**Terraform & HashiCorp Vault Integration: Seamless Secrets Management**



**what is Hashicorp vault.?**

Imagine HashiCorp Vault as a secure digital vault for all your sensitive information like passwords, API keys, and encryption certificates. It acts like a central location to store, access, and manage these secrets. Here’s why it’s useful:

**Strong Security:** Vault encrypts your secrets and controls access with different permission levels. This minimizes the 1. risk of unauthorized access and keeps your sensitive information safe.

2. **Centralized Management:** No more scattered secrets! Vault keeps everything in one place, making it easier to control and audit who can access what.

3. **Flexibility:** It can store various secrets, from database credentials to cloud API keys. Vault also integrates with different tools and platforms you might already be using.

Overall, using Vault helps organizations improve security, simplify secret management, and streamline access control for critical information. While Vault has a free open-source version with limited features, most businesses opt for the paid edition with additional functionalities like advanced auditing and disaster recovery

1. In today’s demo, we will explore Terraform Vault integration with a real example and troubleshooting. Let’s get started!

2. Launch one EC2 instance with basic configuration and access it.

3. Then update your machine and install GPG as well.

sudo apt update && sudo apt install gpg

Then download the signing key to a new keyring.

wget -O- <https://apt.releases.hashicorp.com/gpg> | sudo gpg — dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

Then verify the key’s fingerprint.

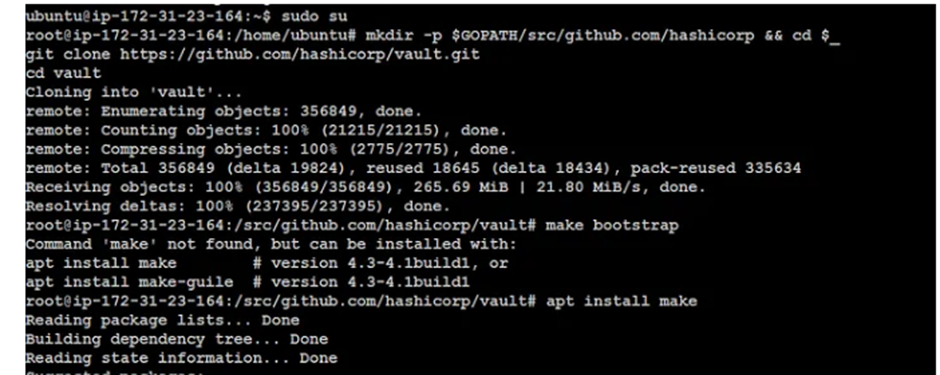
gpg — no-default-keyring — keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg –fingerprint

Clone the Vault repository from GitHub.

mkdir -p $GOPATH/src/github.com/hashicorp && cd $\_

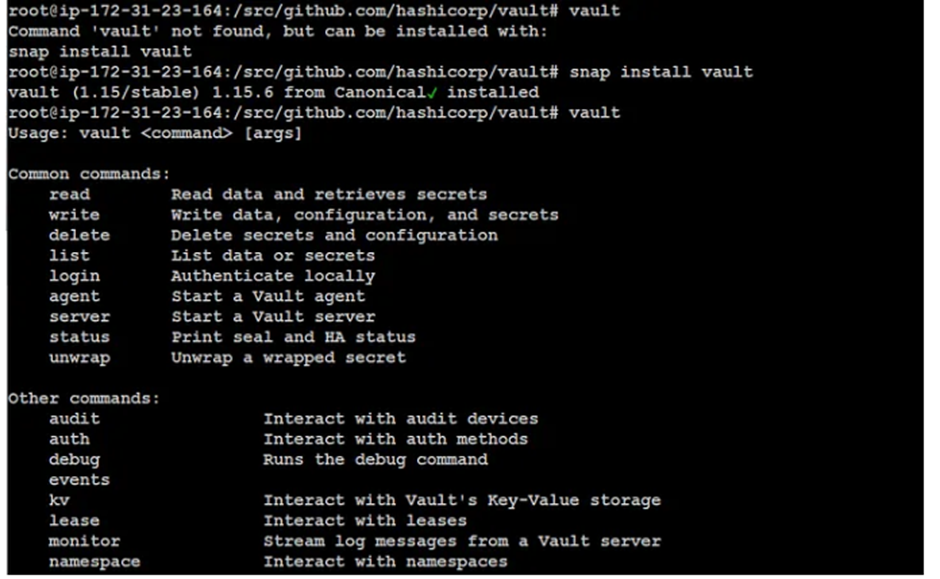
git clone <https://github.com/hashicorp/vault.git>

