Learn AMAZON WEB SERVICES IN A MONTH OF LUNCHES

SAMPLE CHAPTER

TUESDAY
WEDNESDAY
THURSDAY
1 Before you begin
2 The 10-minute EC2 web server
radical
3 Provisioning a more robust EC2 website
7 DNS: what's in a name?
8 S3: cheap, fast file storage
omg
9 S3: cheap, fast system backups
10 AWS security: working with IAM users, groups, and roles
11 Managing growth
14 Pushing back against the chaos: using resource tags
15 CloudWatch: monitoring AWS resources for fun and profit
16 Another way to play: the command-line interface
17 Keeping ahead of user demand
18 High availability: working with AWS networking tools
19
21 High availability: load balancing
22 High availability: auto scaling
23 High availability: content-delivery networks
24 Building hybrid infrastructure
25 Cloud automation: working with Elastic Beanstalk, Docker and Lambda
26
29 Everything else (nearly)
30 Never the end

DAVID CLINTON

MANNING

Learn Amazon Web Services
in a Month of Lunches
by David Clinton

Chapter 3

brief contents

3

Provisioning a more robust EC2 website

Creating the previous chapter's simple Hello World web page was all very well; but considering what's on display, do you suppose anyone will drop by for a visit? Adding the kind of killer content that'll have people falling over each other to get a good look at your site is your department. But assuming the hordes are on their way, what's next?

In this chapter, you'll learn how to prepare for those hordes by choosing the right blend of AWS resources to meet your unique needs. Then, using your new and improved web server, you'll install and launch a WordPress site to take your project to the next level.

3.1 Calculating capacity needs

How can you be sure that your humble AWS server has everything it needs to handle the expected load? What *is* the expected load?

I'll start with some basic assumptions about the nature of happiness:

- Your users will be happiest when their browser requests are answered quickly and accurately.
- Your virtual server will be happiest when it doesn't have to strain to get its job done.
- You'll be happiest when you can be confident that your users' needs are being served, your server has things under control,

and you're not spending any more money on your AWS resources than is absolutely necessary.

Let's dig a bit deeper. To make sure your server won't have to strain its poor, delicate self, you need to give it enough of these things:

- *Processing power*—CPU capacity
- *Storage space*—Hard drive capacity
- *System memory*—RAM
- *Network bandwidth*—Maximum amount of data that can be transferred into and out of your infrastructure at a given time

I'll translate these into AWS-friendly terms so you'll know we're speaking the same language.

3.2 Getting the measure of EC2's core compute services

Let's see how AWS packages virtual versions of each of the four basic categories of compute services:

3.2.1 vCPU

In the Amazon universe, processing power is measured in virtual central processing units (vCPUs). AWS tells you that a single AWS vCPU is roughly the equivalent of a "1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor." Modern consumer PCs often come with four cores, each running at 2.5 GHz; so, in simplified terms, your PC will deliver approximately eight times the processing power of a single vCPU. Having said that, because a virtual server doesn't have to devote resources to most of the thirsty stuff running on your laptop—like a high-resolution video interface or a web browser with 10 or 15 tabs open—it can get a lot more serious work done with a lot less power.

3.2.2 EBS

The AWS storage that's most comparable to the physical hard drive inside your PC is the EBS volume. Elastic Block Store (EBS) is a vast collection of hundreds of thousands of storage drives kept running in AWS's data centers. An EBS *volume* is a logically defined amount of storage space carved out of that vast storage system that's been set aside for an EC2 customer—which would be you.

Relatively speaking, renting space on an EBS volume is expensive, so you'll generally want to keep application data (data generated either for or by your applications) elsewhere. More often than not, that "elsewhere" will mean AWS's S3 storage service. I'll talk a lot about how S3 works in later chapters.

EBS volumes are generally used for your OS, your application code, and sometimes your database, but nothing more. That's why, unlike the 500–1,000 GB drives you'll see on bargain-basement home PCs, an 8 GB volume is often more than enough for Linux cloud servers. After all, who (besides YouTube) stores hundreds of cute cat videos on a server?

3.2.3 Memory

Memory is memory, no matter where it lives. Because RAM memory is much more instantly accessible than any other widely available option, it's usually used to temporarily store as much system-process and application data as possible. If, for example, serving a single web page to a single customer requires one tenth of a megabyte (100 KB) of RAM, you'll want to make sure you have at least 100 KB times *n* of the stuff available, where *n* is the maximum number of pages you might need to serve at a single time. Naturally, you can run this calculation using your own page size and traffic volumes.

