```
$ aws-vault add dev
Enter Access Key Id: (YOUR_ACCESS_KEY_ID)
Enter Secret Key: (YOUR_SECRET_ACCESS_KEY)
```

Under the hood, `aws-vault` will store these credentials securely in your operating system's native password manager (e.g., Keychain on macOS, Credential Manager on Windows). Once you've stored these credentials, now you can authenticate to AWS for any CLI command as follows:

```
$ aws-vault exec <PROFILE> -- <COMMAND>
```

where `PROFILE` is the name of a profile you created earlier via the `add` command (e.g., `dev`) and `COMMAND` is the command to execute. For example, here's how you can use the `dev` credentials you saved earlier to run `terraform apply`:

```
$ aws-vault exec dev -- terraform apply
```

The `exec` command automatically uses AWS STS to fetch temporary credentials and exposes them as environment variables to the command you're executing (in this case, `terraform apply`). This way, not only are your permanent credentials stored in a secure manner (in your operating system's native password manager), but now, you're also only exposing temporary credentials to any process you run, so the risk of leaking credentials is minimized. `aws-vault` also has native support for assuming IAM roles, using multifactor authentication (MFA), logging into accounts on the web console, and more.

Machine users

Whereas a human user can rely on a memorized password, what do you do in cases where there's no human present? For example, if you're setting up a continuous integration / continuous delivery (CI/CD) pipeline to automatically run Terraform code, how do you securely authenticate that pipeline? In this case, you are dealing with authentication for a *machine user*. The question is, how do you get one machine (e.g., your CI server) to