cd vault



Then install Vault.

snap install vault



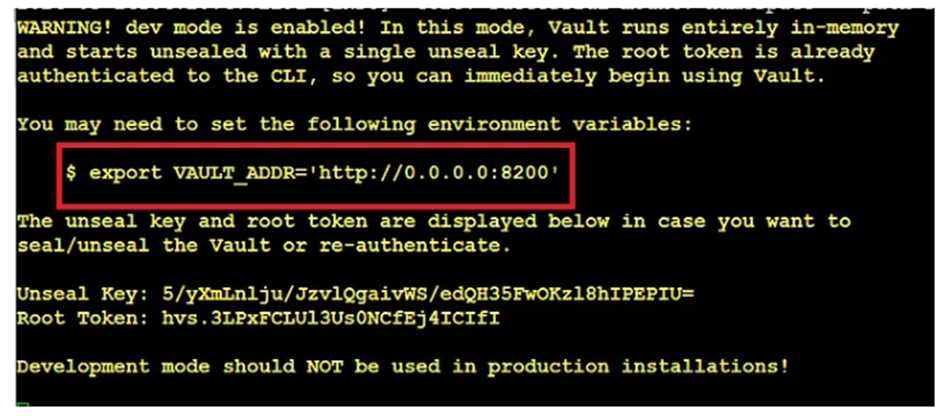
To start Vault, you can use the following command:

vault server -dev -dev-listen-address=”0.0.0.0:8200"



You may need to set the following environment variables. To do so, create a duplicate session and copy the command, then paste it into the new session.

export VAULT\_ADDR=’http://0.0.0.0:8200'

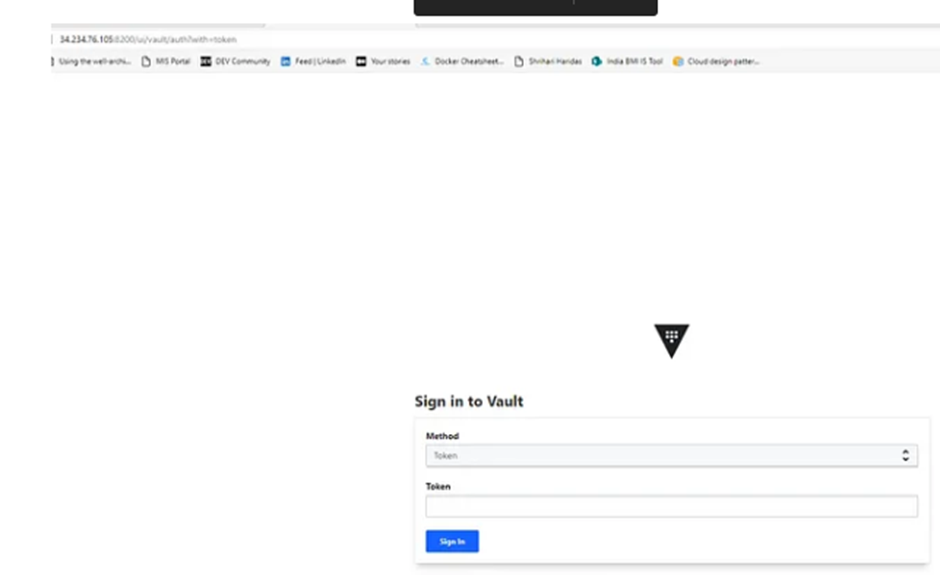


$ export VAULT\_ADDR=’http://0.0.0.0:8200’

