QATIP Intermediate
Azure Lab 03
Setting Up Azure Key Vault and Service Principal for Terraform Authentication

Contents

Lab Objectives

This lab provides guidance and instructions on setting up an Azure Key Vault and a Service Principal (SP) for securely managing Terraform authentication when running Terraform from an external device. This setup ensures that multiple developers can authenticate Terraform dynamically without needing to store or share sensitive credentials directly.

Teaching Points

Step 1 These tasks are typically handled by the **Security Team** to establish a secure authentication mechanism ensuring that only authorized developers can retrieve the necessary service principal credentials from Azure Key Vault.

Step 2 Performed by developers who have been granted the required **Get and List secret permissions** to retrieve authentication credentials used run Terraform commands.

Developers **do not have access to modify secrets** but can retrieve them for authentication.

Before you begin

Ensure you have completed Lab0 before attempting this lab.

In the IDE terminal pane, enter the following command...

```
cd c:\azure-tf-int\lab\03
```

This shifts your current working directory to labs\03. Ensure all commands are executed in this directory

Close any open files and use the Explorer pane to navigate to and open the labs\03 folder.

Step 1: Administrators set up Key Vault and SP

Examine the PowerShell script **setup-terraform-keyvault.ps1** in c:\azure-tf-int\labs\03

Stage 1: Defining Variables and Creating a Resource Group

The first step in the setup process involves defining essential variables that will be used throughout the configuration. These variables include the name of the resource group, the location where resources will be deployed, the Key Vault name, the Azure subscription ID, and the Service Principal name. Once these variables are defined, the script initiates an

authentication process by logging into Azure using the az login command.

After a successful login, the script proceeds to create a resource group using the **az group create** command. The script then verifies the creation of the resource group by using the **az group show** command, ensuring that the resource group has been successfully established

Stage 2: Creating the Azure Key Vault

In this stage, the script creates an Azure Key Vault, which is a secure service that allows for the storage and management of sensitive information such as API keys, passwords, certificates, and secrets. The Key Vault is created within the previously established resource group using the **az keyvault create** command.

Once the Key Vault has been created, the script verifies its existence by running the **az keyvault show** command. This ensures that the Key Vault is properly configured and ready to securely store sensitive Terraform-related credentials.

Stage 3: Creating a Service Principal and Storing Credentials in Key Vault

A Service Principal (SP) is an identity that allows Terraform to authenticate and interact with Azure resources. In this stage, the script creates a Service Principal with the Contributor role assigned to the specified Azure subscription. This is achieved using the **az ad sp create-for-rbac** command. Note that here the sp is given **contributor** permissions, in production environments the role(s) assigned would be more restrictive. The output of this command includes crucial details such as the **appId**, **password**, and **tenantId**, which are necessary for authentication.

These credentials are then securely stored in Azure Key Vault as secrets. The script uses the **az keyvault secret set** command to save the Service Principal credentials, including the client ID, client secret, tenant ID, and subscription ID. To verify that the secrets have been successfully stored,

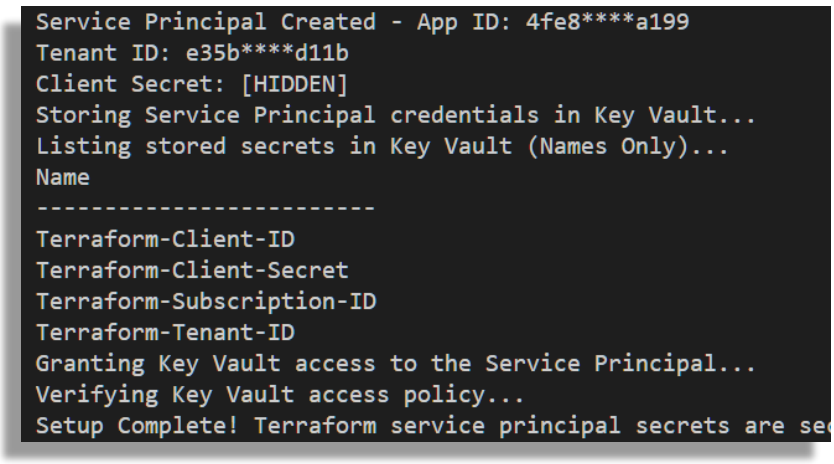
the script lists all stored secrets using the **az keyvault secret list** command.

