```
so executing stage 'teardown_db'.

(...)

PASS
ok      terraform-up-and-running        340.02s
```

Using test stages lets you get rapid feedback from your automated tests, dramatically increasing the speed and quality of iterative development. It won't make any difference in how long tests take in your CI environment, but the impact on the development environment is huge.

Retries

After you start running automated tests for your infrastructure code on a regular basis, you're likely to run into a problem: flaky tests. That is, tests occasionally will fail for transient reasons, such as an EC2 Instance occasionally failing to launch, or a Terraform eventual consistency bug, or a TLS handshake error talking to S3. The infrastructure world is a messy place, so you should expect intermittent failures in your tests and handle them accordingly.

To make your tests a bit more resilient, you can add retries for known errors. For example, while writing this book, I'd occasionally get the following type of error, especially when running many tests in parallel:

```
* error loading the remote state: RequestError: send request
failed
Post https://xxx.amazonaws.com/: dial tcp xx.xx.xx.xx:443:
connect: connection refused
```

To make tests more reliable in the face of such errors, you can enable retries in Terratest using the `MaxRetries`, `TimeBetweenRetries`, and `RetryableTerraformErrors` arguments of `terraform.Options`:

```
func createHelloOpts(
        dbOpts *terraform.Options,
        terraformDir string) *terraform.Options {
```

```
        return &terraform.Options{
                TerraformDir: terraformDir,

                Vars: map[string]interface{}{
                        "db_remote_state_bucket":
dbOpts.BackendConfig["bucket"],
                        "db_remote_state_key":
dbOpts.BackendConfig["key"],
                        "environment":
dbOpts.Vars["db_name"],
                },

                // Retry up to 3 times, with 5 seconds between
retries,
                // on known errors
                MaxRetries:         3,
                TimeBetweenRetries: 5 * time.Second,
                RetryableTerraformErrors: map[string]string{
                        "RequestError: send request failed":
"Throttling issue?",
                },
        }
}
```

In the `RetryableTerraformErrors` argument, you can specify a map of known errors that warrant a retry: the keys of the map are the error messages to look for in the logs (you can use regular expressions here), and the values are additional information to display in the logs when Terratest matches one of these errors and kicks off a retry. Now, whenever your test code hits one of these known errors, you should see a message in your logs, followed by a sleep of `TimeBetweenRetries`, and then your command will rerun:

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

(...)

Running command terraform with args [apply -input=false -
lock=false
-auto-approve]

(...)
```