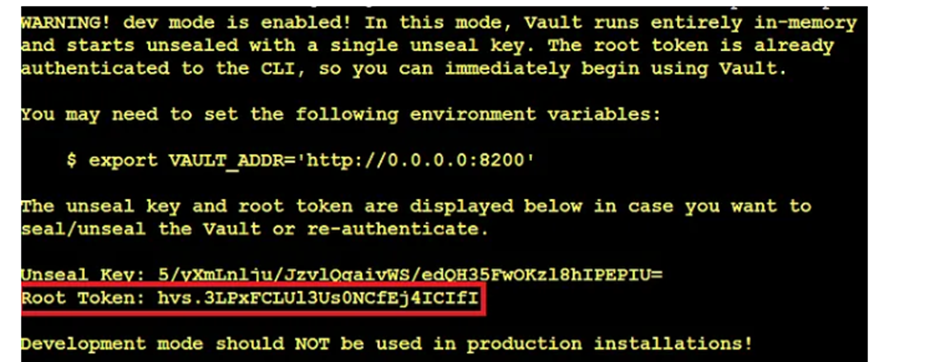
$ export VAULT\_TOKEN=””

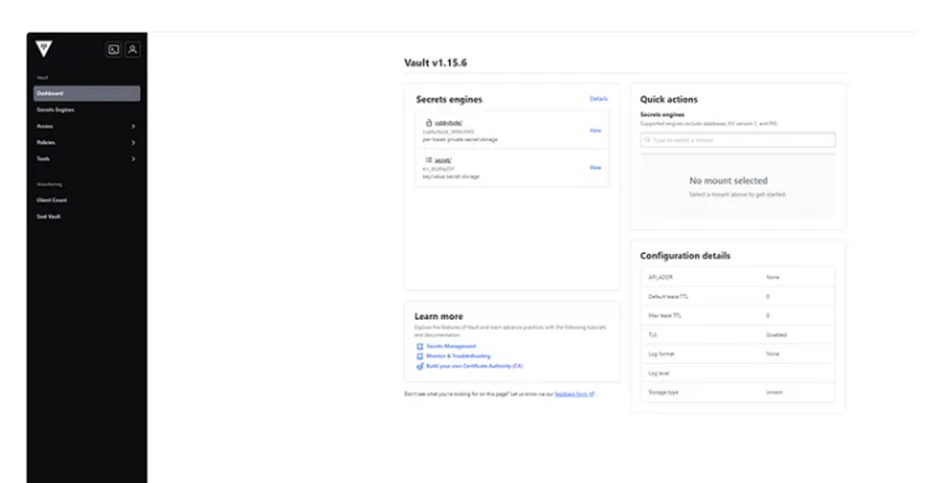
Vault status

And then copy the public IP and paste it into your browser’s address bar followed by port 8200. You will see the login page appear.

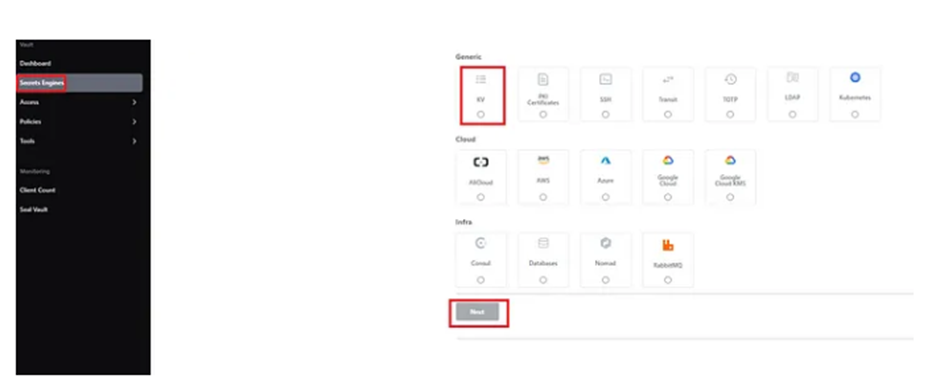


For logging in as root, select ‘Token’ as the default method. Copy the token from your server when you start Vault; it will be displayed to you. After that, you will see the dashboard.

