Figure 4-3. Putting code into modules allows you to reuse that code from multiple environments.

This is a big deal. Modules are the key ingredient to writing reusable, maintainable, and testable Terraform code. Once you start using them, there's no going back. You'll start building everything as a module, creating a library of modules to share within your company, using modules that you find online, and thinking of your entire infrastructure as a collection of reusable modules.

In this chapter, I'll show you how to create and use Terraform modules by covering the following topics:

- Module basics
- Module inputs
- Module locals
- Module outputs
- Module gotchas
- Module versioning

> **EXAMPLE CODE**
>
> As a reminder, you can find all of the code examples in the book on GitHub.

Module Basics

A Terraform module is very simple: any set of Terraform configuration files in a folder is a module. All of the configurations you've written so far have technically been modules, although not particularly interesting ones, since you deployed them directly: if you run `apply` directly on a module, it's referred to as a *root module*. To see what modules are really capable of, you need to create a *reusable module*, which is a module that is meant to be used within other modules.

As an example, let's turn the code in *stage/services/webserver-cluster*, which includes an Auto Scaling Group (ASG), Application Load Balancer (ALB), security groups, and many other resources, into a reusable module.

As a first step, run `terraform destroy` in the *stage/services/webserver-cluster* to clean up any resources that you created earlier. Next, create a new top-level folder called *modules*, and move all of the files from *stage/services/webserver-cluster* to *modules/services/webserver-cluster*. You should end up with a folder structure that looks something like Figure 4-4.

Open up the *main.tf* file in *modules/services/webserver-cluster*, and remove the `provider` definition. Providers should be configured only in root modules and not in reusable modules (you'll learn a lot more about working with providers in Chapter 7).

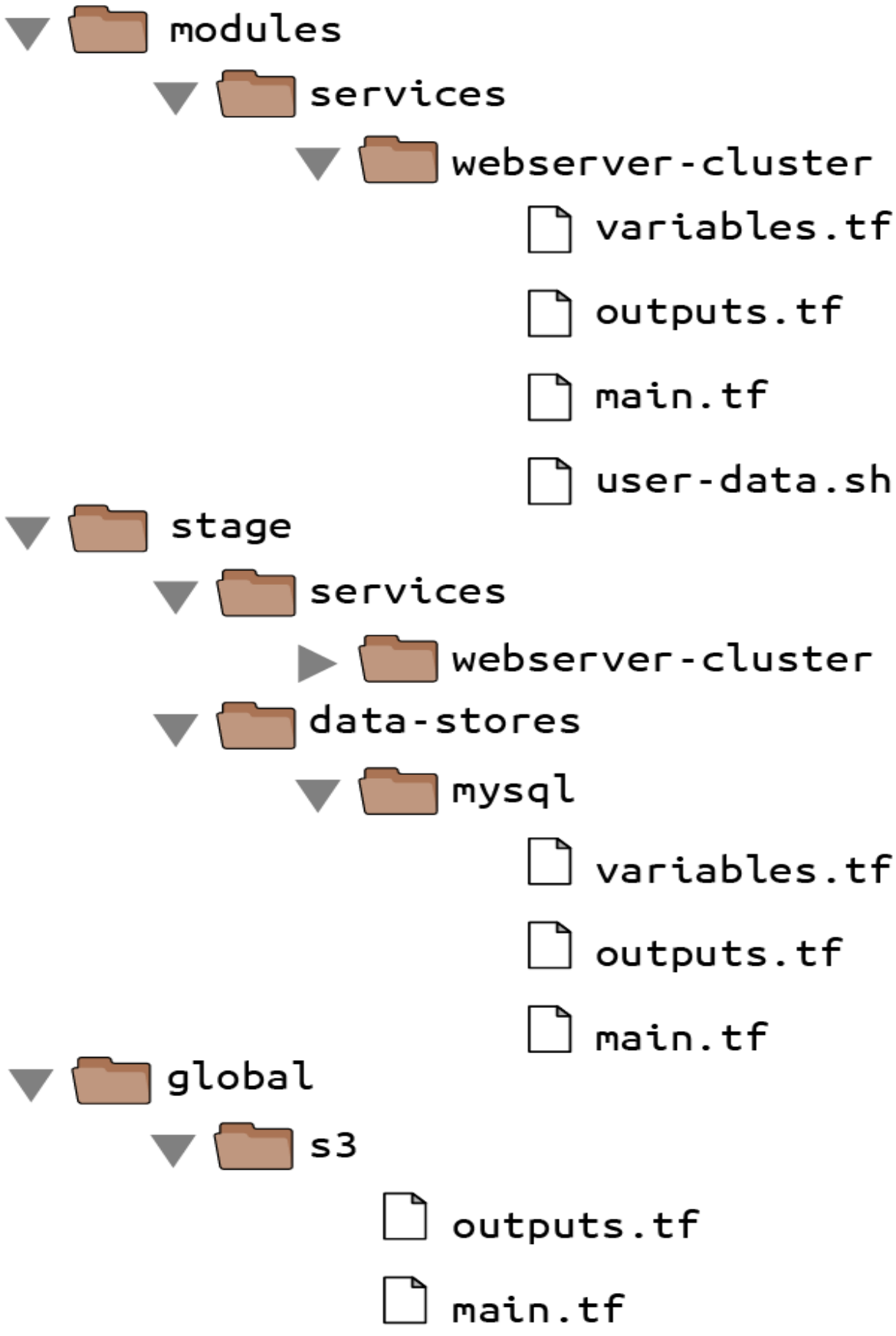
modules
services
webserver-cluster
variables.tf
outputs.tf
main.tf
user-data.sh
stage
services
webserver-cluster
data-stores
mysql
variables.tf
outputs.tf
main.tf
global
s3
outputs.tf
main.tf

Figure 4-4. Move your reusable web server cluster code into a modules/services/web server-cluster folder.

You can now make use of this module in the staging environment. Here's the syntax for using a module:

```
module "<NAME>" {
  source = "<SOURCE>"

  [CONFIG ...]
}
```

where `NAME` is an identifier you can use throughout the Terraform code to refer to this module (e.g., `webserver_cluster`), `SOURCE` is the path where the module code can be found (e.g., *modules/services/webserver-cluster*), and `CONFIG` consists of arguments that are specific to that module. For example, you can create a new file in *stage/services/webserver-cluster/main.tf* and use the `webserver-cluster` module in it as follows:

```
provider "aws" {
  region = "us-east-2"
}

module "webserver_cluster" {
  source = "../../../modules/services/webserver-cluster"
}
```

You can then reuse the exact same module in the production environment by creating a new *prod/services/webserver-cluster/main.tf* file with the following contents:

```
provider "aws" {
  region = "us-east-2"
}

module "webserver_cluster" {
  source = "../../../modules/services/webserver-cluster"
}
```