



LEARN ENOUGH

CUSTOM DOMAINS

TO BE DANGEROUS

DEVELOPER KNOWLEDGE

TUTORIAL BY

LEE DONAHOE AND MICHAEL HARTL

Learn Enough Custom Domains to Be Dangerous

Lee Donahoe and Michael Hartl

Contents

1 A name of our own	1
1.1 Introduction	2
1.2 DNS overview	4
1.2.1 Visiting a site via DNS lookup	10
1.3 Custom domain registration	18
1.3.1 What to register?	20
1.3.2 You've got a domain, now what?	28
2 DNS management	31
2.1 Cloudflare setup	31
2.1.1 Cloudflare features	32
2.1.2 Cloudflare signup	37
2.1.3 Connecting registrar nameservers	37
2.2 Custom domains at GitHub Pages	50
2.2.1 Configuring Cloudflare for GitHub Pages	50
2.2.2 Configuring GitHub Pages	59
2.3 Custom domains at Heroku	66
2.4 Canonical URLs	75
2.4.1 Cloudflare page rules	76
2.4.2 HTTPS everywhere	83
2.5 Profit!!	89
3 Custom email and analytics	91
3.1 Custom email	91

3.1.1	Introduction to email	91
3.1.2	Google mail	95
3.1.3	G Suite signup	98
3.1.4	Verify domain and configure MX records	107
3.2	Site Analytics	120
3.2.1	Add Snippet	124
3.3	Conclusion	129

About the authors

Learn Enough cofounder Lee Donahoe is an entrepreneur, designer, and front-end developer. In addition to doing the design for [Learn Enough](#), [Softcover](#), and the [Ruby on Rails Tutorial](#), he is also a cofounder and front-end developer for [Coveralls](#), a leading test coverage analysis service, and is tech cofounder of [Buck Mason](#), a men's clothing company once featured on ABC's [Shark Tank](#). Lee is a graduate of [USC](#), where he majored in Economics and studied Interactive Multimedia & Technologies.

[Michael Hartl](#) is the creator of the [Ruby on Rails Tutorial](#), one of the leading introductions to web development, and is cofounder and principal author at [Learn Enough](#). Previously, he was a physics instructor at the [California Institute of Technology](#) (Caltech), where he received a [Lifetime Achievement Award for Excellence in Teaching](#). He is a graduate of [Harvard College](#), has a Ph.D. in Physics from [Caltech](#), and is an alumnus of the Y Combinator entrepreneur program.

Chapter 1

A name of our own

Welcome to *Learn Enough Custom Domains to Be Dangerous!*

So, you want to make a custom website? Or perhaps you already have one? This tutorial is designed for anyone who'd rather have their website live at [example.com](#) than at [example.someoneelsesdomain.com](#)—in other words, at a domain you control and that no one can ever take away. As a bonus: your email address can be yourname@example.com instead of yourname1995@-someoneelsesdomain.com.

If you're not sure how to build a real website, we've got you covered there, too: an especially good time to read this tutorial is after you've put together and deployed your first live site on the Internet while following the [Learn Enough tutorial series](#)—especially the Learn Enough tutorials on [CSS & Layout](#), [JavaScript](#), and [Ruby](#), as well as the [Ruby on Rails Tutorial](#).

Finally, this tutorial is designed as a general reference for custom domains, useful even for experienced web professionals. Configuring a website to use a custom domain is generally quite complicated—and every time you need to buy and configure a new domain you'll find yourself following these same steps... which ends up happening for basically every project. This is why we decided it would probably be a good idea to put all that information into a living single document (which we'll update and expand over time), instead of having the content spread out in small sections over a number of tutorials.

1.1 Introduction

At first glance, this tutorial might look fairly long—the print-equivalent of over 100 pages—but most of the length is due to including a lot of screenshots. We've also included supplementary material on things like the Domain Name System ([Section 1.2](#)) because we thought it was worth explaining some of the underlying structure that makes these technologies work. (If you're doing all this a second time, feel free to skip such sections.) Once you know how this process works, you can do all the steps one after another in quick succession, which should let you go quickly from having a test site on [GitHub](#) with a generic email account to a site that is on its own domain with a professional-looking email address. Getting everything done really only takes an afternoon.

The main focus in this tutorial is configuring a custom domain for a statically built site hosted on [GitHub Pages](#), as covered in the Learn Enough tutorials on [Git](#), [HTML](#), [CSS & Layout](#), and [JavaScript](#). There is also a bonus section ([Section 2.3](#)) covering custom domains for dynamic web applications, such as those created in [Learn Enough Ruby to Be Dangerous](#) and the [Ruby on Rails Tutorial](#). The process is similar for both. Additionally, this tutorial shows you how to set up a couple of useful Google services: Gmail, for custom email addresses ([Section 3.1](#)), and Google Analytics, so that you can collect information about who is on your site ([Section 3.2](#)).

Because of how quickly things change on the Web, it is entirely possible that some of the information in this guide might at times be out of date due to services changing interfaces or features ([Figure 1.1](#)).¹ We at Learn Enough are going to try to make sure that this tutorial is always kept up-to-date, but please understand that there is always a degree of variability in the interface for all these services—even if in the end they still mostly work the same. That means you should feel free to contact Learn Enough if you notice that some part of the tutorial is out of date (support@learnenough.com).

Even with the steps detailed here, in all likelihood you'll still have to apply some of your [technical sophistication](#) in order to get everything to work. That's just how it goes on the World Wide Web!

¹Image retrieved from <https://flic.kr/p/2a8hC> on 2019-11-20. Copyright © 2005 by jimthompson and used under the terms of the [Creative Commons Attribution 2.0 Generic](#) license.



Figure 1.1: Sorry we broke your tutorial by changing our interface (not sorry)!

One last note: in the course of doing this tutorial, we are going to use a bunch of different services that you might have to pay for. We aren't making any money from endorsing any of these services; we just find them to be easy to use, and so we are suggesting that you use them, too (Figure 1.2).²

There are ways of doing much of what we discuss entirely for free, but it might come with the cost of dealing with the hassle of constant configuration and monitoring to make sure that everything is working as expected. When it comes to tech services, often it's better to pay someone to make your life a little easier.

1.2 DNS overview

This section gives an overview of how Internet domains work. If you're not interested in these details, or if you've seen them before, at this point you can skip right to custom domain registration in Section 1.3.

You may or may not know this, but when you open up your browser and type in the address of a site like **google.com**, you aren't *actually* entering a real address where the Google server can be found. Instead **google.com** can be thought of more as a screen name that is an easier way to get to Google's web services than if you had to remember the real machine address... which for us at the time of writing this, in the geographic location we're connecting from, is **172.217.4.174**.

You can find a site's IP address by using the **ping** command in your **command line** terminal, as shown in Listing 1.1 (which also has a little bit of the resulting output). Big sites like Google generally have many servers, so which one you connect to at any given time will vary based on the details of Google's system.

Listing 1.1: Pinging google.com from your command line reveals the server IP address.

²Image retrieved from <https://flic.kr/p/brd1K2> on 2019-11-20. Copyright © 2012 and used under the terms of the [Creative Commons Attribution 2.0 Generic license](#).



Figure 1.2: Totally not getting paid to say that.

```
$ ping google.com  
PING google.com (172.217.4.174): 56 data bytes  
64 bytes from 172.217.4.174: icmp_seq=0 ttl=57 time=305.001 ms  
. . .
```

The number in Listing 1.1 is called an [Internet Protocol \(IP\) address](#), and it is made up of four 8-bit numbers (also called *octets*) separated by periods ([Box 1.1](#)). These IP addresses are what define the location of machines on the network and allow for data to be transferred via the underlying system, which is known as [TCP/IP](#) (Transmission Control Protocol/Internet Protocol). TCP/IP is the suite of protocols that governs the flow of information around the network of networks that we call the Internet. The association of the domain name with an IP address is known as an *A record* (where “A” stands for “address”).

Box 1.1. Bits and bytes

A *bit*, or *binary digit*, is the basic unit of information in a computer. It represents one of two values (thus [binary](#)), typically written as either 0 or 1. A set of 8 bits is called a *byte*.

Because bits can take either of two values, the number 2 is especially important in computing, as are various powers of 2. For example, as noted in the text, IP numbers consist of four 8-bit numbers (octets) separated by periods. But what is an 8-bit number?

Well, each bit represents two possible values, so an 8-bit number represents $2^8 = 256$ possibilities. If we started counting at 1, the largest number allowable in an octet would be 256 itself, but because 0 is an allowable number the maximum is actually $2^8 - 1 = 255$.

Thanks to the explosion of Internet-connected devices like phones, computers, cars, and even refrigerators, the world has actually run out of IP addresses. The system described above using four 8 bit numbers is called [IPv4](#), and to add

more addresses we're in the process to a new system called **IPv6** (Box 1.2).³ For now, both systems are still being used during the transition.

Box 1.2. IPv4 to IPv6

Originally, IP addresses were divided up into blocks that were then assigned to corporations, universities, countries, etc. On top of that, some addresses were reserved for specific uses, making them unavailable to be assigned to machines; for instance, `127.0.0.1` is reserved as the internal *home address* for any computer. (Thus the computer joke “There’s no place like `127.0.0.1`”, a reference to the famous quote from *The Wizard of Oz*.)



³Image retrieved from <https://flic.kr/p/2aGJ7q5> on 2019-11-20. Copyright © 2012 and used under the terms of the [Creative Commons Attribution 2.0 Generic license](#).

Even though mathematically there are $256^4 = 4,294,967,296$ (or just over four billion) addresses in the IPv4 system, a lot weren't able to be used, or they are parts of already-assigned blocks and so can't be used even if there is no machine at the address now. For a long time, we were able to get away with not having enough addresses by having entire networks with their own sets of internal IP addresses represented on the Internet by a single IP address, but that was just a strategy to buy time.

We've gradually been making the switch to a new addressing system, IPv6, that uses an address that can consist of 8 groups of 4 [hexadecimal digits](#). The format of the new IP addresses looks like this:

0000:0000:0000:0000:0000:0000:0000:0000

So Google's IPv6 address is much more complicated-looking:

2a00:1450:4007:0813:0000:0000:0000:200e

There are rules for shortening some of the sections if they start with a zero, or shortening the address if it contains multiple consecutive sections that are all zeros, but we aren't going to get into that here. We just wanted you to understand that there was an old system, and that now there is a new system we'll all eventually be moving to.

IP addresses are needed because all information that is sent or received on the Internet is chopped up into little pieces called *packets*, and each packet has an origination address and a destination address. So when you request a web page, the actual content is chopped up by the server into a bunch of chunks, flung out onto the Internet, and each chunk has the address for your computer written on it. Those chunks pass from server to server to get to your computer, and when they reach you, your browser reassembles everything into the web pages you are familiar with.

Imagine how much harder it would be to go to different sites on the Internet

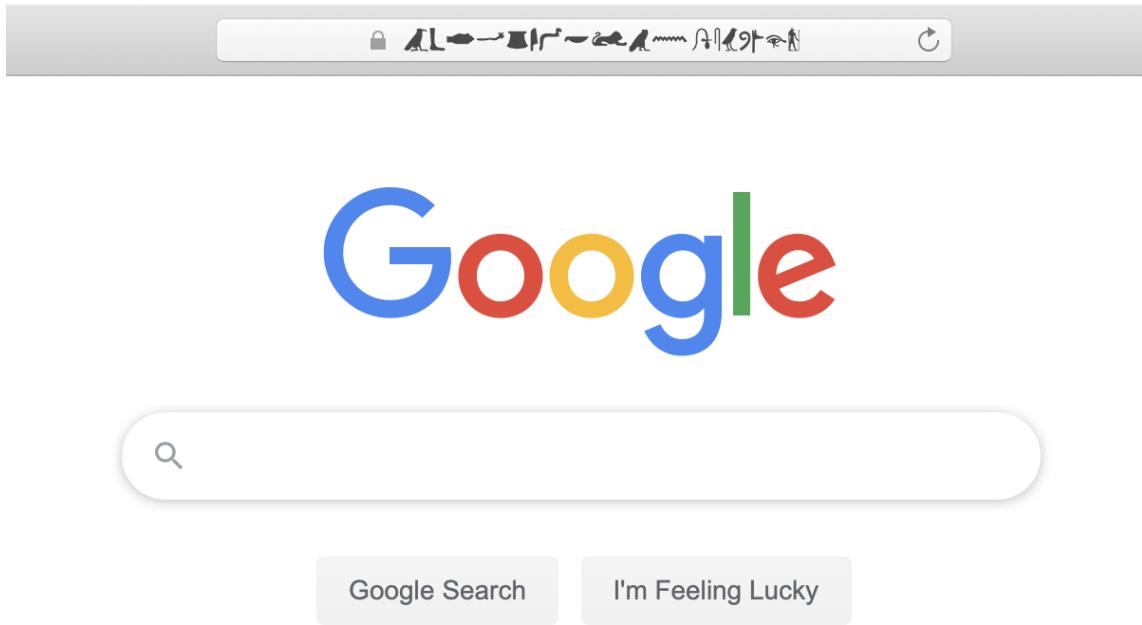


Figure 1.3: Domain names are a lot easier than this would be!

if you had to remember something like the string of numbers in an IP address every time you wanted to go to Google’s site—or any other site (or even something much harder, like [Egyptian hieroglyphs](#) (Figure 1.3)). The people who created the modern Internet knew that navigating around the network using strings of numbers was not going to be a great way to encourage the growth of the network, and so they came up with methods to make things easier.

One of the first methods, and one that actually still exists even on modern computers, was to create a file locally called the *HOSTS* file, which would have the raw IP address and then next to it a more human-readable name. Every user would keep their own little directory of IP addresses and assign an easier-to-remember name for the resource. [Listing 1.2](#) shows what a modern *HOSTS* file looks like with an entry for Google.

Listing 1.2: Adding a host entry for Google.

/etc/hosts

```

## 
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
## 
127.0.0.1 localhost
255.255.255.255 broadcasthost
::1           localhost

216.58.217.206 www.google.com

```

Having to keep a HOSTS file up to date with a bunch of different sites was kind of a pain, though. So, sometime in the ’80s, all the different groups and individuals started coming up with a new global system, called the Domain Name System (DNS), that would work like a phone book for the Internet, and would allow for users to navigate around the network using human-understandable words and phrases which would get **automagically** converted to machine addresses behind the scenes—no more editing local files to save the IP addresses of sites that you want to visit.

1.2.1 Visiting a site via DNS lookup

To get an idea of how DNS works, let’s take a look at what happens when we enter a web address—often called a *URL* (for “Uniform Resource Locator”)—into a browser.

First of all, you’ve seen addresses like **<https://www.google.com>** a million times, but why do we use addresses with all those different parts (Figure 1.4) instead of just entering **google** into the browser? What do the different parts mean?

- *protocol*: The **https://** part of the address tells the browser what sort of connection it should try making to a server. In the case of **http(s)**, we are telling the browser that it is going to make a hypertext transfer protocol request; the optional but common addition of the **s** at the end means that we are requiring the connection to the server to be *secure* and encrypted.

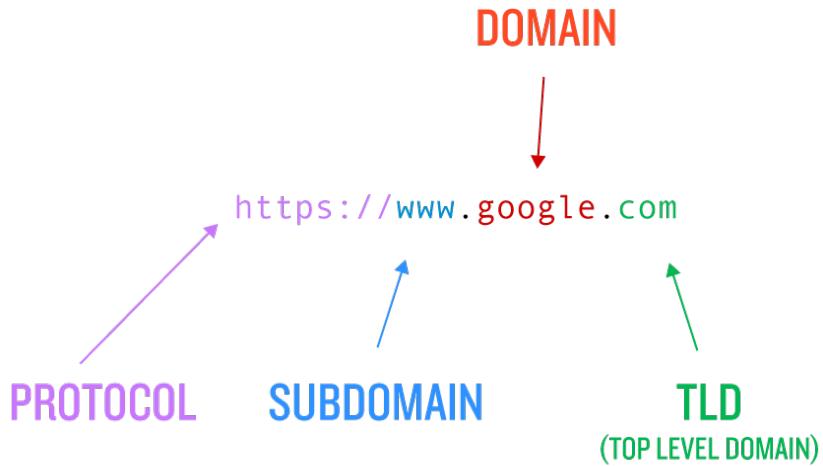


Figure 1.4: The anatomy of a URL.

HTTP requests are the kind of request that ask a server to send back a web page. There are a number of other protocols, like **ftp** (file transfer protocol) for sending and receiving large files and **smtp** (simple mail transfer protocol) for sending email.

- *top-level domain (TLD)*: The top-level domains are the highest level of the domain name system, and you can think of them as completely separate address books that keep a registry of unique names that people have paid for. Each TLD is given the authority to allow people to register domain names for only that TLD—so if you own **example.com** that doesn't mean that you also automatically own **example.org**.

If you want your site available at both addresses, the names would have to be registered with both TLD organizations. There are generic TLDs, like **.com** and **.org**, and then there are TLDs that have been created for individual countries, like **.us** for the United States, **.uk** for the United Kingdom, etc.

Additionally, the body that governs the Internet naming system has al-

lowed organizations to pay to create their own new TLDs, and so there are wild new options like `.ninja` and `.limo`. Ultimately, you can think of TLDs as being like area codes of phone numbers (only at the end instead of at the beginning).

- *domain name*: This is the name of the site or resource that has been registered with a TLD. Domain names have to be unique within a TLD in the same way that there can't be multiple identical phone numbers within a single area code.
- *subdomain*: The subdomain is an extra domain that is fully controlled by the owner of a domain, so there is no need to register or pay for new subdomains. This means that web traffic can go to `www.google.com`, email traffic can go to `mail.google.com`, and customer support traffic can go to `support.google.com`. Each one of those subdomains can be configured to show users a completely different site.

The `www` subdomain—for “World Wide Web”—was created to tell servers that the browser requesting information from that subdomain was looking to receive web pages that conform to the Web standard—in other words, HyperText Markup Language, or HTML, the language of the World Wide Web (see *Learn Enough HTML to Be Dangerous*). While it *is* possible to configure a site to serve web content to a browser request without a subdomain, like `https://google.com`—which is called a “naked” or “root” domain—there are *complicated reasons* why you probably don’t want to run your site on a root domain, and indeed most big sites permanently *redirect* the root domain to the corresponding address using `www`. Examples of such redirected URLs include `google.com`, `amazon.com`, `apple.com`, `microsoft.com`, and `facebook.com`. (We’ll learn how to make such redirects ourselves in [Section 2.4.1.](#))

After you finish typing the address and hit enter, your computer first checks to see if you’ve visited that site before (or if it’s in the HOSTS file). If so, your computer already has the IP address and can thus connect immediately ([Figure 1.5](#)).

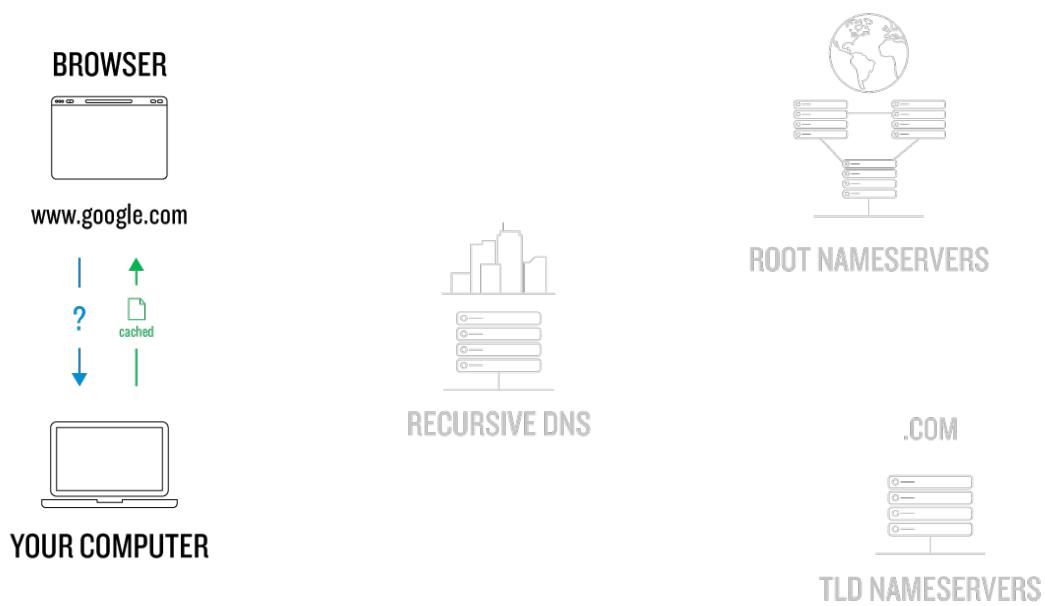


Figure 1.5: Connecting with a locally stored IP address.

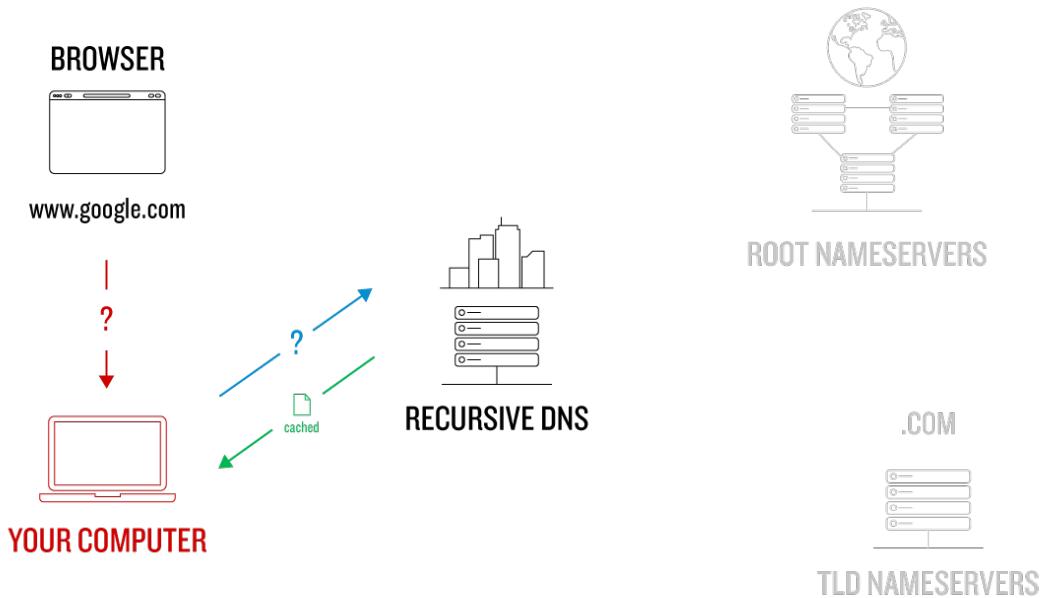


Figure 1.6: Obtaining an address from the nearest recursive DNS.

If the address isn't stored locally, the browser next reaches out to a server (run either by your Internet service provider or by a nearby publicly accessible server) called a *recursive DNS server*. It asks that server, "Do you know the machine address that corresponds to www.google.com?" These recursive servers have their own **cache** of site names and addresses, so most of the time your request is quickly found (Figure 1.6). As a side note, you can configure your computer or router to use specific recursive DNS servers (if you are concerned about security), and Google actually runs their own public servers that are a good option (Box 1.3).

Box 1.3. Using the Google DNS servers

You might not realize it, but every time that you connect to Wi-Fi or use the Internet, by default your device is given the address for a recursive DNS server. This address might be set on a modem that is provided by your Internet service

provider (ISP), it could be from the settings for a Wi-Fi router in a café that set their own custom DNS address, or it could be the DNS that your wireless phone provider gives you. While this system often works well in practice, one potential problem is that you have to trust that the server you are connecting to is going to give you the right addresses.

What do we mean by that?

Well, if you are connecting to sites through an unscrupulous service, like let's say your Internet service provider (because some actually got caught doing this), they could have their DNS server point you to their own web servers instead of to the site you wanted to connect to. Then they request the page you were looking to load, add their own content (like ads) onto the page (or otherwise mess with the content), and then pass everything back to you without your realizing that something bad happened. So you think you are getting Google search results, and you do, but your ISP also injected a bunch of their own ads alongside.

This is called DNS hijacking and it is clearly not good. One way to avoid this is to go into the settings for your router, your computer, and your phone, and set a custom trusted DNS server as the default.

Google runs their own DNS servers that you are free to use if you trust them (and we generally do), but that's up to you. The address for the servers are **8.8.8.8**, and **8.8.4.4**. You can always search on Google to find the addresses if you forget.

The method to add these servers as your default DNS differs by device and operating system, but if you look online you'll be able to find guides on how to do it. Once you set a preferred DNS server on your device, that setting will override any potentially malicious DNS server address set by a router, modem, or ISP.

If a recursive DNS server doesn't have a machine address listing that corresponds to the name you entered in your browser, then your request will be kicked up to the global domain name system. When that happens, the request is first sent to the global *root nameservers*. This is a network of thirteen servers that act kind of like a **switchboard**, and all they care about is directing your request to the correct top-level domain network. They don't know the answer to

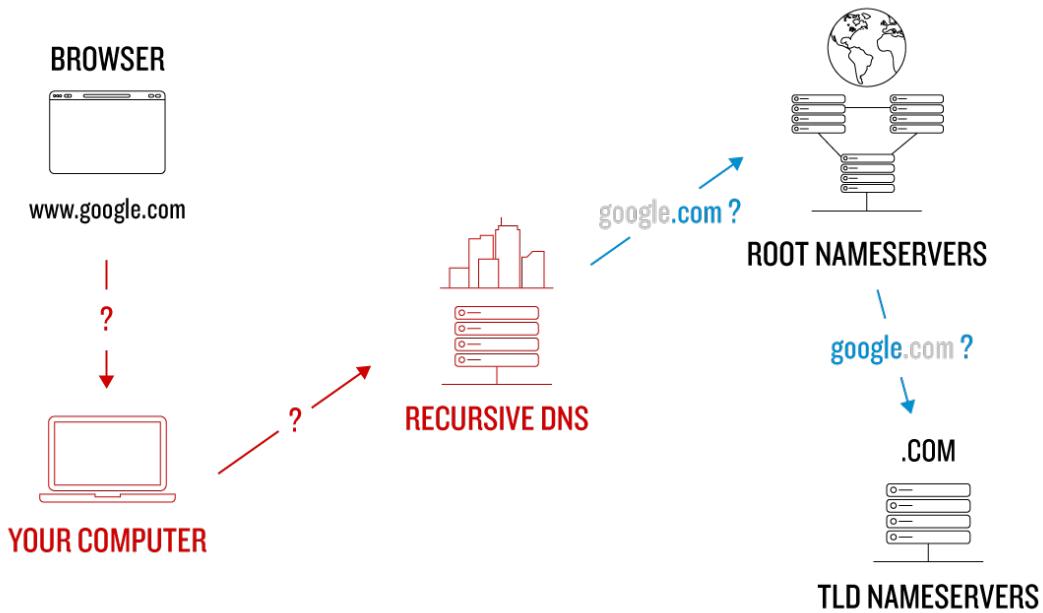


Figure 1.7: Using the root nameservers find the TLD’s nameservers.

“What is the IP address for `www.google.com`?”, but what they do know is the addresses for the network of servers that manage the `.com` TLD (Figure 1.7).

Your request then travels from the root nameservers down to the network of nameservers run by the top-level domain of the site that you are trying to reach. Your request for `www.google.com` is routed around inside the `.com` network until the address record for the site you are trying to reach is found. This will contain an entry that ultimately has the IP address for the server you are trying to reach.

A reply is sent back to your computer through the recursive DNS server that you originally contacted, so that the record can be cached on that server. As a result, the next time someone wants to go to the site you requested the whole lookup process won’t be necessary (Figure 1.8). At the end of the process, your computer receives the content of the DNS record, and your browser opens up a connection to the server.

All this happens in a fraction of a second, and the requests can bounce all

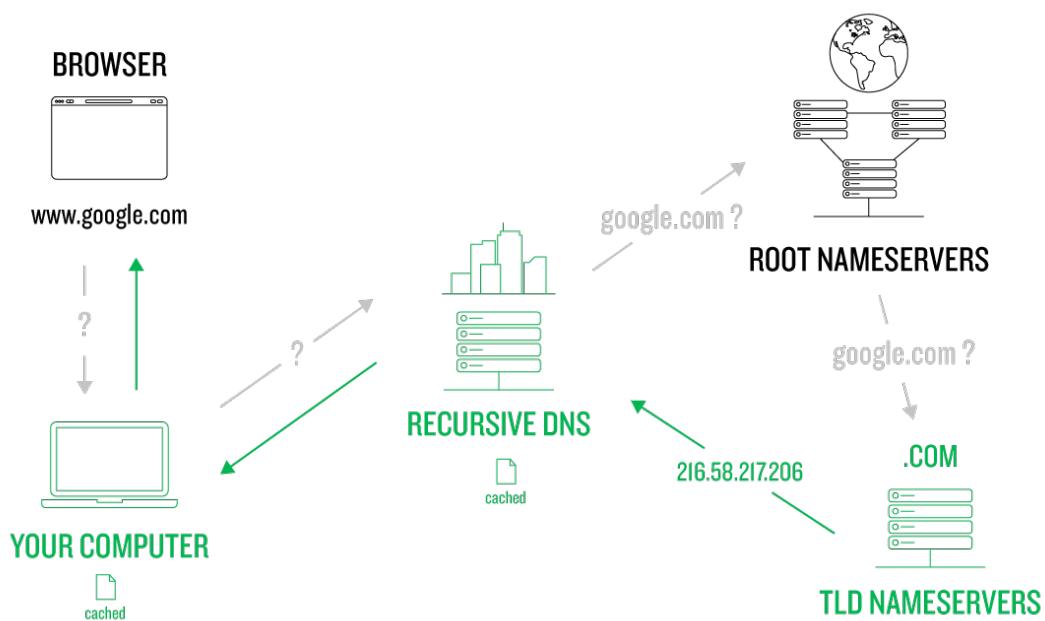


Figure 1.8: Caching the address for future use.

the way around the world without your noticing... Isn't technology amazing?

1.3 Custom domain registration

It's pretty hard to configure a custom domain unless you actually have one, so our first step is to *register* one.

Domain names have to be registered through an authorized registrar, and there are a bunch of them. You may even have seen some of their **bad commercials** on TV. Most of the big names—GoDaddy, Network Solutions, Hover, etc.—allow you to register domains with the majority of TLDs, including those TLDs managed by countries.

In general, domains aren't property that you own outright, like real estate. Instead, domains are more like an office you lease from a property owner, but with the right to renew the lease in perpetuity. The rights to the domain can be transferred and sold, but ultimately someone has to regularly pay the registrar for the right to the domain; otherwise, it will be released back to the pool of possible domains, where someone else can buy it.

The cost of the domain depends on the domain's TLD, and most registrars offer discounts on the yearly cost if you pay for a number of years at once. You'll also have to provide a bunch of required information when you register the domain, as the registrars are required by law to know who owns domains, and they have to make this public information. Most registrars offer a privacy service that will keep your personal information off of the Internet, though.

The domain registrar we recommend is [Hover.com](#), which has good customer service, reasonable prices, an intuitive interface, and a free domain privacy service (Figure 1.9).⁴ It's also nice that, unlike some [other registrars](#), Hover doesn't try to [nickel and dime](#) its customers with unnecessary services.

⁴Image retrieved from <https://flic.kr/p/89UQJ> on 2019-11-20. Copyright © 2006 and used under the terms of the [Creative Commons Attribution 2.0 Generic](#) license.



Figure 1.9: Totally not getting paid to say that (seriously).

1.3.1 What to register?

You've probably noticed that most global services and corporations use **.com**, which is generally considered to be the most desirable TLD. This TLD was created for general commercial activity by **ICANN** ("EYE-can"), the organization responsible for managing Internet names.⁵

So, which TLD should you pick? Well, a lot of that depends on what you want to do online, who your audience is, and the availability of the domain name that you want. If you are a big enough company or have enough name recognition in other media, you could just pick any TLD that has a domain name available that incorporates your established name. Unless you have a good reason to choose otherwise, a **.com** domain is probably the way to go, though there are a lot of different possibilities:

- There are *generic* TLDs, like **.com**, **.info**, **.net**, and **.org**, which can be used for general purposes.
- There are *generic restricted* TLDs, like **.biz**, **.name**, **.pro**, where there are rules on how domains registered to the TLD can be used.
- Every country has their own TLD (too many to list) like **.us** (USA), **.uk** (UK),⁶ **.it** (Italy), **.ly** (Libya), **.co** (Colombia), and **.io** (Indian Ocean territories). The **.io** domain has become popular in the tech industry, especially for a **quality domain name** whose **.com** equivalent is already in use or (especially) when it is being held hostage by a **domain squatter**.⁷
- There are the current *sponsored generic* TLDs, like **.aero**, **.asia**, **.cat** (not about cats but rather the **Catalan** linguistic and cultural community), **.coop**, **.edu**, **.gov**, **.in**, **.jobs**, **.mil**, **.mobi**, **.tel**, **.travel**, and **.xxx**, which can be used only for purposes within a particular industry.

⁵ICANN was originally under the control of the US Government (via the Department of Commerce), but since 2016 it has been free from US Government oversight.

⁶The United Kingdom actually has its own sub-TLD, so many UK sites end in something like **.co.uk**.

⁷Technically, **domain squatting** refers only to the bad-faith registration of domains, such as those corresponding to known trademarks. In practice, though, the supposedly benign practice of "**domaining**" (buying up domains for future resale but doing nothing useful with them in the interim) is so annoying that many people use the term "squatter" to refer to both cases.

- Finally, there is the “new generic TLD (gTLD) program”, which seeks to add an unlimited number of gTLDs (i.e., non-country TLDs). Any person or organization that wishes to have their own TLD can pay a hefty fee and apply to have their own TLD. Currently, 1409 new TLDs have been granted, such as `.xyz`, `.ninja`, `.limo`, as well as a large assortment of TLDs that use writing systems other than the Latin alphabet. Most of these new TLDs will be for entirely generic use, and some will be sponsored generic TLDs that can be used only for a specific purpose. There’s no telling at this point exactly what these purposes might be, so *caveat emptor*.

You should know, though, that many people consider these new domains, and any business that uses one, to be less reliable than the old ones... probably because they aren’t as familiar. Some of the country TLDs have become pretty well-accepted, such as `.io` mentioned above, though they often still cause confusion for less tech-savvy people. The problem is that if you tell a non-tech literate person to go to a site like `http://bit.ly`, you might find that they try to go to `http://bit.ly.com`. The take-away here is that you need to know your audience: you don’t want a domain that scares off potential users, you don’t want a TLD with lots of bad sites, and you don’t want a name that confuses people.

With those considerations in mind, let’s get back to registering a domain. The first step is to visit the [Hover.com](#) home page. At this point, you’ll have to choose a domain. For this tutorial, we’ll use `codedangerously.com`, but of course you’ll need your own (since we already registered that one!).

If you don’t already have an idea for the domain name, there are actually a lot free tools that are specifically designed to help you come up with one. They generally allow you to enter a bunch of different words or phrases to generate suggested domains that are available to be registered. A few popular sites that we’ve used are [instantdomainsearch.com](#), [namemesh.com](#), and [domainr.com](#). Or you can just use the domain registrar itself, which is what we’ll do here.

On Hover, finding out if the domain name is available is as easy as entering the name in the giant input field on the home page ([Figure 1.10](#)) and then clicking the green button with the magnifying glass (or pressing enter).

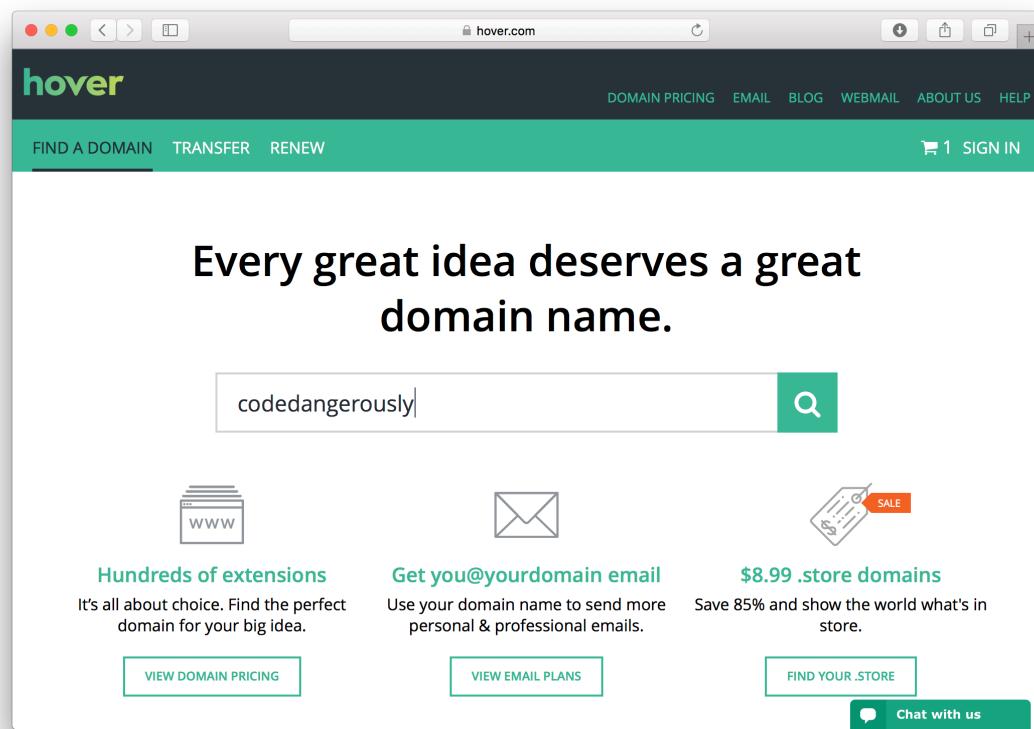


Figure 1.10: Starting a domain search is easier than finishing it.

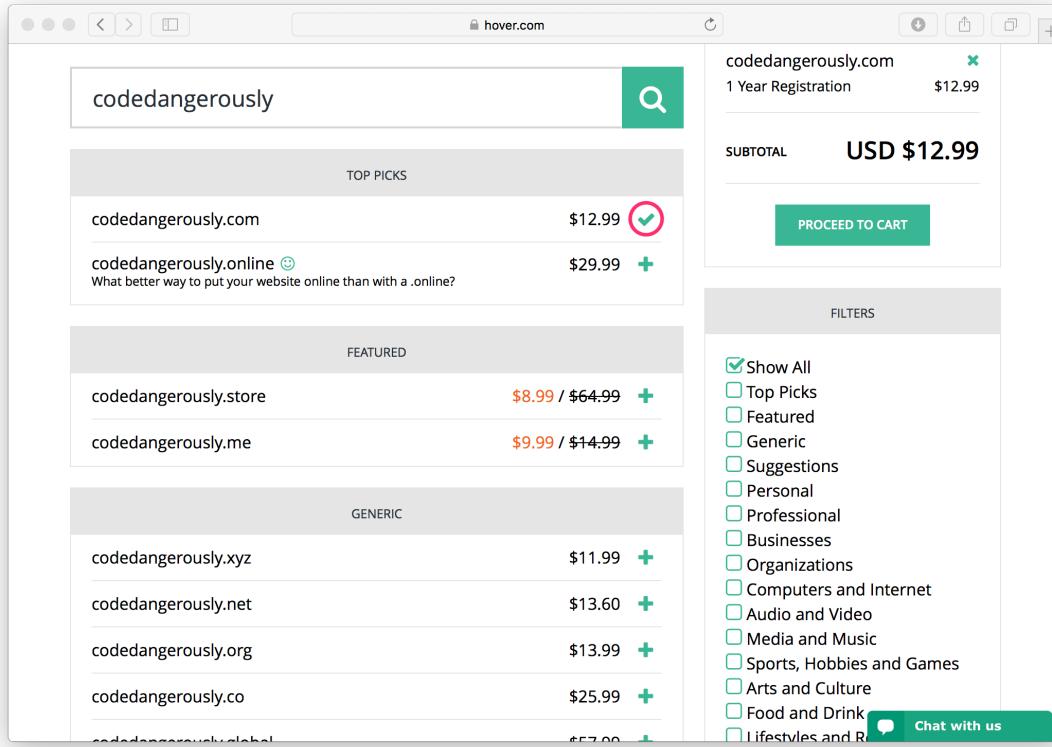


Figure 1.11: The different TLD options that are available to register.

At this point, Hover will give you a long list TLDs for which your name of choice is still available (Figure 1.11), as well as suggested domains containing word substitutions (farther down the page).

If you see that the domain name you want is available, and on the TLD you are looking for, you can click the green plus icon on the right-hand side of the list to add the domain to your cart. You'll know it worked if the plus sign turns into a check-mark (as indicated by the circle Figure 1.11).

When you've selected all the domains that you want to buy (just the **.com** is probably fine), click the “Proceed to Cart”, which will show you a summary of the domains you are about to buy. If everything looks correct, click the “Secure Checkout” button to move on to finalizing the purchase (Figure 1.12).

You might have noticed that even when you find a domain name that you

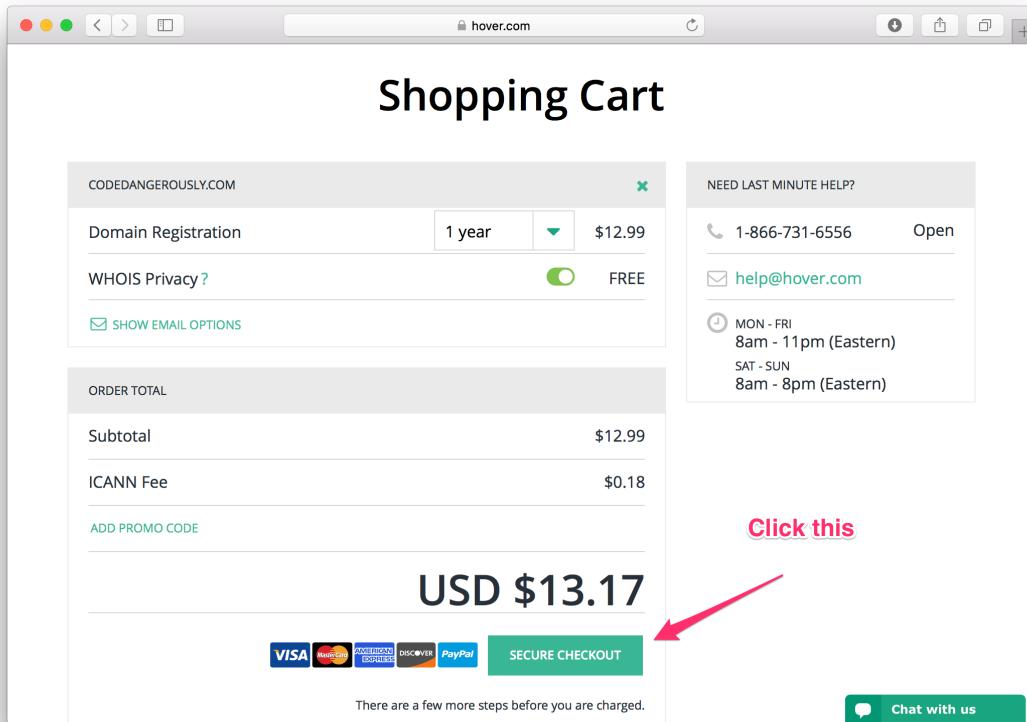


Figure 1.12: Reviewing the cart before paying.

like, there are often a lot of very similar variations available on different TLDs, as well as domains that are close in spelling to your domain. If you have the extra money to spend, you can buy as many misspelled domains as you want and then permanently redirect users to the appropriate domain. For example, we own the domain [railstutorial.org](#)—which contains a common misspelling of “tutorial”—and automatically redirect it to [railstutorial.org](#). (As noted in [Section 1.2](#), we’ll look at how to do domain forwarding ourselves in [Section 2.4.1](#).)

Assuming that you don’t already have an account with Hover, the first step of the checkout process will ask you to create an account. Fill out the contact information as in [Figure 1.13](#) (obviously with your own information), and then click the “Create Account” button.

After your account has been created, Hover will ask you to fill out the

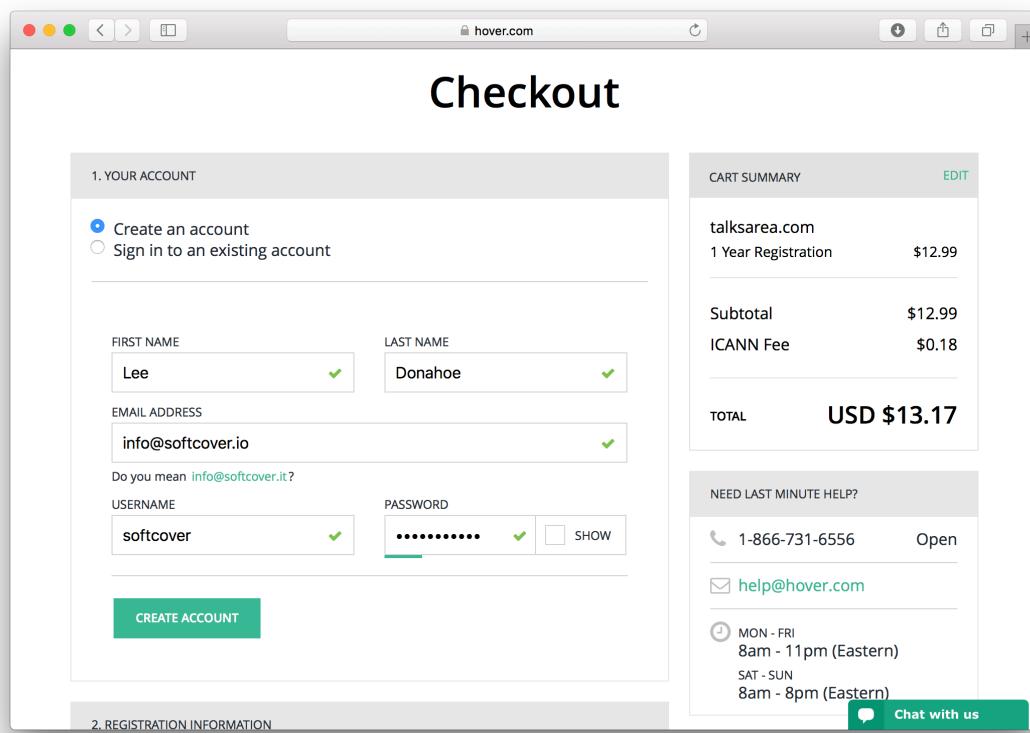


Figure 1.13: Creating your Hover account during the checkout process.

The screenshot shows the Hover.com checkout process. On the left, under '1. YOUR ACCOUNT', the user's details are listed: Full Name (Lee Donahoe), Email (info@softcover.io), and Username (softcover). There is also a link to 'USE A DIFFERENT ACCOUNT'. On the right, the 'CART SUMMARY' section shows a domain registration for 'codedangerously.com' with a 1-year registration cost of \$12.99. The subtotal is \$12.99 and the ICANN fee is \$0.18, totaling USD \$13.17. Below the cart summary, there is a 'NEED LAST MINUTE HELP?' section with contact information: a phone number (1-866-731-6556) and email (help@hover.com), along with operating hours (MON - FRI: 8am - 11pm (Eastern); SAT - SUN: 8am - 8pm (Eastern)). At the bottom right is a green 'Chat with us' button.

Figure 1.14: Adding the necessary domain registration information.

required registration information that they have to gather for all new domain names ([Figure 1.14](#)). Thanks to Hover’s free anonymization service, you don’t have to worry about this information being made public.

After you’ve entered your registration information and pressed the button at the end of the form, you’ll be taken to the payment page. We’re not going to screenshot this step since you are probably already familiar with how payment pages work. Enter your card information, and hit the button at the bottom. You’ll be asked to agree to Hover’s Terms of Service and confirm your order.

In addition to confirming your payment information, Hover requires you to verify the email address used to register the domain ([Figure 1.15](#)). Check your email for the verification email, then follow the link it contains to verify your email address.

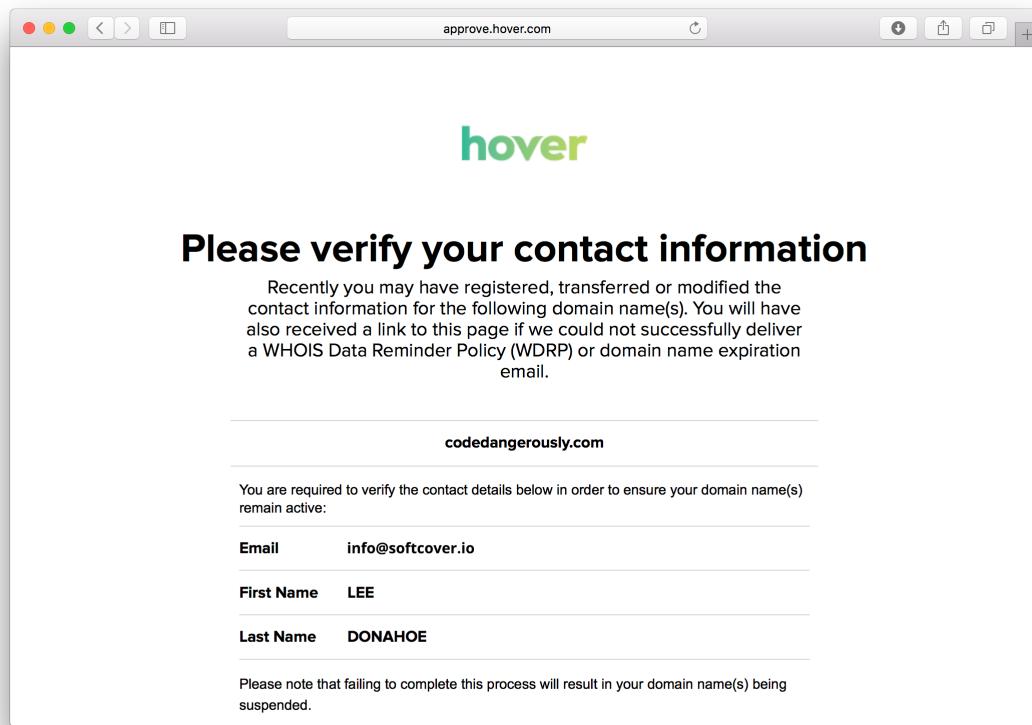


Figure 1.15: Hover's required email verification.



Figure 1.16: Your very own fantasy castle floating in the Internet sky!

If you've made it through all these steps, congratulations! You now own a domain name on the Internet. You are the proud owner (or at least long-term leaser) of virtual digital property (Figure 1.16).⁸

1.3.2 You've got a domain, now what?

Once you've registered a domain, the next step is to set up the *DNS records* for your domain and edit the so-called *nameservers*. The nameserver property for your domain tells the DNS system exactly which server should be queried to find the domain details, such as which subdomains exist, which URLs should

⁸Image retrieved from <https://flic.kr/p/AkF7w> on 2019-11-20. Copyright © 2007 by jimthompson and used under the terms of the [Creative Commons Attribution 2.0 Generic](#) license.

be forwarded, etc.

Right after you purchase a domain, the nameservers are usually set to the registrar's servers, and in the case of Hover that would be something like Listing 1.3.

Listing 1.3: Typical Hover.com nameservers

```
ns1.hover.com  
ns2.hover.com
```

Although you would be perfectly fine using the default DNS controls to complete this tutorial, we are going to set up a third-party service that provides a bunch of additional useful features. Setting up this service will involve changing the nameservers on the registrar to point to the new nameservers so that they can be the first point of contact for domain name in DNS lookups (Figure 1.17). Making these changes, and configuring the result, is the subject of Chapter 2.

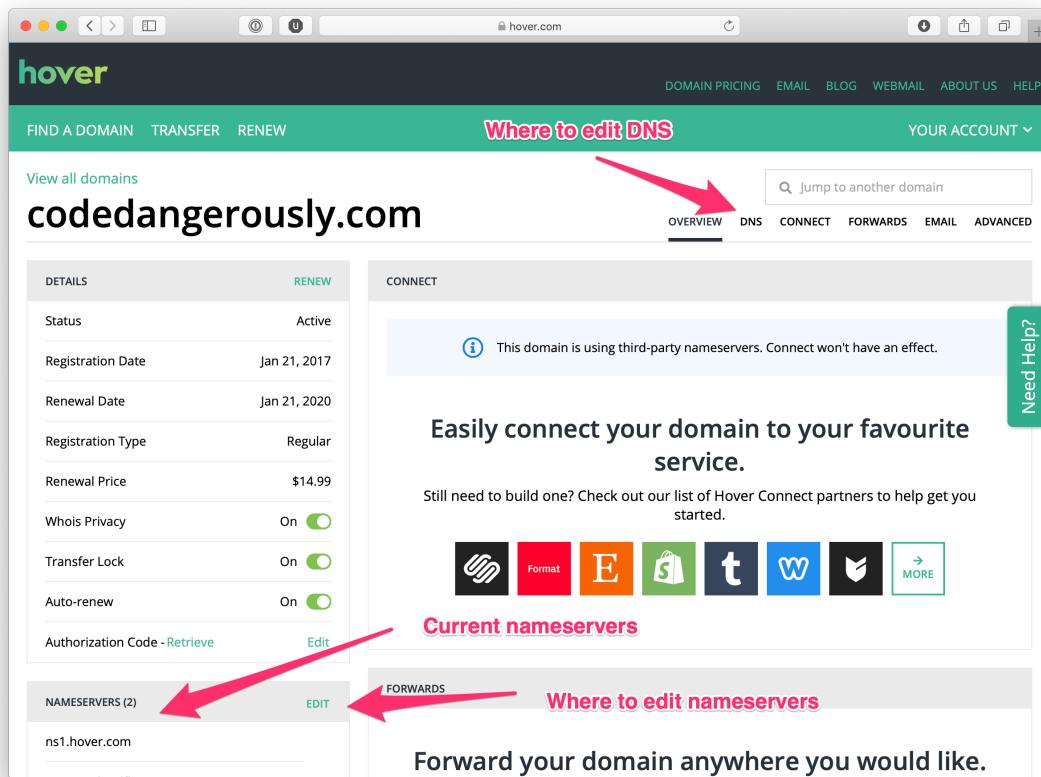


Figure 1.17: Where to find the two main features that you need to control your domain.

Chapter 2

DNS management

Chapter 1 ended by noting that custom domains come with default DNS name-servers (Section 1.3.2) but indicated that we’re going to use a third-party service instead. In this chapter, we’ll set up this service, called *Cloudflare* (Section 2.1), and then configure it to associate custom domains with static websites located at GitHub Pages (Section 2.2). As a bonus, we’ll show how to make the same association for dynamic web applications deployed at Heroku (Section 2.3).

While GitHub Pages and Heroku are only two of an enormous number of options for hosting websites, the principles we cover here are generally applicable to web deployment. After completing this chapter, you’ll have the knowledge needed to configure pretty much any other custom domain setup as well.

2.1 Cloudflare setup

So what is Cloudflare, and why are we going to be using it?

[Cloudflare](#) is a service that sits between the Internet and a website to provides the following benefits:

- Fast DNS management using a nice interface
- SSL support for setting up a secure connection to our site
- *Edge caching* to make the site’s content easy to access across the world

- Protecting the site from anyone who might try to take it offline using a *Distributed Denial of Service* (DDoS) attack
- ...and the basic service is **free to use** (with inexpensive pay plans as your website's needs grow)

It would probably be useful to quickly explain what these features are in case you aren't familiar with them.

2.1.1 Cloudflare features

First, most registrars allow you to do DNS management, but the interfaces can be less than optimal (to put it generously). Additionally, when you make DNS record changes on a registrar (like pointing to a new server IP address), the changes often aren't immediate. Rather, they have to propagate throughout the DNS system—a process which can take 24–48 hours to complete as your update bounces around the DNS network ([Figure 2.1](#)).

Because Cloudflare sits between your servers and the world while passing web requests through, any server changes stay internal to the Cloudflare service. To the outside world, all requests still go first to the Cloudflare IP address. This means that the propagation of DNS changes (like subdomains and redirects) is nearly instantaneous—a feature that can be critical when administering a working site.

The next feature on the list, SSL (or [Secure Sockets Layer](#)), is a method for encrypting web traffic from a user's computer to the server ([Figure 2.2](#)).

You've actually been using SSL for years, possibly without even knowing it: every time you go to a site with a little lock next to the site's domain name, you're using SSL ([Figure 2.3](#)).¹

SSL works by turning the packets of information sent between your browser and a server from plain old text into a jumble of letters and other characters. This makes it virtually impossible for someone to intercept the traffic and see what is being sent back and forth. In many cases, there is a lot of ceremony

¹The Internet is a dangerous place, and you can't ever be certain that you really are safe, but SSL increased the chances considerably.

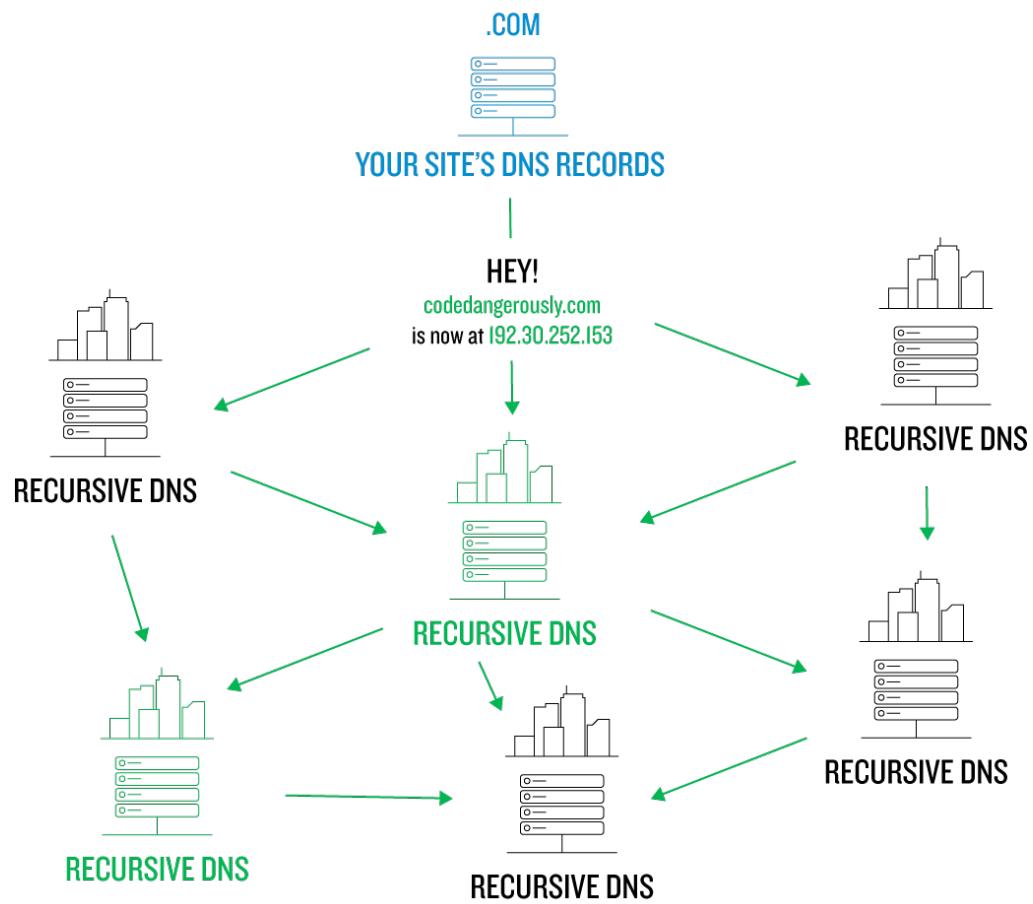


Figure 2.1: Potentially slow DNS propagation.

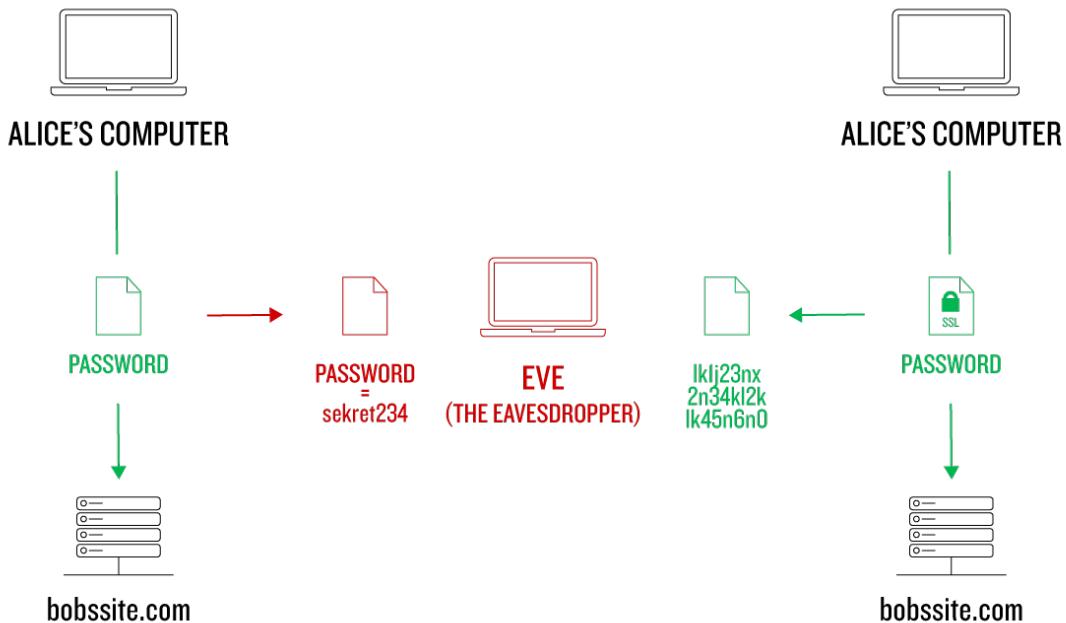


Figure 2.2: Encrypting data using SSL.

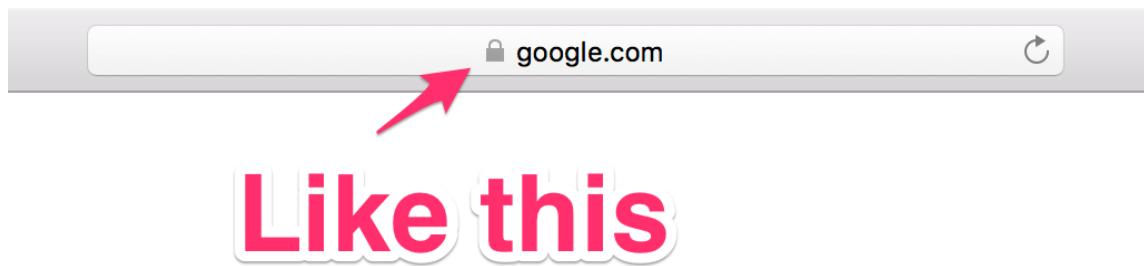


Figure 2.3: An SSL lock icon.

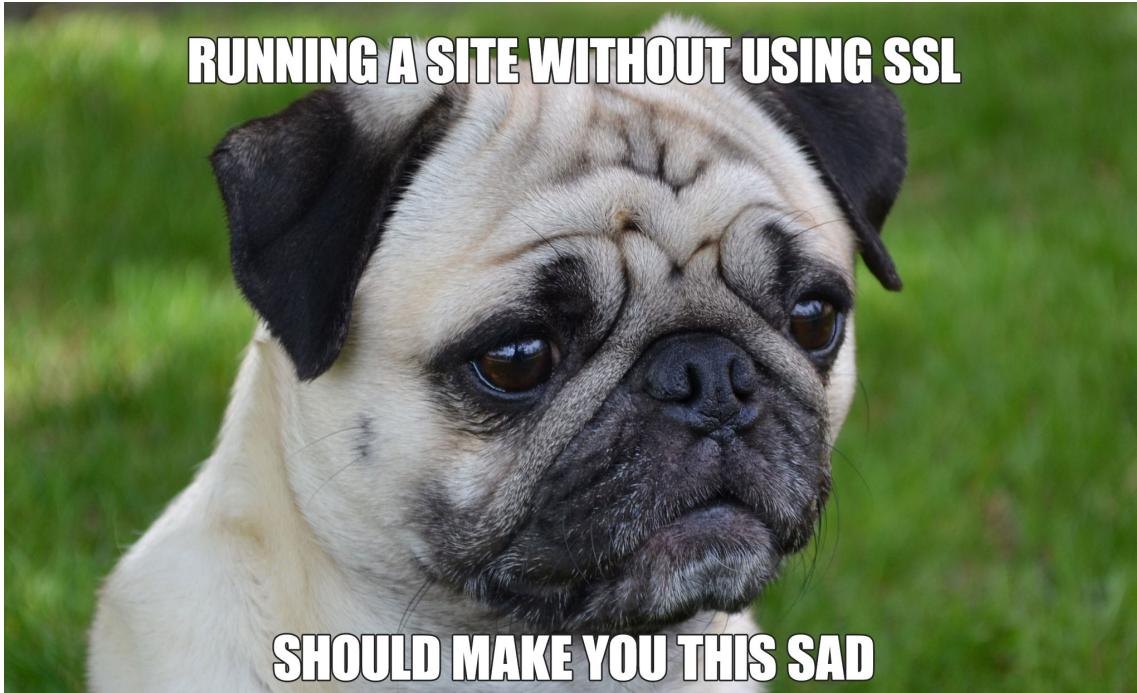


Figure 2.4: Don't do it! Bad dog!

and pain associated with setting up SSL for a site, but when you use Cloudflare everything is already set up for you for free. Because non-SSL sites are so insecure, you shouldn't run a site these days without SSL—not even a static site like a blog (Figure 2.4).²

Next up on the Cloudflare feature list is *edge caching* (Figure 2.5). Edge caching involves automatically saving your site's content on servers around the world, allowing users to access the content from the server nearest to them (which speeds load times).

Lastly, Cloudflare protects the site from [DDoS](#) attacks. You might have heard of these on the news in reference to some group being accused of “hacking” another organization. DDoSes are online attacks that try to make entire sites inaccessible by flooding their servers with fake requests—imagine mil-

²Image retrieved from <https://flic.kr/p/2g1Yi5g> on 2020-02-18. Copyright © 2019 by DaPuglet and used under the terms of the [Creative Commons Attribution 2.0 Generic](#) license.

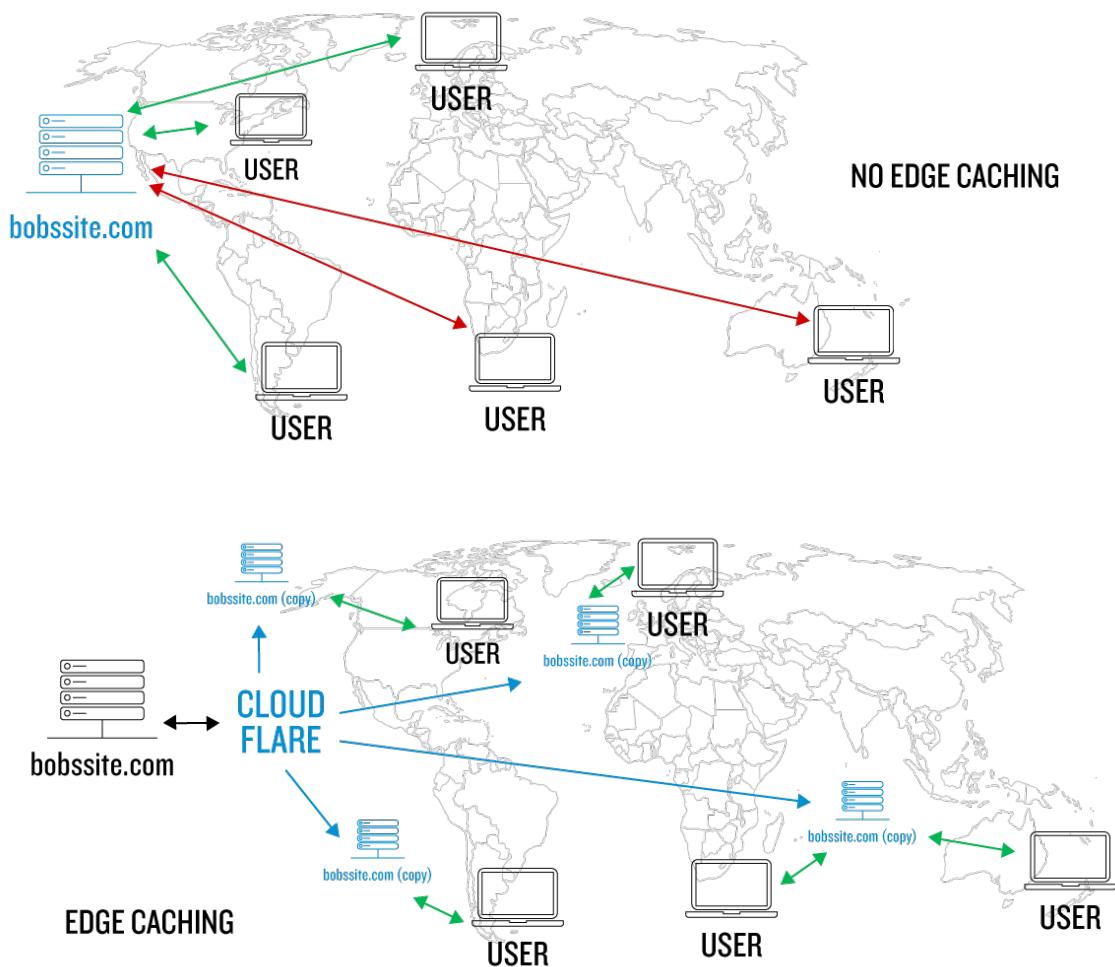


Figure 2.5: Optimizing load times with edge caching.

lions of fake users that all ask a server for a site at the same time.

There are a limited amount of requests that a single server can handle at once, so if there are enough fake users making requests, real users can't get to the site. Cloudflare has the ability to notice when this type of attack is happening and can filter out the fake users to keep your site online and available. Again, you don't have to configure anything to get this benefit.

All this sounds pretty good, right? No one wants to have their site overrun by computer zombies ([Figure 2.7](#)).³

These days, our first step after registering a domain is immediately setting up Cloudflare. The benefits are great, and unless you have a fairly complicated application the service is free.⁴ What's not to love? Let's get started.

2.1.2 Cloudflare signup

The first step in using Cloudflare (just like pretty much any service on the Internet) is to create an account. One added benefit, though, is that the signup process also walks you through adding a domain to Cloudflare. So, head on over to cloudflare.com and click the “Sign Up” button at the top right of the page in the site header to get to the registration page ([Figure 2.8](#)).

2.1.3 Connecting registrar nameservers

When you hit the “Create Account” button, Cloudflare automatically kicks you into the first step to add a domain to their service. As seen in [Figure 2.9](#), you just add in your newly purchased domain in the box and press the “Add Site” button.

After hitting the “Add Site” button, you'll be taken to the plan selection page. Since we are just going to use the free plan in this tutorial, select the “Free” option as in [Figure 2.10](#), and then hit “Confirm Plan”.

³Image retrieved from <https://flic.kr/p/9QeRNF> on 2020-02-18. Copyright © 2011 by Marilyn Sherman and used under the terms of the [Creative Commons Attribution 2.0 Generic](#) license.

⁴We use the professional, paid tier for more complicated sites like learnenough.com and railstutorial.org, which is but one of many indications that Cloudflare's [freemium](#) business model is working well.

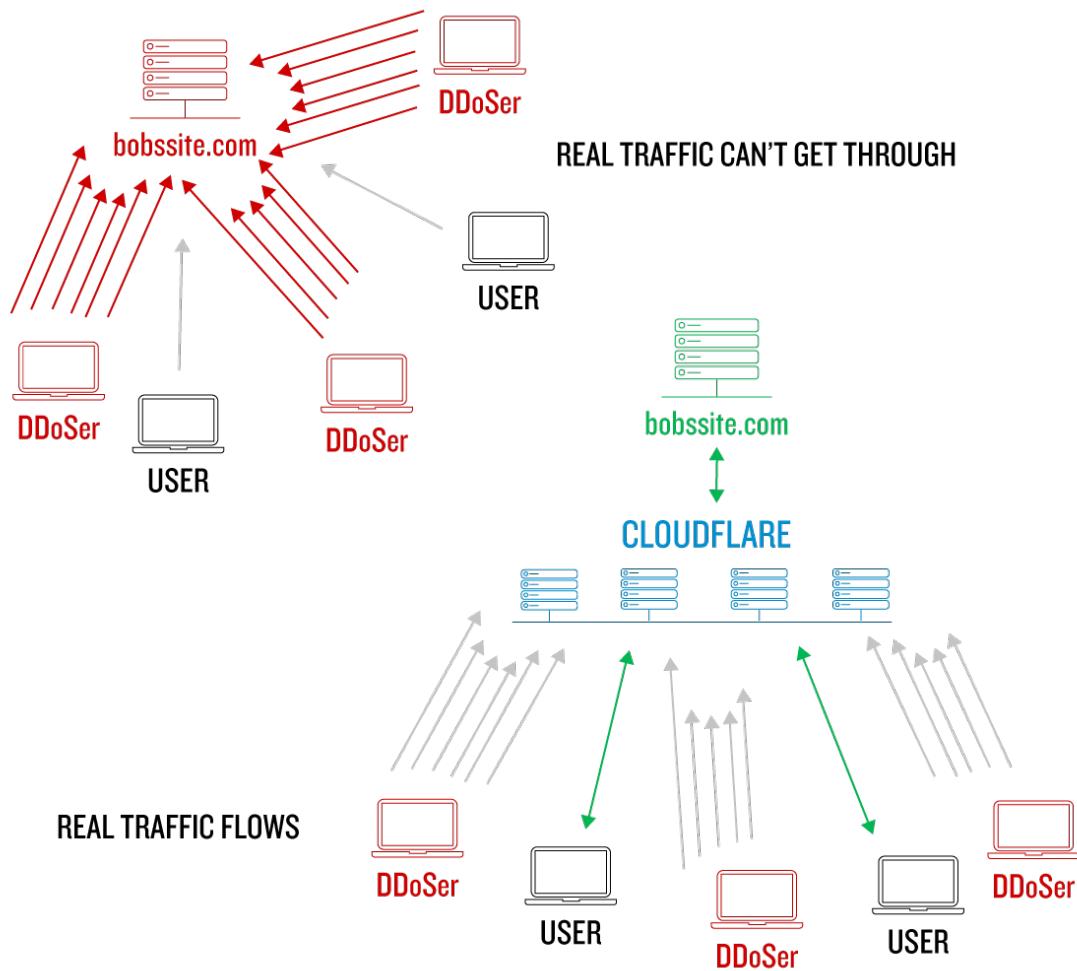


Figure 2.6: Protecting against DDoS attacks.



Figure 2.7: Or real zombies using computers – really any zombies are bad.

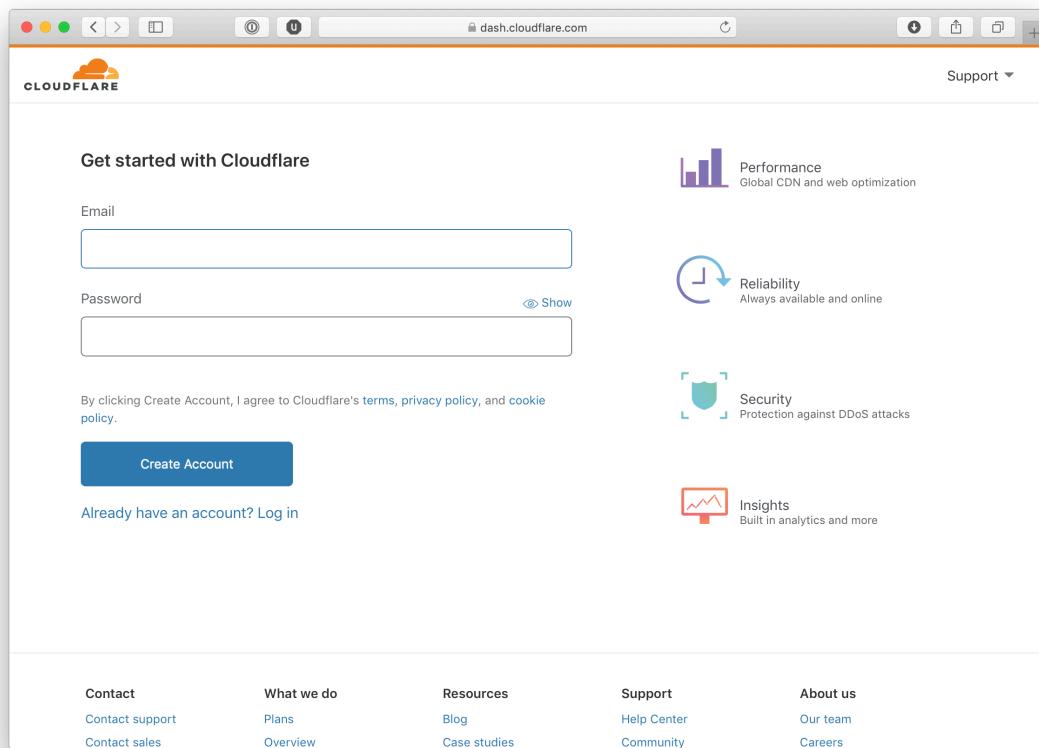


Figure 2.8: First step of the Cloudflare registration process.

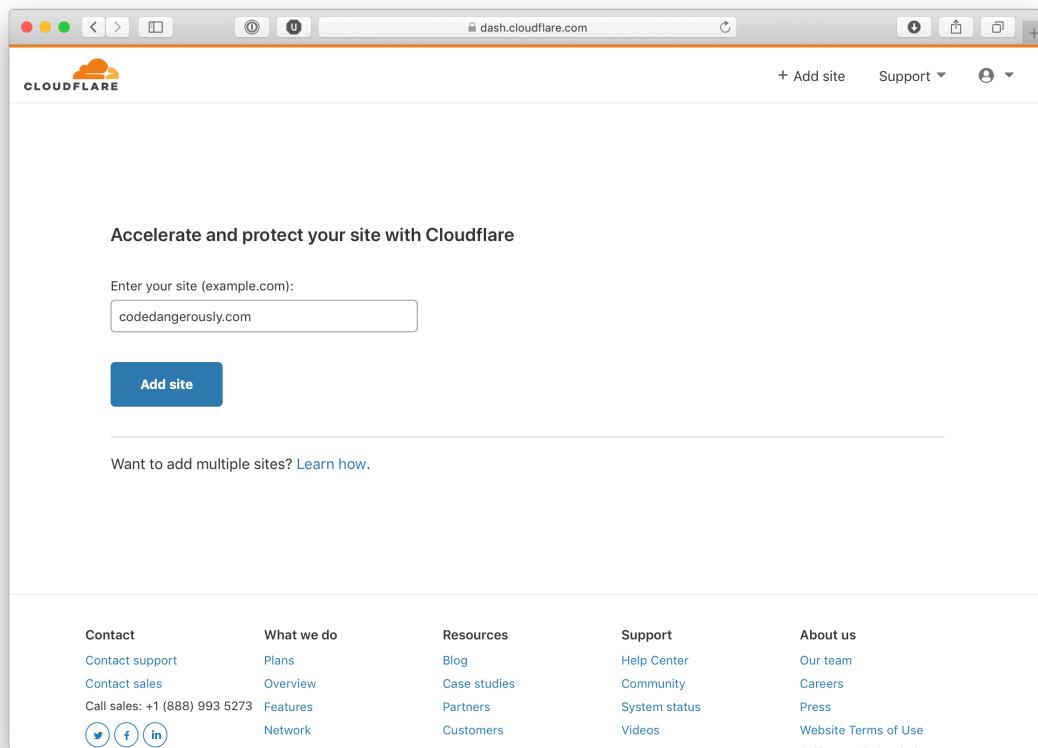


Figure 2.9: Starting Cloudflare's automated process to add your domain.

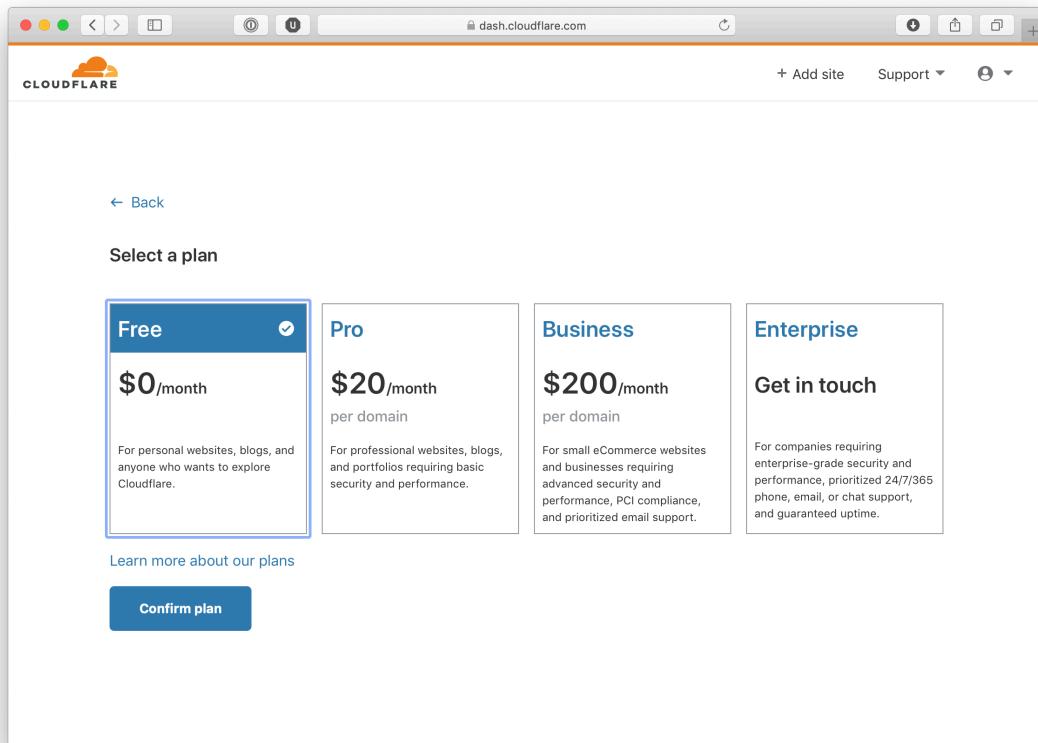
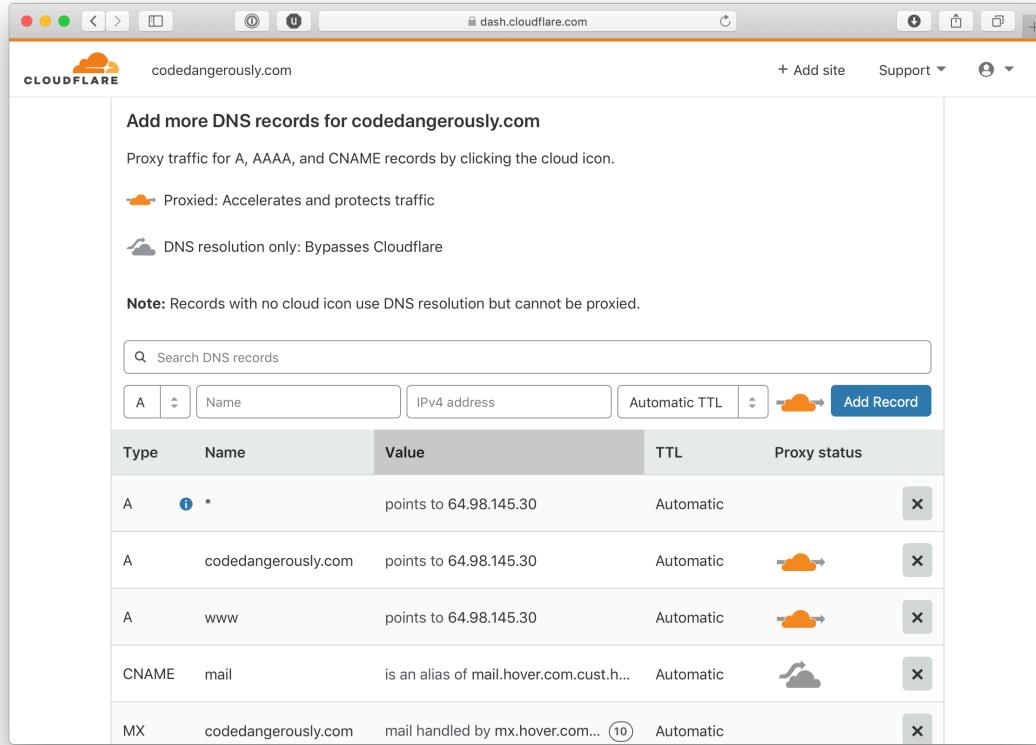


Figure 2.10: Selecting the free Cloudflare plan.

A screenshot of a web browser displaying the Cloudflare dashboard at dash.cloudflare.com. The domain 'codedangerously.com' is selected. The main content area shows a table of DNS records. A note at the top says 'Add more DNS records for codedangerously.com' and 'Proxy traffic for A, AAAA, and CNAME records by clicking the cloud icon.' Below this are two icons: a yellow cloud with arrows labeled 'Proxied: Accelerates and protects traffic' and a gray cloud labeled 'DNS resolution only: Bypasses Cloudflare'. A note below the icons says 'Note: Records with no cloud icon use DNS resolution but cannot be proxied.' The table has columns for Type, Name, Value, TTL, and Proxy status. It contains the following records:

Type	Name	Value	TTL	Proxy status
A	*	points to 64.98.145.30	Automatic	
A	codedangerously.com	points to 64.98.145.30	Automatic	
A	www	points to 64.98.145.30	Automatic	
CNAME	mail	is an alias of mail.hover.com.cust.h...	Automatic	
MX	codedangerously.com	mail handled by mx.hover.com...	Automatic	

Figure 2.11: Our domain's DNS settings report.

It should take less than a minute for Cloudflare to pull in the domain records, and when it's done the next page will show you the result of the scan from the last step: a report with all of the DNS records associated with your domain (Figure 2.11).

As you can see in Figure 2.11, there are a few A records (mentioned briefly in Section 1.2), which are set to flow through Cloudflare (the orange cloud icon). There is also a so-called *CNAME record*, which is set up to skip the Cloudflare service (the gray cloud icon). (We'll discuss more about these records types in Section 2.2.1.)

Since we just purchased this domain, the current records aren't very complicated, and there's nothing to configure on this step. However, if you were

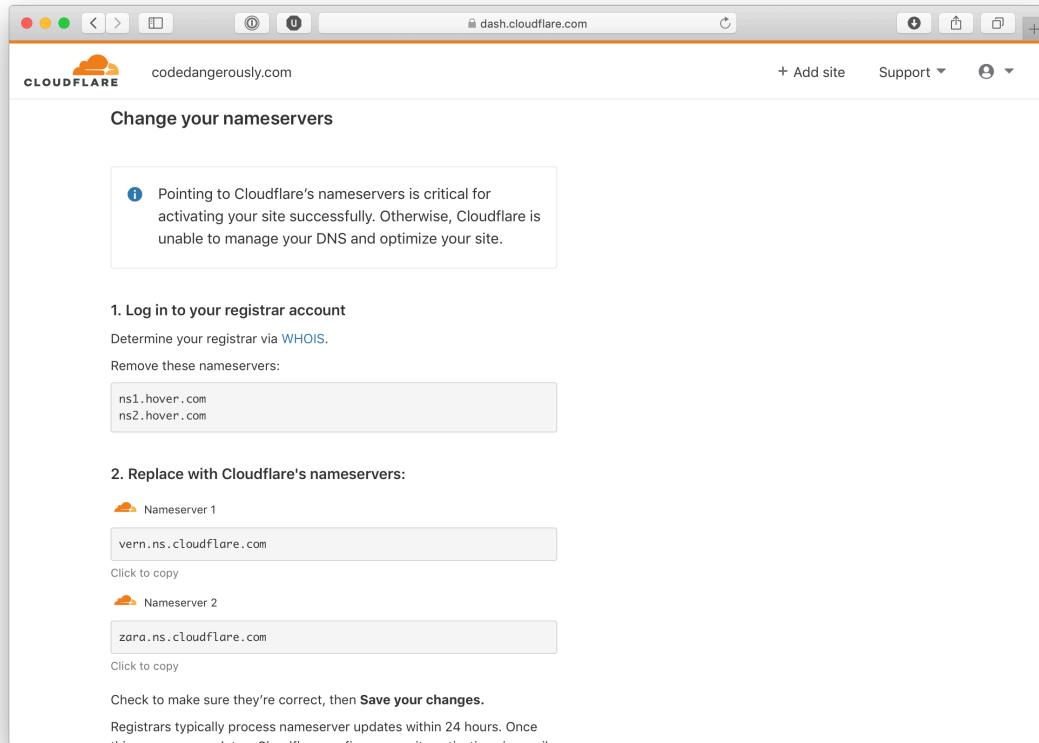


Figure 2.12: The Cloudflare nameservers.

importing a domain that had a bunch of custom settings, this would be a good place to carefully look over the records before hitting “Continue”.

At this point, you’ll be taken to the last step of the sign up and domain-addition process: the nameserver configuration. On this page, you’ll see a listing of the current nameservers for your domain (the ones from Hover), and the new Cloudflare servers that you need to set for your domain ([Listing 2.12](#)).

In the case of [Figure 2.12](#), the nameservers appear as follows:

```
vern.ns.cloudflare.com
zara.ns.cloudflare.com
```

Cloudflare has many different nameservers, though, so your results may vary.

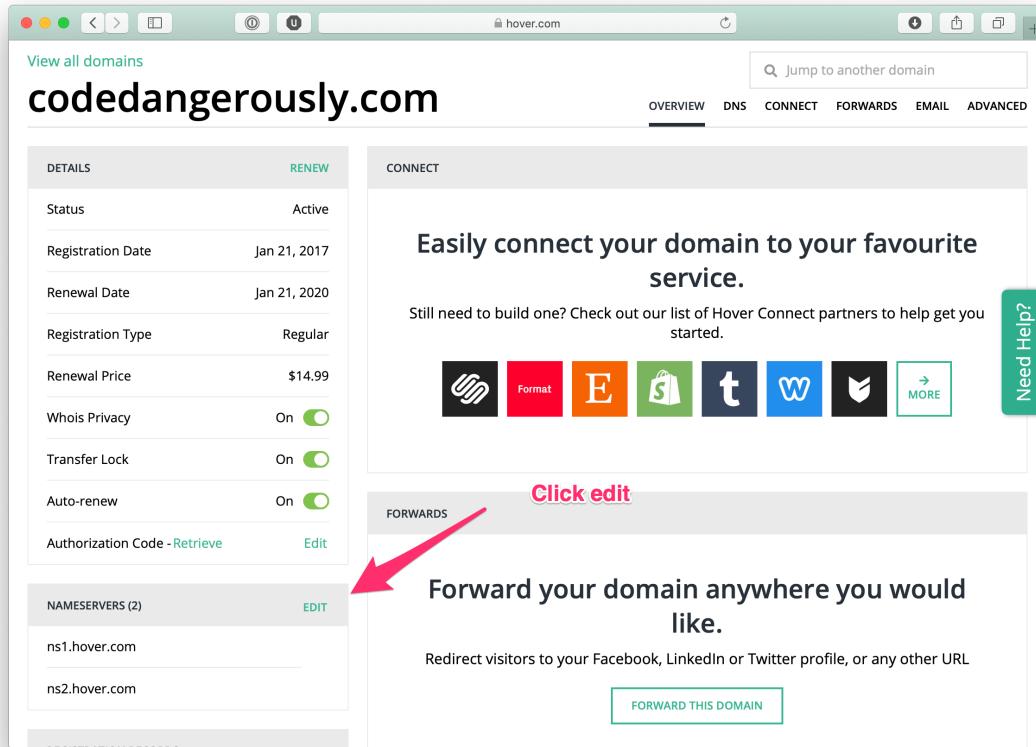


Figure 2.13: The button for editing a domain's nameservers.

At this point, you'll have to switch back to Hover (either in a different window or browser tab) and go to the detail page for your new domain:

https://www.hover.com/control_panel/domain/<yourdomain.com>

Then click the “Edit” button to the right of the nameservers ([Figure 2.13](#)).

This will open up a popup modal with the input boxes to allow you to make changes to your domain's nameservers. The first two boxes should already have text in them (the old nameservers). Switch back to the Cloudflare tab, copy the Cloudflare nameservers, and then switch back to Hover and paste the new addresses into the two fields labeled “Nameserver 1” and “Nameserver 2”

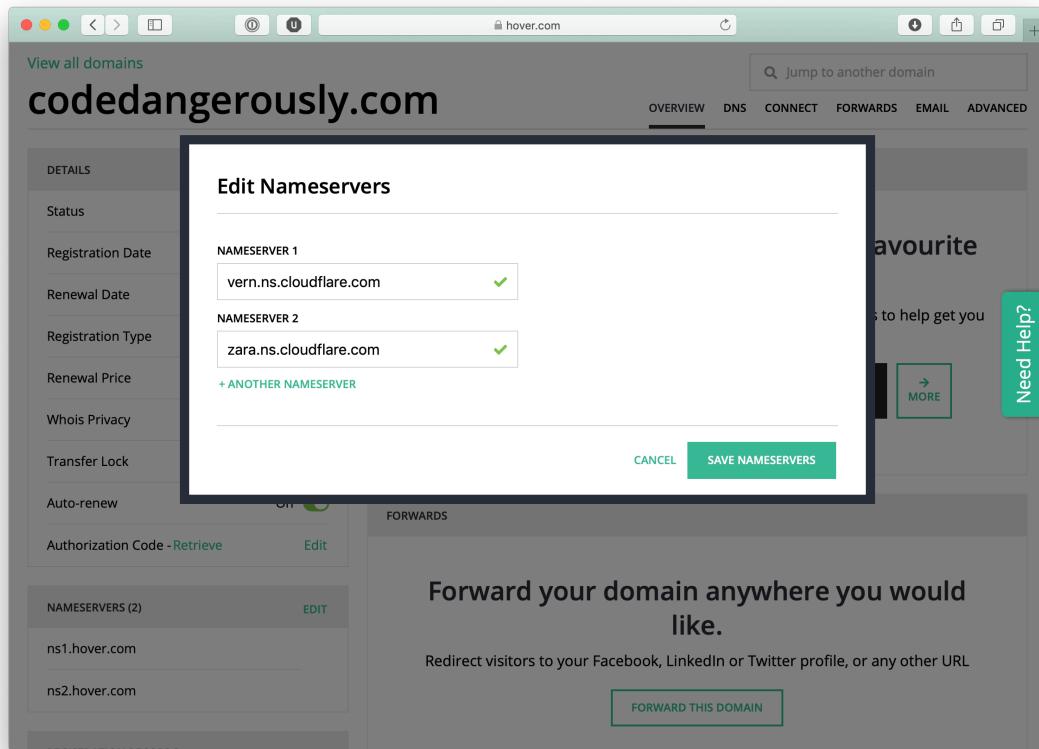


Figure 2.14: Adding the new Cloudflare nameservers to the Hover record for our domain.

(Figure 2.14). Click “Save Nameservers”.

The page will reload, and you should see your new name servers on the page in Hover (Figure 2.15).

Now, switch back to the Cloudflare window or tab and press the “Done, check nameservers” button at the bottom of the “Change your nameservers” page. You’ll be taken to the dashboard page for your domain on Cloudflare, which will be the page for a pending domain (Figure 2.16).

It shouldn’t take too long for your domain’s nameserver change to resolve; you can hit the “Re-check now” button on the pending page (as seen at the bottom of Figure 2.16) every so often until it updates. When the updates are

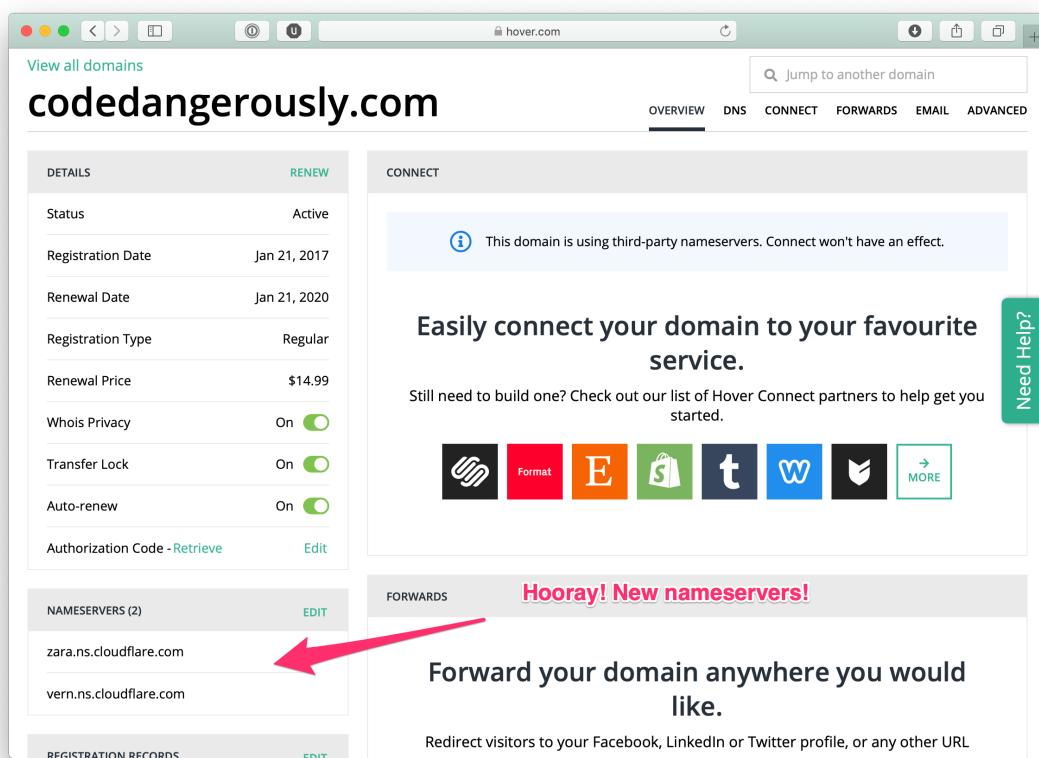


Figure 2.15: Your new nameservers have been added.

