expected response, run `terraform destroy` to clean everything up. All told, it should take only a few minutes, and you now have a reasonable unit test for the “Hello, World” app.

Running tests in parallel

In the previous section, you ran just a single test using the `-run` argument of the `go test` command. If you had omitted that argument, Go would’ve run all of your tests—sequentially. Although four to five minutes to run a single test isn’t too bad for testing infrastructure code, if you have dozens of tests, and each one runs sequentially, it could take hours to run your entire test suite. To shorten the feedback loop, you want to run as many tests in parallel as you can.

To instruct Go to run your tests in parallel, the only change you need to make is to add `t.Parallel()` to the top of each test. Here it is in *test/hello_world_app_example_test.go*:

```
func TestHelloWorldAppExample(t *testing.T) {
        t.Parallel()

        opts := &terraform.Options{
                // You should update this relative path to point
at your
                // hello-world-app example directory!
                TerraformDir: "../examples/hello-world-
app/standalone",

                Vars: map[string]interface{}{
                        "mysql_config": map[string]interface{}{
                                "address": "mock-value-for-test",
                                "port":    3306,
                        },
                },
        }

        // (...)
}
```

And similarly in *test/alb_example_test.go*:

```
func TestAlbExample(t *testing.T) {
        t.Parallel()

        opts := &terraform.Options{
                // You should update this relative path to point
at your alb
                // example directory!
                TerraformDir: "../examples/alb",
        }

        // (...)
}
```

If you run `go test` now, both of those tests will execute in parallel. However, there's one gotcha: some of the resources created by those tests—for example, the ASG, security group, and ALB—use the same name, which will cause the tests to fail due to the name clashes. Even if you weren't using `t.Parallel()` in your tests, if multiple people on your team were running the same tests or if you had tests running in a CI environment, these sorts of name clashes would be inevitable.

This leads to *key testing takeaway #4*: you must namespace all of your resources.

That is, design modules and examples so that the name of every resource is (optionally) configurable. With the `alb` example, this means that you need to make the name of the ALB configurable. Add a new input variable in *examples/alb/variables.tf* with a reasonable default:

```
variable "alb_name" {
  description = "The name of the ALB and all its resources"
  type        = string
  default     = "terraform-up-and-running"
}
```

Next, pass this value through to the `alb` module in *examples/alb/main.tf*:

```
module "alb" {
  source = "../../modules/networking/alb"

  alb_name    = var.alb_name
  subnet_ids = data.aws_subnets.default.ids
}
```

Now, set this variable to a unique value in *test/alb_example_test.go*:

```
package test

import (
        "fmt"
        "github.com/stretchr/testify/require"

        "github.com/gruntwork-io/terratest/modules/http-helper"
        "github.com/gruntwork-io/terratest/modules/random"
        "github.com/gruntwork-io/terratest/modules/terraform"
        "testing"
        "time"
)

func TestAlbExample(t *testing.T) {
        t.Parallel()

        opts := &terraform.Options{
                // You should update this relative path to point
at your alb
                // example directory!
                TerraformDir: "../examples/alb",

                Vars: map[string]interface{}{
                        "alb_name": fmt.Sprintf("test-%s",
random.UniqueId()),
                },
        }

        // (...)
}
```

This code sets the `alb_name` var to `test-<RANDOM_ID>`, where `RANDOM_ID` is a random unique ID returned by the `random.UniqueId()` helper in Terratest. This helper returns a randomized, six-character base-62 string. The idea is that it’s a short identifier you can add to the names of most resources without hitting length-limit issues but random enough to make conflicts very unlikely (62^6

= 56+ billion combinations). This ensures that you can run a huge number of ALB tests in parallel with no concern of having a name conflict.

Make a similar change to the "Hello, World" app example, first by adding a new input variable in *examples/hello-world-app/variables.tf*:

```
variable "environment" {
  description = "The name of the environment we're deploying to"
  type        = string
  default     = "example"
}
```

Then by passing that variable through to the `hello-world-app` module:

```
module "hello_world_app" {
  source = "../../../modules/services/hello-world-app"

  server_text = "Hello, World"

  environment = var.environment

  mysql_config = var.mysql_config

  instance_type      = "t2.micro"
  min_size           = 2
  max_size           = 2
  enable_autoscaling = false
  ami                = data.aws_ami.ubuntu.id
}
```

Finally, setting `environment` to a value that includes `random.UniqueId()` in *test/hello_world_app_example_test.go*:

```
func TestHelloWorldAppExample(t *testing.T) {
        t.Parallel()

        opts := &terraform.Options{
                // You should update this relative path to point
at your
                // hello-world-app example directory!
                TerraformDir: "../examples/hello-world-
app/standalone",
                Vars: map[string]interface{}{
                        "mysql_config": map[string]interface{}{
                                "address": "mock-value-for-test",
                                "port":    3306,
                        },
                        "environment": fmt.Sprintf("test-%s",
random.UniqueId()),
                },
        }

        // (...)
}
```

With these changes complete, it should now be safe to run all your tests in parallel:

```
$ go test -v -timeout 30m

TestAlbExample 2019-05-26T17:57:21+01:00 (...)
TestHelloWorldAppExample 2019-05-26T17:57:21+01:00 (...)
TestAlbExample 2019-05-26T17:57:21+01:00 (...)
TestHelloWorldAppExample 2019-05-26T17:57:21+01:00 (...)
TestHelloWorldAppExample 2019-05-26T17:57:21+01:00 (...)

(...)

PASS
ok      terraform-up-and-running        216.090s
```

You should see both tests running at the same time so that the entire test suite takes roughly as long as the slowest of the tests, rather than the combined time of all the tests running back to back.

Note that, by default, the number of tests Go will run in parallel is equal to how many CPUs you have on your computer. So if you only have one CPU, then by default, the tests will still run serially, rather than in parallel. You can override this setting by setting the `GOMAXPROCS` environment variable or by passing the `-parallel` argument to the `go test` command. For example, to force Go to run up to two tests in parallel, you would run the following:

```
$ go test -v -timeout 30m -parallel 2
```