's Restful API, you can access your content both inside and outside of Rails. This is handy for developers using frontend Frameworks like Angular or Ember.

Multi-Lingual

Out of the box, LocomotiveCMS supports a cool 13 languages (English, German, French, Polish, Brazilian Portuguese, Italian, Dutch, Norwegian, Spanish, Russian, Estonian, Japanese, and Chinese), meaning your clients can use the back-office interface in their own language.

LocomotiveCMS also supports full localization of your site and adding content in as many languages as your want. No hay problema.

2. Wagon and Engine

Developing a CMS site is often tedious, because most solutions get in the way of your workflow. To solve this problem, LocomotiveCMS is divided into two parts: Engine and Wagon. Let's quickly go over what each one does.

Engine

The LocomotiveCMS Engine is a Rails engine that's installed into a Rails application. This engine runs in production, dealing with page requests, handling the restful API, and running the back-office where site editors can update the site's content.

In general, the only time you'll spend with Engine is at the beginning when you're setting up LocomotiveCMS. You'll have to install the gem, do any needed configuration, and get Rails running, but after that, you're pretty much done with the Engine.

And if you're using LocomotiveHosting, you won't need to worry about Engine, since we take care of setting up Rails and the LocomotiveCMS Engine for you.

Wagon

Wagon is a command line tool that let's you develop for LocomotiveCMS on your local machine.

With Wagon you can generate the scaffolding for a new LocomotiveCMS site and start adding the content types and templates you need using any text editor. You can also preview the site with your computer's browser by using the `wagon serve` command.

Once everything looks good, you can deploy your site to any LocomotiveCMS engine using the `wagon push` command. Your changes will immediately be reflected on that site without restarting or making any changes to the Rails app.

To help you work faster, Wagon comes with support for tools like SASS, LESS, HAML, and CoffeeScript and works well with source version control systems like git or svn.

By now hopefully you're itching to get your hands dirty and start making a LocomotiveCMS site, because tomorrow we'll be covering how to install Wagon on your local machine and create a LocomotiveCMS site.

3. Getting started with Wagon

It's finally time to get your hands dirty and setup the LocomotiveCMS development environment. Installing Wagon and its dependencies can be a little tricky if you're new to the command line, but the step-by-step guides below will get you up and running in no time.

Installing Wagon on Mac OS X / Windows / Linux

Wagon installed? Great. To create a new site let's open a terminal shell and run the `wagon init` command.

```
$ wagon init acme_corp
Do you prefer HAML templates ? no
```

This command creates a new directory, `acme_corp`, with the scaffolding for a LocomotiveCMS site inside. The command asks us if we prefer HAML templates over HTML templates. If you prefer HAML, just type yes. If you don't or you're not sure what HAML is, type no.

The command's output gives us some instructions to get us started.

```
cd ./acme_corp
bundle install
bundle exec wagon serve
open http://0.0.0.0:3333
```

The first command will move us into the `acme_corp` directory that was just created and running `bundle install` installs Wagon's gem dependencies.

From here, you could start developing, but just to check that everything is working, let's run `bundle exec wagon serve`.

```
$ bundle exec wagon serve
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:3333, CTRL+C to stop
```

This launches the web server included in the Wagon gem. When the server is running, you won't be able to use the command line in this shell, so if you want to run a command you'll need to open another terminal window.

The last step is to open http://localhost:3333 in your web browser. If everything went well, you should see a relatively blank page except for the words *Home Page* and *Lorem ipsum*...., as per the contents of `app/views/pages/index.liquid`.

Using this address, we can instantly preview changes we make to our LocomotiveCMS site. Pretty neat, huh?
When you're finished testing your site, go back to the terminal and type `ctrl + c` to close the web server.

4. Creating a site with Wagon, Part 1

We installed Wagon, initiated a site called *acme*\corp_, and previewed it with the built in web server. Now, we'll follow that by actually building a site with wagon.

