There are three main techniques you can use:

- Environment variables
- Encrypted files
- Secret stores

Environment variables

This first technique, which you saw back in Chapter 3, as well as earlier in this chapter when talking about providers, keeps plain-text secrets out of your code by taking advantage of Terraform's native support for reading environment variables.

To use this technique, declare variables for the secrets you wish to pass in:

```
variable "db_username" {
  description = "The username for the database"
  type        = string
  sensitive   = true
}

variable "db_password" {
  description = "The password for the database"
  type        = string
  sensitive   = true
}
```

Just as in Chapter 3, these variables are marked with `sensitive = true` to indicate they contain secrets (so Terraform won't log the values when you run `plan` or `apply`), and these variables do not have a `default` (so as not to store secrets in plain text). Next, pass the variables to the Terraform resources that need those secrets:

```
resource "aws_db_instance" "example" {
  identifier_prefix   = "terraform-up-and-running"
  engine              = "mysql"
  allocated_storage   = 10
  instance_class      = "db.t2.micro"
  skip_final_snapshot = true
  db_name             = var.db_name
```

```
  # Pass the secrets to the resource
  username = var.db_username
  password = var.db_password
}
```

Now you can pass in a value for each variable `foo` by setting the environment variable `TF_VAR_foo`:

```
$  export TF_VAR_db_username=(DB_USERNAME)
$  export TF_VAR_db_password=(DB_PASSWORD)
```

Passing in secrets via environment variables helps you avoid storing secrets in plain text in your code, but it doesn't answer an important question: How do you store the secrets securely? One nice thing about using environment variables is that they work with almost any type of secrets management solution. For example, one option is to store the secrets in a personal secrets manager (e.g., 1Password) and manually set those secrets as environment variables in your terminal. Another option is to store the secrets in a centralized secret store (e.g., HashiCorp Vault) and write a script that uses that secret store's API or CLI to read those secrets out and set them as environment variables.

Using environment variables has the following advantages:

- Keep plain-text secrets out of your code and version control system.
- Storing secrets is easy, as you can use just about any other secret management solution. That is, if your company already has a way to manage secrets, you can typically find a way to make it work with environment variables.
- Retrieving secrets is easy, as reading environment variables is straightforward in every language.
- Integrating with automated tests is easy, as you can easily set the environment variables to mock values.