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

  # DO NOT DO THIS!!!
  access_key = "(ACCESS_KEY)"
  secret_key = "(SECRET_KEY)"
  # DO NOT DO THIS!!!
}
```

Storing credentials this way, in plain text, is *not* secure, as discussed earlier in this chapter. Moreover, it's also not practical, as this would hardcode you to using one set of credentials for all users of this module, whereas in most cases, you'll need different credentials on different computers (e.g., when different developers or your CI server runs `apply`) and in different environments (dev, stage, prod).

There are several techniques that are far more secure for storing your credentials and making them accessible to Terraform providers. Let's take a look at these techniques, grouping them based on the user who is running Terraform:

Human users

Developers running Terraform on their own computers.

Machine users

Automated systems (e.g., a CI server) running Terraform with no humans present.

Human users

Just about all Terraform providers allow you to specify your credentials in some way other than putting them directly into the code. The most common option is to use environment variables. For example, here's how you use environment variables to authenticate to AWS:

```
$  export AWS_ACCESS_KEY_ID=(YOUR_ACCESS_KEY_ID)
$  export AWS_SECRET_ACCESS_KEY=(YOUR_SECRET_ACCESS_KEY)
```

Setting your credentials as environment variables keeps plain-text secrets out of your code, ensures that everyone running Terraform has to provide their own credentials, and ensures that credentials are only ever stored in memory, and not on disk.[1]

One important question you may ask is where to store the access key ID and secret access key in the first place. They are too long and random to memorize, but if you store them on your computer in plain text, then you're still putting those secrets at risk. Since this section is focused on human users, the solution is to store your access keys (and other secrets) in a secret manager designed for personal secrets. For example, you could store your access keys in 1Password or LastPass and copy/paste them into the `export` commands in your terminal.

If you're using these credentials frequently on the CLI, an even more convenient option is to use a secret manager that supports a CLI interface. For example, 1Password offers a CLI tool called `op`. On Mac and Linux, you can use `op` to authenticate to 1Password on the CLI as follows:

```
$ eval $(op signin my)
```

Once you've authenticated, assuming you had used the 1Password app to store your access keys under the name "aws-dev" with fields "id" and "secret", here's how you can use `op` to set those access keys as environment variables:

```
$ export AWS_ACCESS_KEY_ID=$(op get item 'aws-dev' --fields 'id')
$ export AWS_SECRET_ACCESS_KEY=$(op get item 'aws-dev' --fields
'secret')
```

While tools like 1Password and `op` are great for general-purpose secrets management, for certain providers, there are dedicated CLI tools to make this even easier. For example, for authenticating to AWS, you can use the open source tool `aws-vault`. You can save your access keys using the `aws-vault add` command under a *profile* named `dev` as follows: