then Terraform will store those credentials in your *terraform.tfstate* file, in plain text. This has been an open issue since 2014, with no clear plans for a first-class solution. There are some workarounds out there that can scrub secrets from your state files, but these are brittle and likely to break with each new Terraform release, so I don't recommend them.

For the time being, no matter which of the techniques discussed you end up using to manage secrets, you must do the following:

Store Terraform state in a backend that supports encryption

> Instead of storing your state in a local *terraform.tfstate* file and checking it into version control, you should use one of the backends Terraform supports that natively supports encryption, such as S3, GCS, and Azure Blob Storage. These backends will encrypt your state files, both in transit (e.g., via TLS) and on disk (e.g., via AES-256).

Strictly control who can access your Terraform backend

> Since Terraform state files may contain secrets, you'll want to control who has access to your backend with *at least* as much care as you control access to the secrets themselves. For example, if you're using S3 as a backend, you'll want to configure an IAM Policy that solely grants access to the S3 bucket for production to a small handful of trusted devs, or perhaps solely just the CI server you use to deploy to prod.

Plan files

You've seen the `terraform plan` command many times. One feature you may not have seen yet is that you can store the output of the plan command (the "diff") in a file:

```
$ terraform plan -out=example.plan
```

The preceding command stores the plan in a file called *example.plan*. You can then run the `apply` command on this saved plan file to ensure that Terraform applies *exactly* the changes you saw originally: