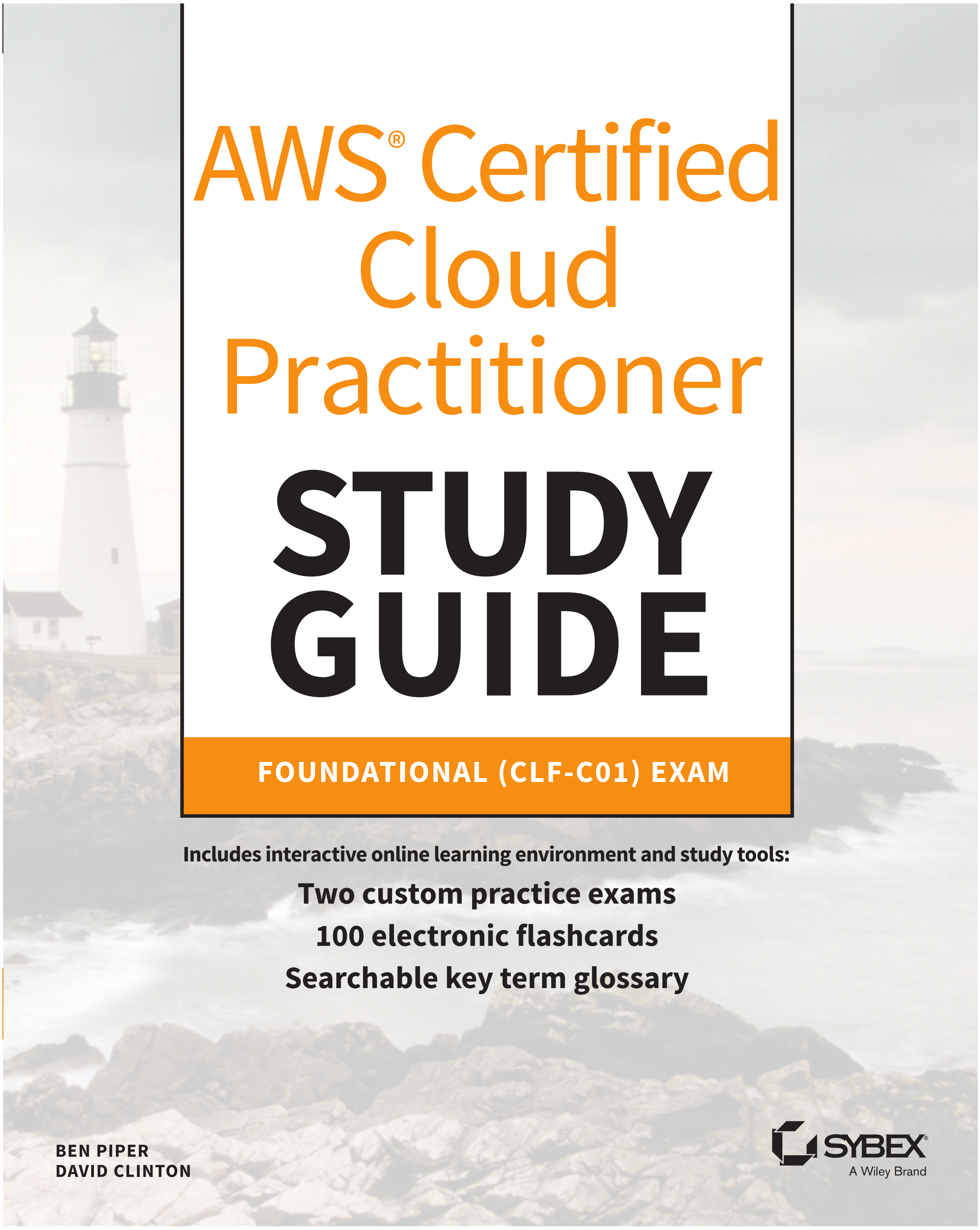

AWS® Certified Cloud Practitioner STUDY GUIDE

FOUNDATIONAL (CLF-C01) EXAM

Includes interactive online learning environment and study tools:

Two custom practice exams

100 electronic flashcards

Searchable key term glossary

BEN PIPER
DAVID CLINTON

SYBEX®
A Wiley Brand

through the Secure Shell (SSH) protocol for this demo. Otherwise, acknowledge the consequences of launching an instance without a key pair (something that would normally be a bit crazy), and go ahead with the launch.

7. The blue View Instances button at the bottom of the confirmation screen will take you to the EC2 Instances Dashboard where you should see your instance status. When the Instance State indicator turns green and the word *running* appears next to it, your web server should be live.
8. Making sure this instance is selected, find the IPv4 Public IP entry on the Description tab in the bottom half of the screen. Hover your mouse over the IP address that's listed there, and choose the Copy To Clipboard icon to its right. Open a new tab in your browser, and paste the IP address into the URL field. You should see the default Apache test page.
9. When you're done admiring your handiwork, you can shut down the instance from the Actions menu at the top of the Instances page. With your instance selected, choose Action, then Instance State, and finally Terminate. When you confirm the action, your instance will be shut down, and you won't be billed any further. If your account is still in its first year and you haven't been running similar instances for more than 750 hours this month, this Free Tier instance won't have cost you anything anyway.

Simplified Deployments Through Managed Services

Building and administrating software applications can be complex wherever you deploy them. Whether they're on-premises or in the cloud, you'll face a lot of moving parts existing in a universe where they all have to play nicely together or the whole thing can collapse. To help lower the bar for entry into the cloud, some AWS services will handle much of the underlying infrastructure for you, allowing you to focus on your application needs. The benefits of a managed service are sometimes offset by premium pricing. But it's often well worth it.

One important example of such a managed service is the Relational Database Service (RDS). RDS, as you'll see in Chapter 9, "The Core Database Services," lets you set the basic configuration parameters for the database engine of your choice, gives you an endpoint address through which your applications can connect to the database, and takes care of all the details invisibly. Besides being responsible for making top-level configuration decisions, you won't need to worry about maintaining the database instance hosting the database, software updates, or data replication.

One step beyond a managed service—which handles only *one part* of your deployment stack for you—is a managed *deployment*, where the *whole stack* is taken care of behind the scenes. All you'll need to make things work with one of these services is your application code or, if it's a website you're after, your content. Until Amazon figures out how to secretly listen in on your organization's planning meetings and then automatically convert your ideas to code without you knowing, things are unlikely to get any simpler than this.

Two AWS services created with managed deployments in mind are Amazon Lightsail and AWS Elastic Beanstalk.

Amazon Lightsail

Lightsail is promoted as a low-stress way to enter the Amazon cloud world. It offers *blueprints* that, when launched, will automatically provision all the compute, storage, database, and network resources needed to make your deployment work. You set the pricing level you're after (currently that'll cost you somewhere between $3.50 and $160 USD each month) and add an optional script that will be run on your instance at startup, and AWS will take over. For context, $3.50 will get you 512 MB of memory, 1 vCPU, a 20 GB solid-state drive (SSD) storage volume, and 1 TB of transfers.

Lightsail uses all the same tools—such as the AMIs and instances you saw earlier in the chapter—to convert your plans to reality. Since it's all AWS from top to bottom, you're also covered should you later decide to move your stack directly to EC2, where you'll have all the added access and flexibility standard to that platform.

Because things are packaged in blueprints, you won't have the unlimited range of tools for your deployments that you'd get from EC2 itself. But, as you can see from these lists, there's still a nice selection:

- **Operating systems:** Amazon Linux, Ubuntu, Debian, FreeBSD, OpenSUSE, and Windows Server
- **Applications:** WordPress, Magento, Drupal, Joomla, Redmine, and Plesk
- **Stacks:** Node.js, GitLab, LAMP, MEAN, and Nginx

NOTE In case you're curious, a LAMP stack is a web server built on Linux, Apache, MySQL (or MariaDB), and PHP (or Python). By contrast, MEAN is a JavaScript stack for dynamic websites consisting of MongoDB, Express.js, AngularJS (or Angular), and Node.js.

AWS Elastic Beanstalk

If anything, Elastic Beanstalk is even simpler than Lightsail. All that's expected from you is to define the application platform and then upload your code. That's it. You can choose between preconfigured environments (including Go, .NET, Java Node.js, and PHP) and a number of Docker container environments. The "code" for Docker applications is defined with specially formatted `Dockerrun.aws.json` files.

One key difference between the two services is that while Lightsail bills at a flat rate (between $3.50 and $160 per month, as you saw), Beanstalk generates costs according to how resources are consumed. You don't get to choose how many vCPUs or how much memory you will use. Instead, your application will scale its resource consumption according to demand. Should, say, your WordPress site go viral and attract millions of viewers one day, AWS will invisibly ramp up the infrastructure to meet demand. As demand falls, your infrastructure will similarly drop. Keep this in mind, as such variations in demand will determine how much you'll be billed each month.

Deploying Container and Serverless Workloads

Even virtualized servers like EC2 instances tend to be resource-hungry. They do, after all, act like discrete, stand-alone machines running on top of a full-stack operating system. That means that having 5 or 10 of those virtual servers on a single physical host involves some serious duplication because each one will require its own OS kernel and device drivers.

Containers

Container technologies such as Docker avoid a lot of that overhead by allowing individual containers to share the Linux kernel with the physical host. They're also able to share common elements (called *layers*) with other containers running on a single host. This makes Docker containers fast to load and execute and also lets you pack many more container workloads on a single hardware platform.

You're always free to fire up one or more EC2 instances, install Docker, and use them to run as many containers as you'd like. But keeping all the bits and pieces talking to each other can get complicated. Instead, you can use either *Amazon Elastic Container Service* (ECS) or *Amazon Elastic Container Service for Kubernetes* (EKS) to orchestrate swarms of Docker containers on AWS using EC2 resources. Both of those services manage the underlying infrastructure for you, allowing you to ignore the messy details and concentrate on administrating Docker itself.

What's the difference between ECS and EKS? Broadly speaking, they both have the same big-picture goals. But EKS gets there by using the popular open source Kubernetes orchestration tool. They are different paths to the same place.

Serverless Functions

The serverless computing model uses a resource footprint that's even smaller than the one left by containers. Not only do *serverless functions* not require their own OS kernel, but they tend to spring into existence, perform some task, and then just as quickly die within minutes, if not seconds.

On the surface, Amazon's serverless service—AWS Lambda—looks a bit like Elastic Beanstalk. You define your function by setting a runtime environment (like Node.js, .NET, or Python) and uploading the code you want the function to run. But, unlike Beanstalk,