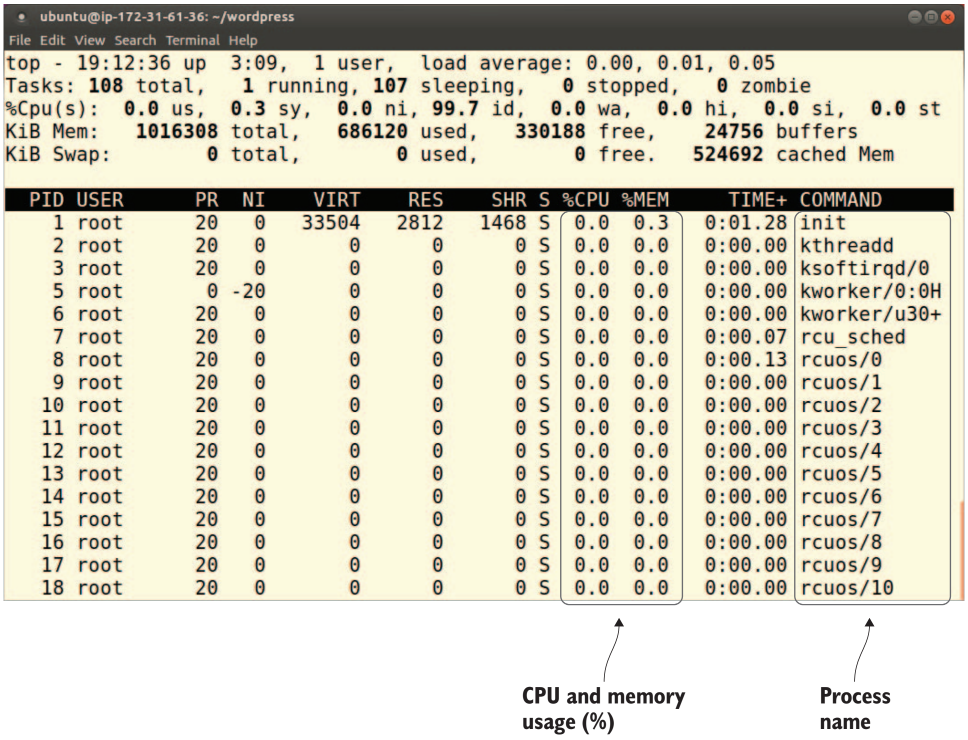
3.2.4 Bandwidth

The quality of an EC2 instance's connectivity to its network is described as its *network performance*. For most instance types, the value is Low, Moderate, or High.

The real-world performance you'll experience will depend on a wide variety of factors, but it's been estimated that you'll get anywhere from 2 to 100 Mbps for Low, 10 to 250 Mbps for Moderate, and 95 to 1000 Mbps for High. Compare those numbers with the amount of data you expect you'll need to transfer in and out of your server to keep your customers happy, to determine what level of network performance to select.

3.3 Assessing your application

In order for all that information to be useful, you'll need to get a sense of how much computing power your application will need. By far the most reliable approach is to try it out for yourself. From within an SSH



```
top - 19:12:36 up  3:09,  1 user,  load average: 0.00, 0.01, 0.05
Tasks: 108 total,   1 running, 107 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   1016308 total,   686120 used,   330188 free,    24756 buffers
KiB Swap:        0 total,        0 used,        0 free.   524692 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
    1 root      20   0   33504   2812   1468 S  0.0  0.3   0:01.28 init
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.00 ksoftirqd/0
    5 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0H
    6 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kworker/u30+
    7 root      20   0       0      0      0 S  0.0  0.0   0:00.07 rcu_sched
    8 root      20   0       0      0      0 S  0.0  0.0   0:00.13 rcuos/0
    9 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/1
   10 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/2
   11 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/3
   12 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/4
   13 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/5
   14 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/6
   15 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/7
   16 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/8
   17 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/9
   18 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/10
```

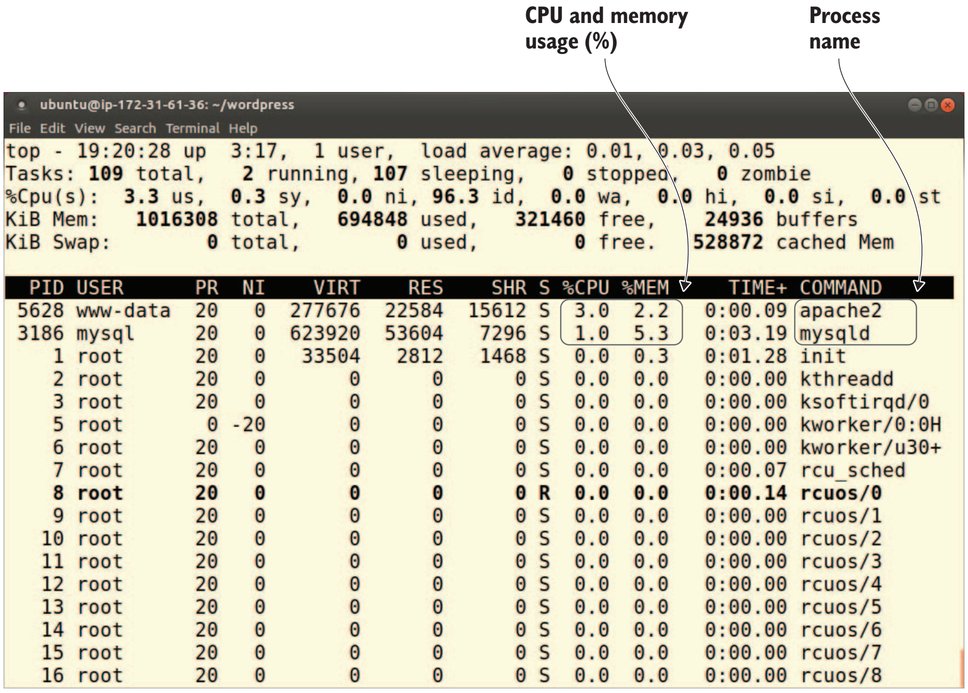
ubuntu@ip-172-31-61-36: ~/wordpress
File Edit View Search Terminal Help
CPU and memory usage (%)
Process name

Figure 3.1 A quiet EC2 web server, as measured by `top`

session on an AWS Ubuntu instance, figure 3.1 shows the output of the `top` command with the system at rest.

top is a Unix/Linux program that reports on the processes that are currently using the most system resources; it can be a great way to quickly assess the state of your instance. The two columns of most interest are %CPU, which indicates the percentage of total CPU capacity taken up by the most active processes; and %MEM, which tells you the percentage of system memory used by the most active processes. In this case, very little of either resource is currently being used.

But figure 3.2 shows the same `top` command run immediately after I loaded a web page served by this instance in my browser. This time, the Command column at far right shows that the two most active processes are associated with the Apache web server software and MySQL: between them, they account for 4% of the CPU and around 7.5% of memory.


CPU and memory usage (%)
Process name
ubuntu@ip-172-31-61-36: ~/wordpress
File Edit View Search Terminal Help

```
top - 19:20:28 up  3:17,  1 user,  load average: 0.01, 0.03, 0.05
Tasks: 109 total,   2 running, 107 sleeping,   0 stopped,   0 zombie
%Cpu(s):  3.3 us,  0.3 sy,  0.0 ni, 96.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   1016308 total,   694848 used,   321460 free,    24936 buffers
KiB Swap:        0 total,        0 used,        0 free.   528872 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 5628 www-data  20   0  277676  22584  15612 S  3.0  2.2   0:00.09 apache2
 3186 mysql     20   0  623920  53604   7296 S  1.0  5.3   0:03.19 mysqld
    1 root      20   0   33504   2812   1468 S  0.0  0.3   0:01.28 init
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.00 ksoftirqd/0
    5 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0H
    6 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kworker/u30+
    7 root      20   0       0      0      0 S  0.0  0.0   0:00.07 rcu_sched
    8 root      20   0       0      0      0 R  0.0  0.0   0:00.14 rcuos/0
    9 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/1
   10 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/2
   11 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/3
   12 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/4
   13 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/5
   14 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/6
   15 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/7
   16 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/8
```

Figure 3.2 `top` **output illustrating a busier web server. Note the MySQL and Apache activity.**

At this rate, you could certainly handle a dozen or so users launching requests within a short time; but after that, assuming you're not using some clever optimization program, you'll be pretty much hitting the limit. In other words, if you're expecting serious traffic to head your way, you'd better select a more robust instance type. Having said that, the virtualization underlying AWS makes it simple to upgrade resources later if your needs change.

Naturally, you'll also want to experiment with the quality of network response times and keep an eye on available storage on your EBS volume. Need a good tool to monitor network capacity? Look no further than EC2. With your instance selected in the EC2 Instances dashboard, select the Monitoring tab and scroll down through the data charts. You'll find accurate, up-to-date representations of network activity displayed, based on a number of metrics.

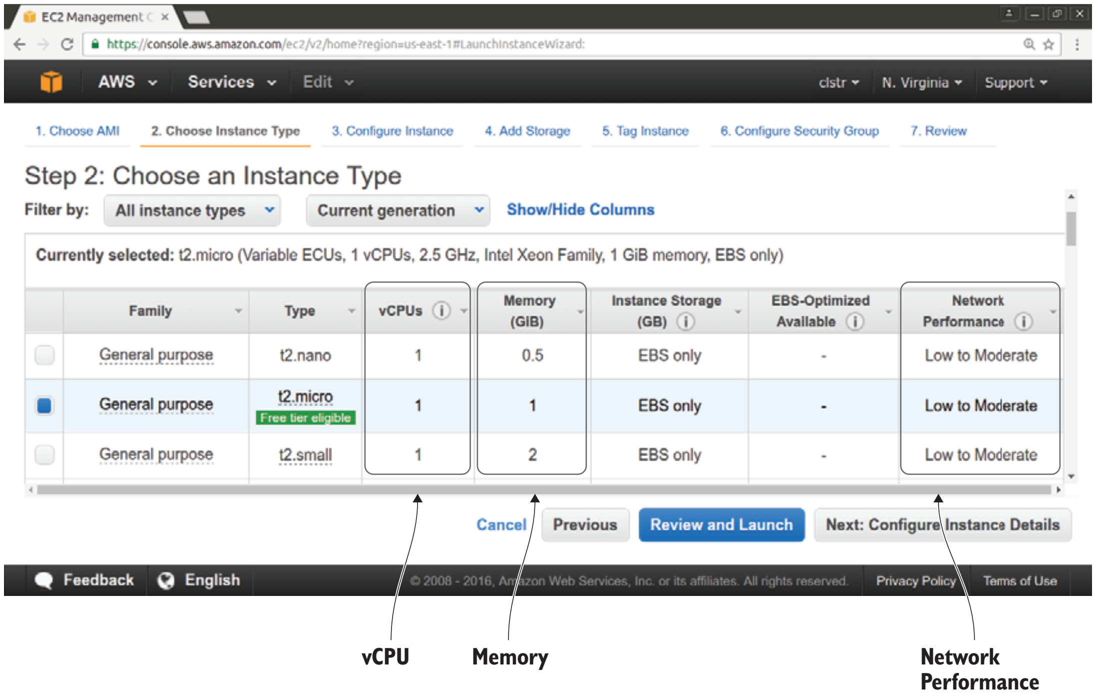
Try it now

Why not take some time right now and see if you can use these methods to work out the kind of resource levels you'll need for your project? How much traffic can your server handle before it will choke?

3.4 Choosing the right instance for your project

In addition to storage—which is a separate decision—the CPU, memory, and network-performance capacity you get will all depend on the instance type you choose. You probably remember seeing the Choose an Instance Type page (figure 3.3) in the previous chapter.

