Lab: Secure Secrets in Terraform Code

When working with Terraform, it's very likely you'll be working with sensitive values. This lab goes over the most common techniques you can use to safely and securely manage such secrets.

- Task 1: Do Not Store Secrets in Plain Text
- Task 2: Mark Variables as Sensitive
- Task 3: Environment Variables
- Task 4: Secret Stores (e.g., Vault, AWS Secrets manager)

Task 1: Do Not Store Secrets in Plain Text

Never put secret values, like passwords or access tokens, in .tf files or other files that are checked into source control. If you store secrets in plain text, you are giving the bad actors countless ways to access sensitive data. Ramifications for placing secrets in plain text include:

- Anyone who has access to the version control system has access to that secret.
- Every computer that has access to the version control system keeps a copy of that secret
- Every piece of software you run has access to that secret.
- No way to audit or revoke access to that secret.

Task 2: Mark Variables as Sensitive

The first line of defense here is to mark the variable as sensitive so Terraform won't output the value in the Terraform CLI. Remember that this value will still show up in the Terraform state file:

In your `variables.tf file`, add the following code:

```
variable "phone_number" {
  type      = string
  sensitive = true
  default   = "867-5309"
}

output "phone_number" {
  value     = var.phone_number
  sensitive = true
}
```


Run a `terraform apply` to see the results of the sensitive variable. Notice how Terraform marks this as sensitive.

Task 3: Environment Variables

Another way to protect secrets is to simply keep plain text secrets out of your code by taking advantage of Terraform's native support for reading environment variables. By setting the `TF_VAR_<name>` environment variable, Terraform will use that value rather than having to add that directly to your code.

In your `variables.tf` file, modify the `phone_number` variable and remove the default value so the sensitive value is no longer in cleartext:

variable "phone_number" { type = string sensitive = true }

In your terminal, export the following environment variable and set the value:

```
export TF_VAR_phone_number="867-5309"
```

Note: If you are still using Terraform Cloud as your remote backend, you will need to set this environment variable in your Terraform Cloud workspace instead.

Now, run a `terraform apply` and see that the plan runs just the same, since Terraform picked up the value of the sensitive variable using the environment variable. This strategy prevents us from having to add the value directly in our Terraform files and likely being committed to a code repository.

Task 4: Inject Secrets into Terraform using HashiCorp Vault

Another way to protect your secrets is to store them in secrets management solution, like HashiCorp Vault. By storing them in Vault, you can use the Terraform Vault provider to quickly retrieve values from Vault and use them in your Terraform code.

Download HashiCorp Vault for your operating system at vaultproject.io. Make sure the binary is moved to your $PATH so it can be executed from any directory. For help, check out https://www.vaultproject.io/docs/install. Alternatively, you can use Homebrew (MacOS) or Chocolatey (Windows). There are also RPMs available for Linux.

Validate you have Vault installed by running:

```
vault version
```


You should get back the version of Vault you have downloaded and installed.

In your terminal, run `vault server -dev` to start a Vault dev server. This will launch Vault in a pre-configured state so we can easily use it for this lab. Note that you should never run Vault in a production deployment by starting it this way.

Open a second terminal, and set the `VAULT_ADDR` environment variable. By default, this is set to HTTPS, but since we're using a dev server, TLS is not supported.

```
export VAULT_ADDR="http://127.0.0.1:8200"
```

Now, log in to Vault using the root token from the output of our Vault dev server. An example is below, but your root token and unseal key will be different:

```
WARNING! dev mode is enabled! In this mode, Vault runs entirely in-memory
and starts unsealed with a single unseal key. The root token is already
authenticated to the CLI, so you can immediately begin using Vault.

You may need to set the following environment variable:

    $ export VAULT_ADDR='http://127.0.0.1:8200'

The unseal key and root token are displayed below in case you want to
seal/unseal the Vault or re-authenticate.

Unseal Key: 1zqTTWCHyAvEhTqOLurR2nQPeeoR1Sk2FMp95fRNEaU=
Root Token: s.Oi1tQPY98uwWQ6HOf9T7Elkg

Development mode should NOT be used in production installations!
```

Log in to Vault using the following command:

```
vault login <root token>
```

Now that we are logged into Vault, we can quickly add our sensitive values to be stored in Vault's KV store. Use the following command to write the sensitive value to Vault:

```
vault kv put /secret/app phone_number=867-5309
```

Back in Terraform, let's add the code to use Vault to retrieve our secrets. Create a new directory called `vault` and add a `main.tf` file. In your `main.tf` file, add the following code:

```
provider "vault" {
  address = "http://127.0.0.1:8200"
  token = <root token>
}
```


By the way, note that I am only using the root token as an example here. Root tokens should NEVER be used on a day-to-day basis, nor should you use a root token for Terraform access. Please use a different auth method, such as AppRole, which is a better solution to establish connectivity between Terraform and Vault.

Now, add the following data block, which will use the Vault provider and token to retrieve the sensitive values we need:

```
data "vault_generic_secret" "phone_number" {
  path = "secret/app"
}
```

Finally, let's an a new output block that uses the data retrieved from Vault. In your main.tf, add the following code:

```
output "phone_number" {
  value = data.vault_generic_secret.phone_number
}
```

Run a `terraform init` and a `terraform apply`. Notice that Terraform is smart enough to understand that since the value was retrieved from Vault, it needs to be marked as sensitive since it likely contains sensitive information.

Add the `sensitive` configuration to the output block as follows:

```
output "phone_number" {
  value = data.vault_generic_secret.phone_number
  sensitive = true
}
```

Run a `terraform apply` again so Terraform retrieves that data from Vault. Note that the output will be shown as sensitive, but we can still easily display the data. In the terminal, run the following command to display the sensitive data retrieved from Vault:

```
terraform output phone_number
```

If you need ONLY the value of the data retrieved, rather than both the key and value and related JSON, you can update the output to the following:

```
output "phone_number" {
  value = data.vault_generic_secret.phone_number.data["phone_number"]
  sensitive = true
}
```

Run a `terraform output phone_number` again to see that the value is now ONLY the

phone_number string, rather than the JSON output as we saw before.