**INJECT SECRETS INTO TERRAFORM USING THE VAULT PROVIDER**

Traditionally, developers looking to safely provision infrastructure using Terraform are given their own set of long-lived, scoped AWS credentials. While this enables the developer's freedom, using long-lived credentials can be dangerous and difficult to secure.

1. Operators need to manage a large number of static, long-lived AWS IAM credentials with varying scope.
2. Long-lived credentials on a developer's local machine creates a large attack surface area. If a malicious actor gains access to the credentials, they could used them to damage resources.

You can address both concerns by storing your long-lived AWS credentials in HashiCorp's Vault's AWS Secrets Engine, then leverage [Terraform's Vault provider](https://www.terraform.io/docs/providers/vault/) to generate appropriately scoped & short-lived AWS credentials to be used by Terraform to provision resources in AWS.

As a result, operators (Vault Admin) are able to avoid managing static, long-lived secrets with varying scope and developers (Terraform Operator) are able to provision resources without having direct access to the secrets.

In this guide, you assume the role of both the **Vault Admin** and the **Terraform Operator**.

1. First, as an Vault Admin, you will configure AWS Secrets Engine in Vault.
2. Then, as a Terraform Operator, you will use connect to the Vault instance to retrieve dynamic, short-lived AWS credentials generated by the AWS Secrets Engine to provision an Ubuntu EC2 instance.
3. Finally, as an Vault Admin, you will remove the Terraform Operator's ability to manipulate EC2 instances by modifying the policy for the corresponding Vault role.

## **PREREQUISITES**

**Terraform installation**

Install Terraform on Ubuntu 18.04

To update the system and packages, you can use the built-in software updater, or manually update the system with:

$ sudo apt-get update

Again, we will install wget and unzip packages if they’re not already installed:

$ sudo apt-get install wget unzip

Also next, we will run the same commands as we did with CentOS 7:

$ wget https://releases.hashicorp.com/terraform/0.13.0/terraform\_0.13.0\_linux\_amd64.zip

$ sudo unzip ./terraform\_0.13.0\_linux\_amd64.zip -d /usr/local/bin/

And finally, to test if our installation was successful:

$ terraform -v

**Vault Installation**

Add the HashiCorp [GPG key](https://apt.releases.hashicorp.com/gpg).

**$** curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -

Add the official HashiCorp Linux repository.

**$** sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb\_release -cs) main"

Update and install.

**$** sudo apt-get update && sudo apt-get install vault

After installing Vault, verify the installation worked by opening a new terminal session and checking that the vault binary is available. By executing vault, you should see help output similar to the following:

**$** vault

Usage: vault <command> [args]

Common commands:

read Read data and retrieves secrets

write Write data, configuration, and secrets

delete Delete secrets and configuration

list List data or secrets

login Authenticate locally

server Start a Vault server

status Print seal and HA status

unwrap Unwrap a wrapped secret

Other commands:

audit Interact with audit devices

auth Interact with auth methods

lease Interact with leases

operator Perform operator-specific tasks

path-help Retrieve API help for paths

policy Interact with policies

secrets Interact with secrets engines

ssh Initiate an SSH session

token Interact with tokens

**START VAULT SERVER**

We have to start the vault server and also need to enable the UI of the server, inorder to do so,

Vi config.hcl

ui = true  
disable\_mlock = true

storage "inmem"{  
    
}

listener "tcp" {  
  address     = "0.0.0.0:8200"  
  tls\_disable = 1  
}

api\_addr = "http://10.44.0.72:8200"

cluster\_addr = <https://vault-0.vault-internal:8201>

Start the vault server in dev mode as follows:

vault server -dev -config=config.hcl

To check the vault is running, type the following url and login into the instance using your root token

<http://13.212.74.73:8200>

**TERRAFORM**

git clone https://github.com/hashicorp/learn-terraform-inject-secrets-aws-vault && cd learn-terraform-inject-secrets-aws-vault

Navigate to the Vault Admin directory.

**$** cd vault-admin-workspace

In the main.tf file, you will find 2 resources:

1. the [*vault\_aws\_secret\_backend.aws*](https://github.com/hashicorp/learn-terraform-inject-secrets-aws-vault/blob/master/vault-admin-workspace/main.tf#L13) resource configures AWS Secrets Engine to generate a dynamic token that lasts for 2 minutes.
2. the [*vault\_aws\_secret\_backend\_role.admin*](https://github.com/hashicorp/learn-terraform-inject-secrets-aws-vault/blob/master/vault-admin-workspace/main.tf#L22) resource configures a role for the AWS Secrets Engine named dynamic-aws-creds-vault-admin-role with an IAM policy that allows it iam:\* and ec2:\* permissions.