Keeping in mind what you've just learned about these values, spend some time looking through the vCPUs, Memory, and Network Performance columns and thinking about which type you're likely to need. To



Step 2: Choose an Instance Type

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate
■	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate
	General purpose	t2.small	1	2	EBS only	-	Low to Moderate

Figure 3.3 The EC2 Choose an Instance Type page

help you through that process, I'll give you an overview of AWS's instance type families.

Because AWS knows that the specific needs of individual users' projects are wildly different, it offers various categories of instance types, known as *families*, each focusing on a specific profile range. The general overview in this section should help you interpret what might otherwise feel a bit overwhelming.

As listed in table 3.1, AWS currently organizes its instance types into 10 families, each designated by a single letter: T, M, and so on. The identifying name of a particular type within a family consists of its family letter along with a number to differentiate it from any other types that exist in the family. The missing numbers (where's T1, for instance?) are probably older types that have since been deprecated. The Focus column describes the particular strength peculiar to each family.

Table 3.1 **EC2 instance type families**

Type family	Member types	Focus
T	T2	Baseline general purpose (*burstable performance*)[a]
M	M3, M4	General purpose: balance between compute, memory, and network
C	C3, C4	Computer optimized: high-performance processors
X	X1	Memory optimized for enterprise-class, in-memory applications
R	R3, R4	Memory optimized for memory-intensive applications
P	P2	Graphics accelerated for GPU-intensive applications
G	G2	Graphics-heavy processing
F	F1	Hardware acceleration with field-programmable arrays (FPGAs)
I	I2, I3	Storage optimized: very fast storage volumes for efficient I/O operations
D	D2	Storage optimized: high disk throughput for very large data stores

[a] *Burstable* means the vCPU of a T2 instance is capable of spending *CPU credits* to achieve brief bursts of higher performance to meet periods of higher demand. An instance acquires these credits during stretches of sub-baseline activity.

Each member type, such as M4, is generally available in a number of variations: M4.large, M4.xlarge, M4.2xlarge, and so on.

Suppose it's your job to keep Netflix.com's customer account database running smoothly. Based on all this information, if you're going to be running that sort of large, high-performance database operation, you'll want to choose a flavor of R3 instance—or, more accurately, many hundreds of them. But what if you're responsible for a busy multitiered web server, such as a popular blog that attracts more than 100,000 unique users each month? Then M4, or even C3, instances might work better. The good news is that AWS has a profile to match your needs. The better news is that if you choose badly or your needs evolve, updating your profile later isn't difficult.

That wasn't so complicated, was it? Maybe not, but only because I left out some stuff. Take another look at the AWS Console, and let your eyes wander toward the upper-right corner and the drop-down menu to the right of your account name. In my case (as you can see in figure 3.4), N. Virginia is the menu's current title, but clicking the menu shows a long list of other geographic areas.

NOTE *N. Virginia* isn't short for North Virginia. Last time I looked, there was no such state in the United States. The AWS folk probably mean northern Virginia.

The area (or region) you select will determine where in the world the resources you subsequently launch will be physically based. This is important for many reasons, including the fact that you may prefer to host your virtual servers on hardware that's as close as possible to your customers. But it's also significant because not every instance type is available in every region.

Try it now

Take a few minutes to browse through the instance types that are available in your region and—bearing in mind their specific attributes like vCPU count, memory, and storage—try to imagine which will work best for your needs. When you're finished, you should have a decent idea of which type you'll choose.

Whether you're going to use the same instance you created in the previous chapter or launch a new one, you'll need to log in to it again using

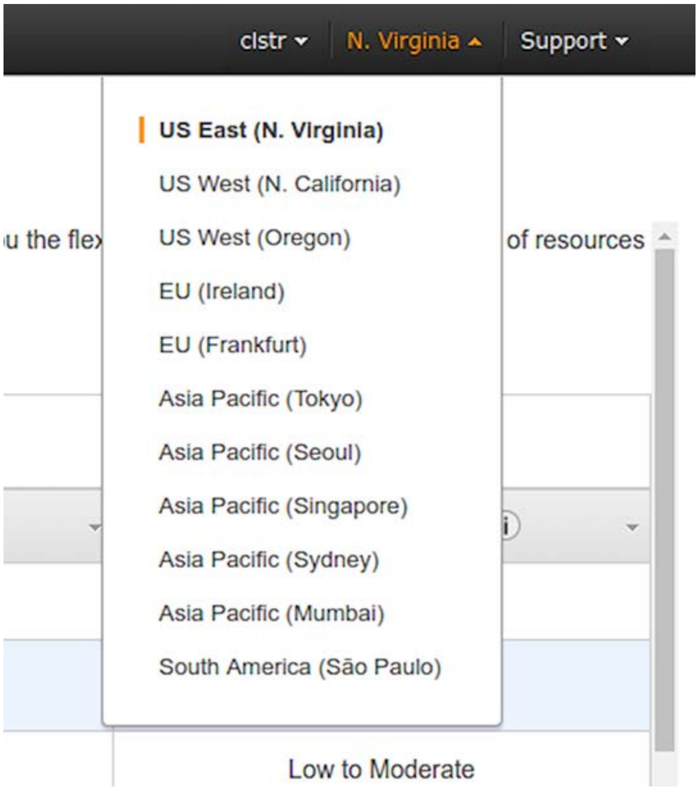


Figure 3.4 Select an AWS region.

SSH or PuTTY. While you're configuring a new instance, if AWS complains that no existing keys are found, check to make sure you're still using the same AWS geographic region as when you created the keys in the first place.

3.5 Adding WordPress

What is WordPress? Nothing much, really. Just the content management system (CMS) and website-creation tool used to build more than a quarter of all the internet's websites, as measured by the technology-usage trackers at w3techs.com (https://w3techs.com/technologies/overview/content_management/all).