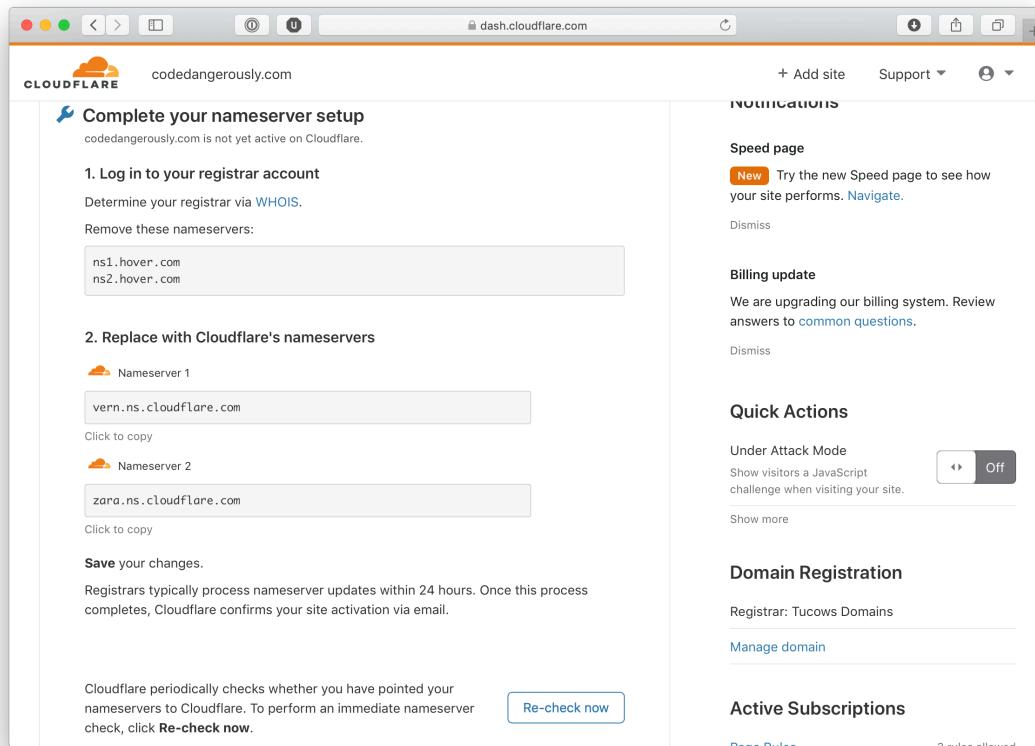


Figure 2.16: The page for a pending domain.

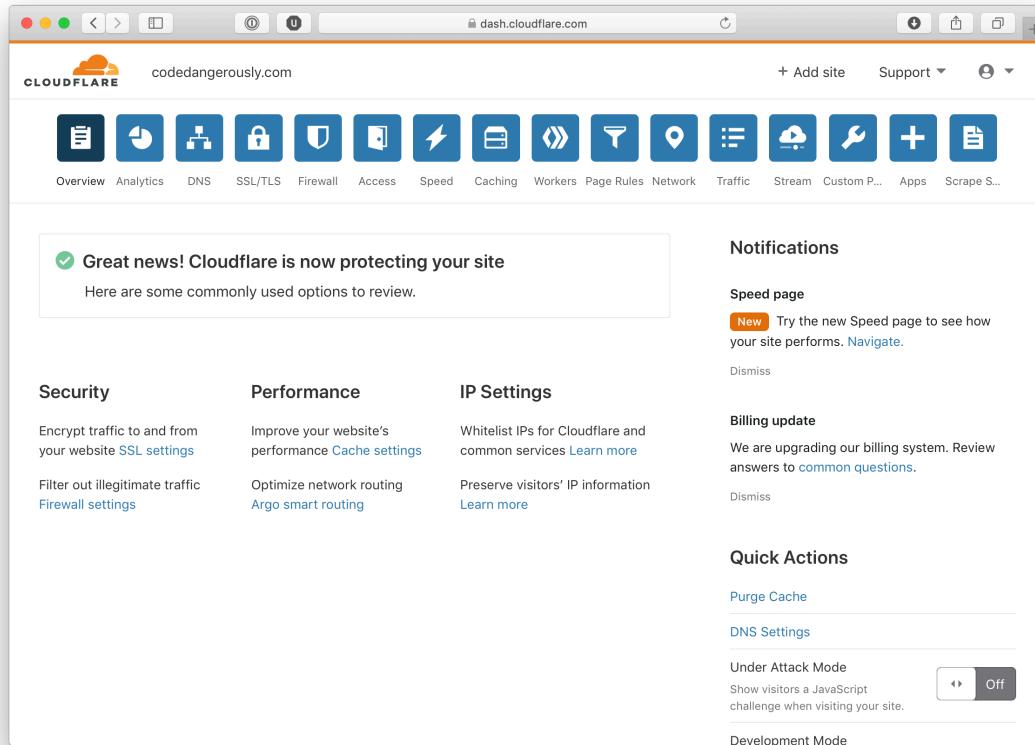


Figure 2.17: A newly active domain.

public on your registrar, your domain will be active on Cloudflare (Figure 2.17)!

So, congratulations!

But what on Earth did we just do? Well, now all page or DNS requests that are directed at your domain will first go through the Cloudflare service, which means that we can change the way those requests are handled using Cloudflare's user interface. For example, we'll be able to set up subdomains (Section 2.2.1), URL redirects (Section 2.4.1), and automatic forwarding to the secure version of our site (Section 2.4.2).

2.2 Custom domains at GitHub Pages

In this section, we'll explain how to connect a custom domain to GitHub Pages. This is a great (free!) option for hosting static websites on custom domains.⁵ (Indeed, the [Learn Enough blog](#) runs on a custom domain at GitHub Pages.) It's also a good exercise even if your primary interest is hosting dynamic websites ([Section 2.3](#)), since many of the principles are the same, so we suggest completing this section in any case.

If you don't have a GitHub account, you should follow the steps from [Chapter 2](#) of [Learn Enough Git to Be Dangerous](#) at this time. If you don't already have an example website at GitHub Pages, we suggest following the steps in [Chapter 1](#) of [Learn Enough HTML to Be Dangerous](#) to create and deploy a simple "hello, world!" app. ([Learn Enough CSS & Layout to Be Dangerous](#) develops a much more sophisticated website using the same basic technique.) In what follows, we'll assume you've set up such a site (or equivalent).

2.2.1 Configuring Cloudflare for GitHub Pages

Our first step is to tell Cloudflare that our site is located at GitHub Pages. To get started, click on the "DNS" menu item to go to the DNS settings for your domain ([Figure 2.18](#)). At this point, your screen should look something like [Figure 2.19](#).

So, what are these different types of records? We'll go in order and make the changes needed to get our site running off of GitHub Pages.

The *A records* that you see are also known as *address records* ([Section 2.1.3](#)); the top A record is known as an *apex record*. These settings are usually used to define the way that the *root domain* (e.g., `codedangerously.com`, without the `www`.) is handled, and they should always point to a valid IP address.

In the default settings transferred over from Hover, the `www` subdomain is set up with an A record that also points to a server's IP address. As was briefly

⁵Here *static* refers to static server-side assets like HTML, CSS, JavaScript, etc. The site's behavior itself still might be dynamic, typically due to JavaScript executing on the user's browser. An example of a statically generated site yielding dynamic client-side behavior appears in the image gallery developed in the [final chapter](#) of [Learn Enough JavaScript to Be Dangerous](#).

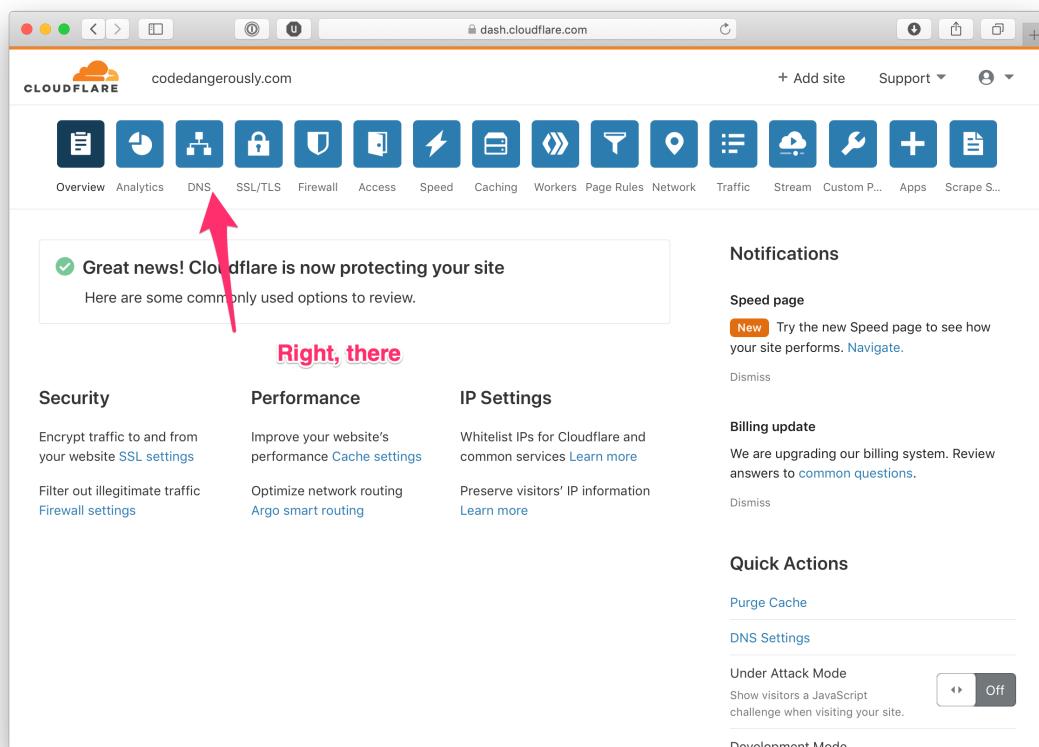


Figure 2.18: The Cloudflare feature icons.

The screenshot shows the Cloudflare DNS management interface for the domain `codedangerously.com`. The interface includes a search bar, an 'Advanced' button, and a table of DNS records. Below the table are sections for Cloudflare nameservers and custom nameservers.

Type	Name	Content	TTL	Proxy status	
A	*	64.98.145.30	Auto	DNS only - wildcard	
A	codedangerously.com	64.98.145.30	Auto	Proxied	
A	www	64.98.145.30	Auto	Proxied	
CNAME	mail	mail.hover.com.cust.hostedemail....	Auto	DNS only	
MX	codedangerously.com	mx.hover.com.cust.hostedemail....	10	Auto	DNS only

Cloudflare nameservers
To use Cloudflare, ensure your authoritative DNS servers, or nameservers have been changed. These are your assigned Cloudflare nameservers.

Type	Value
NS	vern.ns.cloudflare.com
NS	zara.ns.cloudflare.com

Custom Nameservers

Figure 2.19: How your DNS settings should look before we start making changes.

DNS management for **codedangerously.com**

Type	Name	Content	TTL	Proxy status	
A	*	64.98.145.30	Auto	DNS only - wildcard	
A	codedangerously.com	64.98.145.30	Auto	Proxied	
A	www	64.98.145.30	Auto	Proxied	
CNAME	mail	mail.hover.com.cust.hostedemail.com	Auto	DNS only	
MX	codedangerously.com	mx.hover.com.cust.hostedemail....	10	Auto	DNS only

Cloudflare nameservers

To use Cloudflare, ensure your authoritative DNS servers, or nameservers have been changed. These are your assigned Cloudflare nameservers.

Type	Value
NS	vern.ns.cloudflare.com
NS	zara.ns.cloudflare.com

Custom Nameservers

Figure 2.20: Removing the old record for the **www** subdomain.

mentioned in [Section 1.2.1](#), there are a lot of technical reasons why it’s a good idea to use a **www** subdomain, but the easiest way to describe why you should direct traffic there, and not to the root domain, is that down the road it gives you much more flexibility in how you deal with the traffic coming to your site. For our example site, we are going to get rid of this A record for the subdomain, and instead add a CNAME record for the **www** subdomain.

Let’s start the cleanup. Go ahead and click the “X” on the right-hand side of the screen for the **www** DNS record ([Figure 2.20](#)). When Cloudflare asks you to confirm, press the “OK” button. (If you are using a registrar that didn’t automatically add an A record for **www**, you don’t need to do anything at this point.)

DNS management for **codedangerously.com**

A few more steps are required to complete your setup.

- ✓ Add an A, AAAA, or CNAME record for www so that www.codedangerously.com will resolve.

Type	Name	Content	TTL	Proxy status
A	codedangerously.com	64.98.145.30	Auto	Proxied
MX	codedangerously.com	mx.hover.com.cust.hostedemail....	10	DNS only

Cloudflare nameservers

To use Cloudflare, ensure your authoritative DNS servers, or nameservers have been changed. These are your assigned Cloudflare nameservers.

Type	Value
NS	vern.ns.cloudflare.com

Figure 2.21: The cleaned-up DNS record.

Depending on which registrar you used, you may or may not also have an A record with the `*` in the name field. That's a wildcard record that handles requests to any random subdomain of your URL. That means that it allows random requests like `blarglebargle.codedangerously.com` to pass through to the server, but notice that there is no little orange cloud icon by the record. That means the traffic to those random addresses is not being handled by Cloudflare, and will instead flow straight to the server—which we definitely don't want.

To fix this, remove the wildcard A record (if present), as well as the CNAME record pointing to `mail.hover.com.cust.hostedmail`. This should leave your Cloudflare DNS records looking like Figure 2.21.

GitHub Pages IP addresses

Now, to have our site be served from GitHub Pages, we need to point the A record to the IP addresses of GitHub's servers instead of to the current address (yes, you could have just edited the A records, but we wanted to step you through removing and adding things from scratch). You can find the addresses in the [GitHub Pages documentation](#); at the time of this writing, the IP numbers for the GitHub Pages servers are as follows:

```
185.199.108.153  
185.199.109.153  
185.199.110.153  
185.199.111.153
```

Where there was a single A record before, we are going to add a second backup server address (you don't want your site to be unavailable because of a server issue that you can't control, do you?). To change the IP address for the record, just click on the IP address in the value column (it will turn into an input field), type in one of the GitHub IPs (like **185.199.108.153**), and then click somewhere else on the page to save the changes. Your DNS settings should look like [Figure 2.22](#).

Now let's add the second record. Above the list of records, you'll see a button that is labeled "+Add record". Clicking that causes Cloudflare to reveal a row of inputs. Make sure that the dropdown menu is set to **A**, and then add **yourdomain.com** in the Name field and the second GitHub IP number (e.g., **185.199.109.153**) into the address field ([Figure 2.23](#)). Then click "Save" to finish.

Your records should look like [Figure 2.24](#).

CNAME records

Let's now take a look at CNAME records, which are *canonical name* records. They are a type of DNS record that allows you to point traffic to any domain you want (such as a URL like **codedangerously.com**). This is in contrast to A records, which point only at IP addresses (like **185.199.108.153**).

Type	Name	Content	TTL	Proxy status
A	codedangerously.com	185.199.108.153	Auto	Proxied
MX	codedangerously.com	mx.hover.com.cust.hostedemail....	10	DNS only

Figure 2.22: Changed the A record to the first GitHub Pages IP.

The screenshot shows the Cloudflare dashboard for the domain `codedangerously.com`. A modal window titled "Adding a new 'A' record" is open. Inside the modal, there is a note: "codelangerously.com points to 185.199.109.153 and has its traffic proxied through Cloudflare." Below this note, the form fields are filled out: Type is set to "A", Name is "codedangerously.com", and IPv4 address is "185.199.109.153". The TTL is set to "Auto" and the Proxy status is "Proxied". At the bottom of the modal, there are "Cancel" and "Save" buttons. Below the modal, a table shows existing DNS records:

Type	Name	Content	TTL	Proxy status
A	codedangerously.com	185.199.108.153	Auto	Proxied
MX	codedangerously.com	mx.hover.com.cust.hostedemail....	10	Auto
				DNS only

At the bottom of the page, under "Cloudflare nameservers", it says: "To use Cloudflare, ensure your authoritative DNS servers, or nameservers have been changed. These are your assigned Cloudflare nameservers." The table shows one entry: NS vern.ns.cloudflare.com.

Figure 2.23: Adding the second alternate GitHub Pages server IP.

A few more steps are required to complete your setup.

✓ Add an A, AAAA, or CNAME record for **www** so that **www.codedangerously.com** will resolve.

Type	Name *	IPv4 address *	TTL	Proxy status
A	Use @ for root		Auto	Proxied
A	codedangerously.com	185.199.109.153	Auto	Proxied
A	codedangerously.com	185.199.108.153	Auto	Proxied
MX	codedangerously.com	mx.hover.com.cust.hostedemail....	10	DNS only

Cloudflare nameservers

To use Cloudflare, ensure your authoritative DNS servers, or nameservers have been changed. These are your assigned Cloudflare nameservers.

Figure 2.24: The DNS records with two GitHub servers added.

CNAMEs are often used to create [aliases](#) on sites so that visitors who go to a subdomain like `help.codedangerously.com` are instead permanently redirected to, say, `docs.codedangerously.com`. In our case, we want traffic that comes to our site using a `www` subdomain—i.e., `www.codedangerously.com`—to see the site’s homepage, which we can do by adding a CNAME record called `www` that points to the root domain.

To start, click the “+Add record” once again if the add record input fields aren’t showing (if you never left the Cloudflare page, they should still be visible). To add the CNAME record, click the dropdown at the beginning of the new record input form and select CNAME from the list. Then add `www` in the “Name” field and add your site domain in the “Target” field. Hit the “Save” button and your DNS settings should look something like [Figure 2.25](#).

Nice! That’s all the DNS setup needed to get our new domain to work with GitHub Pages. There is one last DNS record at the bottom, the *MX record*. That is a *mail exchanger* record that directs email traffic to your email server. We’ll be editing that record in [Section 3.1](#).

2.2.2 Configuring GitHub Pages

Now that we’ve configured Cloudflare for GitHub Pages, all we need to do is tell Pages about our custom domain. To get started, point your browser at your repository’s page at GitHub and locate the “Settings” link ([Figure 2.26](#)).

After clicking on “Settings”, scroll down to the “GitHub Pages” configuration panel, click on the dropdown menu labeled “None”, and select the “**master branch**” option ([Figure 2.27](#)).

After clicking on the “master branch” selection, scroll down to the Pages settings again and add your site’s domain name into the “Custom Domain” input field (in our case, `www.codedangerously.com`) ([Figure 2.28](#)).

After clicking “Save”, scroll down one more time to confirm that the Pages site has been published at the custom domain we specified ([Figure 2.29](#)).

That’s it! If you open up a new tab or window in your browser and go to your custom domain, you should now see your GitHub Pages site ([Figure 2.30](#)).

At this point, there’s only one significant task left, which is to arrange for our site to use only *canonical URLs*, a step that is covered in [Section 2.4](#). Unless

The screenshot shows the Cloudflare DNS management interface for the domain `codedangerously.com`. The interface includes a search bar, an 'Add record' button, and an 'Advanced' filter. The main table displays the following DNS records:

Type	Name	Content	TTL	Proxy status
CNAME	www	codedangerously.com	Auto	Proxied
A	codedangerously.com	185.199.109.153	Auto	Proxied
A	codedangerously.com	185.199.108.153	Auto	Proxied
MX	codedangerously.com	mx.hover.com.cust.hostedemail....	10	DNS only

Below the table, a section titled "Cloudflare nameservers" lists the assigned nameservers:

Type	Value
NS	vern.ns.cloudflare.com

Figure 2.25: The DNS records with both GitHub servers added.

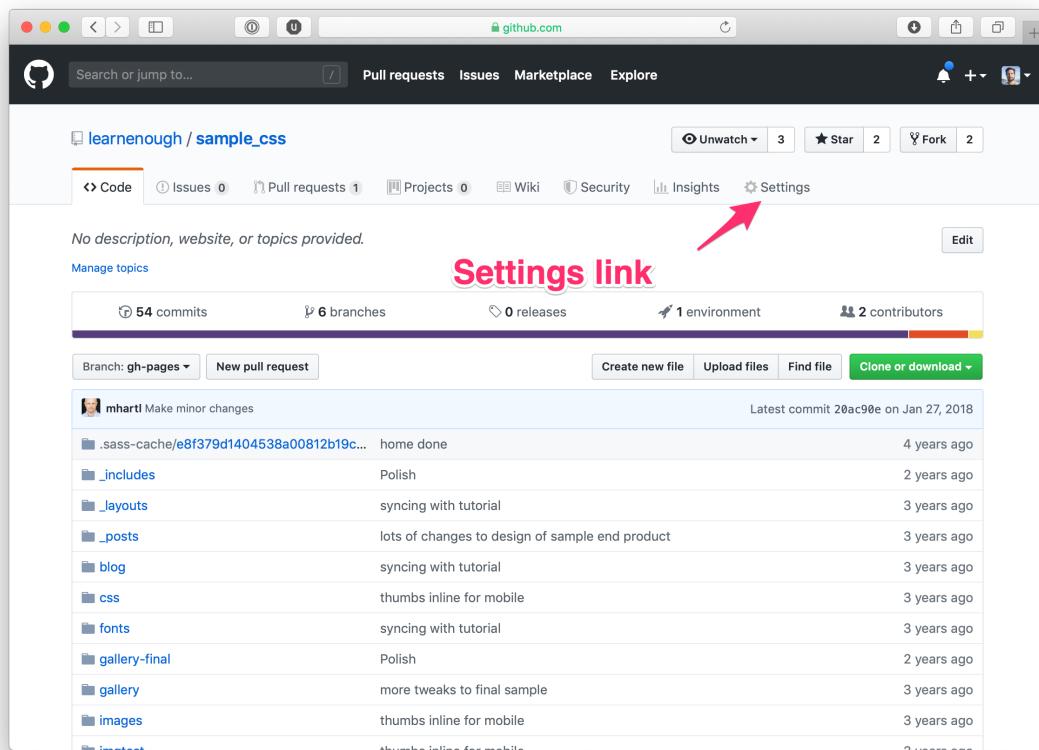


Figure 2.26: A GitHub repository's "Settings" link.

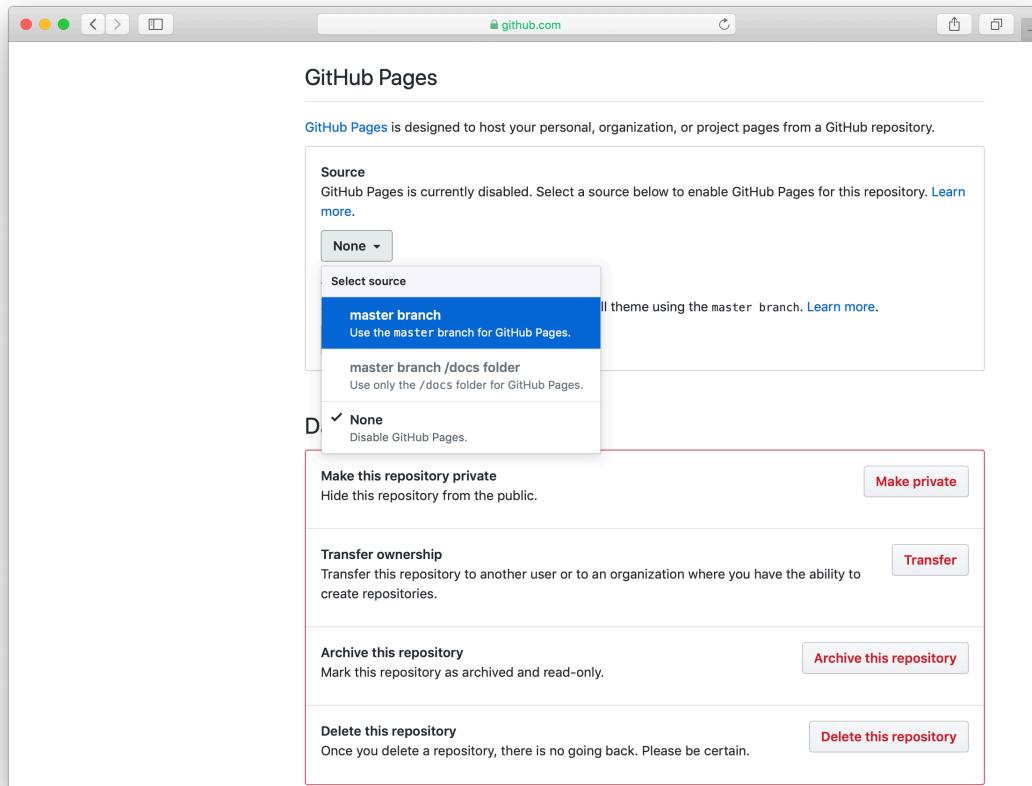


Figure 2.27: Serving the Pages site from the **master** branch.

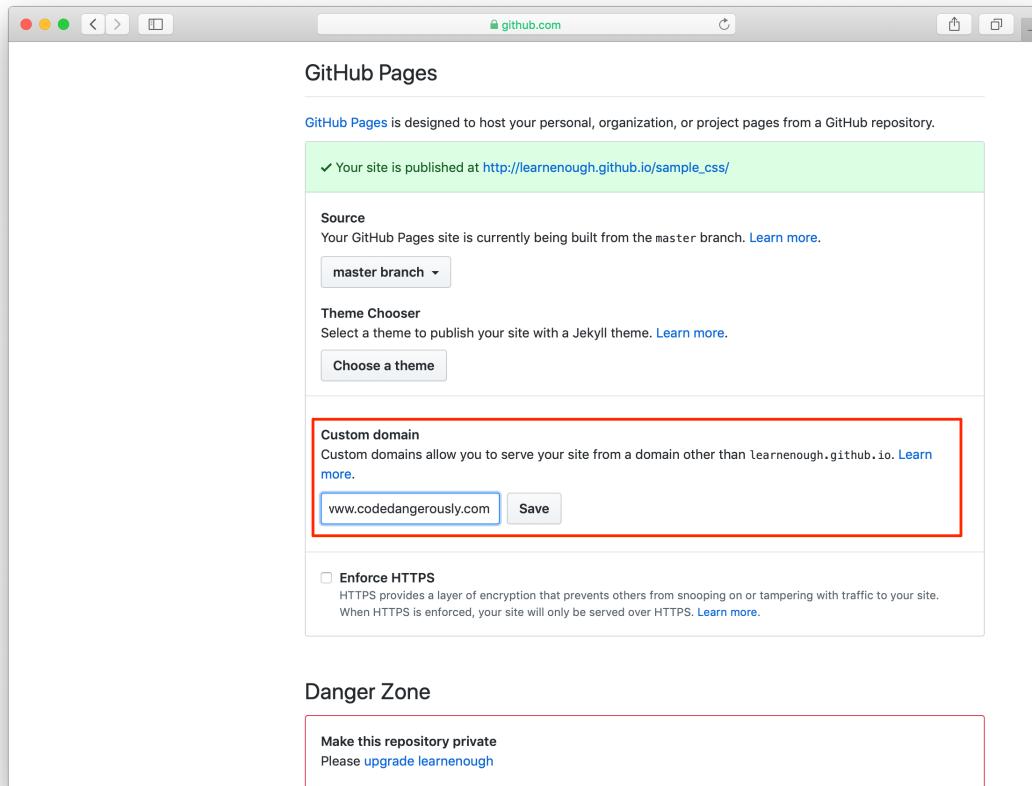


Figure 2.28: Configuring Pages to use a custom domain.

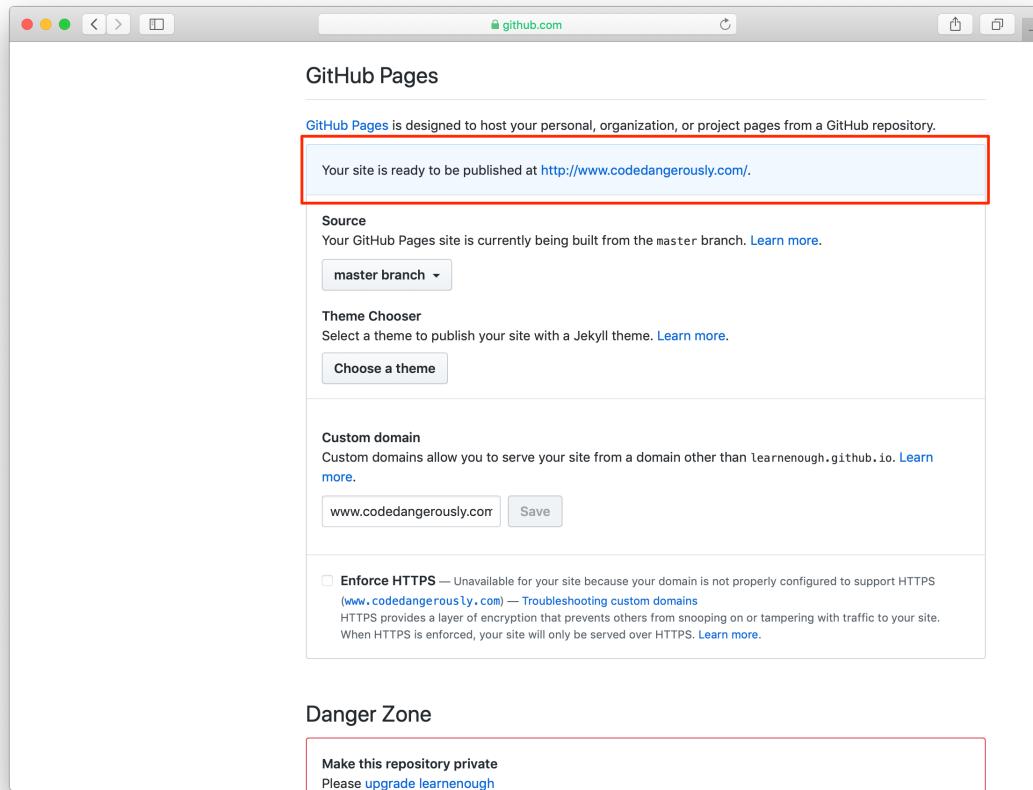


Figure 2.29: Confirming that the Pages site is published at a custom domain.

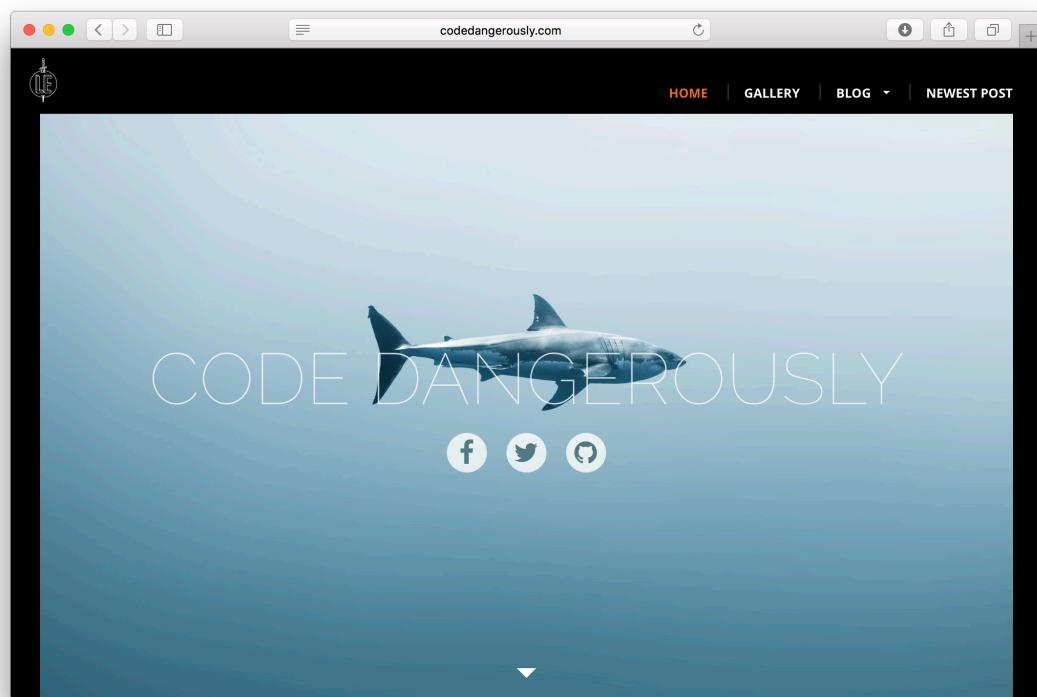


Figure 2.30: Can you believe all the steps it took to get to this place?

you want to deploy a web application to Heroku in Section 2.3, you can skip right to Section 2.4 at this time.

2.3 Custom domains at Heroku

In this section, we'll explain how to use a custom domain for a dynamic web application deployed to [Heroku](#). If you don't have such an app at present, you can create one by following Chapter 1 of the [Ruby on Rails Tutorial](#). If you don't want to follow these steps right now, you can skip to Section 2.4 without loss of continuity.

To get started, go to Heroku and open up the page for your application. Then click the “Settings” link in the navigation (Figure 2.31).

Next, scroll down to the “Domains and Certificates” section and click the “Add domain” button (Figure 2.32) to bring up the sidebar interface to add a new custom domain (Figure 2.33). (When you do this, note that it means you won't be serving that URL from GitHub Pages any longer. To deploy both GitHub Pages and Heroku apps with custom domains, you can either use two different custom domains or use different subdomains.)

After clicking “Next”, Heroku will show you a unique “DNS target” URL, which you should copy for future use (Figure 2.34). A fragment of the URL appears on the domain dashboard (Figure 2.35); to see the whole string again, click on the pencil icon (circled in Figure 2.35) to bring up the Edit Domain page (Figure 2.36).

At this point, there's only one step left: tell Cloudflare about the Heroku configuration. When configuring Cloudflare for a Heroku-deployed site, we can actually remove the A records entirely and use a CNAME for the root domain, with content equal to the domain target:

```
protected-ravine-fx46nfntjzuy2rqy8zx0sdu.herokuapp.com
```

This can be copied from the interface shown in Figure 2.34 and pasted into Cloudflare as in Figure 2.37.

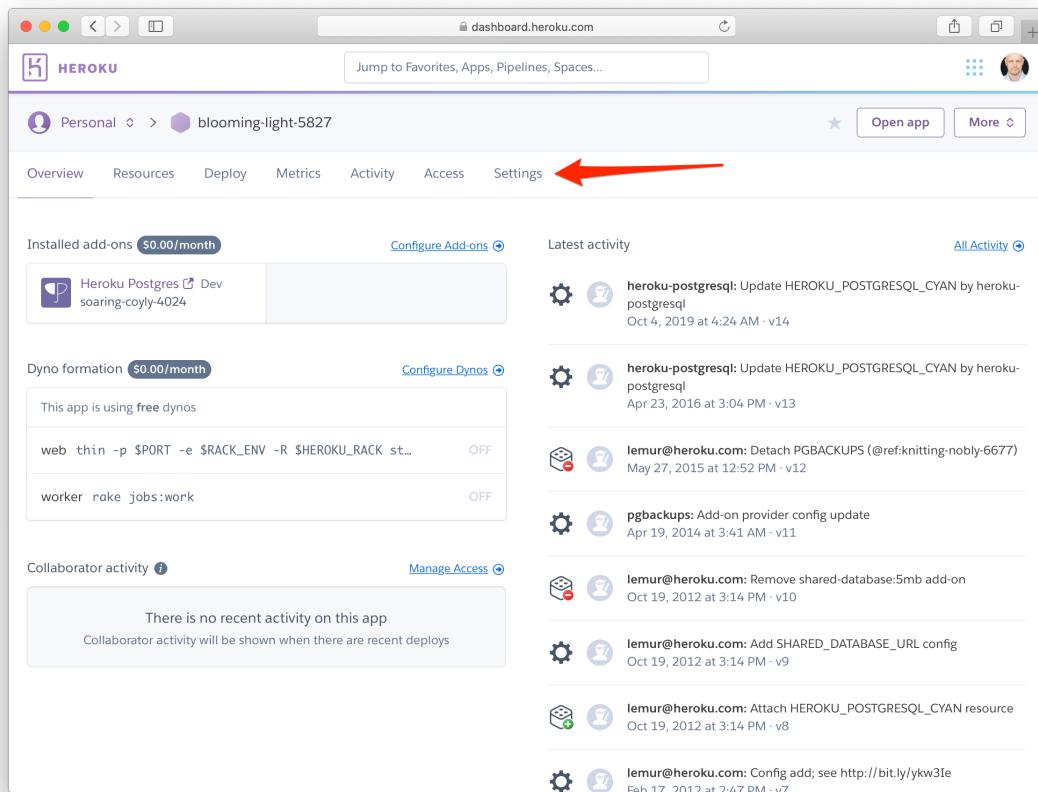


Figure 2.31: The Settings link for a Heroku app

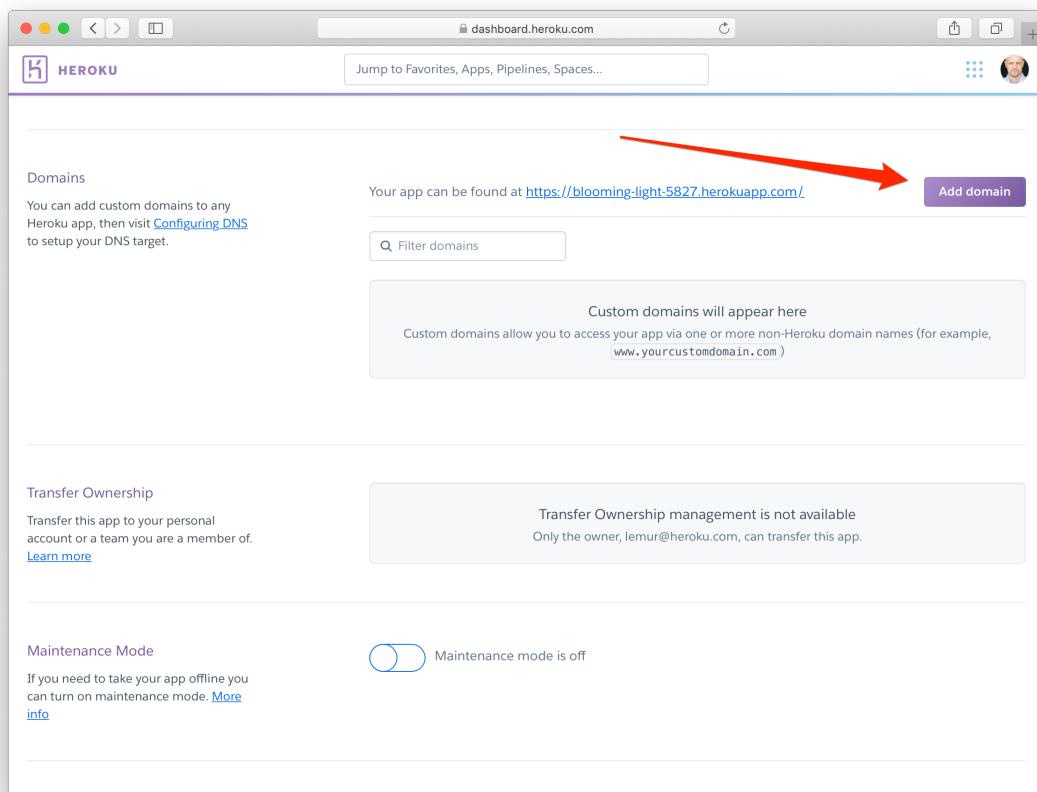


Figure 2.32: The link to add a Heroku domain.

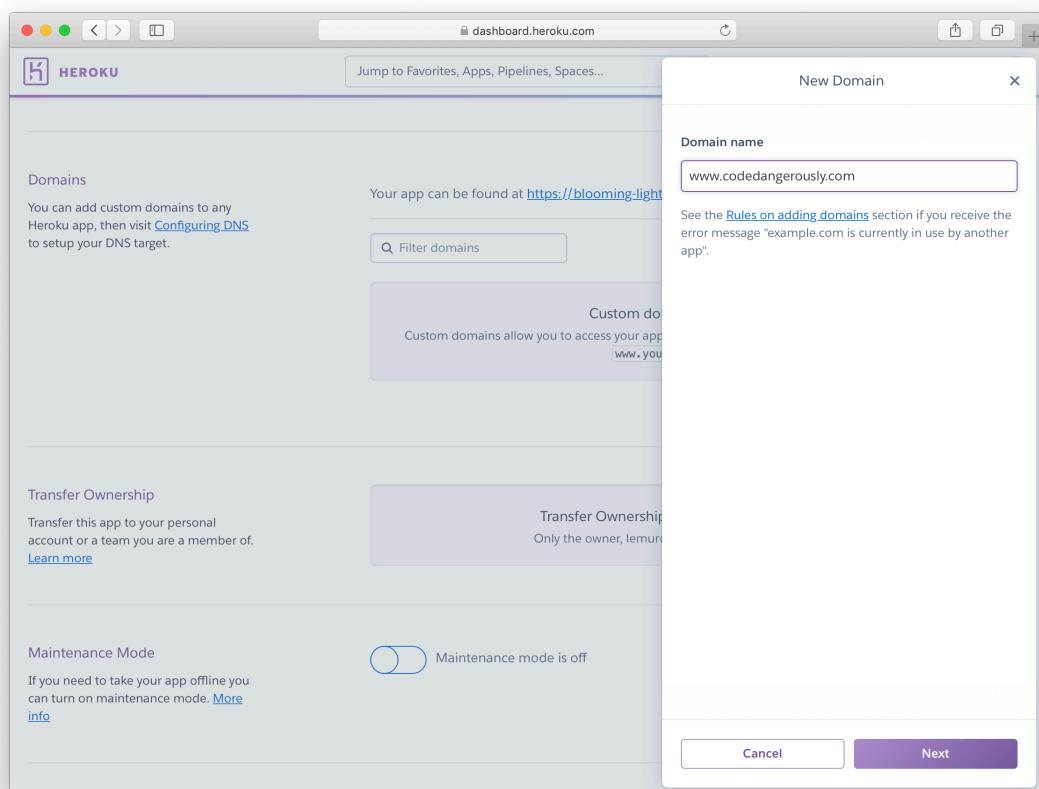


Figure 2.33: The interface for adding a Heroku custom domain.

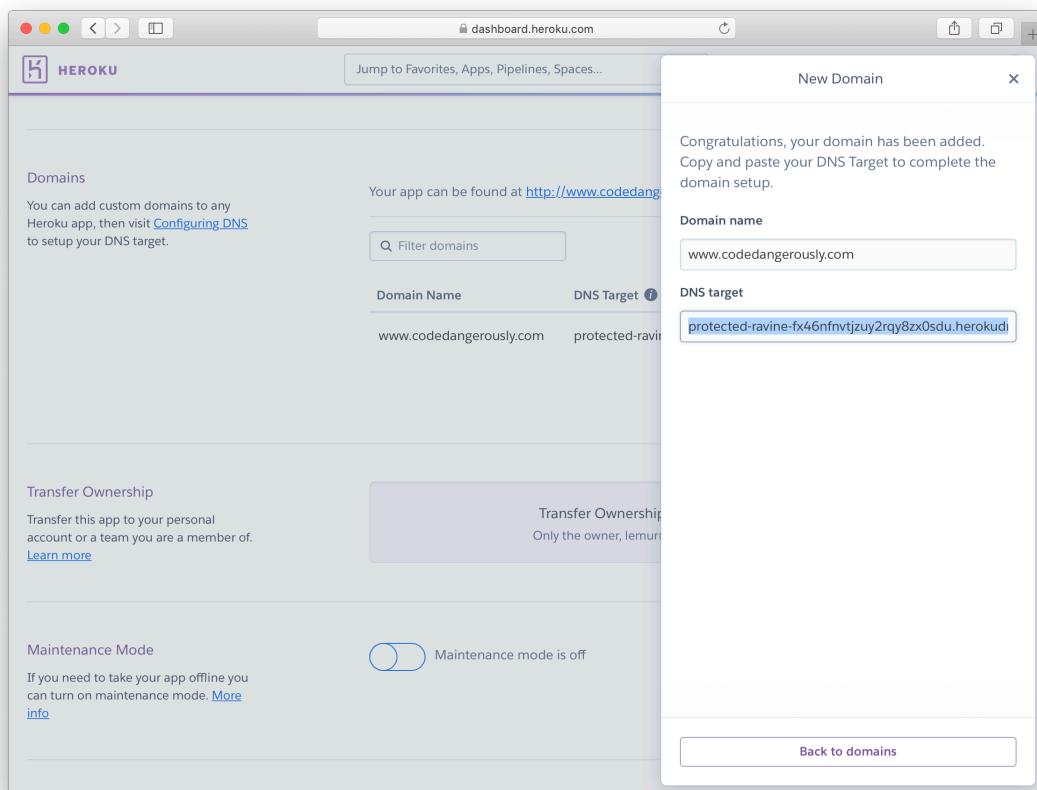


Figure 2.34: Copying the DNS target on the New Domain page

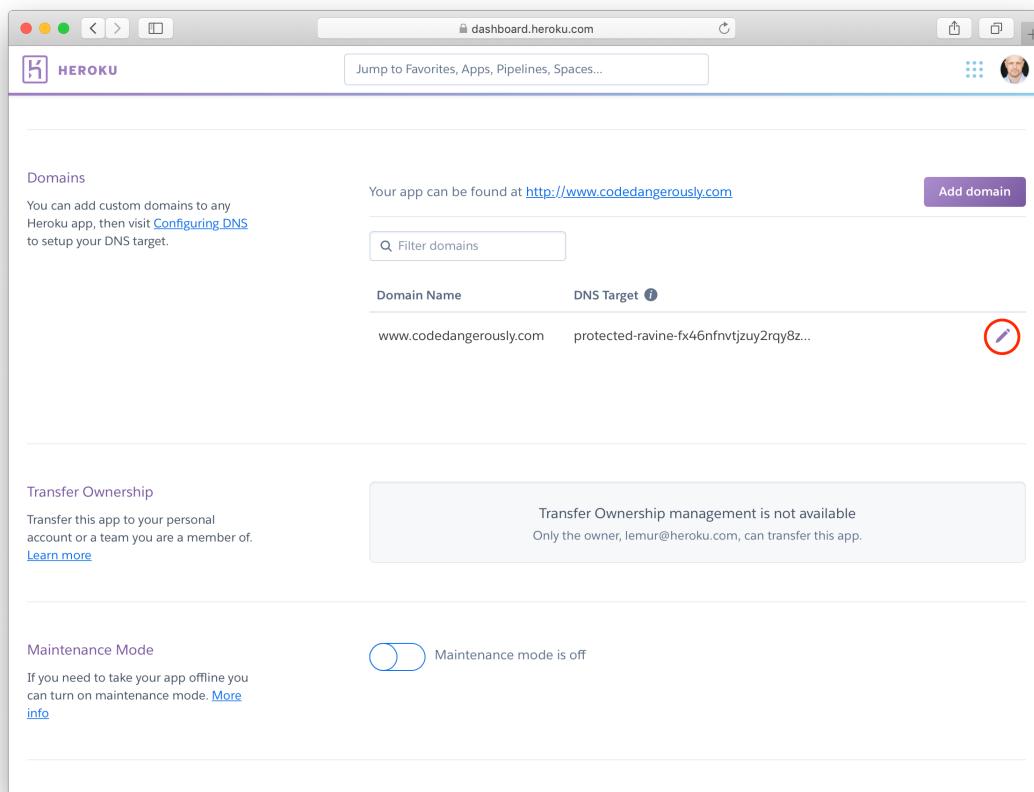


Figure 2.35: The Heroku dashboard with a custom domain.

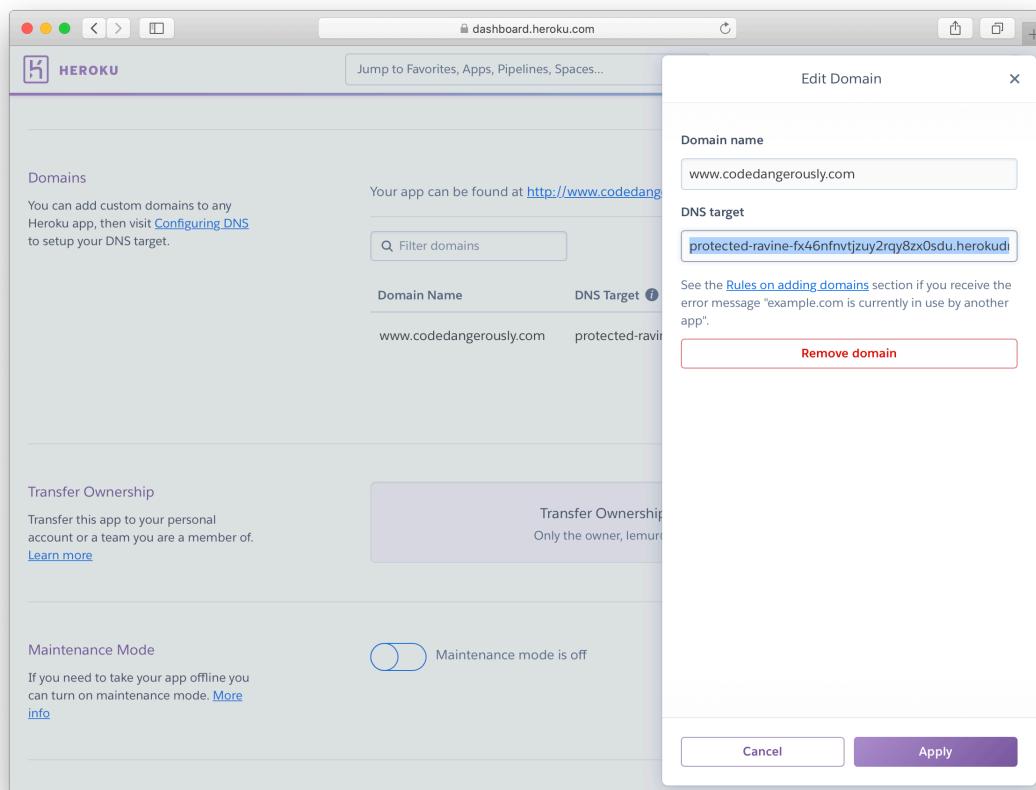


Figure 2.36: Viewing the DNS target on the Edit Domain page.

DNS management for **codedangerously.com**

[name] is an alias of [target] and has its traffic proxied through Cloudflare.

Type	Name *	Target *	TTL	Proxy status
CNAME	Use @ for root		Auto	Proxied
Cancel Save				
Type	Name	Content	TTL	Proxy status
1 CNAME	codedangerously.com	protected-ravine-fx46fnvljzuy2rqy8zx0...	Auto	Proxied
CNAME	www	codedangerously.com	Auto	Proxied
MX	codedangerously.com	mx.hover.com.cust.hostedemail...	10	Auto
				DNS only

Cloudflare nameservers

To use Cloudflare, ensure your authoritative DNS servers, or nameservers have been changed. These are your assigned Cloudflare nameservers.

Type	Value
NS	vern.ns.cloudflare.com
NS	zero.ns.cloudflare.com

Figure 2.37: Adding the Heroku domain target at Cloudflare.

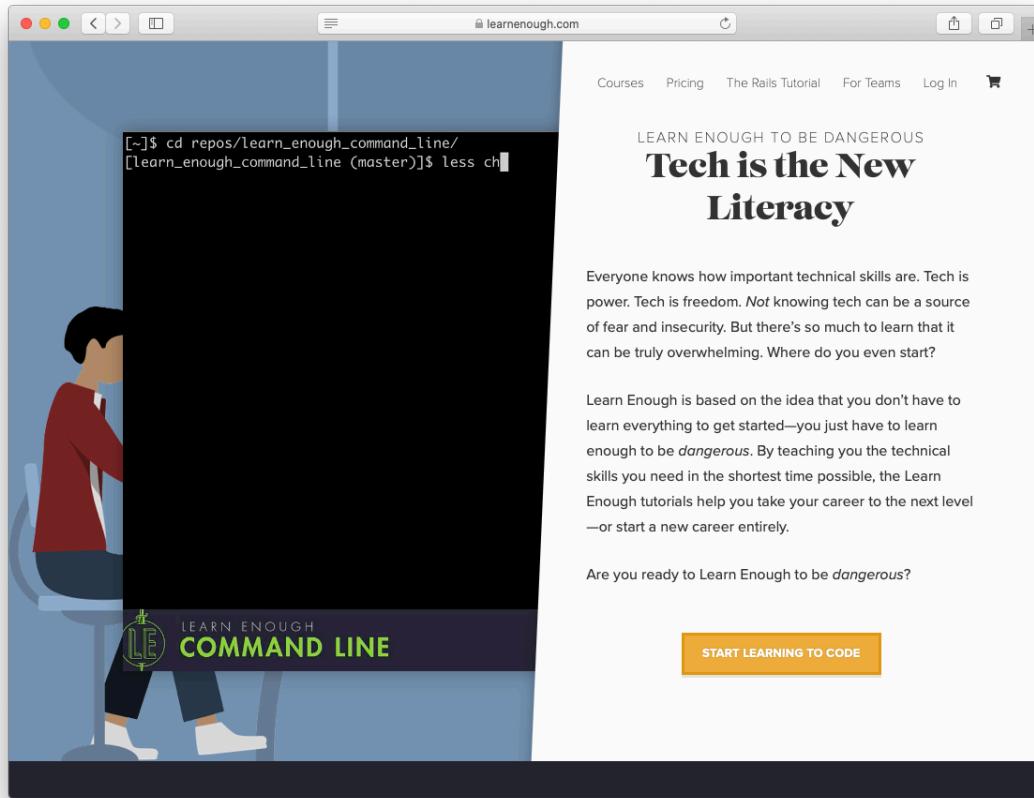


Figure 2.38: An example of a Heroku-deployed app running on a custom domain.

That should be the last step in making your Heroku app available on the Internet at your custom domain. As one of many examples of such a site, consider... learnenough.com itself (Figure 2.38)! That's right: learnenough.com—which (you guessed it!) gets forwarded to www.learnenough.com—is a Heroku app configured at Cloudflare and running on a custom domain.⁶ Ensuring things like that final detail—namely, forwarding the root domain to its **www** equivalent—is the subject of Section 2.4.

⁶The learnenough.com domain is actually registered at [Namecheap](#) rather than at [Hover](#), but the principles are the same.



Figure 2.39: This is the wrong type of cannon to be thinking of.

2.4 Canonical URLs

Whether you’re using a static site as developed in [Section 2.2](#) or a dynamic web application as developed in [Section 2.3](#), there’s one detail left to finish: ensure that the site uses *canonical URLs* to ensure a consistent and secure experience for all users.

The word “canonical” has its origins in religious [canon](#), and is now used in a variety of [technical contexts](#) ([Figure 2.39](#)).⁷

In our case, standardizing our site’s URLs to use a canonical form involves two principal tasks:

1. Ensure that all pages use the standard [www](#) form of the site rather than the root domain.
2. Ensure that connections to the site are always secure.

⁷Image retrieved from <https://flic.kr/p/dmDByU> on 2020-02-18. Copyright © 2012 by Ozzy Delaney and used under the terms of the [Creative Commons Attribution 2.0 Generic](#) license.

We'll accomplish both tasks using features of Cloudflare: the first with *page rules* (Section 2.4.1) and the second by enforcing HTTPS everywhere (Section 2.4.2).

2.4.1 Cloudflare page rules

At this point, our website is properly configured to serve the home page from the address `www.codedangerously.com`. In order to avoid serving content from two different domains (which can complicate things like session cookies), we'd like to arrange for the `www` version to be the *only* address for the home page for our website. In particular, we want any traffic pointed at the root domain `codedangerously.com` to be automatically redirected to `www.codedangerously.com`. We can accomplish this using Cloudflare *page rules*.

Page rules are a powerful and flexible tool, and we'll barely be scratching the surface here, but performing a redirect is one of their most common and important applications. To get started, click the “Page Rules” icon in the Cloudflare navigation, and then click the “Create Page Rule” button to open the new rule interface (Figure 2.40).

One thing we could do is to forward *everything* at the root domain to the `www` equivalent:

```
codedangerously.com -> www.codedangerously.com
```

There's one problem, though: what if someone hits a subpage of our site? Rather than do this:

```
codedangerously.com/blarg -> www.codedangerously.com
```

We'd rather do this:

```
codedangerously.com/blarg -> www.codedangerously.com/blarg
```

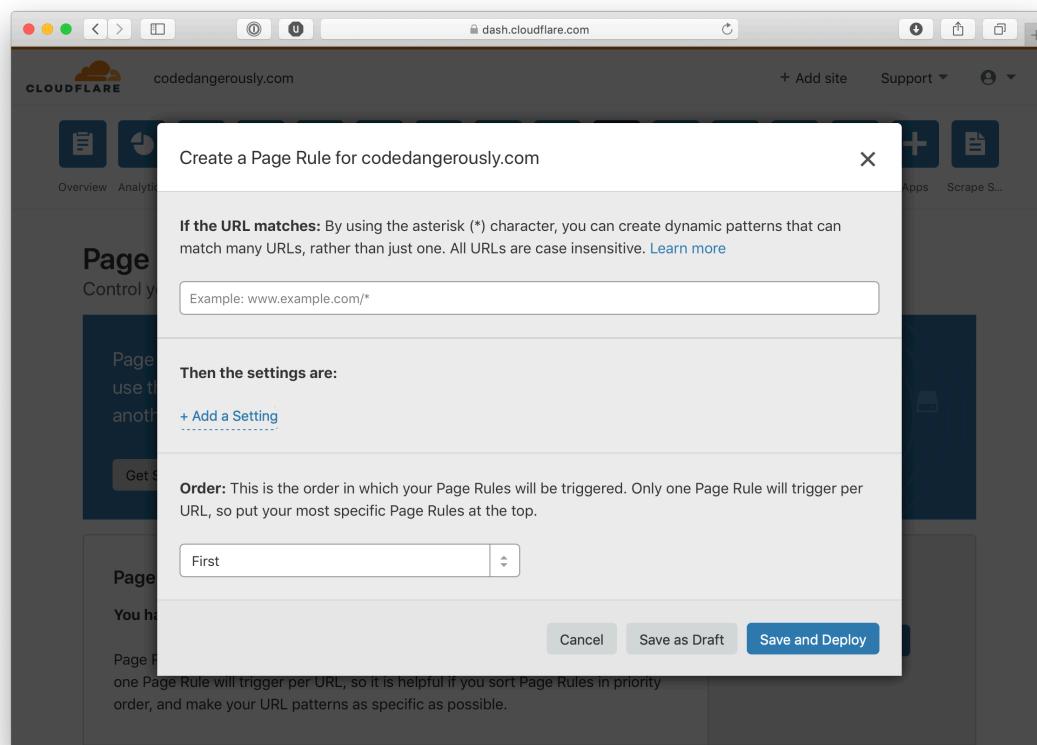


Figure 2.40: Getting started with a page rule.

This way the root and `www` domains will truly be equivalent, and *any* attempt to access the site via the root URL will end up on the corresponding `www` version.

We can arrange for this behavior using a *wildcard* (a concept mentioned briefly in [Section 2.2.1](#)), which will dynamically match whatever the user enters after the domain name, and keep it intact after the redirect. So if they enter `codedangerously.com/blarg`, the page rule will make sure they are forwarded automatically to `www.codedangerously.com/blarg`. We don't want to lose that `/blarg` at the end of the address!

To configure Cloudflare to use a wildcard and make a redirect, we'll start with the interface shown in [Figure 2.40](#). In the big input field, add your domain name, a forward slash, and then an asterisk to represent the wildcard—in our case, `codedangerously.com/*`. Then click the “+ Add a Setting” link, and in the dropdown choose “Forwarding URL”. You'll now see more options, including two kinds of *redirects*: 301 and 302 ([Box 2.1](#)).

Box 2.1. 301 versus 302 redirects

As we saw in the page rule dropdown, there are two types of redirects: *301 - permanent* and *302 - temporary*. You can probably guess the intended purpose of each redirect from the names, but what you might not realize is that your choice has an effect on search engine results.

If you choose a 302 redirect, then search engines like Google will assume that at some point the redirect will be removed and the page in question will be a destination that users can reach. That means that the search engines don't combine the traffic—both URLs exist as separate objects. That isn't good for analytics or search engine optimization (SEO) if you really intended for the redirect to be permanent.

If you set the redirect to 301, then search engines consider the URLs to be exactly the same. So even if a user came to your site through a redirected URL, your main URL would still get credit for purposes of traffic estimates and SEO.

In practice, the vast majority of page rule redirects will be 301s. Indeed, such permanent redirects are so common that “301” is often used as a verb, as in “[Please 301 your old links instead of breaking them.](#)”

For canonical URLs, we want the redirects to be permanent, so set “Status Code” to “301 - Permanent Redirect”, and in the “Destination URL” field add in the full URL for your site (including the protocol string `https://`) followed by `/$1`:

```
https://www.example.com/$1
```

Here `$1` represents the text matched by the first wildcard. In other words,

```
example.com/*
```

matches

```
example.com/blarg
```

and makes the string `blarg` available as `$1`. As a result, the code

```
https://www.example.com/$1
```

produces the canonical URL

```
https://www.example.com/blarg
```

In the present case, there is only one wildcard, but it’s possible to match on multiple wildcards as well (Box 2.2).

Box 2.2. Matching URLs

As you might be able to guess, page rule matches are numbered sequentially, so if we had two wildcards in a rule they would be available as \$1 and \$2. For example, `*.codedangerously.com/*` is a pattern to match any subdomain; in this case, `foo.codedangerously.com/bar` would put `foo` in \$1 and `bar` in \$2. (Note that the free Cloudflare tier supports subdomain wildcards only for CNAME records that are defined explicitly. Redirecting *all* subdomains requires upgrading to the [Business plan](#).)

The same permanent forwarding can be used to direct users who requested a completely different domain over to your main site. So if we purchased `dangerouscoding.com`, we could forward any `dangerouscoding.com` URL to `codedangerously.com` using the above wildcard method to preserve the specific page they requested. The matching rule in this case would be

```
*dangerouscoding.com/*
```

where the leading wildcard matches all subdomains as well as the root domain (i.e., both `subdomain.dangerouscoding.com` and `dangerouscoding.com`). Assuming we want to ignore the subdomain, the 301 redirect would then be

```
https://www.codedangerously.com/$2
```

where we use \$2 because the subpage is matched by the second wildcard.

In our case, the forwarding URL is

```
https://www.codedangerously.com/$1
```

as seen in [Figure 2.41](#). Upon clicking the “Save and Deploy” button to save your new rule, your page rules should look something like [Figure 2.42](#).

If you want to see whether or not the redirect worked, you can try visiting the root domain in your browser. For example, if we enter `codedangerous-`

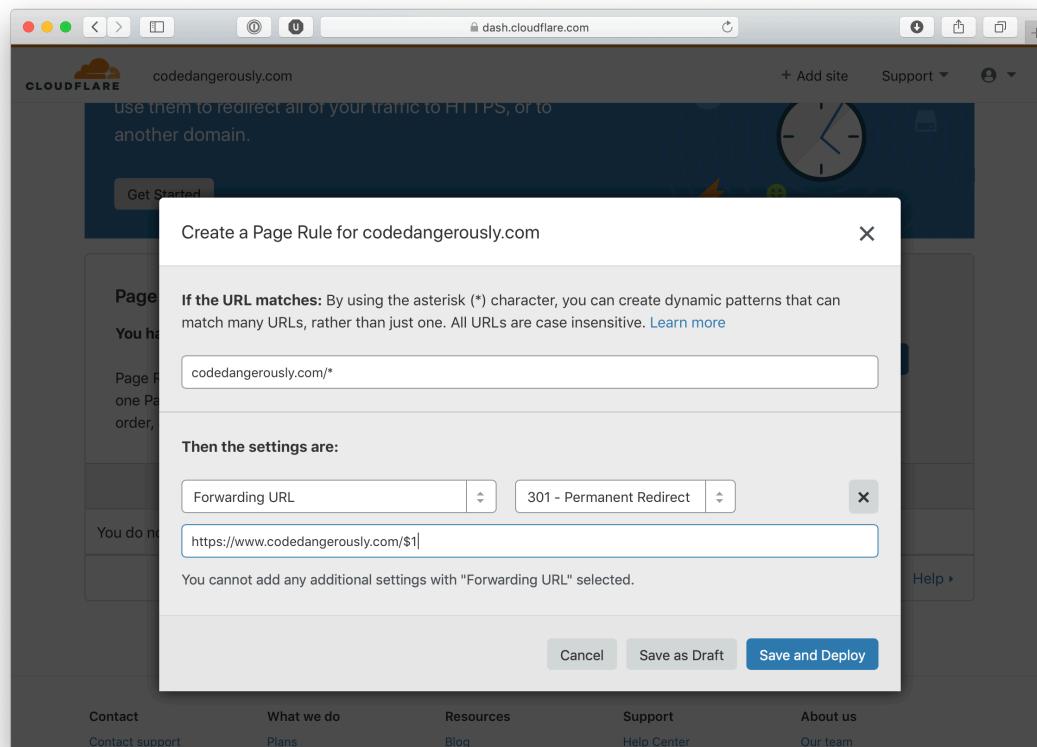


Figure 2.41: The new page rule before being submitted.