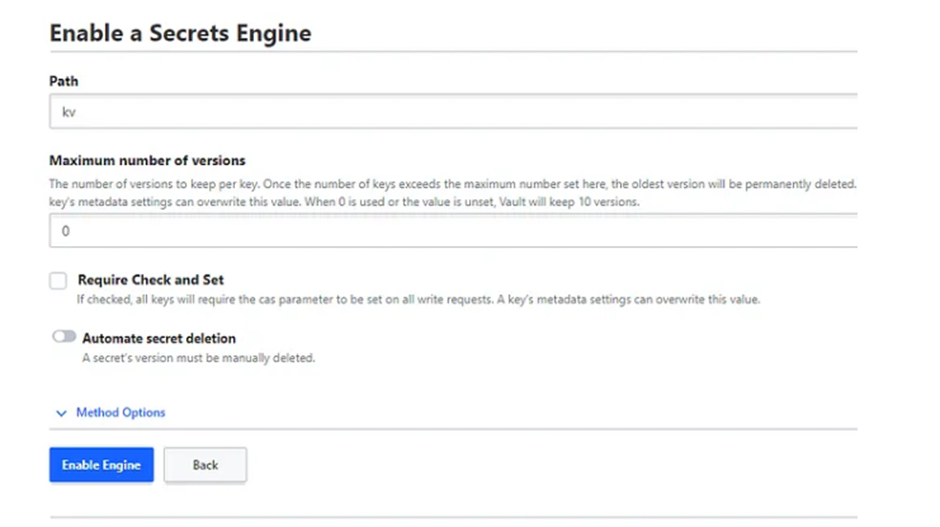


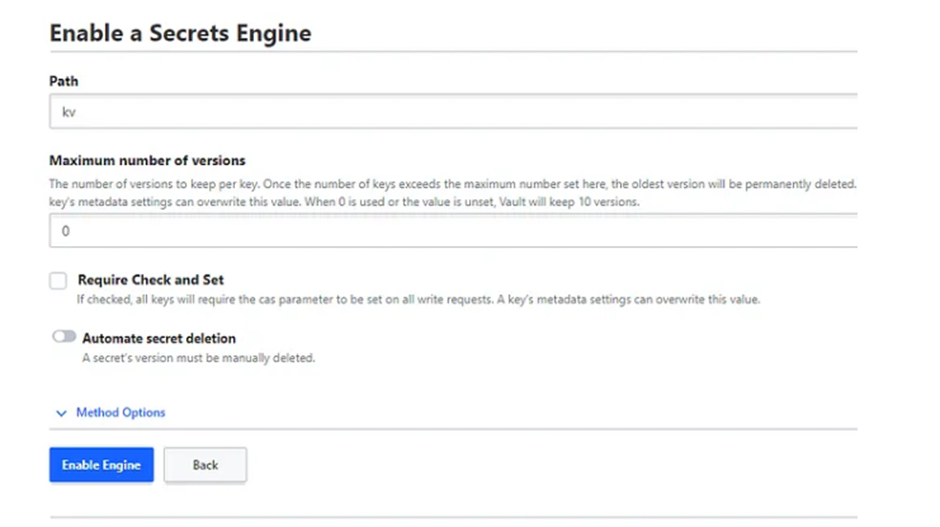


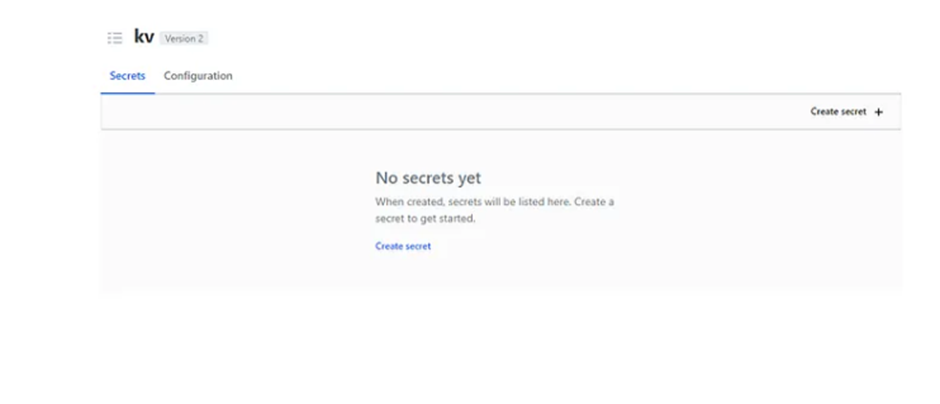
Then, in this demo, we will store the secret as ‘Key & value’. To do so, click on ‘Secret Engines’, then select ‘KV’ which stands for ‘Key/value’, and click on ‘Next’.