First task: go to the *acme*\corp_ project directory that we created yesterday and open `config/site.yml`. Let's change the `name` property to the name of our site, Acme Corporation.

```
name: Acme Corporation
```

Next, open *index.liquid* from the `app/pages/views` directory, where all the page templates are stored. Delete the file's current contents and paste in the following code.

```
---
title: Welcome to Acme
---
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>{{ page.title }} | {{ site.name }}</title>
    {{ 'styles.css' | stylesheet_tag }}
  </head>
  <body>
    <div id="logo"><img src="{{ 'logo.png' | theme_image_url }}" alt="Acme" /></div>
    <nav id="main-nav">
        <a href="/">Home</a>
        <a href="/products">Products</a>
    </nav>
    <div id="content">
      <h1>{{ page.title }}</h1>
      {% block 'main' %}
        <p>
          {% editable_text "welcome_message" %}
            Welcome to Acme, the world's favorite cartoon company.
          {% endeditable_text %}
        </p>
      {% endblock %}
    </div>
  </body>
</html>
```

Ok, so what's all this stuff do?

First thing to notice is the YAML header at the top. There are lots of header options we can use, but this file has just one, `title`, which sets the page's title.

Moving on, we'll encounter our first bit of Liquid Markup in the page's title tag: `{{ page.title }} - {{ site.name }}`. Those double curly brackets are Liquid's *output markup*, meaning whatever is in between the curly brackets will be output.

`page.title` is a variable to access the `title` property in the page's YAML header and `site.name` accesses the `name` property in `site.yml`. So why use these variables and not just <title>Welcome to Acme - Acme Corporation</title>?

Two reasons: the YAML header and `site.yml` variables are editable through the site's back-office GUI, so if a non-programming site editor ever wants to change these values, no coding, no problems.
Also `index.liquid` is going to serve as a template for other pages, so we'll want the contents of the title to change depending on what page is being rendered.

The last thing to note in the `<head>` is `{{ 'styles.css' | stylesheet*tag }}`, *which uses the* `*stylesheet*\tag` Liquid filter to insert a `<link>` tag to the `public/stylesheets/styles.css` stylesheet.

I won't go all the way into filters now, but here's an overview: filters come after a pipe character inside Liquid output markup and in some way modify that output.

We haven't made `public/stylesheets/styles.css`, so create the file and paste in the following contents.

```
body { text-align: center }
#logo, #main-nav, #content { width:600px; margin:10px auto; }
#content { text-align: left; }
```

Onto the `<body>`! You'll notice another liquid filter: `{{ 'logo.jpg' | theme_image_url }}`. This outputs an absolute path to the`public/images/logo.png` file. That file doesn't exist yet, so go ahead and find an image to put in there.

The next piece of Liquid markup we run into is `{% block 'main' %}`, and if you go down a bit further, you'll also see the closing `{% endblock %}`. This time instead of the output markup, we have single curly brackets and percentage signs, which is Liquid's *tag markup*.

Block tags define blocks, which are sections of Liquid code that can be overwritten by child templates. If that's unclear, don't worry, we'll come back to this.

Next, we have another Liquid tag: `editable_text`.

The contents of this tag is some text that we want to put in this <p> tag, so what's this editabletext stuff?

Well, what if next week the folks at Acme Corp. decide they'd prefer the text to read "*the world's greatest cartoon company*" instead of "*the world's favorite cartoon company*."

Normally, this means they email you, you change the copy, and re-deploy. Or we can use this handy editable\text tag, which let's the site editors change this text at any time using the back-office GUI. Sweet.

Let's double check that everything is looking the way we'd like so far.

Open a terminal, navigate to the `acme_corp` directory, and run run `bundle exec wagon serve`. Then go to http://localhost:3333 in your browser.

Ok, you won't be winning any awards for this website, but we've covered the bare necessities all in one email without breaking a sweat. If you're looking to dig a little deeper into what we covered today, feel free to stop by the LocomotiveCMS Documentation Site and look around.

5. Creating a Site with Wagon, Part 2

Let’s now add a little more to our website, for instance, some products.

We could create a products page and manage its content manually, but that wouldn't be very flexible. For example, what if we want different options for sorting the products? Or want to use the same product data on other pages?

A much better solution is to normalize the product data using a LocomotiveCMS content type.

Open a terminal, navigate back to the *acme*\corp project directory, and run the following command.

```
bundle exec wagon generate content_type products title:string description:text
```

This command generates a *products* content type and gives it two fields: title and description.

Title is a string field and *description* is a text field (text is a really long string). Running this command isn't required to make content types; it's just a convenient way to generate content type YAML files.

Let's open the generated `app/content_types/products.yml` and take a look.

Whoa nelly, look at all that YAML!

For your conveinence, `wagon generate` has added many common options to the file.

You can comment in and comment out these options as fits the needs of your content type.

For the time being you don't need to concern yourself with most of these options, but take note of several key ones.

```
- name: the human friendly name of this content type that will be displayed in the back-office
- slug: the underscored string used to refer to this content type in templates
- fields: an array of fields or properties for the content type. Fields should be underscored strings.
Each field is then further described by another YAML array of various options.
```

So our products content type is setup, but we need to add some products data if we want to properly test this. We can do that in the `data/products.yml` file, which has already been created for us by `wagon generate`.

The `wagon generate` command preloaded file with some filler entries. Let's replace that with our own data from Acme's infamous product catalog.

```
- "Earthquake Pills":
    description: "Why wait? Make your own earthquakes--loads of fun!"
- "Do-it Yourself Tornado Kit":
    description: "Seed your own Tornadoes."
- "Iron Carrot":
    description: "Fool your friends!"
- "Christmas Package Machine":
    description: "Makes neat, small packages."
```

Alright, now that we have products, we're ready to add a products page. Create a new file `app/pages/products.liquid`, and paste in the following code.

```
---
title: Products
---
{% extends parent %}
{% block 'main' %}
  {% for product in contents.products %}
    <div class="product">
      <h2>{{ product.title }}</h2>
      <p>{{ product.description }}</p>
    </div>
  {% endfor %}
{% endblock %}
```

What's this `{% extends parent %}` business?

In LocomotiveCMS, all page templates are part of an inheritance tree, at the base of which is`index.liquid`. Since we put `products.liquid` in the root page directory, its parent is `index.liquid` by default.

Adding the `{% extends parent %}` Liquid tag to a page makes that page, by default, a copy of the parent page. If this file were blank except for the YAML Header and `{% extends parent %}`, the */products* page would look exactly like the root page, except the places where we used the`page.title` variable.

Of course, having an exact copy wouldn't be very useful, which is why yesterday we defined a *main* block in `index.liquid`. By defining a *main* block in `products.liquid`, the `index.liquid` *main* block is overriden.

Inside the main block, we have a simple for loop that iterates over each product and outputs its title and description. Notice that we can access the data of content types using the *contents* global variable.

Go back to the terminal, run `bundle exec wagon serve`, and open the site at http://localhost:3333. As you can see, we now have both a home page and a page listing Acme's products. Since the products page extends the index page, the home page's header, logo, and menu have all been included automatically.

6. Hosting your LocomotiveCMS Site

We spent the past two chapters getting a super simple LocomotiveCMS site built and thanks to Wagon we were able to develop and test it quickly on our local machine.

It's time to put that site on the web. As you might remember from chapter 2 of the this crash course, Wagon is just one half of LocomotiveCMS. The other half is the Engine, the Rails app that serves the site's pages and the back-office editor in production.

One of the great things about LocomotiveCMS is that there are so many ways to host it. You can use PaaS providers or deploy LocomotiveCMS on dedicated or virtual private servers. In short, options abound.

We can't cover setting up all of these options in this email, but we've done our best to guide users through setting up a LocomotiveCMS Engine on our documentation site. See the Installing Engine Locally and Engine in Production guides for more information.

However, from the beginning, one of the biggest pain points LocomotiveCMS users have communicated to us, is how difficult it is to deploy and maintain LocomotiveCMS Engines.

We're not knocking LocomotiveCMS, it's just that LocomotiveCMS is built on top of some technologies which can be tricky to configure and maintain: Ruby, Rails, and MongoDB.