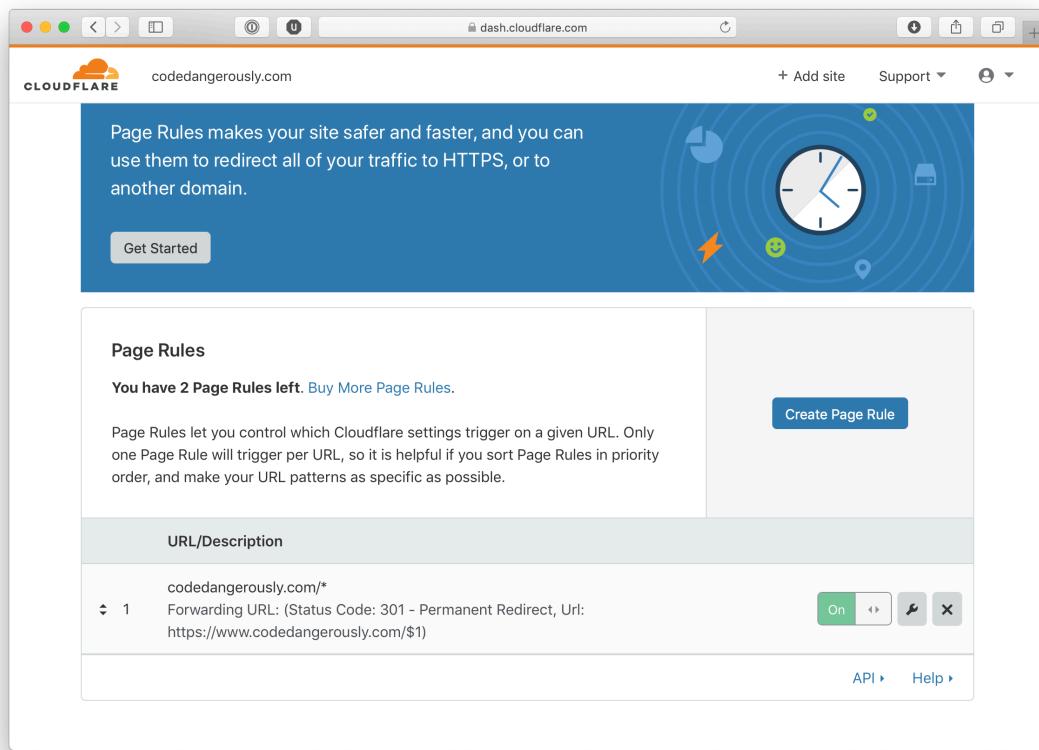


Figure 2.42: The page rule added onto the domain.



Figure 2.43: It works, it really works!

ly.com into a browser we get redirected to **www.codedangerously.com**, as shown in Figure 2.43.

Another convenient way of checking URL forwarding is to use the **curl** command at the command line (as covered in *Learn Enough Command Line to Be Dangerous*). In particular, we can use the **--head** option to return just the HTTP header (rather than the whole page):

```

1 $ curl --head codedangerously.com
2 HTTP/1.1 301 Moved Permanently
3 Date: Thu, 13 Feb 2020 16:47:00 GMT
4 Connection: keep-alive
5 Cache-Control: max-age=3600
6 Expires: Thu, 13 Feb 2020 17:47:00 GMT
7 Location: https://www.codedangerously.com/
8 Server: cloudflare
9 CF-RAY: 5648481ded567896-LAX

```

We see in line 2 that the HTTP status code is **301** as required, with the forwarding URL (line 7) including the **www** subdomain and the required secure **https://** protocol indicator.

2.4.2 HTTPS everywhere

Even though the forwarding rule in Section 2.4.1 automatically forwards to a secure **https** URL, we still have a problem: what if a user explicitly types in an insecure URL using **http** (with no **s**)? This would happen if someone typed this into their browser's address bar:

```
http://www.codedangerously.com/
```

The answer at present is not good: our system in fact would return the *insecure* page (Figure 2.44), which as noted in Section 2.1.1 is a **Bad Thing**.

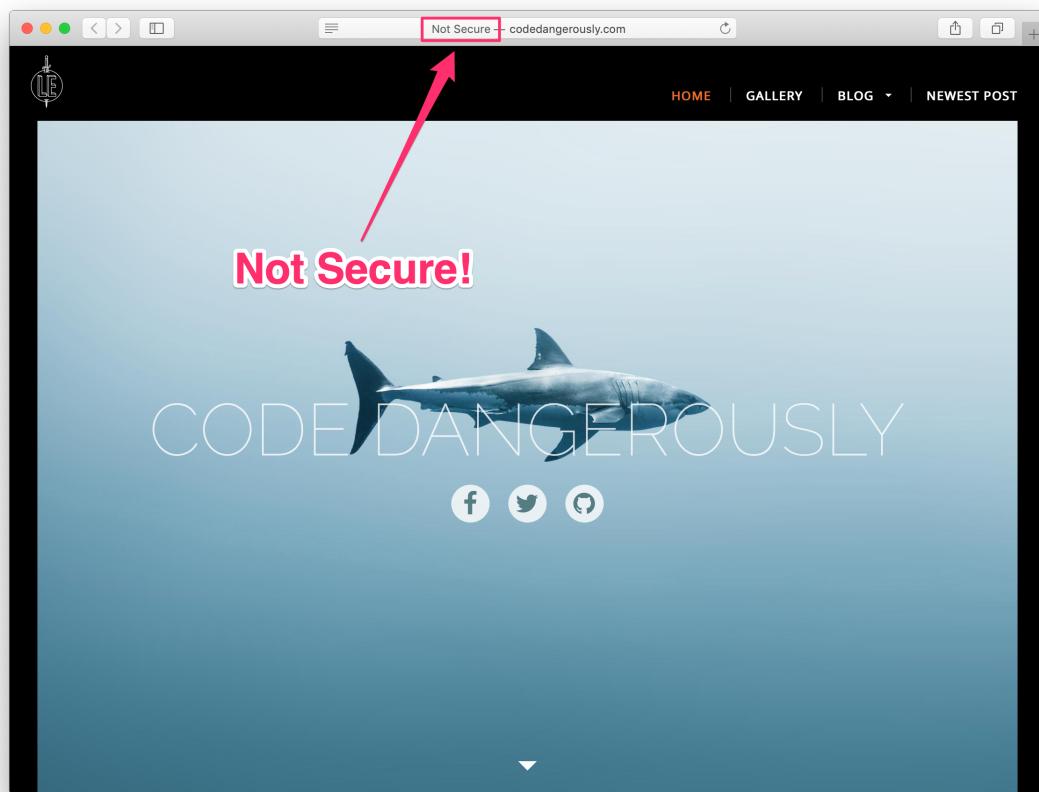


Figure 2.44: Our current site using **http** (without the **s**)—insecure!

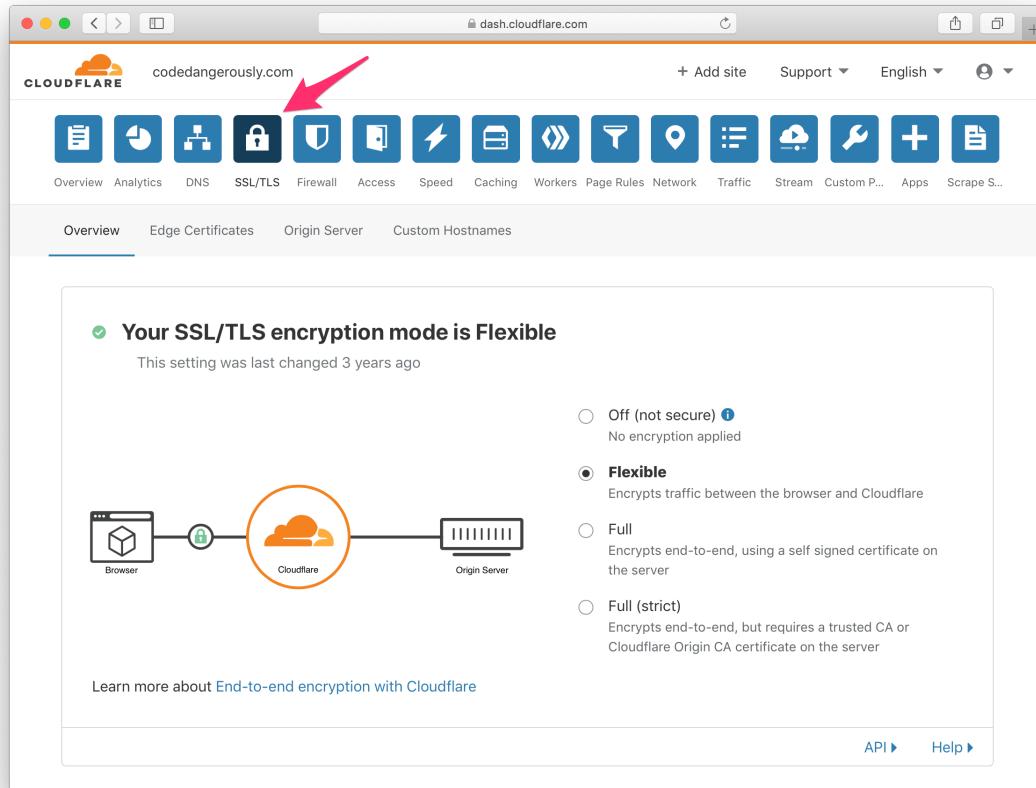


Figure 2.45: The SSL/TLS page at Cloudflare.

We can fix this security issue by using HTTPS everywhere, automatically forwarding *all* **http** requests to the corresponding **https** URL. This used to require a separate page rule—a real inconvenience, since Cloudflare’s free tier allows a total of only three such rules—but this operation is so common and so important that Cloudflare now includes a special setting just for HTTPS forwarding.

The steps are simple. First, click on the SSL/TLS icon (Figure 2.45) and then select the Edge Certificates tab (Figure 2.46).

Next, scroll down to “Always Use HTTPS” (Figure 2.47) and move the slider from “Off” to “On” (Figure 2.48).

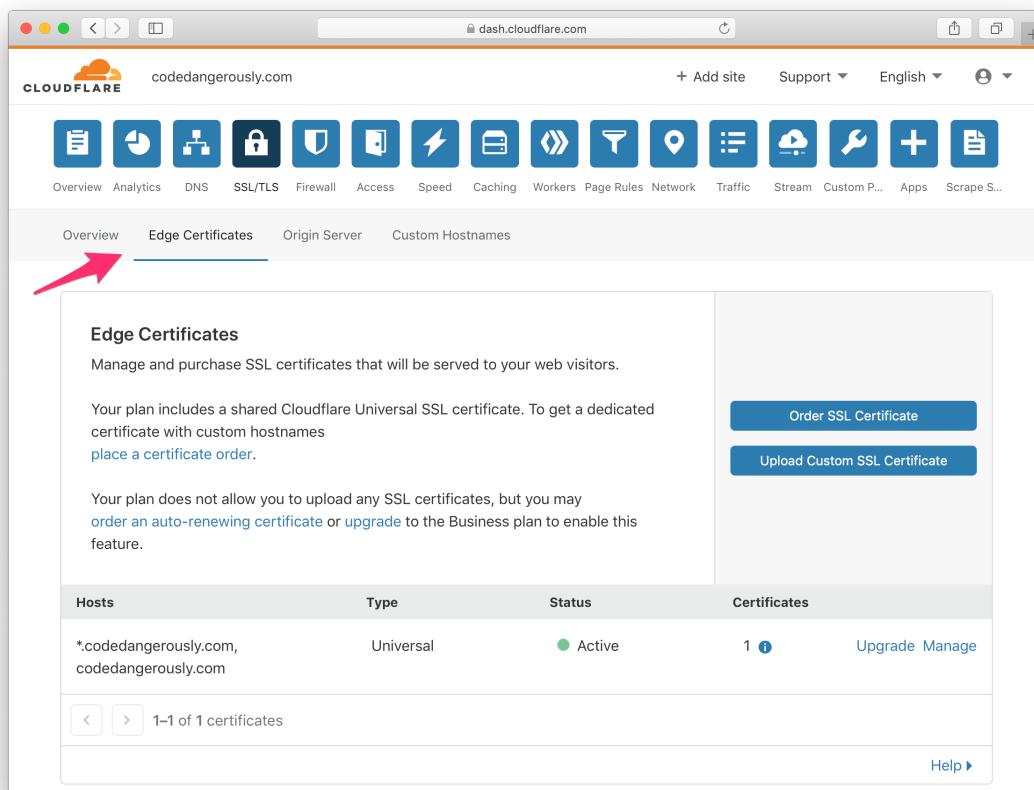


Figure 2.46: The Edge Certificates tab.

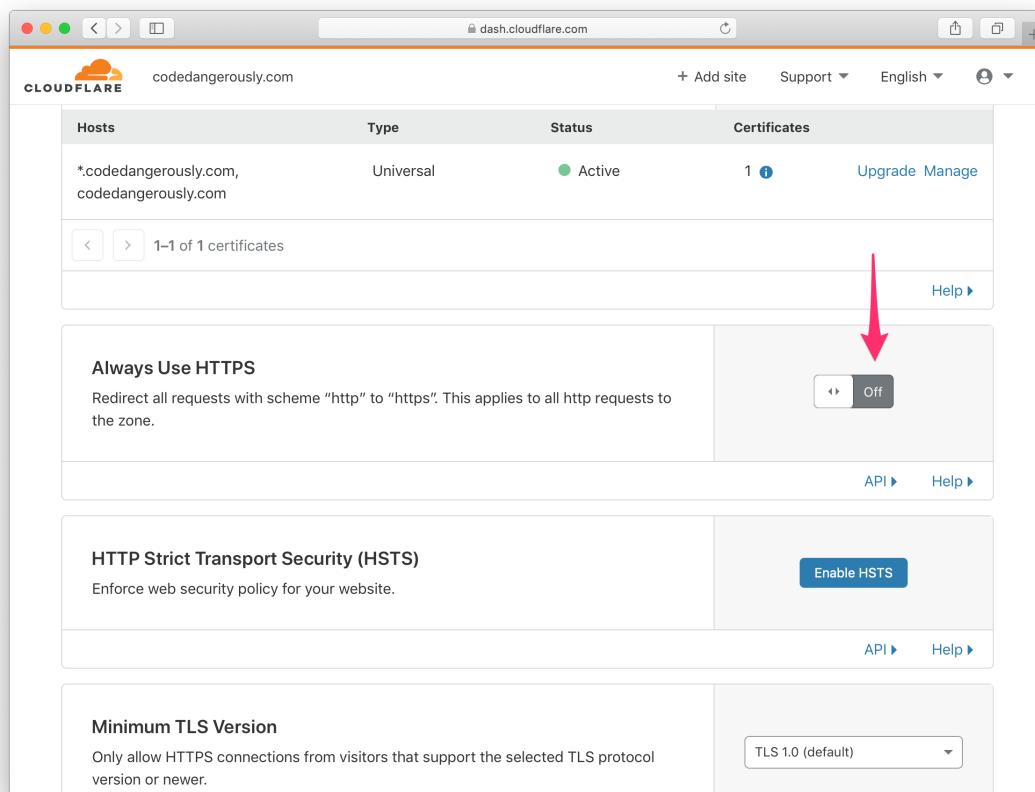


Figure 2.47: The “Always Use HTTPS” feature.

The screenshot shows the Cloudflare dashboard for the domain `codedangerously.com`. The top navigation bar includes links for '+ Add site', 'Support', 'English', and a user icon. Below the navigation is a table titled 'Hosts' with columns for 'Hosts', 'Type', 'Status', and 'Certificates'. A single entry is listed: `*.codedangerously.com, codedangerously.com`, which is a Universal host and currently Active, with one certificate. A red arrow points down from the status bar to the 'Always Use HTTPS' section.

Always Use HTTPS

Redirect all requests with scheme "http" to "https". This applies to all http requests to the zone.

This setting was last changed a few seconds ago

HTTP Strict Transport Security (HSTS)

Enforce web security policy for your website.

Enable HSTS

Minimum TLS Version

Only allow HTTPS connections from visitors that support the selected TLS protocol

TLS 1.0 (default)

Figure 2.48: Turning “Always Use HTTPS” on.

At this point, we're done! Cloudflare now forwards all requests for **http** URLs to the corresponding **https** versions (Figure 2.49):

```
1 $ curl --head http://www.codedangerously.com/
2 HTTP/1.1 301 Moved Permanently
3 Date: Thu, 13 Feb 2020 17:08:38 GMT
4 Connection: keep-alive
5 Cache-Control: max-age=3600
6 Expires: Thu, 13 Feb 2020 18:08:38 GMT
7 Location: https://www.codedangerously.com/
8 Server: cloudflare
9 CF-RAY: 564867ceafabd38e-LAX
```

As you can see on line 2, and as you might have guessed from the discussion in Section 2.4.1, HTTPS everywhere is enforced by a 301 permanent redirect (Box 2.1) from the **http** URL to the one with **https**.

2.5 Profit!!

Whether you got to this point after setting up a GitHub Pages site or a Heroku app, you now have an edge-cached site that protects you from DDoS attacks while making DNS changes easy and fast. Congratulations!

The only steps left, should you wish to follow them, are setting up email at your domain and adding analytics to collect data on your site's users. These are the subjects of Chapter 3.

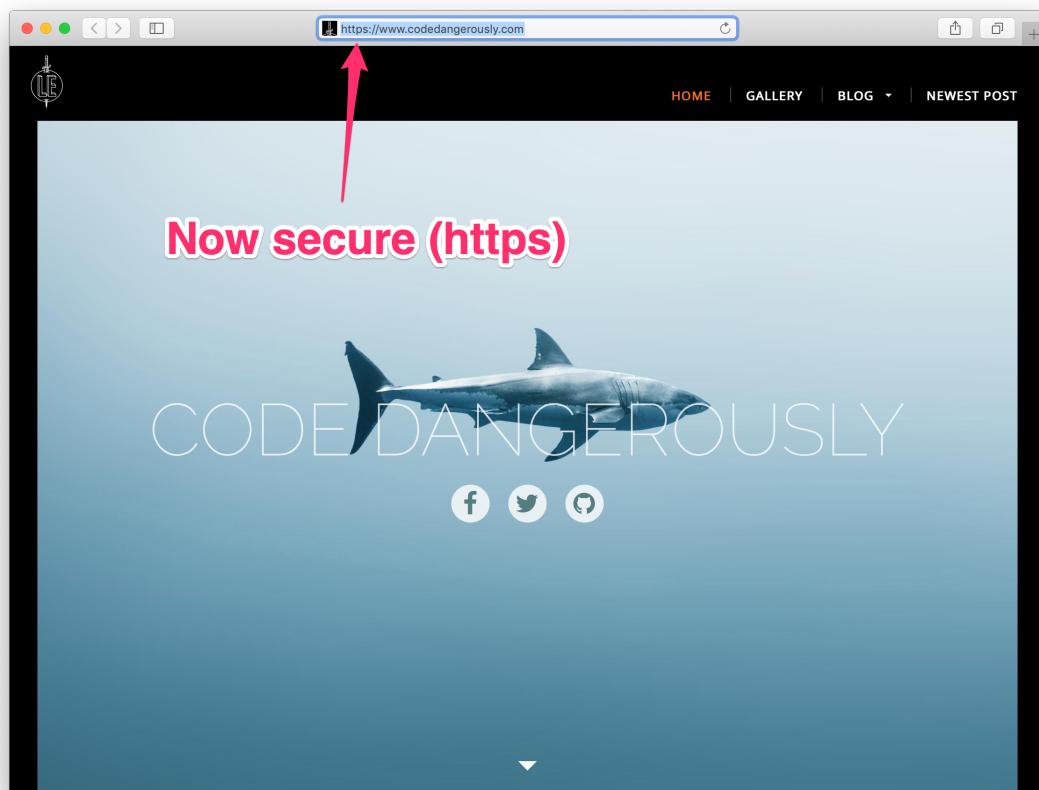


Figure 2.49: Automatically forwarding to **https**.

Chapter 3

Custom email and analytics

By the end of [Chapter 2](#), we've accomplished the main task we set out to do: we have our own website on a custom domain. In this chapter, we include two refinements: custom email ([Section 3.1](#)) and site analytics ([Section 3.2](#)). They're in the same chapter because the solutions we recommend are both provided by the same company (Google) and therefore fit nicely together.

3.1 Custom email

In this section, we'll first give an overview of how email works and then go into the details of using Google's [G Suite](#) to support custom email address at your domain. The end result will be the ability to receive email at `yourname@yourdomain.com`.

3.1.1 Introduction to email

Everyone uses email, but few know how it works. Ultimately, the system is surprisingly simple—email is really just plain text documents being sent back and forth with a little extra data needed to make things work.

The first real email message was sent by Ray Tomlinson in 1971 after he came up with an innovative way to format email addresses using the `@` sign. This



Figure 3.1: The two parts of an email.

allowed people to specify both the name and the domain for the destination for any given email (Figure 3.1).

Before Tomlinson came up with his addressing idea, there wasn't really any standardized way to send messages to people who were on a different network or computer (this was the time of **mainframes**, so often many people would share one giant room-sized computer). You could send messages to people on the same system as you, but not outside of it (Figure 2.39).¹

The convention of using the @ sign to denote the computer or network made email capable of being a universal messaging protocol. Eventually, groups of engineers came together to create a set of properties that every properly formatted email needed to have in order to be delivered—things like “to:” and “from:” fields. All that extra information is contained in what is called the *email header*. The message, and any other files like attachments, are a part of the *email body*.

When you send an email message, your properly formatted email is transferred as a text file to an outgoing email SMTP (Simple Mail Transfer Protocol) server. This is a software application that collects emails being sent by authorized users of that server and places them in a **queue**. The server then looks at each email's header to see what the destination is. Next, the SMTP server queries the DNS system to find the destination domain's *mail exchanger (MX) records*, which specify the mail server responsible for accepting email messages on that domain. When the location of the destination server is returned,

¹Image retrieved from <https://flic.kr/p/bu2gfG> on 2020-02-18. Copyright © 2012 by Robert Nix and used under the terms of the [Creative Commons Attribution 2.0 Generic](#) license.



Figure 3.2: Well Bob, if I'm reading this right, you'll be able to send messages alllllll the way over to Mary there. Let's call it 10ft.

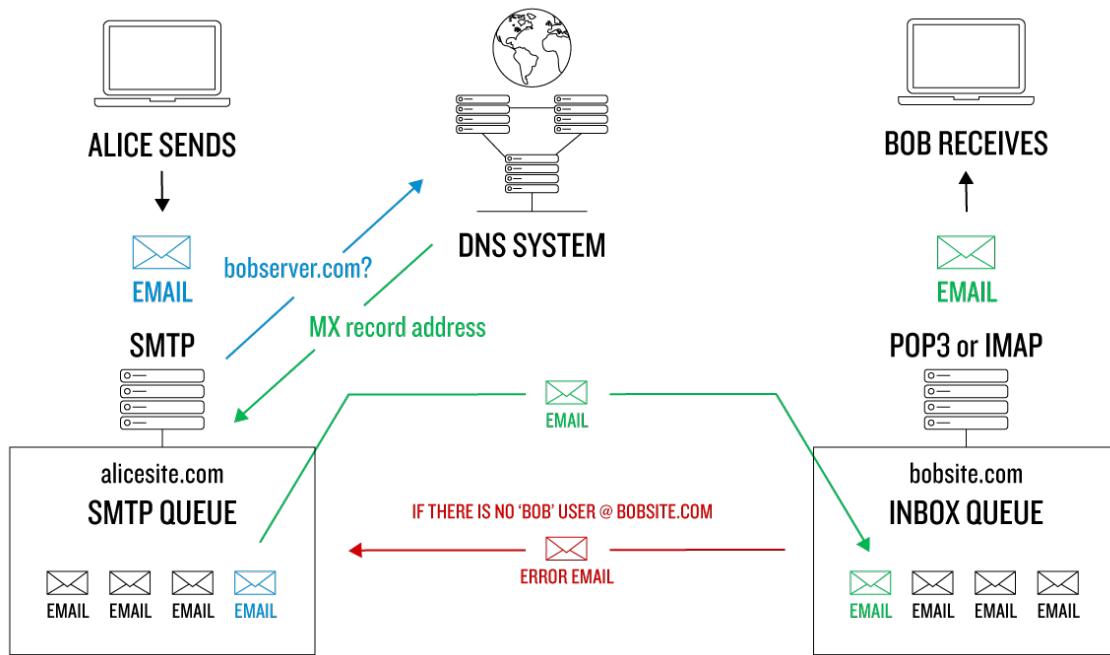


Figure 3.3: Sending an email between outgoing and incoming email servers.

the SMTP server opens up a connection to the receiving server and sends the file over the Internet (Figure 3.3).

It's certainly possible to run an email server on your own hardware, and then set your DNS for your domain to point messages to that server... but why would you want to give yourself the headache of having to deal with managing and updating hardware and software?

If you are running your own server, then more than likely you are going to be tied to a single machine that might go down or lose access to the Internet. That's to say nothing of the risk you run of having your servers hacked (there's an [American politician](#) you might be able to ask [how poorly that can go](#) (Figure 3.4)).²

It's a lot easier to use a global email service that can take care of all the security and uptime requirements, while also providing additional features like

²Image retrieved from <https://flic.kr/p/zQ8tnC> on 2019-11-20. Copyright © 2015 by jimthompson and used under the terms of the [Creative Commons Attribution 2.0 Generic](#)) license.



Figure 3.4: [No comment.](#)

advanced spam filtering. This is the option we recommend.

3.1.2 Google mail

The specific global email service we recommend is Google mail (Gmail). We use Gmail because, as one of Google's core consumer-facing businesses, there is support for Gmail on a large variety of devices and other services. It is also available and accessible basically anywhere at any time since it is running on Google's core infrastructure—that means you don't have to worry about some rinky-dink server outage taking down your access to email. Things have to go very wrong on the Internet to prevent you from being able to access Gmail.

Gmail's spam filtering is also top-notch, as they leverage both Google's expertise in machine learning and the millions of users that are on Gmail to figure out how to distinguish legitimate email from [spam](#). If you are one of the global few who never has had a Gmail account, and has had to deal with inadequate spam filtering, we can promise you that with Gmail you rarely end up with spam in your inbox, or with real messages in your spam folder. It feels



Figure 3.5: Spam is always full of lies.

like a solved problem (Figure 3.5).³

Lastly, we actually like the Gmail interface... We know that some people don't, but hey, you know what they say about opinions. We find that searching email on Google is always fast and accurate (something that we can't say about other systems that we've used in the past), and there are nice little power-user features that make life easier (like the ability to unsend a message within 30 seconds or the dots-and-pluses trick (Box 3.1)).

Box 3.1. Gmail dots and pluses trick

Many people don't know this, but when you have a Gmail email address, you actually have a basically infinite set of sub-email addresses that can tie into your account without needing to do any setup. This is a nice way to pre-filter your email by giving different people, or websites that you don't fully trust, a slightly different address. Then you can easily have Gmail move any mail sent to an email variant into a folder.

What do we mean? Let's say you have the email address `yourname@gmail.com`.

The first trick is that Gmail doesn't care about dots in your email address. So if you tell someone to email you at `your.name@gmail.com` that still will end up in the inbox for `yourname@gmail.com`—and you'll be able to see that it was sent to `your.name@gmail.com`. The lone caveat here is that you can't have

³Image retrieved from <https://flic.kr/p/8jBWuu> on 2020-02-18. Copyright © 2010 by GuillermoJM and used under the terms of the [Creative Commons Attribution 2.0 Generic](#) license.

two periods in a row. So `your..name@gmail.com` wouldn't work, but there would be nothing wrong with `y.o.u.r.n.a.m.e@gmail.com`. Eventually, though, you run out of variations...

The second trick is that you have at ability to create a limitless number of *ad hoc* emails by adding a plus sign, +, to the end of your regular email and then whatever additional text you desire.

In other words

`yourname+travelsite@gmail.com`

and

`yourname+shadysite@gmail.com`

both end up going to `yourname@gmail.com`. This lets you use distinct email addresses for different sites.

So what are the downsides of using Gmail for your custom domain? Well, unlike the free service (the one where your email is `something@gmail.com`), you have to pay for it. Google offers custom Gmail via its general suite of cloud productivity tools called [G Suite](#), which (as of this writing) costs \$6/user/mo. for the Basic plan. (Note that each user can have up to 30 *aliases*, or other email addresses that all go to the same inbox, so there's no need to pay for multiple users in this case.) For more details, see the [G Pricing page](#).

Some people have also expressed privacy concerns about Google's services. While this may apply to Gmail's consumer service, which serves ads based on email content, G Suite is different, as noted in Google's [FAQ](#):

Does Google use my organization's data in G Suite services or Cloud Platform for advertising purposes?

No. There are no ads in G Suite [Services](#) or Google Cloud Platform, and we have no plans to change this in the future. We

do not scan for advertising purposes in Gmail or other G Suite services. Google does not collect or use data in G Suite services for advertising purposes.

The process is different for our free offerings and the consumer space. For information on our free consumer products, be sure to check Google’s [Privacy and Terms](#) page for more consumer tools and information relating to consumer privacy.

There’s an old axiom that applies here: “If you aren’t paying for a product or service, then you are the product.” (For example, Facebook is “free”, but their product is your personal information, which they use for ad targeting.) With Gmail for work, you *are* paying, so you are *not* the product.

If you’d really rather use an alternate service (such as privacy-focused [ProtonMail](#)), the setup is going to end up being similar to what we cover here. This means that the steps discussed in this section will be useful in any case.

Let’s get started!

3.1.3 G Suite signup

Before we start, a little caveat: Some of these screenshots might not be accurate to your sign up experience simply because of the way that Google constantly tests different interfaces for their sites. We visited the site one day to go through all the steps and create an account, and then the next day went back to double check a step in the process only to find that the sign up flow was entirely changed. They still wanted the same information, everything was just arranged totally differently! There’s simply no substitute for applying your [technical sophistication](#).

To sign up for G Suite, head over to the [G Suite home page](#) and get started by clicking the “Get Started” button... or whatever the equivalent is today. Again, Google might be testing different language for buttons—who knows?

The first thing that you’ll be asked about is the name and size of your business ([Figure 3.6](#)). (If this is just a personal website, your “business” is just you.)

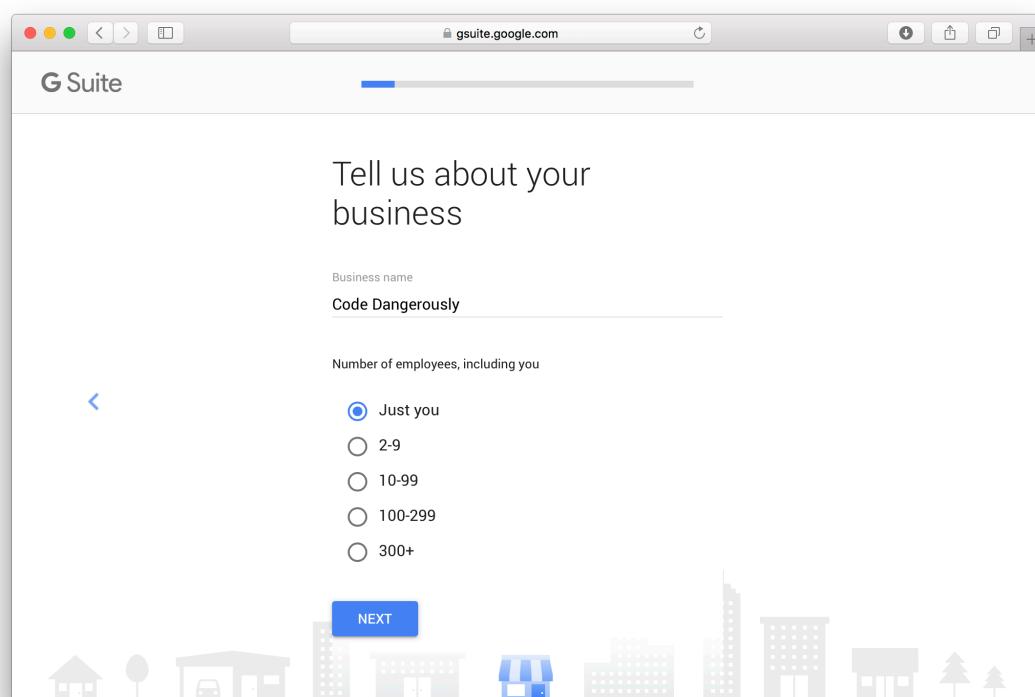


Figure 3.6: Submitting basic information about your “business” (possibly just you).

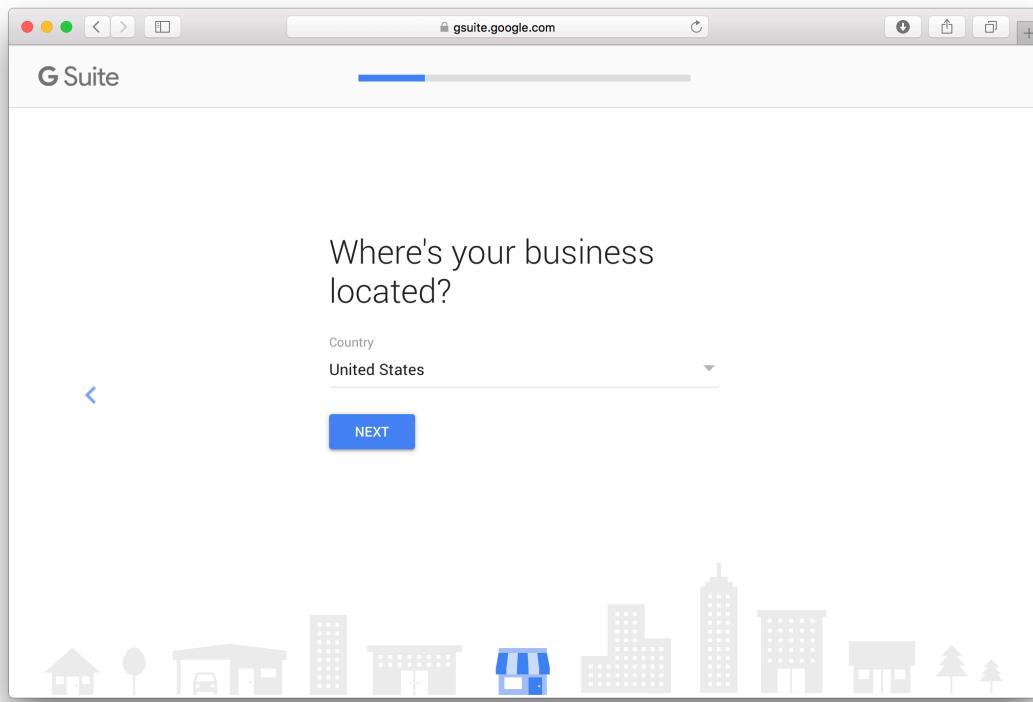


Figure 3.7: Adding a “business” location.