Why should you use WordPress for a privately hosted site? Of course, you don't need to use WordPress if you don't want to, and I promise not to take it badly. But because WordPress makes it a lot easier to produce a professional-looking, reliable, secure website of just about any kind without needing a strong web development background, I think it's a good way to illustrate how your website might evolve. So here we go.

3.5.1 Preparing the server

There are a few Linux command line–based steps you need to walk through to get this thing going, but they won't be difficult. And if you keep your eyes open, they *will* be highly educational. Just so you know what's coming, here's what you'll do:

1. Install the LAMP server the same way you did in the last chapter (before you terminated that instance).
2. Create a new database in MySQL for WordPress to use.
3. Download and configure the latest version of WordPress.
4. Copy the WordPress files to your Apache document root directory.
5. Log in to your new WordPress site.

The first step is a repeat of the process you went through in chapter 2. If you chose to leave that instance running, then you can skip this part. Either way, the goal is to SSH in to your Ubuntu AWS instance:

```
$ ssh -i keyname.pem ubuntu@<your-instance-IP-address>
```

Assuming this is a brand-new instance, update your repositories. Then, once again, install the LAMP server package:

```
$ sudo apt update
$ sudo apt install lamp-server^
```

Don't forget to open port 80 (HTTP) in the AWS security group used by your instance; otherwise, your users will be unable to reach your site.

3.5.2 Preparing MySQL

You'll have to create a new user account and database for WordPress to use. Technically, you could use the MySQL root user for this purpose, but it's always best to use root accounts sparingly; they possess more power than they need, and using them more than absolutely necessary exposes you to an unnecessary security risk.

Use the root user (and the root password that you created and didn't forget during the MySQL installation) to log in to the MySQL shell. `-u` tells MySQL that you'll assume the root user, and `-p` says that you want to be prompted for your password:

```
$ mysql -u root -p
```

Once in the shell, you need to create a new database—I call it `wordpressdb`, but the name makes no difference. Then create a new user (which I call wpuser), assign a password, and grant the user privileges on the new database. Remember to end each MySQL command with a semicolon, or the shell will have absolutely no idea what you're trying to tell it. Finally, exit the shell:

```
mysql> CREATE DATABASE wordpressdb;
Query OK, 1 row affected (0.00 sec)
mysql> CREATE USER 'wpuser'@'localhost' IDENTIFIED BY 'mypassword';
Query OK, 0 rows affected (0.00 sec)
mysql> GRANT ALL PRIVILEGES ON wordpressdb.* To 'wpuser'@'localhost';
Query OK, 0 rows affected (0.00 sec)
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
mysql> exit
```

3.5.3 Downloading and configuring WordPress

In your browser, visit the WordPress download page (http://wordpress.org/download) to get the address of the package you're after. You don't want to download the package directly to your personal computer, because that's not where you're going to be installing WordPress. You could right-click the Download WordPress button and select Copy Link (or some variation) from the menu that appears. That option points to a .zip file, which would work fine but would require an extra step on your Ubuntu instance to install the unzip program. Instead, right-click Download .tar.gz to get the same package, but archived in the more Linux-friendly tar format.

Back in the SSH terminal session, use the `wget` program to download WordPress, using the address you copied from the WordPress website:

```
$ wget https://wordpress.org/latest.tar.gz
--2017-03-21 19:47:20--  https://wordpress.org/latest.tar.gz
Resolving wordpress.org (wordpress.org)... 66.155.40.249, 66.155.40.250
Connecting to wordpress.org (wordpress.org)|66.155.40.249|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8008833 (7.6M) [application/octet-stream]
Saving to: 'latest.tar.gz.1'

latest.tar.gz.1     100%[===================>]   7.64M  1.35MB/s    in 6.6s

2017-03-21 19:47:27 (1.16 MB/s) - 'latest.tar.gz.1' saved [8008833/8008833]
```

Once that's finished (and it shouldn't take more than a few seconds), use the tar program to decompress and unpack the file. The x tells tar to extract the archived contents, z means it's going to need decompression as well, and f points to the filename:

```
$ tar xzf latest.tar.gz
```

Run ls to list the current contents of your directory. You'll see that, in addition to the latest.tar.gz file you just downloaded, there's a new subdirectory called wordpress. Use cd to change to that directory and ls again to list its contents:

```
ls
cd wordpress
ls
```

See the file called wp-config-sample.php? That's a template you can use to manage your WordPress configuration. You'll use cp to create a copy of that file named wp-config.php (so WordPress will know that this is the configuration file you want it to use) and then edit its contents:

```
$ cp wp-config-sample.php wp-config.php
```

You can use the nano text editor to work with the config file:

```
$ nano wp-config.php
```

