KPLABS Course

HashiCorp Certified: Terraform Associate

Domain 6

ISSUED BY
Zeal

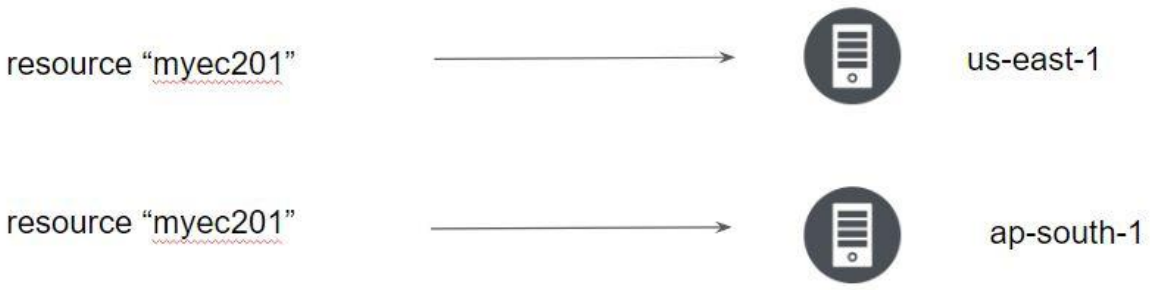
REPRESENTATIVE
instructors@kplabs.in

Domain 6 - Security Primer

Module 1: Provider Configuration

Till now, we have been hardcoding the aws-region parameter within the providers.tf

This means that resources would be created in the region specified in the providers.tf file.



By default, resources use a default provider configuration inferred from the first word of the resource type name.

For example, a resource of type aws_instance uses the default (un-aliased) aws provider configuration unless otherwise stated.

To select an aliased provider for a resource or data source, set its provider meta-argument to a <PROVIDER NAME>.<ALIAS> reference:

```
resource "aws_instance" "foo" {
  provider = aws.west

  # ...
}
```

Module 2: Handling Multiple AWS Profiles in Terraform

You can optionally define multiple configurations for the same provider, and select which one to use on a per-resource or per-module basis.

The primary reason for this is to support multiple regions for a cloud platform.

To include multiple configurations for a given provider, include multiple provider blocks with the same provider name, but set the alias meta-argument to an alias name to use for each additional configuration. For example:

```
# The default provider configuration
provider "aws" {
  region = "us-east-1"
}

# Additional provider configuration for west coast region
provider "aws" {
  alias  = "west"
  region = "us-west-2"
}
```

The provider block without alias set is known as the default provider configuration.

When an alias is set, it creates an additional provider configuration.

For providers that have no required configuration arguments, the implied empty configuration is considered to be the default provider configuration.

Module 3: Sensitive Parameter

With the organization managing its entire infrastructure in terraform, it is likely that you will see some sensitive information embedded in the code.

When working with a field that contains information likely to be considered sensitive, it is best to set the Sensitive property on its schema to true

```
output "db_password" {
  value       = aws_db_instance.db.password
  description = "The password for logging in to the database.
  sensitive   = true
}
```

Setting the sensitive to “true” will prevent the field's values from showing up in CLI output and in Terraform Cloud

It will not encrypt or obscure the value in the state, however.

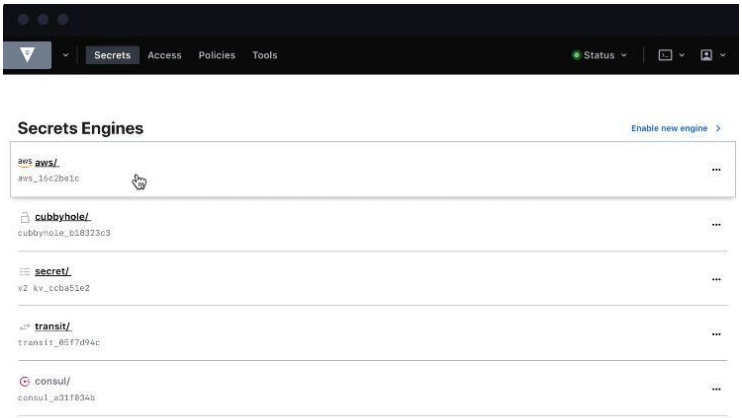
```
C:\Users\Zeal Vora\Desktop\terraform\sensitive data>terraform apply

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

Outputs:

db_password = <sensitive>
```

