should do that itself. Only include comments to offer information that can't be expressed in code, such as how the code is meant to be used or why the code uses a particular design choice. Terraform also allows every input and output variable to declare a `description` parameter, which is a great place to describe how that variable should be used.

Example code

As discussed in Chapter 8, every Terraform module should include example code that shows how that module is meant to be used. This is a great way to highlight the intended usage patterns and give your users a way to try your module without having to write any code, and it's the main way to add automated tests for the module.

Automated tests

All of Chapter 9 focuses on testing Terraform code, so I won't repeat any of that here, other than to say that infrastructure code without tests is broken. Therefore, one of the most important comments you can make in any code review is "How did you test this?"

File layout

Your team should define conventions for where Terraform code is stored and the file layout you use. Because the file layout for Terraform also determines the way Terraform state is stored, you should be especially mindful of how file layout affects your ability to provide isolation guarantees, such as ensuring that changes in a staging environment cannot accidentally cause problems in production. In a code review, you might want to enforce the file layout described in "Isolation via File Layout", which provides isolation between different environments (e.g., stage and prod) and different components (e.g., a network topology for the entire environment and a single app within that environment).

Style guide