Depending on the exact setup, users encounter a host of issues: ranging from optimizing Unicorn performance to implementing security measures and everything in between

.

Which is why we decided to create LocomotiveHosting: a zero configuration, hassle free hosting option for LocomotiveCMS. A few of LocomotiveHosting's features include:

- **Hassle free deployment**: just click the new site button and everything is taken care of for you. Nothing to setup, nothing to configure.
- **Expand easily**: one LocomotiveHosting account hosts unlimited sites.
- **No effort upgrades**: when new versions of LocomotiveCMS are released, your sites are instantly upgraded.
- **Daily backups**: all your sites backed up daily.
- **Expert support**: when you run into problems, get support directly from the LocomotiveCMS developers.
- **White label**: completely re-brand the locomotive back-office for your client or agency without changing a line of code.
- **Wagon Desktop App**: one perk soon to be available exclusively to LocomotiveHosting users, free of charge, is the Wagon Desktop App, which enables you to install Wagon with one click and use it on your desktop.

Fortunately, there's no need to take our word for any of this, because we offer a free trial to all users. You can't publish sites with your own domain until you upgrade, but you can sign up for a free account and deploy sites to a LocomotiveHosting.com subdomain, all for free. There's no time limit on the trial and no credit card is required.

Hopefully by now you've figured out how you want to host your site and have at least considered giving LocomotiveHosting a try. In the next chapter, I'll show you how to to setup your first site on LocomotiveHosting.

7. Setting up a Site with LocomotiveHosting

I'll now show you how easy using LocomotiveHosting actually is.

If you haven't signed up for an account yet, let's quickly do that. Go to http://www.locomotivehosting.com; fill in your first name, email address, and password; and click the *Get Started* button. That's it! You now have an account.

If you just signed up, you'll be redirected to your Workspace automatically. If you signed up before, you can get to your workspace using the *Sign In* link on LocomotiveHosting homepage or by going directly to https://www.locomotivehosting.com/admin/sites.

Your workspace has two tabs: Workspace and Billing. If you click the *Manage my sites* button under the *Workspace* tab, you'll see a *+ new site* button on the right.

Let's click that to add a new site. Just fill in the site's name and the primary language. You can also add custom domains where you would like to be able to access the site. All of this can be changed later, so if things basically look good, click *Create*.

You'll then be redirected to a page with details on setting up Wagon, but we've already done that so we can skip this. Also, if you added a custom domain to your site, before those start working you will need to update the DNS records for that domain so it points to your LocomotiveHosting subdomain. In the meantime we can access these sites using their LocomotiveHosting subdomain.

If you click the *Manage my sites* button again we'll be taken back to a list of our sites, which should now include the one we created. Click on the site's name to go to its back-office. You can also access a site's back-office directly by going to *https://your-subdomain.locomotivehosting.com/admin* or, if you have a custom domain, *http://www.example.com/admin*.

The back-office is a graphical interface for managing our site's content and settings.

Under the *Contents* tab you can add, edit, and remove page templates as well as content types and content entries.

Under the *Settings* tab you can modify your site's language, time zones, and SEO settings. You can also manage asset files, translations, and user accounts here. I'll let you poke around the back-office more on your own later.

Let's see what our site looks like by going directly to it's LocomotiveHosting subdomain, i.e. http://your-subdomain.locomotivehosting.com. Since we haven't added anything, it will just be a blank page that says "Content of the home page." Not to exciting, but hey, it's working!

You've now setup a site with LocomotiveHosting and all you had to do was click a few buttons. In the final crash course chapter, I'll walk you through how to deploy the Acme Corportation site we made using Wagon to LocomotiveHosting.

8. Deploying your Site

Yesterday, we covered how to get setup a site with LocomotiveHosting and today we'll show you how to deploy the site we made with Wagon three days ago to our LocomotiveHosting site.

Back on your computer, find the *acme*\corp_ site and open the `config/deploy.yml` file. This YAML file specifies connection details for LocomtiveCMS Engines. To start you out, several dummy Engines are already entered: development, staging, and production.

We'll ignore development and staging for the moment. Under production, first fill in *host* with the your LocmotiveHosting subdmain, which probably looks something like this: `yellow-josh-22.locomotivehosting.com`. Next, we need to specify authentication details either in the form of an API key or the email address and password we use to login to the back-office.

You can find your API key in your site's back-office. To jog your memory, you can login to the site's back-office at *http://your-subdomain.locomotivehosting.com/admin*. Once logged in, go to the *Settings* tab, click the *My Account* button, and the API key will be under the *Credentials* section.

At this point, your *deploy.yml* might have something like this:

```
production:
  host: yellow-hermina-22.locomotivehosting.com
  api_key: d215b15d1a1280137bc394433d310df34b73f83c
```

Or something like this if you didn't use an API key:

```
production:
  host: yellow-hermina-22.locomotivehosting.com
  email: joe@example.com
  password: abc123
```

Ok, let's deploy to our server. Open the terminal, navigate to the project's root directory and run the `bundle exec wagon push production –data` command.

```
$ bundle exec wagon push production --data
* Pushing Site
  en
    updating Acme Corporation.....................................................[done]
* Pushing Snippets
  en
* Pushing ContentTypes
    updating Products.............................................................[done]
* Pushing ContentEntries
  en
    creating products / earthquake-pills..........................................[done]
    creating products / do-it-yourself-tornado-kit................................[done]
    creating products / iron-carrot...............................................[done]
    creating products / christmas-package-machine.................................[done]
* Pushing Pages
  en
    updating index................................................................[done]
    updating 404..................................................................[done]
    updating products.............................................................[done]
* Pushing ThemeAssets
    updating images/logo.jpg......................................................[done]
    updating stylesheets/styles.css...............................................[done]
```

For those wondering what that `--data` business was, here's the skinny: by default `wagon push` deploys everything except content entries and data for editable elements. This is usually a good idea because you don't want to override changes made to the content by site editors, however for this initial deployment, we want to deploy our `data/products.yml` data, so I added `--data`.

Now, if you login into your site's back-office, you'll notice you can now manage the pages and content entries we created in wagon. And if you take a look at the actual site, by going to *http://your-subdomain.locomotivehosting.com*, everything appears just as it did when we tested it with `bundle exec wagon serve` three days ago.

And we're done! You've deployed a LocomotiveCMS site. Of course, if this were a real site we wouldn't want to access it at the LocomotiveHosting subdomain; you would want to use your own domain.

To do that you'll need to upgrade your LocomotiveHosting account. Just go to your LocomotiveHosting workspace and click the *Publish now* for any of your sites and you'll be walked through selecting one of our affordable plans so that you can publish your site with your own domain.

Well, this is the end of the crash course. How does it feel to be a LocomotiveCMS expert? I hope you've enjoyed the ride and please feel free to contact us (contact@locomotivehosting.com) if you still have any questions!

What's next?

I hope this guide was a great read for you and you now feel ready to create your websites with LocomotiveCMS.

I suggest you now visit our Documentation website and head on over to our thorough and complete primer on developing with LocomotiveCMS: the Making a Blog tutorial.

The tutorial assumes you've setup an Engine somewhere, so you'll want to look at your options and get something setup to use when doing the tutorial, even if it's just a local development engine or a free LocomotiveHosting site.

Once you've mastered the basics, you can find step by step guides for specific development tasks in the Guides section. You'll also want to make heavy use of the Reference section, which provides detailed information about Template features, API usage, and Configuration options.

For support resources and more from the worldwide LocomotiveCMS community, check out the Community section. When you're stuck, the Google Forum is the best place to get support for your questions. You can also help out by filing bug reports or making pull requests to our GitHub account.

LocomotiveCMS is totally free for you to use but it costs us a lot of time, sweat and money to develop. By subscribing to LocomotiveHosting, you show us your support and we reinvest all the money into new features that will make your life as a developer easier.

So please sign up to LocomotiveHosting, we'd really appreciate it.

Thanks for reading this guide and enjoy developing with LocomotiveCMS!

Didier and the LocomotiveCMS team.