going through health checks. Wait for the Status indicator to indicate "healthy" for both of them. This typically takes one to two minutes. When you see it, test the `alb_dns_name` output you copied earlier:

```
$ curl http://<alb_dns_name>
Hello, World
```

Success! The ALB is routing traffic to your EC2 Instances. Each time you access the URL, it'll pick a different Instance to handle the request. You now have a fully working cluster of web servers!

At this point, you can see how your cluster responds to firing up new Instances or shutting down old ones. For example, go to the Instances tab and terminate one of the Instances by selecting its checkbox, clicking the Actions button at the top, and then setting the Instance State to Terminate. Continue to test the ALB URL, and you should get a 200 OK for each request, even while terminating an Instance, because the ALB will automatically detect that the Instance is down and stop routing to it. Even more interesting, a short time after the Instance shuts down, the ASG will detect that fewer than two Instances are running and automatically launch a new one to replace it (self-healing!). You can also see how the ASG resizes itself by adding a `desired_capacity` parameter to your Terraform code and rerunning `apply`.

Cleanup

When you're done experimenting with Terraform, either at the end of this chapter, or at the end of future chapters, it's a good idea to remove all of the resources you created so that AWS doesn't charge you for them. Because Terraform keeps track of what resources you created, cleanup is simple. All you need to do is run the `destroy` command:

```
$ terraform destroy

(...)
```

```
Terraform will perform the following actions:

  # aws_autoscaling_group.example will be destroyed
  - resource "aws_autoscaling_group" "example" {
      (...)
    }

  # aws_launch_configuration.example will be destroyed
  - resource "aws_launch_configuration" "example" {
      (...)
    }

  # aws_lb.example will be destroyed
  - resource "aws_lb" "example" {
      (...)
    }

  (...)

Plan: 0 to add, 0 to change, 8 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as
shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value:
```

It goes without saying that you should rarely, if ever, run `destroy` in a production environment! There's no "undo" for the `destroy` command, so Terraform gives you one final chance to review what you're doing, showing you the list of all the resources you're about to delete, and prompting you to confirm the deletion. If everything looks good, type **`yes`** and hit Enter; Terraform will build the dependency graph and delete all of the resources in the correct order, using as much parallelism as possible. In a minute or two, your AWS account should be clean again.

Note that later in the book, you will continue to develop this example, so don't delete the Terraform code! However, feel free to run `destroy` on the actual deployed resources whenever you want. After all, the beauty of infrastructure as code is that all of the information about those resources is captured in code, so you can re-create all of them at any time with a single