Next, on the following window, provide a path and click ‘Next’ to proceed, where you will notice that no credentials are created or stored.

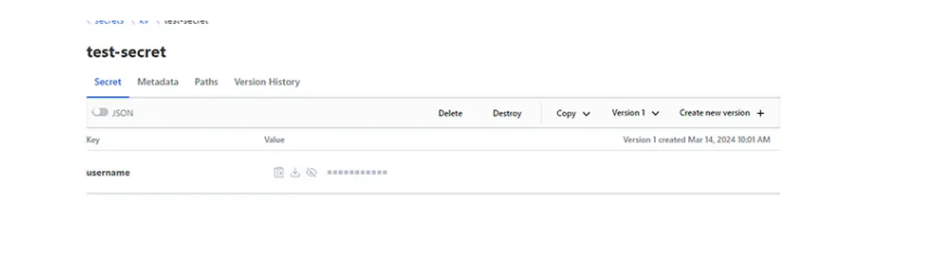






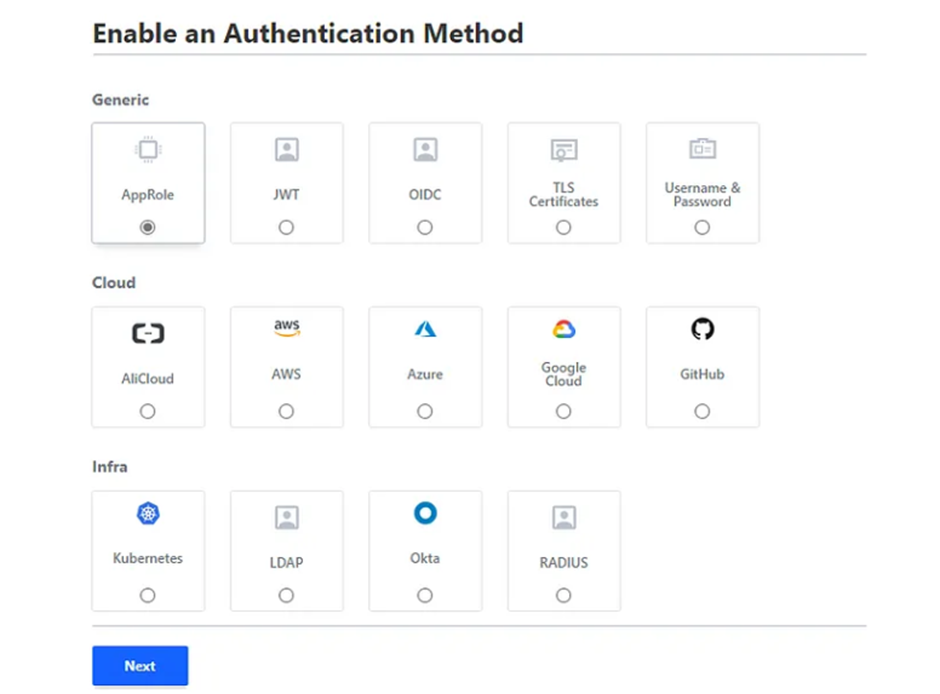
Now, click on ‘Create Secret’ and input your secret. I will use ‘Username & password’ as an example.