Module 4: Overview of HashiCorp Vault

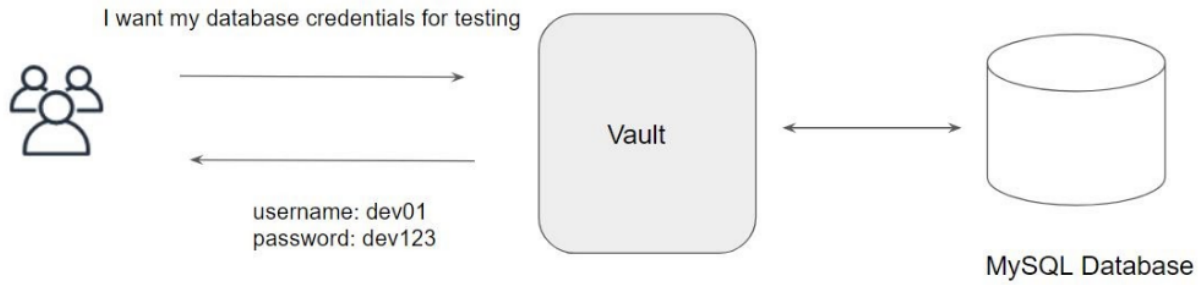
HashiCorp Vault allows organizations to securely store secrets like tokens, passwords, certificates along with access management for protecting secrets.

One of the common challenges nowadays in an organization is “Secrets Management”

Secrets can include database passwords, AWS access/secret keys, API Tokens, encryption keys and others.



Vault can also generate dynamic secrets like AWS Access/Secret keys that has validity for a limited time.

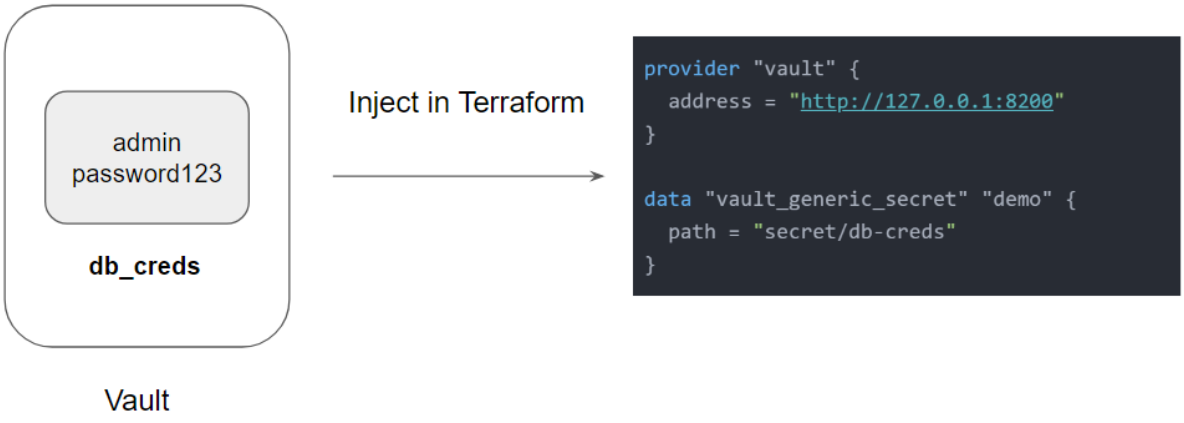


Once Vault is integrated with multiple backends, your life will become much easier and you can focus more on the right work.

Major aspects related to Access Management can be taken over by vault.

Module 5: Terraform and Vault Integration

The Vault provider allows Terraform to read from, write to, and configure HashiCorp Vault.



Important Note:

Interacting with Vault from Terraform causes any secrets that you read and write to be persisted in Terraform's state file.