Lab: Terraform Cloud Secure Variables

Terraform Cloud has built in support for encryption and storage of variables used within your Terraform configuration. This allows you to centrally manage variables per workspace or organization as well as store sensitive items (such as cloud credentials, passwords, etc.) securely during the provisioning process without exposing them in plaintext or storing them on someone's laptop.

This lab demonstrates how to store variables to Terraform Cloud.

- Task 1: Utilize Terraform Cloud Variables
- Task 2: Run a plan locally
- Task 3: Change a variable
- Task 4: Run an apply remotely
- Task 5: Utilize Variable sets

This lab demonstrates how to store variables in Terraform Cloud.

Task 1: Utilize Terraform Cloud Variables

Now that we have our state stored in Terraform Cloud in our workspace, we will take the next logical step and store our variables in Terraform Cloud

Step 1.1 - Create Terraform Variable

> Note: If your infrastructure has not yet been deployed, execute a `terraform apply` to deploy the infrastructure.

- Navigate to your Terraform Cloud `my-aws-app` in the UI.
- Once there, navigate to the `Variables` tab.
- In the `Variables` tab, you can add variables related to the state file that was previously migrated.
- To do this, first select the + `Add variable` button
- Let's add a Terraform variable named `aws_region` with a value of `us-east-1`
- Let's add a second Terraform variable named `vpc_name` with a value of `demo_vpc`

Step 1.2 - Create Environment Variable

Terraform requires credentials in order to communicate with your cloud provider's API. These API keys should never be stored directly in your terraform code. We will use Terraform Cloud environment

variables to store our sensitive cloud credentials for AWS.

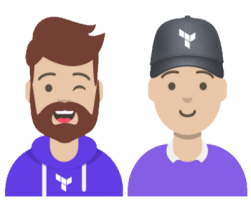
- In the `Variables` tab of your `my-aws-app` you can add environment variables for the AWS Credentials
- To do this, first select the `+ Add variable` button
- Lets add a Environment variable named `AWS_ACCESS_KEY_ID` with your AWS Access Key
- Lets add a second Environment variable named `AWS_SECRET_ACCESS_KEY` with your AWS Secret Key. Be sure to mark this variable as sensitive. Sensitive variables will not be displayed within the environment, and can only be overwritten - not read.

Task 2: Run a plan locally

Step 2.1 - Run a terraform init

- Next reinitialize your terraform project locally and run a `terraform plan` to validate that refactoring your code to make use of variables within TFC did not introduce any planned infrastructure changes. This can be confirmed by validating a zero change plan.

```
terraform plan

Running plan in the remote backend. Output will stream here. Pressing Ctrl
    -C
will stop streaming the logs, but will not stop the plan running remotely.

Preparing the remote plan...

To view this run in a browser, visit:
https://app.terraform.io/app/Enterprise-Cloud/server-build/runs/run-
    DGXauYrWeB1xwwPx

Waiting for the plan to start...

Terraform v0.14.8
Configuring remote state backend...
Initializing Terraform configuration...
module.keypair.tls_private_key.generated: Refreshing state... [id=34
    a0559a16dc68108d30a76d9a5a7b25f8885e1e]
module.keypair.local_file.public_key_openssh[0]: Refreshing state... [id=0
    e346a51831a9bc96fd9ea142f8c35b4e1ade12b]
module.keypair.local_file.private_key_pem[0]: Refreshing state... [id=
    b7ac3f7125c4e3681fd38539b220c1abf01d1254]
module.keypair.null_resource.chmod[0]: Refreshing state... [id
    =1854006565944356631]
```

```
module.keypair.aws_key_pair.generated: Refreshing state... [id=terraform-
    nyl-ant-key]
module.server[0].aws_instance.web[1]: Refreshing state... [id=i-047
    d4d406e22e87f9]
module.server[1].aws_instance.web[0]: Refreshing state... [id=i-003
    fcdb877e575b26]
module.server[0].aws_instance.web[0]: Refreshing state... [id=i-0
    d16b6f6eda6b834e]
module.server[1].aws_instance.web[1]: Refreshing state... [id=i-03
    be3bf6ee29f5633]

No changes. Infrastructure is up-to-date.

This means that Terraform did not detect any differences between your
configuration and real physical resources that exist. As a result, no
actions need to be performed.
```

Task 3: Change a Variable

Next, we will be testing to make sure that our variables in TFC are working correctly.

Step 3.1 - Modify the vpc name by changing the `vpc_name` variable in TFC

- Simply click on the `***` button to the right of your variables, and then select the `Edit` option
- Change the value of your `vpc_name` variable and save to apply a new name for your VPC.
- Run a `terraform apply` locally in your code editor.
- When this is ran locally, you should see that Terraform is looking at the variable in TFC, and it will update your server tags with the `vpc_name` value.

Task 4: Run an apply remotely

Now that we've tested that our configuration is still working and that our variables can be stored and manipulated withing Terraform Cloud, let's also test an apply within Terraform Cloud itself.

Step 4.1 - Queue a plan in Terraform Cloud

Now that your variable value has changed, let's queue the plan in Terraform Cloud

- Navigate to the `Actions` box in the top right corner and choose `Start new plan`

- In the `Reason for starting plan` dialogue box, type something meaningful to your use case. For this, we can simply say `Updating VPC Name`.
- In the `Choose plan type` option, make sure that `Plan (most common)` is selected
- Follow the dialogue boxes through this process. Review your plan to ensure that it will be doing exactly what you expect, and then confirm and apply when given the opportunity.

Task 5: Utilize Variable sets

Variable sets allow Terraform Cloud users to reuse both Terraform-defined and environment variables not just from root to child modules, but across certain workspaces or an entire organization. One of the most common use cases for variable sets is credential and identity management. Variable sets allow for a convienent way to reuse variables across workspaces within a Terraform Cloud organization.

- In the `Settings` tab of your Terraform Cloud Organization go to `Variable sets`.
- Select the `Create Variable set` button
- Give your variable set a name - `AWS Credentials` and description.
- Apply the variable set to your `my-aws-app` workspace.
- You can add environment variables for the AWS Credentials
- To do this, first select the + `Add variable` button
- Lets add a Environment variable named `AWS_ACCESS_KEY_ID` with your AWS Access Key
- Lets add a second Environment variable named `AWS_SECRET_ACCESS_KEY` with your AWS Secret Key. Be sure to mark this variable as sensitive.

Now you can use these AWS credentials on any new workspace you create in Terraform Cloud by adding the variable set to the workspace. While storing cloud credentials is a common use of variable sets, they are not limited to credentials. Variable sets can be utlized for any set of variables that you want to reuse accross workspaces within the organization.

Congratulations! You're now storing your variable definitions remotely and can control the behavior of your Terraform code by adjusting their values. With Terraform Cloud you are able to centralize and secure the variable definintions for your workspace.