Then you’ll be asked about the location (we’re guessing to help Google localize what part of the world they should store your emails in). Select your location and hit “Next” ([Figure 3.7](#)).

Next you’ll be asked to supply your current email address as a backup way to get in touch with you ([Figure 3.8](#)).

If you didn’t already have a domain name to work with, you’d be able to buy one through Google at this point... but you’ve already got your sweet sweet domain, so hit the “Yes, I have one” button and move on ([Figure 3.9](#)).

Then add your domain name ([Figure 3.10](#)).

Now you are going to be moving on to creating your user account, so Google wants to know your name. This first user is going to be the administrator (admin) for the account ([Figure 3.11](#)).

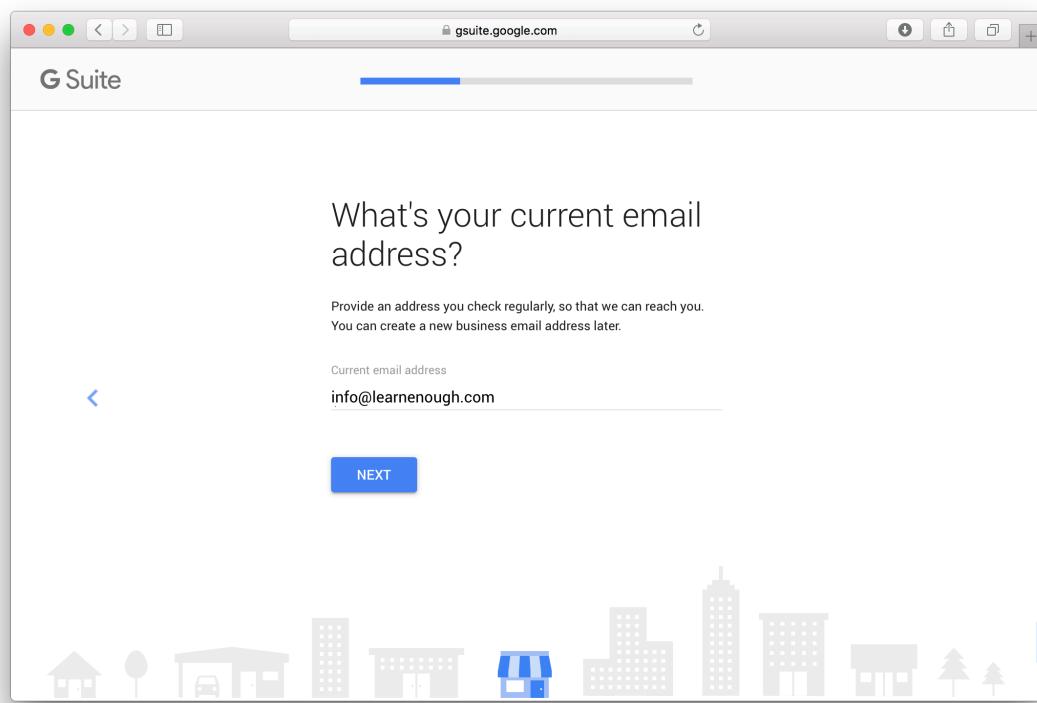


Figure 3.8: Adding a backup email address.

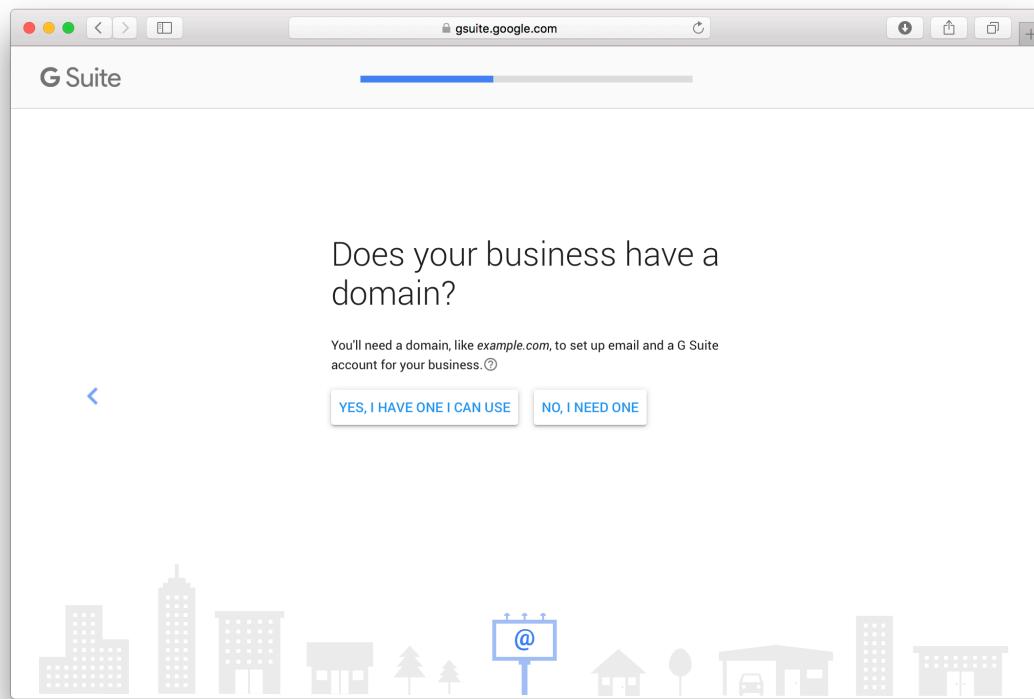


Figure 3.9: You can buy a domain through Google, but you already have one..

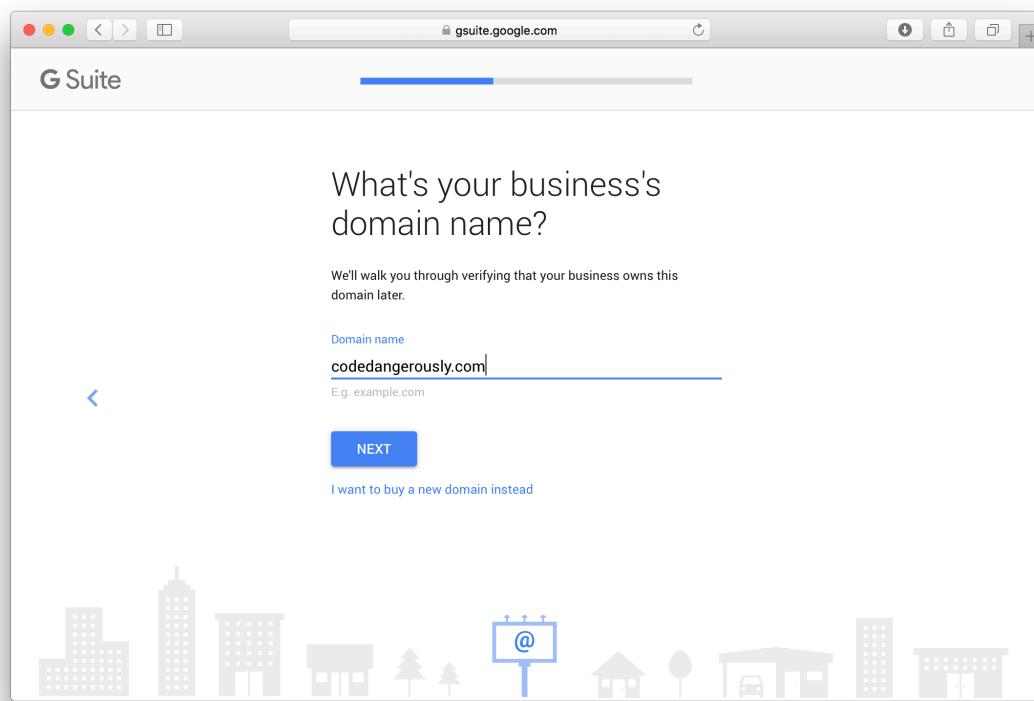


Figure 3.10: Inputting your custom domain.

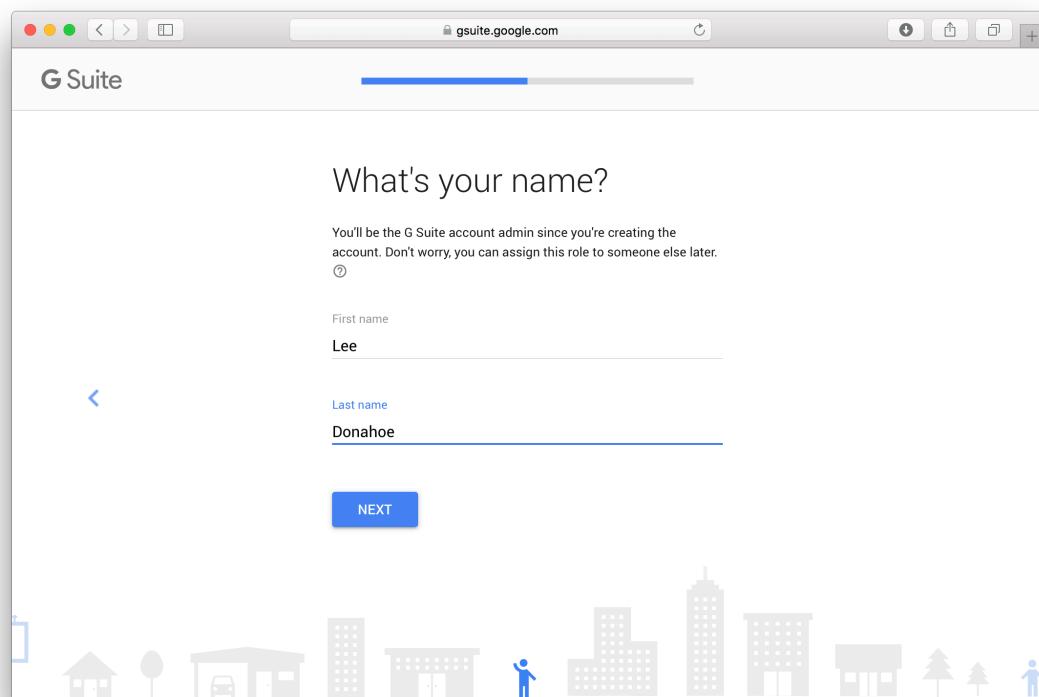


Figure 3.11: Letting Google know who you are!

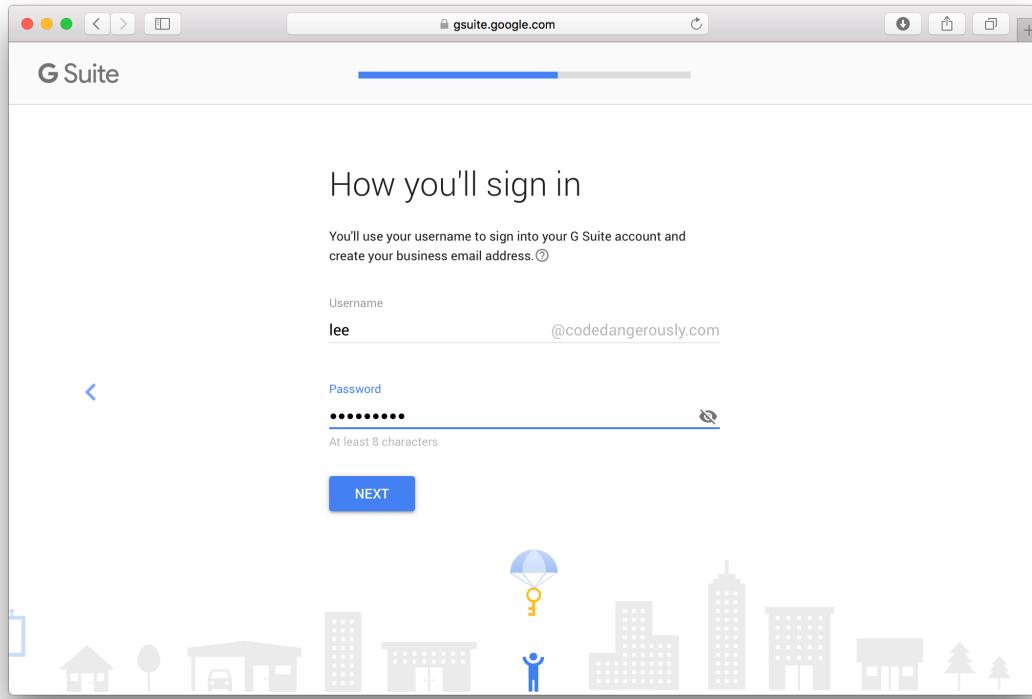


Figure 3.12: First user account creation.

Next, you are going to actually add the user name that will be your email address, and the password that you'll use to log in to Gmail. Pick a good one so you don't get hacked (Figure 3.12).

At this point, you might be asked whether or not you want to receive special offers from Google. If you want that, then say yes, otherwise, choose "No thanks" and then move on to paying for the account. Once you've filled out all your payment info, click the button to agree to the terms and create your account (Figure 3.13).

Once your account is created, you should see a welcome page with a single button that says "Continue to Setup". It would be a good idea to click that! You will be taken to your admin console and presented with yet another popup; click the "Next" button and you'll finally be kicked over to an admin page where your

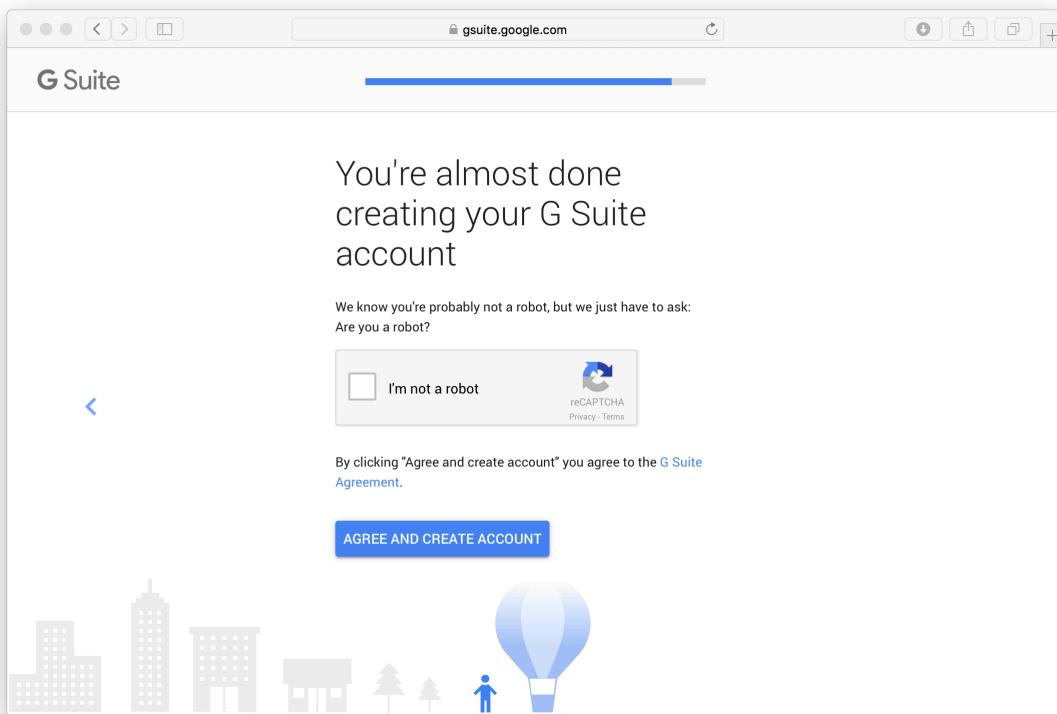


Figure 3.13: The last part of account creatio... unless you are a robot.

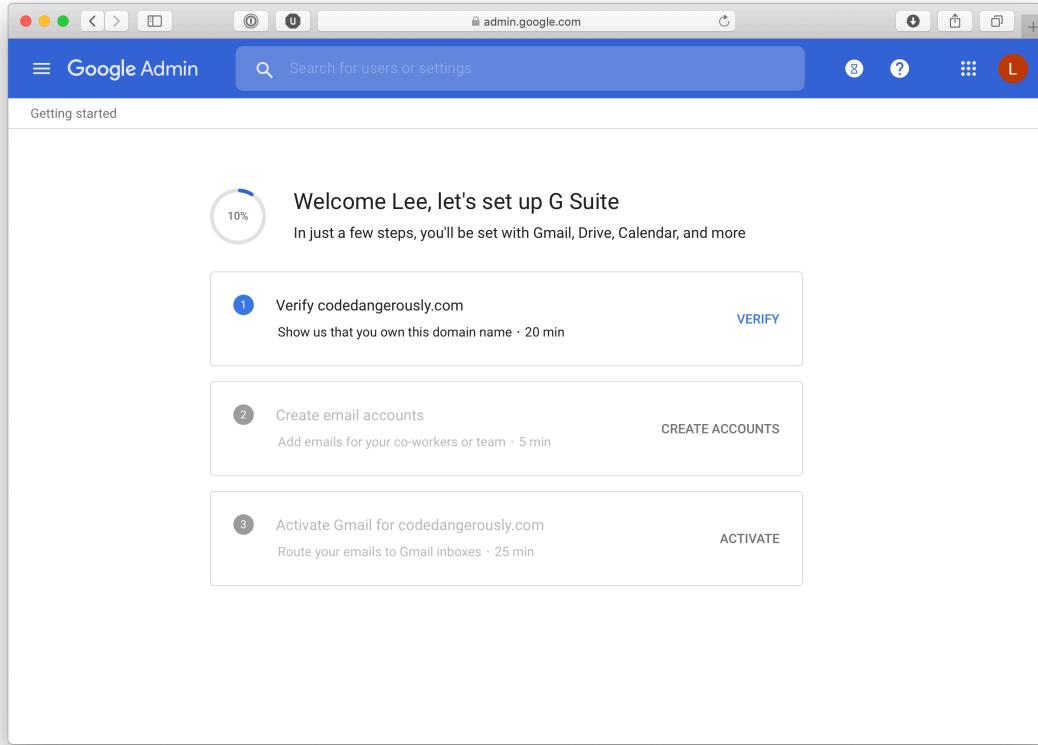


Figure 3.14: Finally arriving the admin console.

only option is to start the verification process (Figure 3.14).

Now that the initial setup is done, Google will want you to make changes to your DNS records (Chapter 2) so that Google can both verify that you actually own the domain and add the correct MX records (Section 3.1.1). That sounds more complicated than it is, but thankfully by now you are an expert at editing DNS records!

3.1.4 Verify domain and configure MX records

As mentioned above, Google wants to make sure that you are in fact the owner of the domain that you are trying to create an email account for, and luckily the

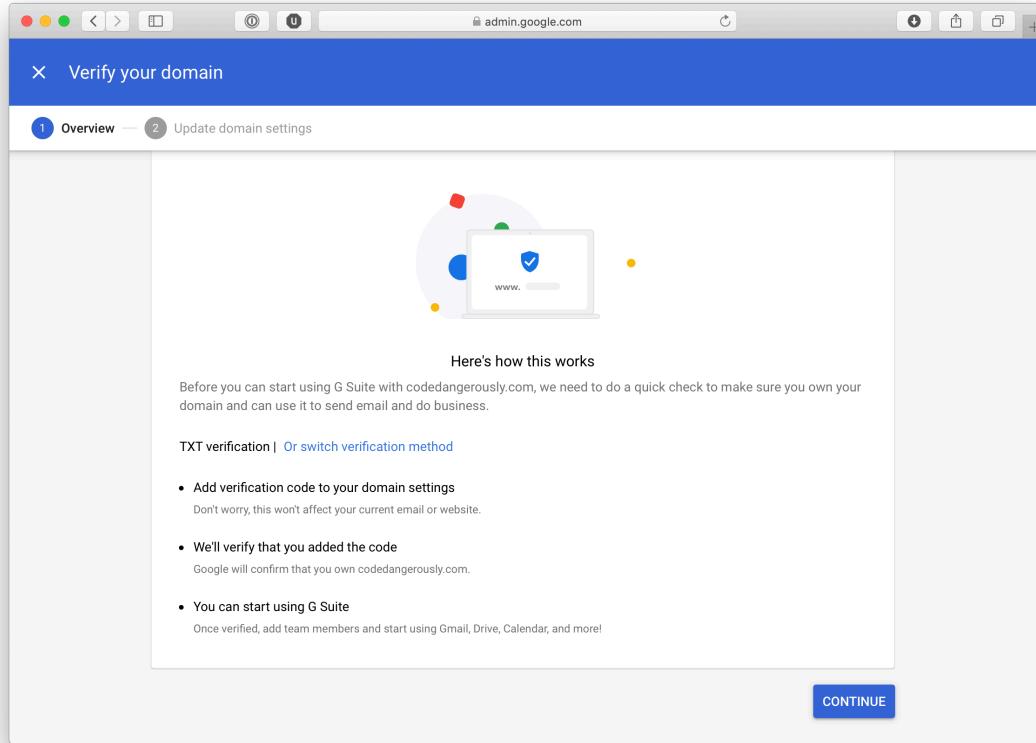


Figure 3.15: Google’s instructions for setting up the domain verification.

DNS system provides a lot of opportunities to add information to a domain’s records. For Gmail, Google just wants you to add an extra TXT record (which is what it sounds like: a text record on your site’s DNS configuration) consisting of a unique string of letters and numbers (Figure 3.15).

Because of the public nature of the DNS records, Google can just query your domain’s information and easily see that the verification strings match, confirming that you are in fact the person who controls the domain. If you haven’t already, click past the verification-instructions welcome page to get to the details on how to make Google happy. Copy the verification text on the page (Figure 3.16).

In another tab or window, open up your site’s DNS records in Cloudflare,

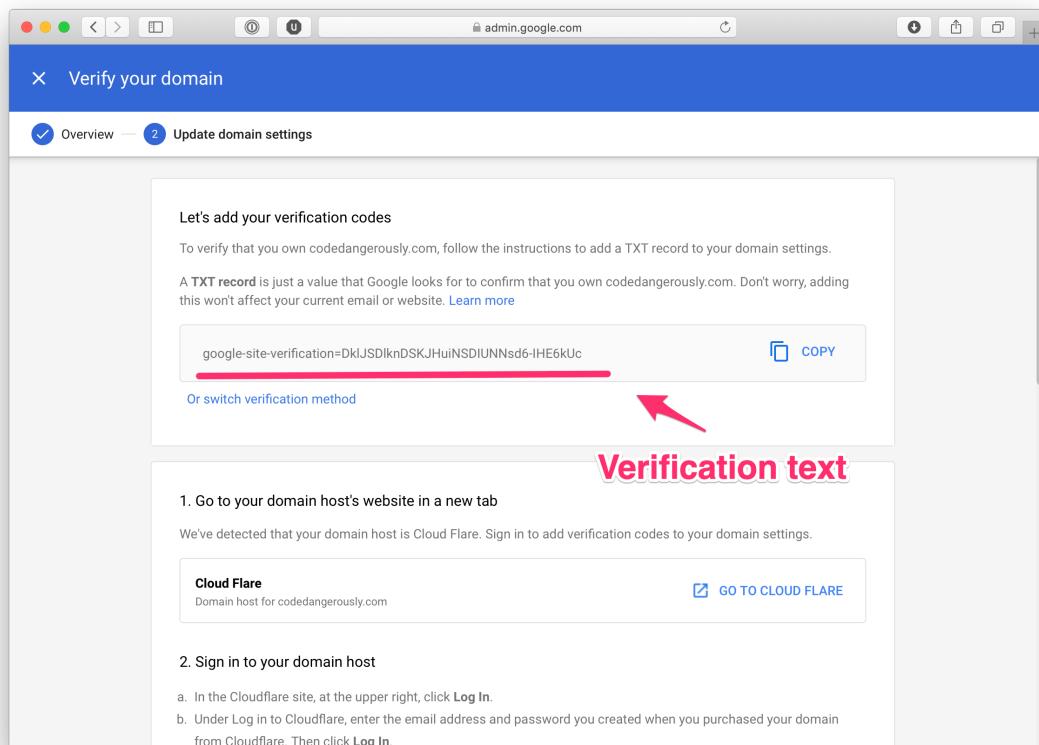


Figure 3.16: The TXT record verification string (not our real one!).

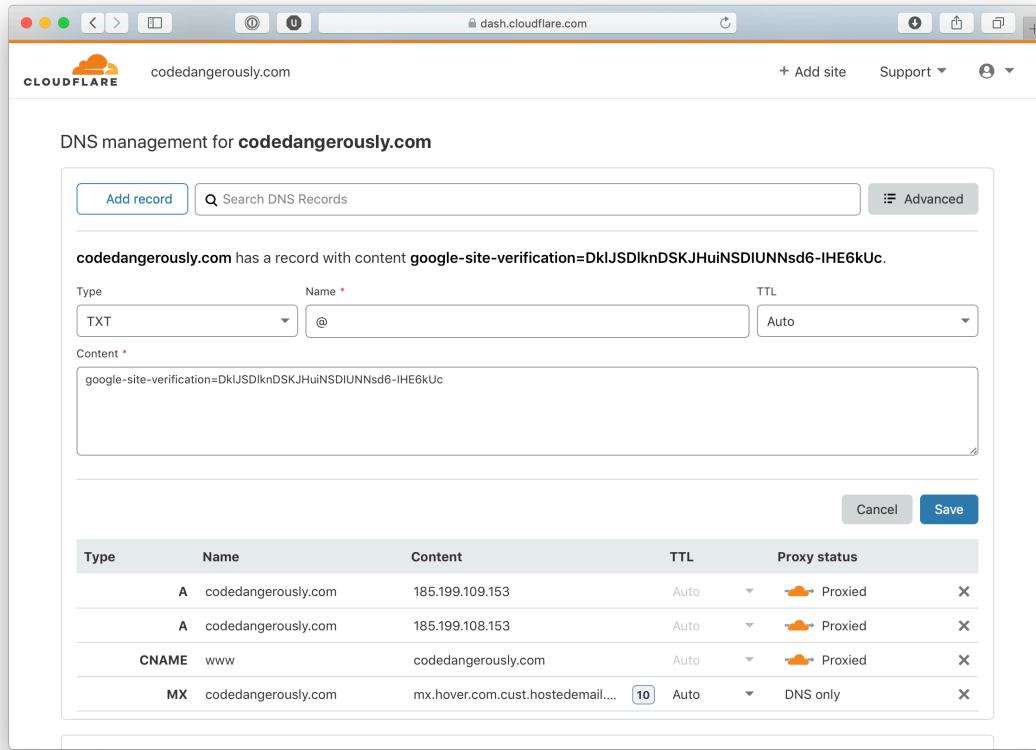


Figure 3.17: Adding the new TXT record.

click the “+Add new record” button and select **TXT** from the new record dropdown. In the “Name” field, add in an **@** sign, which is a shorthand way of saying “use whatever the root domain is for this record”. (In other words, for us **@** is the same as **codedangerously.com**.) Click into the “Content” field and paste in your verification text as shown in Figure 3.17 (still not our real verification key).

After hitting the “Save” button, your updated DNS settings should look like Figure 3.18.

Finally, delete any old MX records that not pointing to the Google servers (Figure 3.19).

That’s all that’s needed on Cloudflare for the moment, but we still haven’t

The screenshot shows the Cloudflare DNS management interface for the domain `codedangerously.com`. A new TXT record is being created with the following details:

Type	Name *	Content	TTL	Proxy status
TXT	Use @ for root		Auto	DNS only
Content *				
<input type="text"/>				

Below the table, the current DNS records for the domain are listed:

Type	Name	Content	TTL	Proxy status
TXT	codedangerously.com	google-site-verification=Tn3NwMaF...	Auto	DNS only
A	codedangerously.com	185.199.109.153	Auto	Proxied
A	codedangerously.com	185.199.108.153	Auto	Proxied
CNAME	www	codedangerously.com	Auto	Proxied
MX	codedangerously.com	mx.hover.com.cust.hostedemail... 10	Auto	DNS only

At the bottom, there is a section titled "Cloudflare nameservers".

Figure 3.18: TXT record added to your domain.

The screenshot shows the Cloudflare DNS management interface for the domain `codedangerously.com`. At the top, there's a search bar labeled "Search DNS Records" and an "Advanced" filter button. Below the search bar is a table of DNS records:

Type	Name	Content	TTL	Proxy status
MX	codedangerously.com	mx.hover.com.cust.hostedmail	10	Auto DNS only
A	codedangerously.com	185.199.108.153	Auto	Proxied
A	codedangerously.com	185.199.109.153	Auto	Proxied
CNAME	www	codedangerously.com	Auto	Proxied
TXT	codedangerously.com	google-site-verification=Tn3NwMaF...	Auto	DNS only

Below the table, there's a section titled "Cloudflare nameservers" with two entries:

Type	Value
NS	vern.ns.cloudflare.com
NS	zara.ns.cloudflare.com

At the bottom, there's a section titled "Custom Nameservers" with a note: "Custom nameservers allow you to create your own, unique nameservers in place of your". A large red arrow points from the right side of the screen towards the "Delete me!" button at the top of the page.

Figure 3.19: See, there's an arrow that says “delete me”—what other motivation do you need?

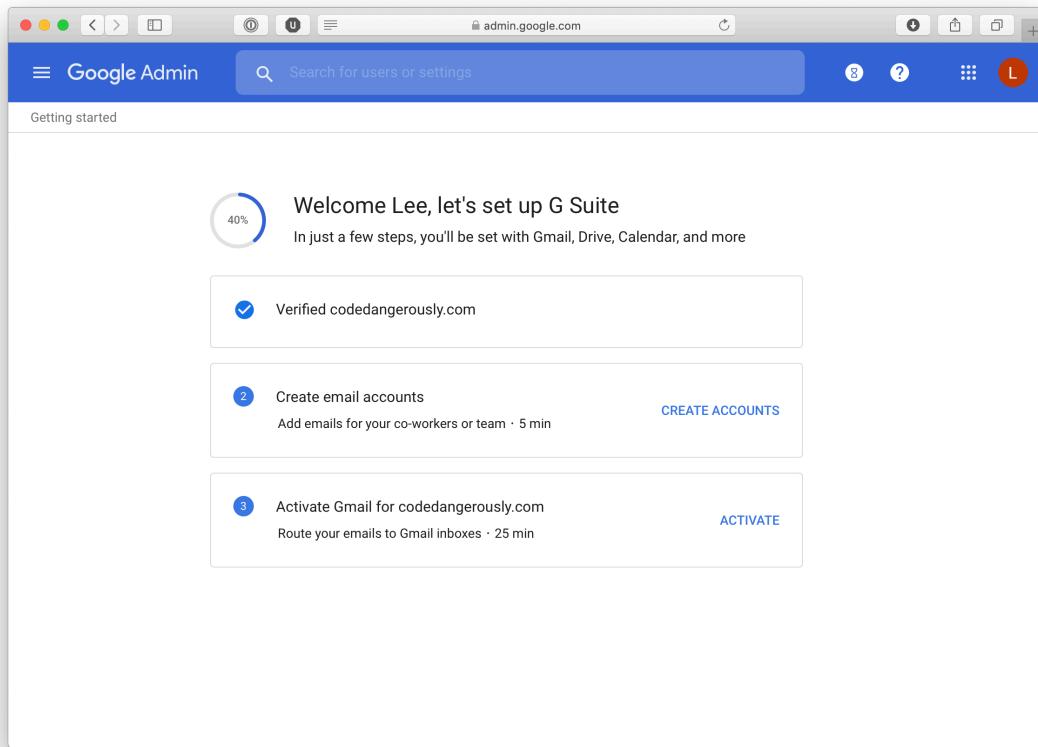


Figure 3.20: The admin panel for a verified domain.

set up the actual DNS records to handle email. Hop back over to your G Suite admin console tab and click the “Verify My Domain” button at the bottom of the page.⁴ When the verification has been processed, your admin panel should look something like [Figure 3.20](#).

If you are working with any other people on your project, you can use step 2 in the admin console setup process to add other users to your G Suite account. It is a pretty self-explanatory process, and if you don’t want to deal with it now, you don’t need to.

If you are ready to get your email up and running, click the “Activate” link

⁴If we weren’t using Cloudflare, the changes could theoretically take up to 72 hours to go into effect—which is all the more reason to use Cloudflare.

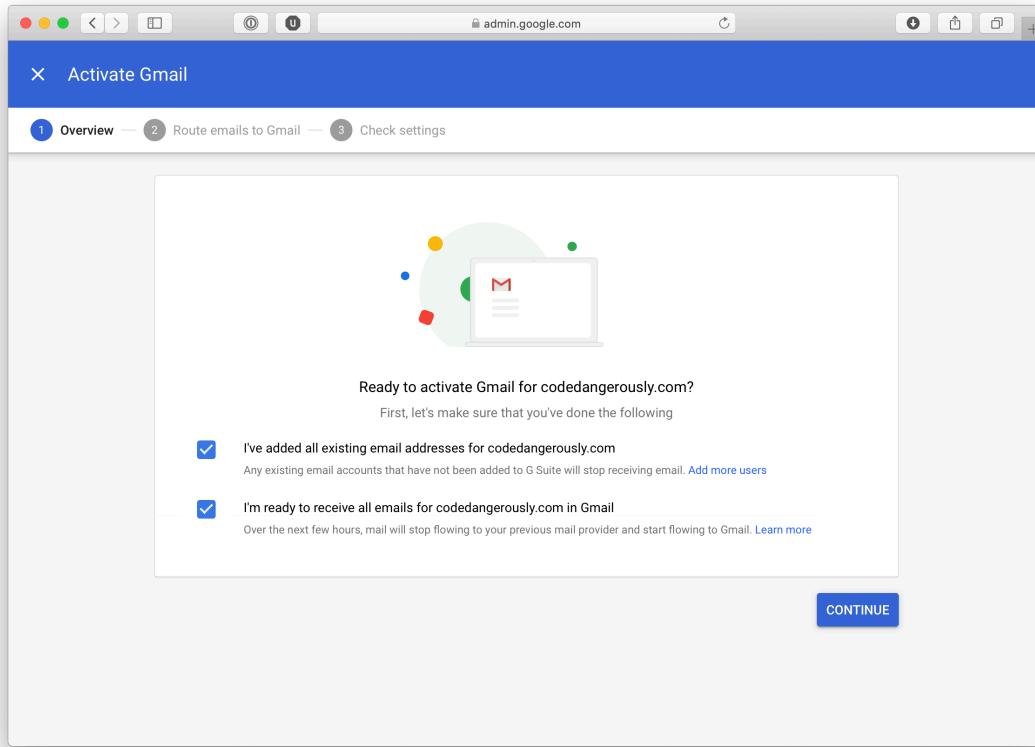


Figure 3.21: So many confirmations steps.

and to start finishing things up. On the next page, check both of the boxes to let Google know you are done with setup, and then click “Continue” (Figure 3.21).

The site will now take you to a page with instructions that detail how to set up your DNS records so that email gets routed to the correct mail servers. To do this you will be adding a handful of MX records to your DNS settings on Cloudflare. The information for the records can be found towards the bottom of the page (Figure 3.22).

Let’s add the first MX record at Cloudflare. Copy the first value from the G Suite setup, which for us is **ASPMX.L.GOOGLE.COM**. Then, on the Cloudflare DNS management page, perform the following steps:

- Add a new record, making sure that **MX** is selected in the “Type” drop-

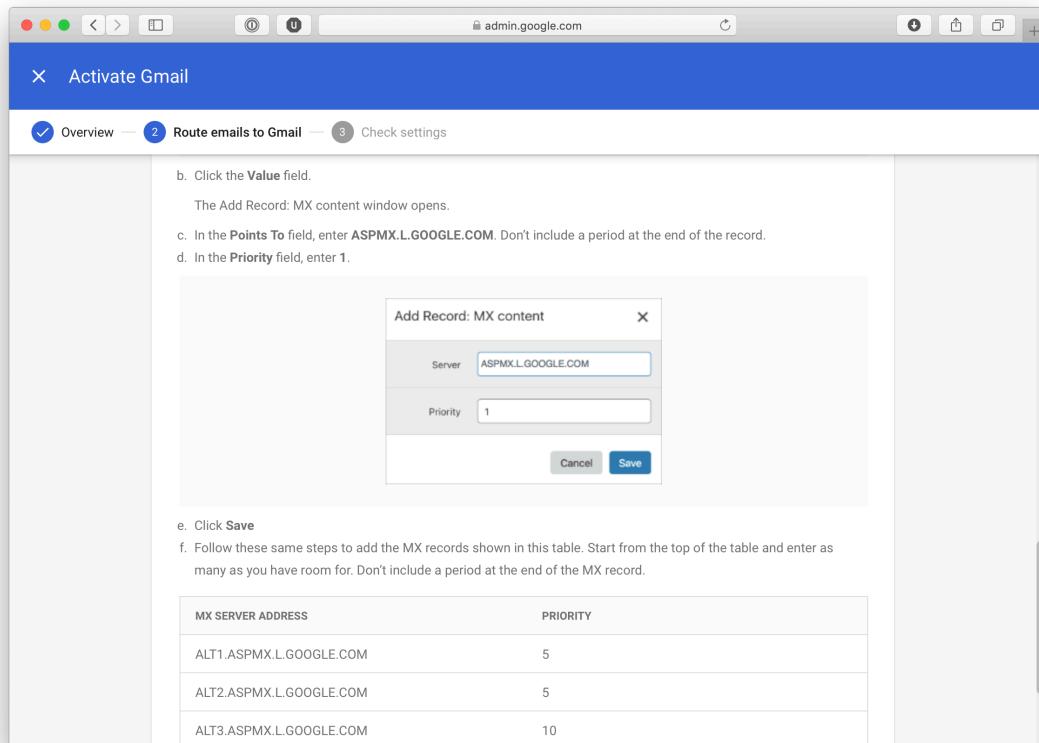


Figure 3.22: Lots of details, and not the easiest thing to read.

Type	Name	Content	TTL	Proxy status
A	codedangerously.com	185.199.108.153	Auto	Proxied
A	codedangerously.com	185.199.109.153	Auto	Proxied
CNAME	www	codedangerously.com	Auto	Proxied
TXT	codedangerously.com	google-site-verification=Tn3NwMaF...	Auto	DNS only

Figure 3.23: Adding the first Gmail MX record in Cloudflare.

down.

- Add @ in the “Name” field so that the record points to the domain root.
- Add the server name that you copied from G Suite into the “Mail server” field.
- For this first record, set a priority of 1.

If your record looks like Figure 3.23, go ahead and click “Save”.

Now head back over to your G Suite setup tab, and then bounce back and forth between that and Cloudflare to finish entering the other MX records listed in the table, making sure that you are setting the priority on each one to the

The screenshot shows the Cloudflare DNS management interface for the domain 'codedangerously.com'. At the top, there's a search bar labeled 'Search DNS Records' and an 'Advanced' button. Below that, a note says '[mail server] handles mail for [name]'. A form allows adding a new record with fields for Type (MX), Name (e.g., 'Use @ for root'), Mail server (e.g., 'alt4.aspmx.l.google.com'), Priority (e.g., 10), and TTL (Auto). A table below lists existing records:

Type	Name	Content	TTL	Proxy status
MX	codedangerously.com	alt4.aspmx.l.google.com	10	Auto DNS only
MX	codedangerously.com	alt3.aspmx.l.google.com	10	Auto DNS only
MX	codedangerously.com	alt2.aspmx.l.google.com	5	Auto DNS only
MX	codedangerously.com	alt1.aspmx.l.google.com	5	Auto DNS only
MX	codedangerously.com	aspmx.l.google.com	1	Auto DNS only
A	codedangerously.com	185.199.108.153	Auto	Proxied
A	codedangerously.com	185.199.109.153	Auto	Proxied
CNAME	www	codedangerously.com	Auto	Proxied
TXT	codedangerously.com	google-site-verification=Tn3NwMaF...	Auto	DNS only

Figure 3.24: All the records successfully added.

specified level (1, 5, or 10 in Figure 3.22). When you are done, your DNS records on Cloudflare should look like Figure 3.24.

When you are done, click the “Activate Gmail” button and let Google check to see if the DNS records have been correctly added (Figure 3.25).

You are all done! At this point, Google will take you to a confirmation page to let you know that your G Suite account is set up. If you want to check out your new Gmail account, click the Google Apps menu icon at the top right, and then select Gmail (Figure 3.26).

In the future, you can access your new email inbox by going to mail.google.com and signing in with the new account that you created during the mail setup process. If you need to get back to the console and can’t remember what the

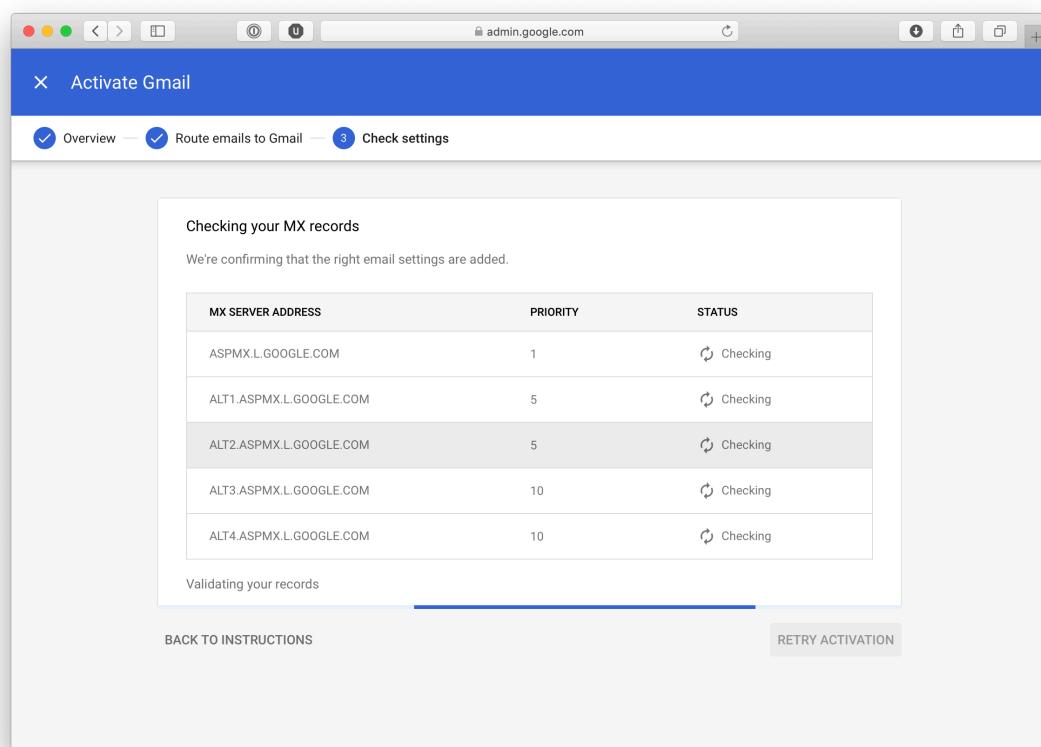


Figure 3.25: All the records successfully added.

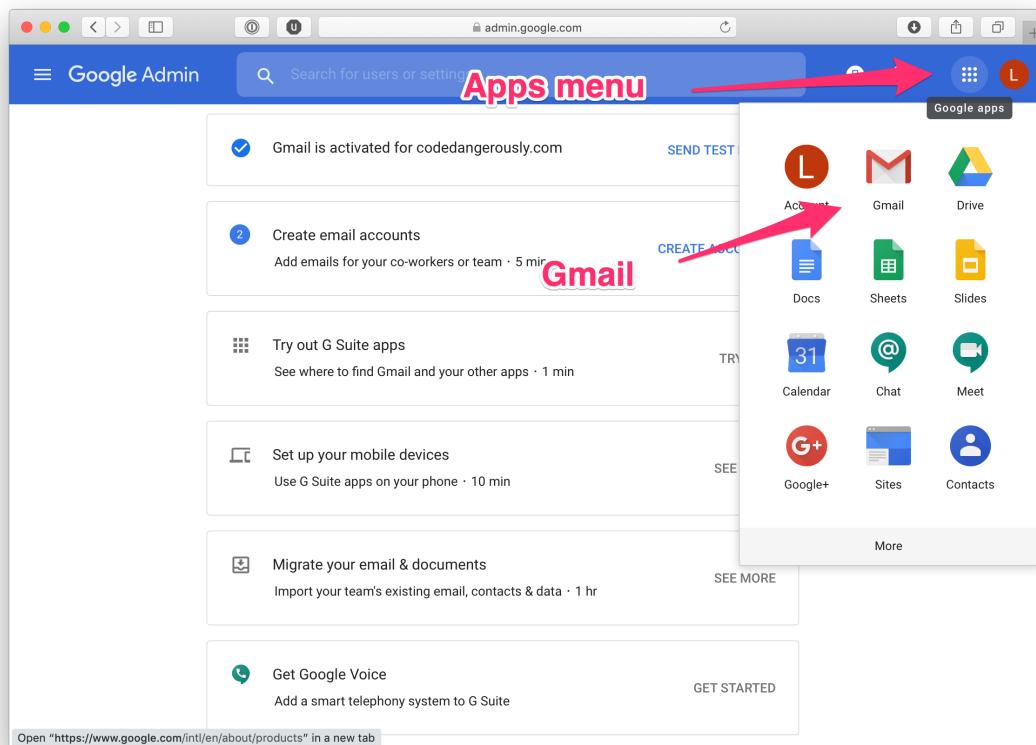


Figure 3.26: It's all done, done.

address is, it's super-easy: admin.google.com. Once you've logged in there, you can add additional users to your account, update billing information, or add additional services.

Speaking of additional services... since you have a Google account now, why not sign up for Google Analytics so that you can get information about who is visiting your site?

3.2 Site Analytics

Google Analytics is probably the most useful free service that Google has for the owners of websites and web applications. By adding just a tiny little snippet of code to your site, you can have Google track not just how many people are visiting your site, but also the browser and operating system they are using, whether they are on desktop or mobile, how they found your site, how long they stayed on your site, which pages they viewed, which page they left your site from... and a lot more.

To get started, visit the [Google Analytics home page](https://www.google.com/analytics) and click the big “Sign Up for Free” button. (As always, use your technical sophistication if Google has changed their interface since we wrote this.) From there, you'll be asked to sign in using the Google account that you just created (and if you have other Google accounts they'll show up too [Figure 3.27](#)).

Once you are in, you'll be on the Analytics set-up track. Click the “Get Started” or “Sign Up” button to go to the site information page. Fill out the fields on the page with the proper information from your site as in [Figure 3.28](#), and then at the bottom of the page press the “Get Tracking ID” button.

If all your information has been entered correctly, you should see a pop-up that asks you to agree to the Terms of Service, so hit the “I Accept” button. That will take you to a page in your new analytics account that will have the little bit of code you need to add to your site in order for the tracking to work ([Figure 3.29](#)).

What you are looking for is the code that looks like [Listing 3.1](#).

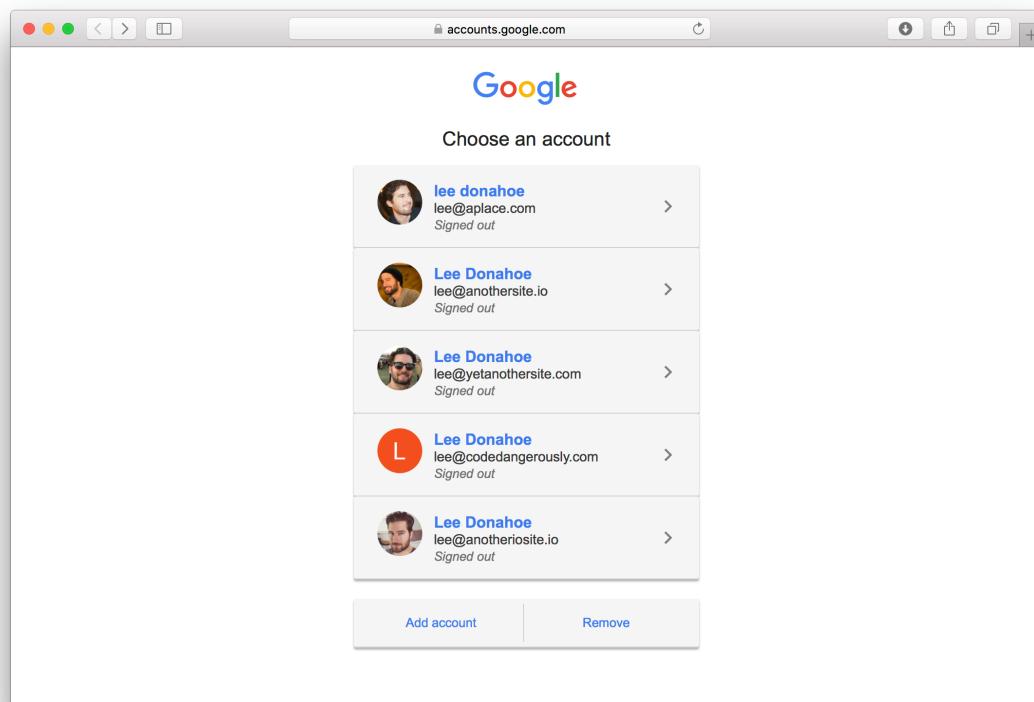


Figure 3.27: The sign in page can get pretty crowded if you have a lot of Google accounts.

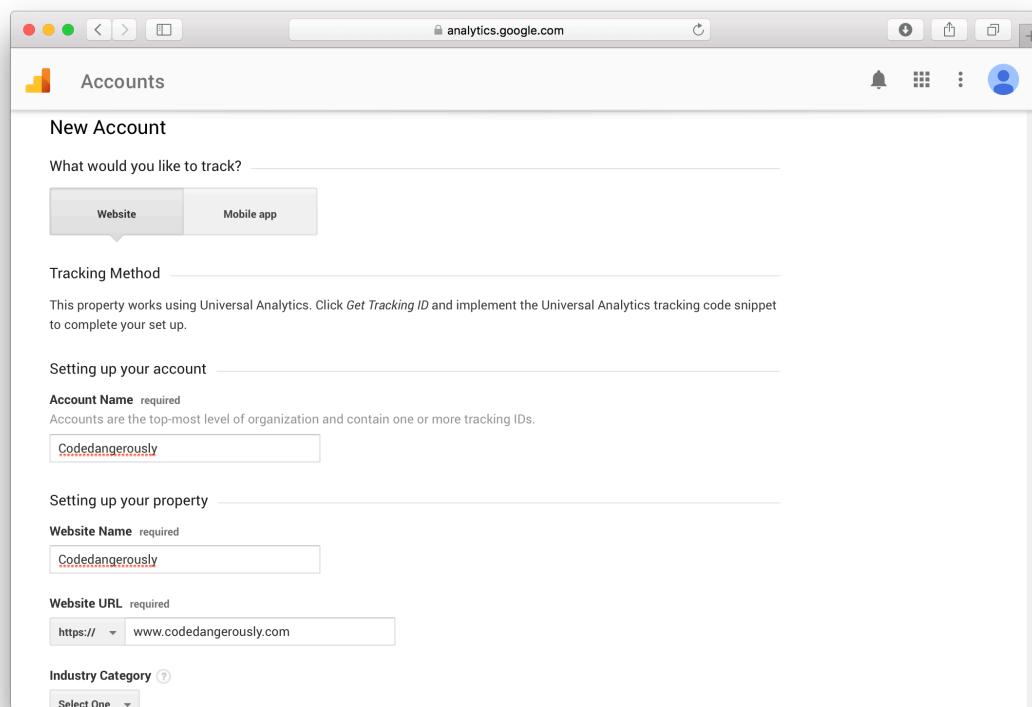


Figure 3.28: Filling out the required information for your analytics account.

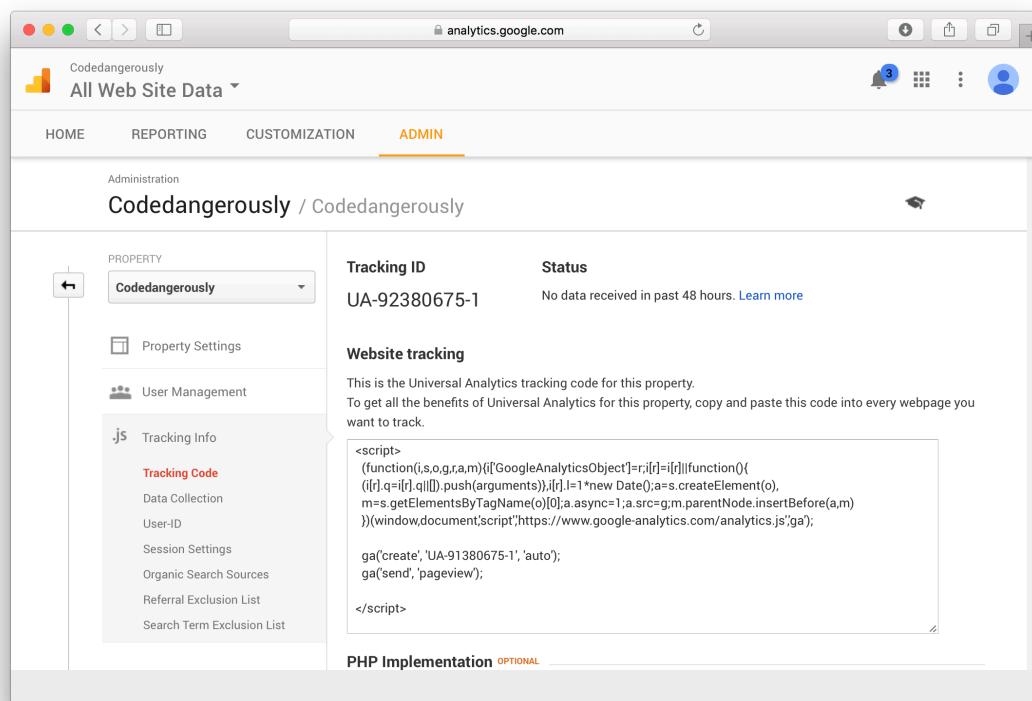


Figure 3.29: The tracking code page for your site.

Listing 3.1: This is the code that makes Analytics work.

```
<script>
(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
(i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
})(window,document,'script','https://www.google-analytics.com/analytics.js','ga');
ga('create', 'UA-XXXXXXX-1', 'auto');
ga('send', 'pageview');
</script>
```

(We've removed one level of indentation from the script to get it to fit on the page, but you can just copy-and-paste from Google.) Note that you should always use the exact code from your version of Figure 3.29; Listing 3.1 is included only for reference.

3.2.1 Add Snippet

One option is to paste the snippet from Figure 3.29 on every page of your site, but in general this is rather inconvenient. Instead, we're going to assume that you are using some sort of templating framework like Jekyll (as covered in *Learn Enough CSS & Layout to Be Dangerous*) or Ruby on Rails (as covered in the *Ruby on Rails Tutorial*).

Whichever system you are using, you should paste the analytics snippet into the `<head>` section of your site. In *Learn Enough CSS & Layout to Be Dangerous*, this section is located in `_includes/head.html`, as shown in Listing 3.2.

Listing 3.2: Where to put the code snippet in your site code.

```
_includes/head.html

<head>
  {% if page.title %}
    <title>{{ page.title }} | Test Page</title>
  {% else %}
    <title>Test Page: Don't Panic</title>
  {% endif %}
  {% if page.description %}
    <meta name="description" content="{{ page.description }}">
```

```
{% else %}
    <meta name="description" content="This is a dangerous site.">
{% endif %}
<link href="/favicon.png" rel="icon">
<meta charset="utf-8">
<meta name=viewport content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="/fonts/font-awesome-4.7.0/css/font-awesome.min.css">
<link rel="stylesheet" href="/css/main.css">

<script>
(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
(i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
})(window,document,'script','https://www.google-analytics.com/analytics.js','ga');
ga('create', 'UA-XXXXXXXXX-1', 'auto');
ga('send', 'pageview');
</script>
</head>
```

Save your changes, deploy your site... and that's it! If you want force Google Analytics to push some test visitors to the site and make sure everything is working, you can click the “Send test traffic” button ([Figure 3.30](#)).

If you've looked for information on how to set up Analytics in the past, you might have seen someone suggest that you should put the Google Analytics code snippet at the very end of your site right above the closing `</body>` tag. You can absolutely do that, and the analytics will still work, but the main reason for doing that has changed.

It used to be the case that when your site tried to load Analytics it would actually hold up the rest of your site from loading until the Analytics code finished downloading—not a good thing if the connection to Google is slow. Since then though, the Google Analytics code has switched to using an *asynchronous loading* method that makes it so that the rest of your site load isn't held up.

That said... of course there is a caveat. With Google Analytics, you can add in a bunch of custom tagging and tracking functionality—things that require parts of your site to load first to set up tracking code. If you are using a lot of custom Analytics tracking, then it might be a good idea to move the Analytics initialization to the bottom of your page template above the closing `</body>` tag.

Once you are all set up, you can sign into the Analytics dashboard by going to analytics.google.com, and after you click on the link that says “All Website

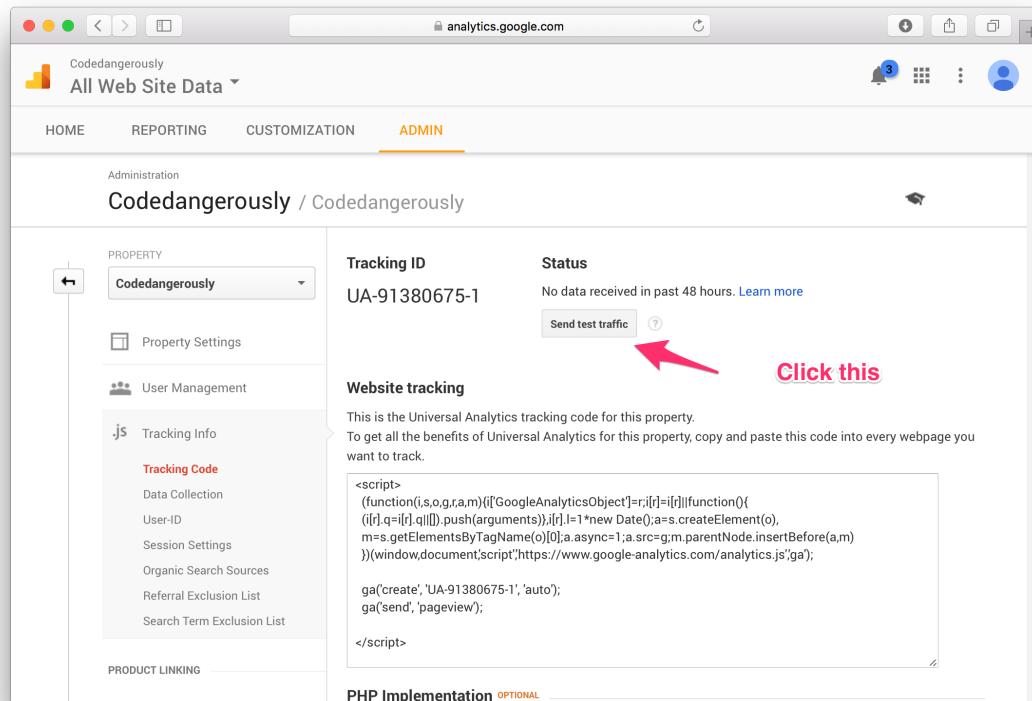


Figure 3.30: The tracking code page for your site.

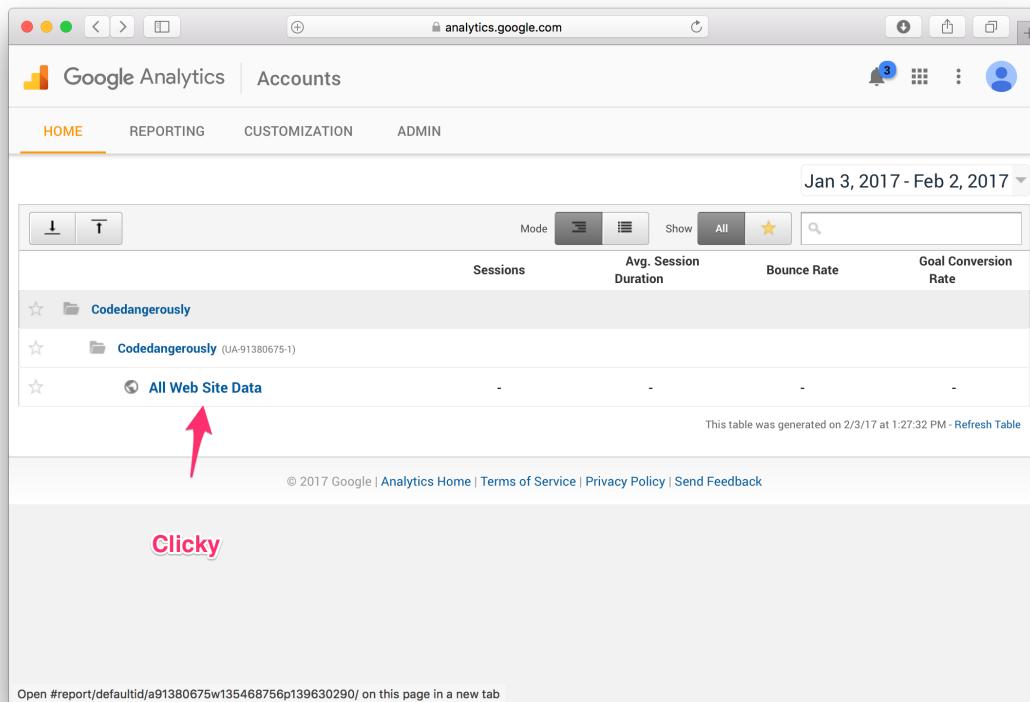


Figure 3.31: After you sign in you still need to click on this silly link to get to your site data.

Data” (Figure 3.31) you’ll be taken to the site data.

Once you are on the dashboard, you will have a ton of options for all the different ways to slice and dice your data, or even see the users who are on the site in real time (Figure 3.32).

Google Analytics is a *really* deep service, and we aren’t going to dive into how to use it—that would require a book of its own (and there are many out there). This goal here was just to get you up and running with the service, and in the future we might write up an Analytics tutorial of our own. Until then, you’ll just need to search around the [interwebs](#) and find guides and tutorials to teach you the dark arts of Google Analytics.

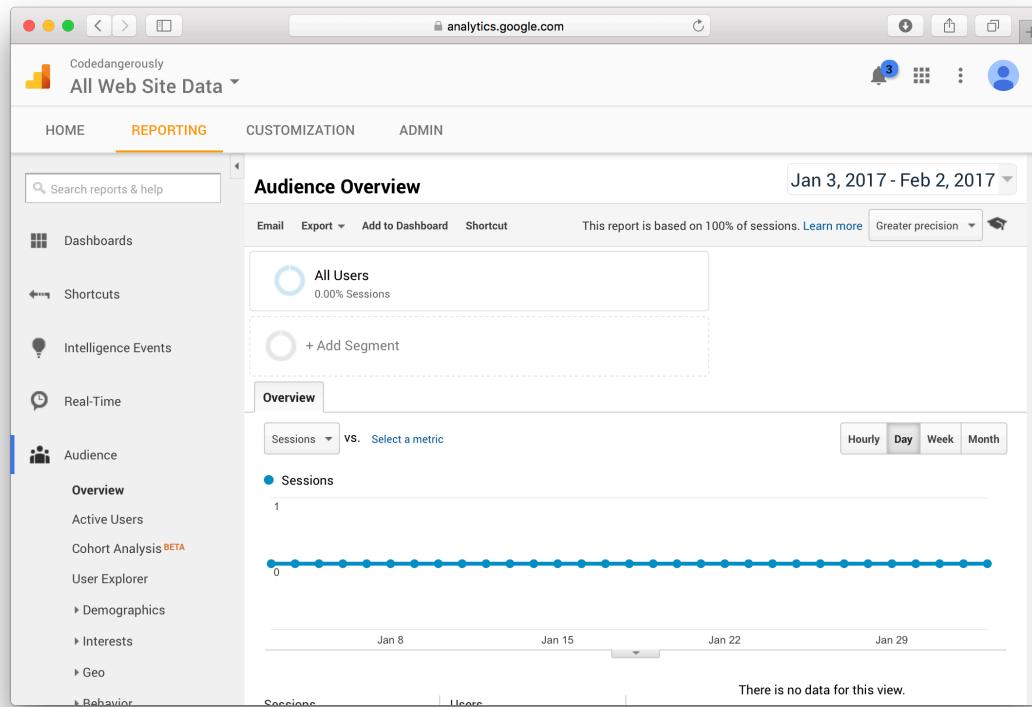


Figure 3.32: Hooray analytics, boo no visitors. So lonely.

3.3 Conclusion

So there you go! If you made it all the way through the tutorial, you now have your site running on a custom domain, you are using Cloudflare for DNS management, you have a custom email address through Google G Suite, and you are using Google Analytics to see who is visiting your site. You've gone from just having a rinky-dink page on the Internet to actually looking like a legitimate site or business. Great work!