Then, to access these credentials in Terraform, you need to assign policies or roles similar to IAM. Click on ‘Policies’, then click on ‘Enable New Method’, and select ‘App Role’ as the authentication method. This is similar to IAM. Enable the role. Using this App Role,

I will authenticate Terraform or whichever tool you are using.





Then, create a duplicate session of your machine and run the following command on it.

18. Because HashiCorp Vault does not support the creation of users in the UI, go to the console and create a user. It’s very easy. Before creating the role, we need to create a policy for that role. This policy enables Terraform to access the ‘kv’ and ‘secret’ folders in Vault.

vault policy write terraform - <<EOF

path "\*" {

capabilities = ["list", "read"]

}

path "secrets/data/\*" {

capabilities = ["create", "read", "update", "delete", "list"]

}

path "kv/data/\*" {

capabilities = ["create", "read", "update", "delete", "list"]

}

path "secret/data/\*" {

capabilities = ["create", "read", "update", "delete", "list"]

}

path "auth/token/create" {

capabilities = ["create", "read", "update", "list"]

}

EOF

vault write auth/approle/role/terraform \

secret\_id\_ttl=10m \

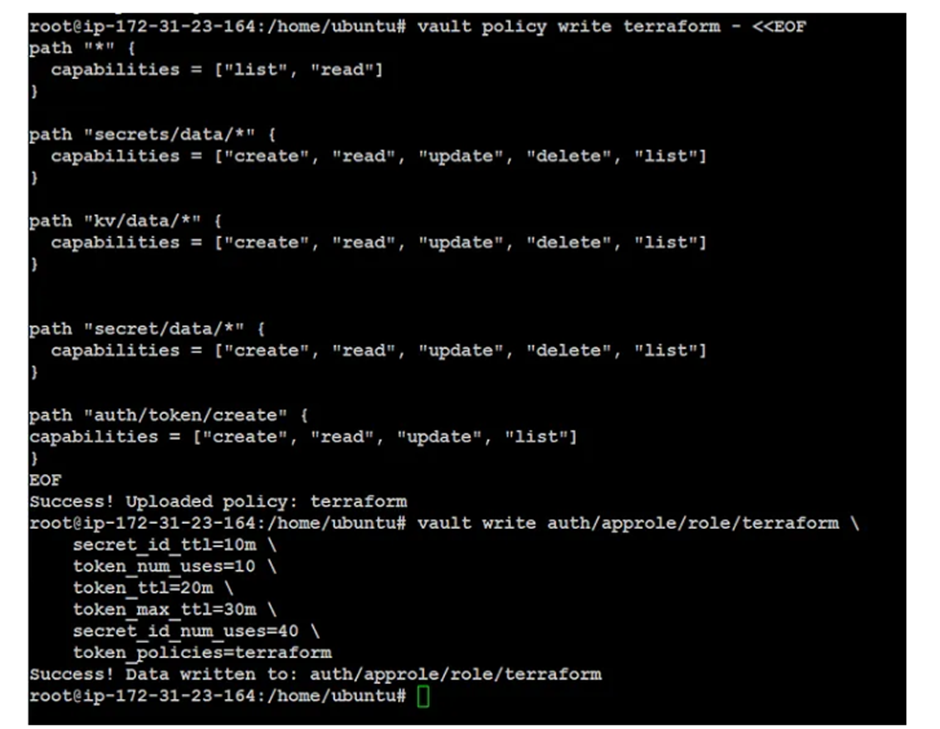
token\_num\_uses=10 \

token\_ttl=20m \

token\_max\_ttl=30m \

secret\_id\_num\_uses=40 \

token\_policies=terraform



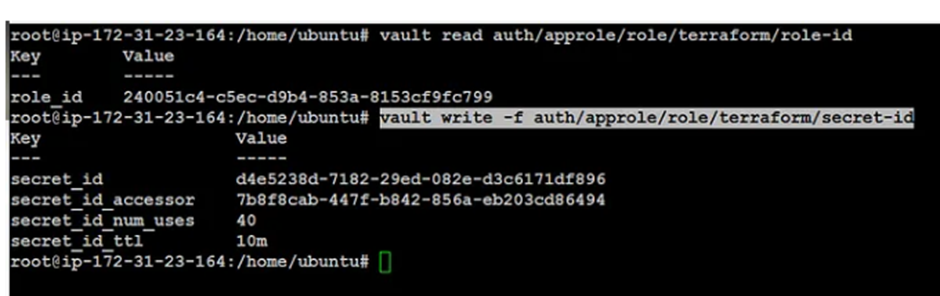
vault auth list

vault auth enable approle

Similar to AWS, in Vault, we have Role ID and Secret ID. This is sensitive information, so do not share it with anyone.