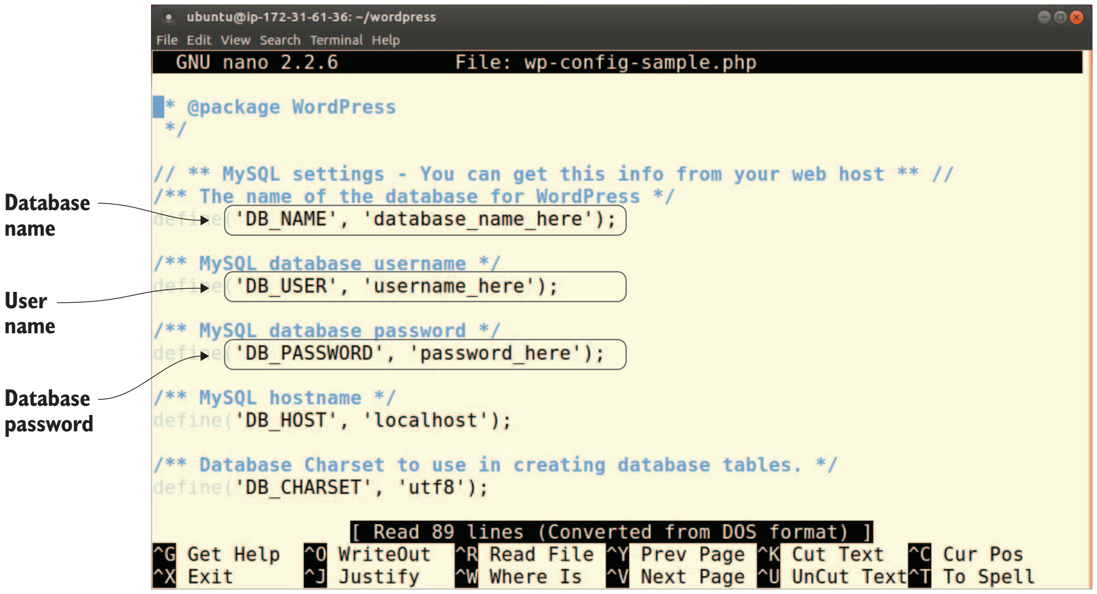
This (as you can see in figure 3.5) is where you tell WordPress about its new neighborhood. Replace database_name_here with the name you'd like to give your database, username_here with the wpuser you created for MySQL, and password_here with that user's new password, so that WordPress can successfully log in and manage its database. For the values I used, those lines now look like this:

```
/** The name of the database for WordPress */
define('DB_NAME', 'wordpressdb');

/** MySQL database username */
define('DB_USER', 'wpuser');

/** MySQL database password */
define('DB_PASSWORD', 'mypassword');
```

TIP Using *mypassword* (or any remotely similar variation) for a password on any working server is nuts. Don't try it.


ubuntu@ip-172-31-61-36: ~/wordpress
File Edit View Search Terminal Help
Database name
User name
Database password

```
  GNU nano 2.2.6            File: wp-config-sample.php

 * @package WordPress
 */

// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define('DB_NAME', 'database_name_here');

/** MySQL database username */
define('DB_USER', 'username_here');

/** MySQL database password */
define('DB_PASSWORD', 'password_here');

/** MySQL hostname */
define('DB_HOST', 'localhost');

/** Database Charset to use in creating database tables. */
define('DB_CHARSET', 'utf8');

                  [ Read 89 lines (Converted from DOS format) ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

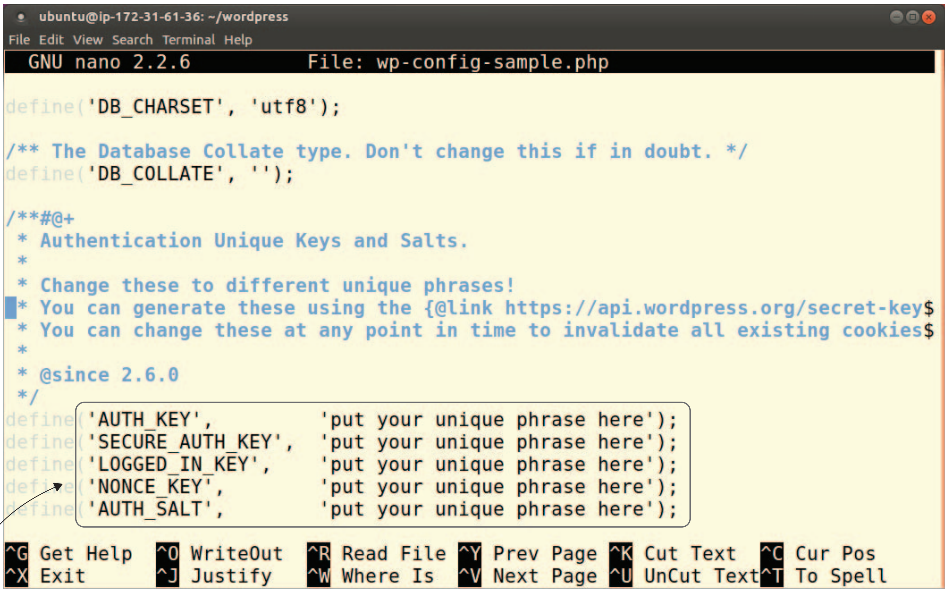
Figure 3.5 Lines from the wp-config.php file that require your attention

You can quickly generate a unique set of keys and salts—character strings used by WordPress to allow secure authentication with the database, as shown in figure 3.6—by loading the WordPress secret-key page (https://api.wordpress.org/secret-key/1.1/salt) in your browser, copying the text that's displayed, and then using that text to replace the existing sample text.

TIP When using nano, rather than pressing the Delete key over and over again to remove the hundreds of characters making up the sample salts lines, move your cursor to anywhere on a line you want to delete and press Ctrl-K. The entire line will magically disappear. Repeat to taste. To save the file and exit, press Ctrl-X.

That's it for the configuration. All that's left is to copy the contents of the wordpress directory to wherever you'd like your WordPress site to live.

TIP The only absolutely reliable way to know if you've configured the file properly is to try it out and see whether you can load the page in your browser (more about that later). If something goes wrong, you should look for any error messages that appear. Consulting system logs like those at /var/log/apache2 and /var/log/mysql can also be helpful.


ubuntu@ip-172-31-61-36: ~/wordpress
File Edit View Search Terminal Help
Keys and salts lines