This role will be used by the Terraform Operator workspace to dynamically generate AWS credentials scoped to this IAM policy.

Before applying this configuration, set the required Terraform variable substituting <AWS\_ACCESS\_KEY\_ID> and <AWS\_SECRET\_ACCESS\_KEY> with your AWS Credentials. Notice that we're also setting the required [Vault Provider arguments](https://www.terraform.io/docs/providers/vault/index.html#provider-arguments) as environment variables: VAULT\_ADDR & VAULT\_TOKEN.

**$** export TF\_VAR\_aws\_access\_key=<AWS\_ACCESS\_KEY\_ID>

**$** export TF\_VAR\_aws\_secret\_key=<AWS\_SECRET\_ACCESS\_KEY>

**$** export VAULT\_ADDR=http:// 3.94.98.140:8200

**$** export VAULT\_TOKEN= s.9ywq6OhmUyrGxl3e6m74dRbp

Initialize the Vault Admin workspace.

**$** terraform init

In your initialized directory, run terraform apply, review the planned actions, and confirm the run with a yes

**$** terraform apply

**#**# ... Output truncated ...

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

The state of your infrastructure has been saved to the path

below. This state is required to modify and destroy your

infrastructure, so keep it safe. To inspect the complete state

use the `terraform show` command.

State path: terraform.tfstate

Outputs:

backend = dynamic-aws-creds-vault-admin-path

role = dynamic-aws-creds-vault-admin-role

Notice that there are two output variables named backend and role. These output variables will be used by the Terraform Operator workspace in a later step.

Now that you have successfully configured Vault's AWS Secrets Engine, you can retrieve dynamic short lived AWS token to provision an EC2 instance.

Navigate to the Terraform Operator workspace.

**$** cd ../operator-workspace

In the main.tf file, you should find the following data and resource blocks:

1. the [terraform\_remote\_state.admin](https://github.com/hashicorp/learn-terraform-inject-secrets-aws-vault/blob/master/operator-workspace/main.tf#L12) data block retrieves the Terraform state file generated from your Vault Admin workspace

2.the [vault\_aws\_access\_credentials.creds](https://github.com/hashicorp/learn-terraform-inject-secrets-aws-vault/blob/master/operator-workspace/main.tf#L20) data block retrieves the dynamic, short-lived AWS credentials from your Vault instance. Notice that this uses the Vault Admin workspace's output variables: backend and role

Initialize the Terraform Operator workspace.

**$** terraform init

Apply the Terraform configuration, remember to confirm the run with a yes. Terraform will provision the EC2 instance using the dynamic credentials generated from Vault.

**$** terraform apply

Refresh the IAM Users and search for the vault-token-terraform-dynamic-aws-creds-vault-admin prefix. You should see a IAM user.

This IAM user was generated by Vault with the appropriate IAM policy configured by the Vault Admin workspace. Because the default\_lease\_ttl\_seconds is set to 120 seconds, Vault will revoke those IAM credentials and they will be removed from the AWS IAM console after 120 seconds.

**$** export VAULT\_ADDR=http://54.89.202.254:8200

**$** export VAULT\_TOKEN=s.NXaBYyDTWfstGtJ67LdOpXcp

vault\_2 (54.89.202.254) | internal: (10.44.0.115)  
- Initialized and unsealed.  
- The root token and recovery key is stored in /tmp/key.json.  
- K/V-V2 secret engine enabled and secret stored.  
- Leader of HA clusters

$ ssh -l ubuntu 54.89.202.254 -i oct19.pem

# Root token:  
$ ssh -l ubuntu 54.89.202.254 -i oct19.pem "cat ~/root\_token"  
# Recovery key:  
$ ssh -l ubuntu 54.89.202.254 -i oct19.pem "cat ~/recovery\_key"

vault\_3 (3.85.147.188) | internal: (10.44.0.236)  
- Started  
- You will join it to cluster started by vault\_2

$ ssh -l ubuntu 3.85.147.188 -i oct19.pem

vault\_4 (3.237.195.243) | internal: (10.44.1.30)  
- Started  
- You will join it to cluster started by vault\_2

$ ssh -l ubuntu 3.237.195.243 -i oct19.pem