vault read auth/approle/role/terraform/role-id

vault write -f auth/approle/role/terraform/secret-id



Now that we are done with Vault, the next step is to write down the Terraform project and check whether Terraform is able to read the secret from Vault.

#main.tf

provider “aws” {

region = “us-east-1”

}

provider “vault” {

address = “http://<your-ip\_address>:8200”

skip\_child\_token = true

auth\_login {

path = “auth/approle/login”

parameters = {

role\_id = “<your-role\_id>”

secret\_id = “<your-secret\_id>”

}

}

}

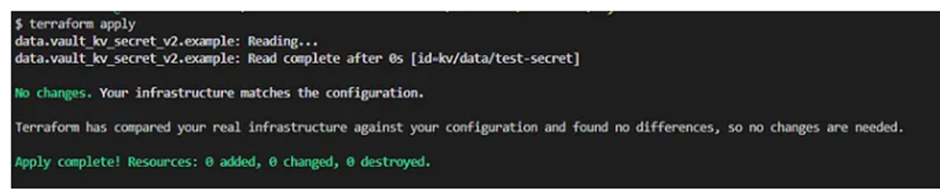
data “vault\_kv\_secret\_v2” “example” {

mount = “kv” // change it according to your mount

name = “test-secret” // change it according to your secret

}

Then, to check if Terraform is able to retrieve the credentials from Vault, run the ‘terraform init’ and ‘terraform apply’ commands.



Now, let’s add the block to create an EC2 instance and apply it again. You will see that in the EC2 tag, our password will be taken. #main.tf

provider “aws” {

region = “us-east-1”

}

provider “vault” {

address = “http://<your-ip\_address>:8200”

skip\_child\_token = true

auth\_login {

path = “auth/approle/login”

parameters = {

role\_id = “<your-role\_id>”

secret\_id = “<your-secret\_id>”

}

}

}

data “vault\_kv\_secret\_v2” “example” {

mount = “kv” // change it according to your mount

name = “test-secret” // change it according to your secret

}

resource “aws\_instance” “example” {

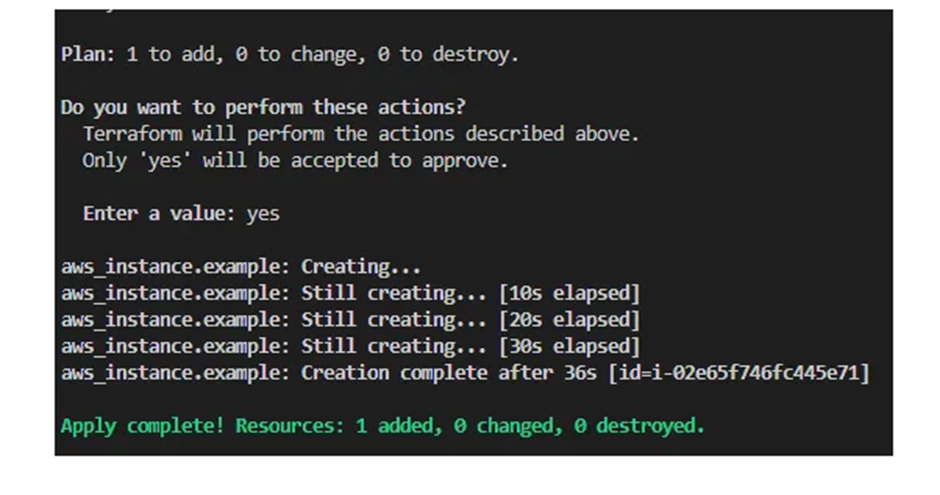
ami = “ami-0c7217cdde317cfec”