```
  GNU nano 2.2.6            File: wp-config-sample.php

define('DB_CHARSET', 'utf8');

/** The Database Collate type. Don't change this if in doubt. */
define('DB_COLLATE', '');

/**#@+
 * Authentication Unique Keys and Salts.
 *
 * Change these to different unique phrases!
 * You can generate these using the {@link https://api.wordpress.org/secret-key$
 * You can change these at any point in time to invalidate all existing cookies$
 *
 * @since 2.6.0
 */
define('AUTH_KEY',         'put your unique phrase here');
define('SECURE_AUTH_KEY',  'put your unique phrase here');
define('LOGGED_IN_KEY',    'put your unique phrase here');
define('NONCE_KEY',        'put your unique phrase here');
define('AUTH_SALT',        'put your unique phrase here');

^G Get Help   ^O WriteOut   ^R Read File ^Y Prev Page ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Figure 3.6 Lines from the wp-config.php file reserved for authentication keys

3.5.4 Setting up the WordPress filesystem

If you want to use WordPress to manage all the content on your domain—perhaps you like the fact that the WordPress ecosystem includes so many resources for closely integrating content and transaction processing—you should copy all the files in the wordpress directory to Apache's document root directory. On Ubuntu machines, you'll recall, this is /var/www/html/. If you do this and your domain is, for instance, bootstrap-it.com, visitors will see the contents of WordPress's main page. But if you're already using bootstrap-it.com to host other content, and you'd prefer to use WordPress, say, only as a blog, then you might create a directory called blog within /var/www/html/. Visitors to bootstrap-it.com/blog will then see that content.

Copying files along with subdirectories and their contents requires the `-r` flag along with the `cp` command, invoking the recursive feature. When run from within the wordpress directory, the following example copies all the files in the directory to the document root:

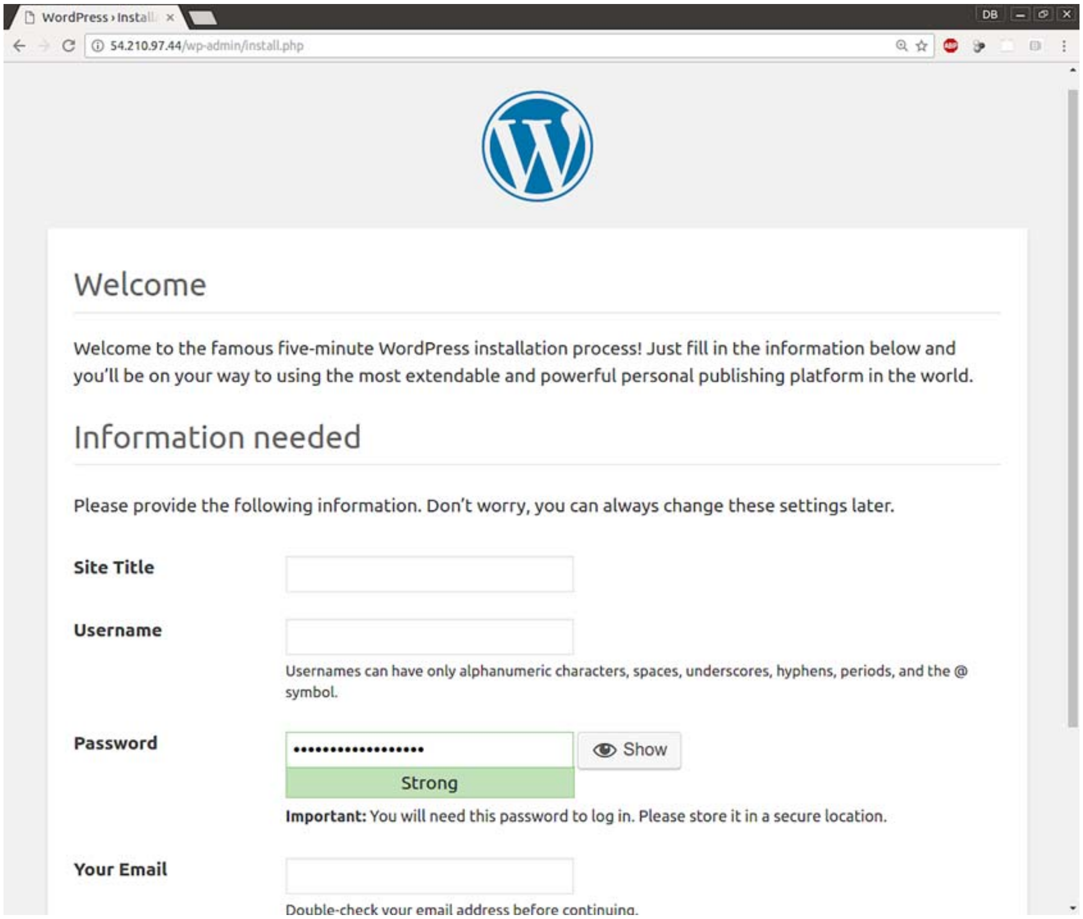
```
$ sudo cp -r * /var/www/html
```

3.5.5 *Testing your application*

When that's done, it's time to head back to your browser and point it to your instance's public IP address (the one you used to SSH in). Just this once, add the route to the install.php file, where you can create a WordPress admin account. The address will look something like this:

```
54.218.241.9/wp-admin/install.php
```

As you can see in figure 3.7, you're off to the races!



Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Don't worry, you can always change these settings later.

Site Title

Username

Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.

Password ••••••••••••••••••• Show

Strong

Important: You will need this password to log in. Please store it in a secure location.

Your Email

Double-check your email address before continuing.

Figure 3.7 The setup screen that will welcome you to your new WordPress site (accessed using *your-ip-or-sitename*/wp-admin/install.php)

Of course, it's possible that you won't be off to the races and the page will fail to load. There's a long list of things that may have gone wrong, but these three troubleshooting steps are nearly always helpful:

1. Note any error messages that are displayed.
2. Look for log messages. On an Ubuntu system, you'll find useful log files in the /var/log/apache2/ and /var/log/mysql/ directories. Remember that although log files can be long, each entry usually has a timestamp.
3. Copy important-looking segments of error or log messages, and paste them into the search bar of your favorite internet search engine. Someone else has almost certainly encountered—and solved—your problem already.

In addition, you may encounter the following error when you try to load the WordPress install page in your browser:

```
Your PHP installation appears to be missing the MySQL extension
➥which is required by WordPress.
```

This probably means that the legacy PHP MySQL module isn't working. You can either enable MySQLi using

```
$ sudo phpenmod mysqli
```

or remove the old mysql-common package and install mysql-server on top:

```
$ sudo apt remove mysql-common
$ sudo apt install mysql-server
```

Either way, you should restart Apache, just to make sure it's up to date with all the latest changes:

```
sudo systemctl restart apache2
```

3.5.6 What next?

Finally, you've got one more tough choice to make: to shut down or not to shut down. It will certainly be helpful to have this instance available for coming chapters, so leaving it running would be nice. But if there's a good chance that you'll put the book aside for some weeks—your boss cancelled your lunch break, for instance—then you may want to terminate the instance and build it again from scratch when you're ready to continue.

If you decide to shut down, remember that you can do so from the EC2 Instances dashboard. Make sure this instance is selected, and click Actions > Instance State > Terminate. Rather than terminate it, you can also click Stop to stop your instance and restart it at a later time. One thing to keep in mind about Stop, though, is that the IP address you've been using to access the instance will no longer be available when you restart.

If that might be a problem (and it often will), you can associate a permanent elastic IP address to the instance. This will be useful later as your AWS resources become more dependent on each other. You can read more about elastic IPs in chapter 5.

3.6 Lab

With a live EC2 instance running WordPress—build it again, if you've shut down the one from the demo—use a number of browsers to load its pages. From inside the instance (via SSH), use `top` (or similar tools) to measure how close you get to your system capacity.

Definitions

- *vCPU*—Virtual central processing units
- *Network bandwidth*—The rate (or volume of data) at which data can travel through a network
- *Instance type*—A standard measure of an AWS EC2 virtual machine
- *AWS region*—A collection of physical AWS data center server farms located within a specific geographic area
- *PuTTY*—A Windows client used to access remote resources through SSH servers.
- *Root account*—The main administration user account for an operating system or database
- *tar*—The Unix/Linux file-archiving program
- *Keys and salts*—WordPress authentication keys

Command-line review

- *Create a database in MySQL*—`mysql> CREATE DATABASE wordpressdb;`
- *Extract an archive using* `tar`—`tar xzf latest.tar.gz`
- *Make a working copy of the sample config file*—`cp wp-config-sample.php wp-config.php`
- *Delete a complete line of text (in nano)*—Ctrl-K
- *Copy the current directory contents to the web root*—`sudo cp -r * /var/www/html`

SOFTWARE ENGINEERING/CLOUD

Learn AMAZON WEB SERVICES IN A MONTH OF LUNCHES

DAVID CLINTON

Free eBook
SEE INSERT

Cloud computing has transformed the way we build and deliver software. With the Amazon Web Services cloud platform, you can trade expensive glass room hardware and custom infrastructure for virtual servers and easy-to-configure storage, security, and networking services. Better, because you don't own the hardware, you only pay for the computing power you need! Just learn a few key ideas and techniques, and you can have applications up and running in AWS in minutes.

Learn Amazon Web Services in a Month of Lunches gets you started with AWS fast. In just 21 bite-size lessons, you'll learn the concepts and practical techniques you need to deploy and manage applications. You'll learn by doing real-world labs that guide you from the core AWS tool set through setting up security and storage and planning for growth. You'll even deploy a public-facing application that's highly available, scalable, and load balanced.

WHAT'S INSIDE

- First steps with AWS—no experience required
- Deploy web apps using EC2, RDS, S3, and Route 53
- Cheap and fast system backups
- Setting up cloud automation

If you know your way around Windows or Linux and have a basic idea of how web applications work, you're ready to start using AWS.

David Clinton is a system administrator and Linux server professional with 15 years of teaching experience.

"An excellent book for those who have often thought about learning AWS but were overwhelmed by options and where to start."

—Conor Redmond
InComm Product Control

"An easy read and straight to the point. A great foundation for every AWS user."

—Alexey Galiullin, GlobalOrange

"A must-read for anyone who wants a robust and practical start with AWS."

—Peter Perlepes, Growth

"A friendly tutorial on what's essential to know in today's cloud/DevOps world."

—Miguel Paraz, Odecee

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit www.manning.com/books/learn-amazon-web-services-in-a-month-of-lunches

ISBN-13: 978-1-61729-444-0
ISBN-10: 1-61729-444-6
53999
9 781617 294440

MANNING $39.99 / Can $52.99 [INCLUDING eBOOK]