Terraform Secure Infra Lab: Security Module Documentation

This documentation details the **Key Vault Module** (responsible for creation) and the **Security Module** (responsible for secrets and access control), explaining how they work together to achieve secure secret management using Infrastructure-as-Code (IaC).

---

1. Key Vault Module: Provisioning the Secure Vault

The Key Vault Module provisions the base Azure Key Vault resource, establishing fundamental security settings like soft-delete and purge protection.

Terraform Code Snippets (`modules/key_vault/main.tf`)

Terraform

```
# Key Vault Module: stores SSH key and secrets


resource "azurerm_key_vault" "this" {
  name                = "kv-secure-${random_string.suffix.result}"
  location            = var.location
  resource_group_name = var.resource_group
  tenant_id           = var.tenant_id
  sku_name            = "standard"


  # For testing - disable purge protection
  purge_protection_enabled   = false
  soft_delete_retention_days = 7


  # This block is essential for setting permissions. (Initial Admin Access)
  access_policy {
    tenant_id = var.tenant_id
    object_id = var.object_id # The Admin/Terraform Principal

    secret_permissions = [
      "Get", "List", "Set", "Delete", "Purge", "Recover"
    ]
  }
}

# to generate the random KV Name
resource "random_string" "suffix" {
  length  = 6
  special = false
  upper   = false
}

output "key_vault_id" {
  description = "The ID of the Key Vault"
  value       = azurerm_key_vault.this.id
}

# ... other outputs
```

Step-by-Step Provisioning Process (Key Vault Module)

1. **Name Generation:** The `random_string.suffix` resource creates a unique 6-character suffix, ensuring the Key Vault name (`kv-secure-xxxxxx`) is globally unique, which is a requirement for Azure Key Vault.
2. **Resource Creation:** The `azurerm_key_vault.this` resource is provisioned in Azure using variables for location, resource group, and tenant ID.
3. **Soft Delete Configuration:** `soft_delete_retention_days = 7` is set. This is a **security best practice**, preventing permanent data loss for 7 days upon deletion.
4. **Initial Access Policy:** An **initial access policy** is created inside the main Key Vault resource block. this policy grants the administrative principal (`var.object_id`) full control over secrets, keys, and certificates, which is necessary for the next module (Security Module) to inject secrets.
5. **Output:** The module exports the unique **ID** (`azurerm_key_vault.this.id`) and **Name** of the Key Vault. This ID is critical, as it's used by the downstream **Security Module**.

---

2. Security Module: Secret and Access Management

The Security Module uses the output from the Key Vault Module to manage **contents** (secrets) and enforce **least privilege access policies** for application identities (e.g., the Private VM's Managed Identity).

Terraform Code Snippets (`modules/security/main.tf` and `variables.tf`)

Terraform

```
# In modules/security/main.tf:


# This block creates a secret inside the Key Vault.
resource "azurerm_key_vault_secret" "database_password" {
  name         = "db-password"
  value        = var.database_password_secret_value # Sensitive input
  key_vault_id = var.key_vault_id                    # Input from KV module
}
```

```
# This data block retrieves the client configuration, which is needed for the tenant_id and object_id.
data "azurerm_client_config" "current" {}


# This block creates an access policy for the Key Vault, granting access to a VM.
# It applies the Least Privilege principle.
resource "azurerm_key_vault_access_policy" "vm_access" {
  key_vault_id = var.key_vault_id
  tenant_id    = var.tenant_id
  object_id    = var.private_vm_object_id # The VM's Managed Identity


  secret_permissions = [
    "Get", # ONLY Get permission is granted.
  ]
}


# In modules/security/variables.tf:


variable "key_vault_id" {
  description = "The ID of the Key Vault."
  type        = string
}


variable "database_password_secret_value" {
  description = "The secret value for the database password."
  type        = string
  sensitive   = true # Crucial for security
}


variable "private_vm_object_id" {
  description = "The Object ID of the private VM's managed identity."
  type        = string
}


# ... other variables
```

Step-by-Step Security Process (Security Module)

1. **Data Retrieval:** The `data "azurerm_client_config" "current"` block retrieves the current execution context, primarily the **Tenant ID**, to ensure the access policy is correctly scoped.
2. **Secret Injection:** The `azurerm_key_vault_secret.database_password` resource is deployed:
   - It uses the **sensitive input** `var.database_password_secret_value`, which should be sourced from a secure location in the root module.
   - It places this secret directly into the Key Vault identified by `var.key_vault_id`.
3. **Least Privilege Enforcement:** The `azurerm_key_vault_access_policy.vm_access` resource is created to grant the application access:
   - **Target Principal:** It uses `var.private_vm_object_id`, which is the **Managed Identity** of the Private VM (the application consuming the secret).
   - **Permission Scope:** It grants **only** the `Get` permission via `secret_permissions = ["Get"]`. This ensures the VM can **retrieve** the database password but *cannot* list, set, delete, or recover other secrets in the vault, thereby strictly adhering to the **Least Privilege Principle**.
4. **Final Integration:** Once applied, the Private VM can use its system-assigned Managed Identity to authenticate to Azure Key Vault and fetch the `db-password` secret value for its application configuration.

Troubleshooting Doc

This guide addresses common errors encountered when working with resource blocks and dependencies within the Security Module of a modular Terraform project.

Issue	Root Cause	Resolution Steps (Simple Format)
Undeclared Variables (`var.example_name` used but not defined)	The variable is used in the module's `.tf` files but is **not explicitly declared** in the module's `variables.tf`.	**1. Declare the variable** in `variables.tf`. **2. Ensure the calling (root) module** passes a value for the variable.
Incorrect File Location (Root resources in module)	Infrastructure resources (like `azurerm_resource_group`, `random_string`) were placed inside the security module's `main.tf`.	**1. Move root-level resources** out of the security module and into the main project or a dedicated infrastructure module. **2. Pass only necessary IDs/values** to the security module via variables.
Missing Data Sources (`data.azurerm_client_config` not found)	A referenced data source is **not defined** within the current module or the required value is not being received from the root module.	**1. Define the required data source** (e.g., `data "azurerm_client_config" "current" {}`) inside the module. **OR 2. Define the data source in the root** and pass its output as a variable to the module.
Unmet Dependencies (Error referencing Managed Identity)	The security module attempts to use a value (like the Private VM's Managed Identity ID) from a resource **that hasn't been created yet**.	**1. Ensure the dependent resource** (e.g., the VM) is provisioned first. **2. The security module call** in the root should explicitly reference the output of the dependent resource to ensure correct ordering.

General Module Structure Fix	Incorrect separation of concerns, causing resources and dependencies to be tangled.	**1. Check module `main.tf`**: It should **only** contain security resources (`azurerm_key_vault_secret`, `azurerm_key_vault_access_policy`). **2. Check module `variables.tf`**: It should **correctly declare all inputs** used in `main.tf`. **3. Check root `main.tf`**: It should **define root resources** and pass outputs as inputs to your module.