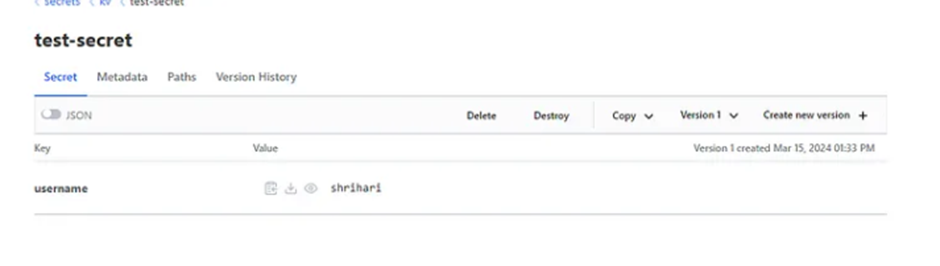
instance\_type = “t2.micro”

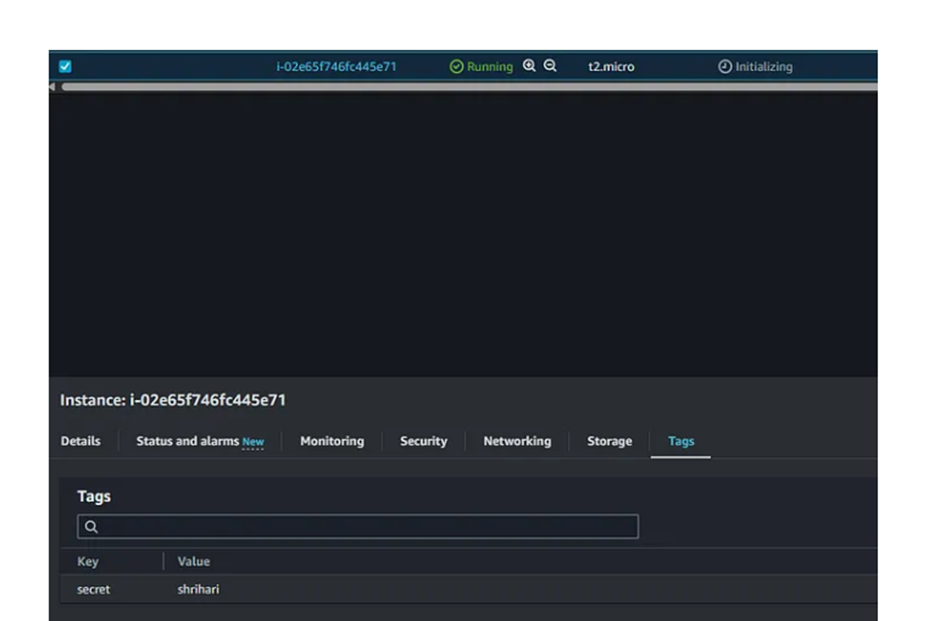
tags = {

secret = data.vault\_kv\_secret\_v2.example.data[“username”]

}







If you encounter the following error while running ‘terraform apply’ command, then run it again to recreate ‘role-id’ and ‘secret-id’ with the same command as mentioned in step no. 20. After that, paste the new credentials into the code. This will solve your problem.



3