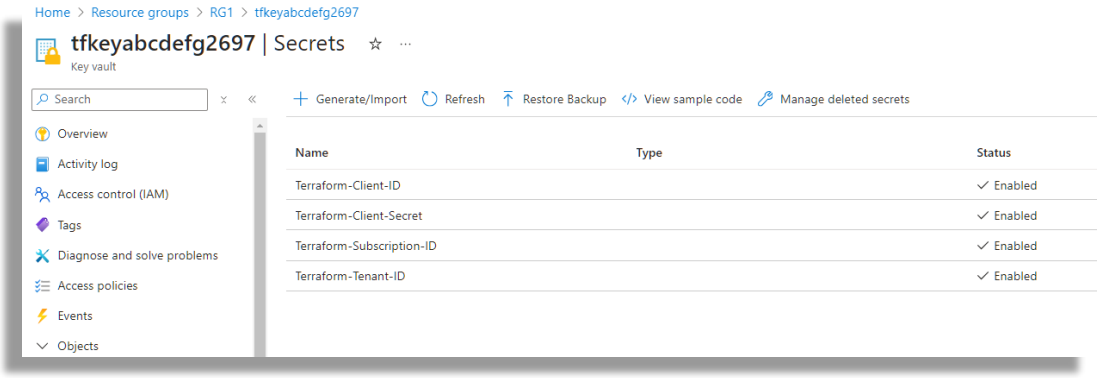
Stage 4: Granting Key Vault Access to the Service Principal

To enable Terraform to retrieve credentials from the Key Vault, the Service Principal must be granted the necessary permissions. This is done using the **az keyvault set-policy** command, which assigns **get** and **list** permissions to the Service Principal. These permissions allow Terraform to access the stored credentials when required.

After setting the access policy, the script verifies that the policy has been correctly applied by running the **az keyvault show** command. This final verification step ensures that the Service Principal has the appropriate permissions to interact with the Key Vault securely.

1. Update line **5** with a Vault name of your choice. This must be unique and can be up to 24 character long. It is suggested you use "**tfkey**" appended with a portion of your subscription_id. Make a note of the name chosen.
2. Update line **6** with your lab subscription id
3. Save your file changes
4. Run **az logout** to demonstrate the full process
5. Run the script **.\setup-terraform-keyvault.ps1**
6. You will be prompted for credentials. In production these would be cloud administrative credentials with authority to create the vault, service principal and secrets. In the lab environment, use your lab credentials.





Step 2: Developers obtain SP credentials from Key Vault

Examine the PowerShell script **retrieve-sp-credentials.ps1** in **c:\azure-tf-int\labs\03**

Stage 1: Define Parameters and Secure Retrieval Function

In this stage, we define the script parameters and a function to securely retrieve secrets from Azure Key Vault. The script requires the Azure Key Vault name as an input parameter and sets a default file (**sp_credentials.env**) to store the retrieved credentials.

The function **Get-SecretValue** is created to securely fetch secrets stored in Azure Key Vault. Since Azure Key Vault encrypts secrets, PowerShell retrieves them as a SecureString. To convert them into plaintext (for Terraform compatibility), we use **.SecretValue** and decrypt it using **.SecureStringToBSTR().** This ensures Terraform can access credentials without exposing them in plaintext logs.

Stage 2: Authenticate to Azure and Retrieve Credentials

In this stage, the script authenticates to Azure and retrieves the Service Principal credentials stored in Azure Key Vault. The authentication is performed using **Connect-AzAccount**, which ensures the script has access to Azure services.

After authentication, the script fetches the Service Principal Client ID, Client Secret, Tenant ID, and Subscription ID from Azure Key Vault using the **Get-SecretValue** function. These credentials are required for

Terraform to authenticate and interact with Azure resources. By securely retrieving these values, we prevent storing static credentials in the script

Stage 3: Set Environment Variables and Store Credentials in a File

Once the credentials are retrieved, they need to be set as environment variables so Terraform can access them during execution. The script assigns the credentials to ARM_CLIENT_ID, ARM_CLIENT_SECRET, ARM_TENANT_ID, and ARM_SUBSCRIPTION_ID using PowerShell's **$env:** syntax. These variables allow Terraform to authenticate automatically without requiring manual input.

Additionally, the credentials are stored in a local .env file, allowing them to persist across sessions or be used in CI/CD pipelines where strict access control is enforced. The .env file format is commonly used to load environment variables dynamically, reducing the need to manually reconfigure Terraform authentication each time.

To ensure security, it is crucial to exclude this file from version control by adding it to .gitignore. This prevents credentials from being accidentally committed to a repository.

1. Run **az logout** to demonstrate the full process
2. Run the script **retrieve-sp-credentials.ps1 -KeyVaultName "<your keyvault name> "**
3. You will be prompted for credentials. In production these would be your developer credentials with authority to read from the vault. In the lab environment, use your lab credentials.
4. At this stage, the SP credentials are downloaded and stored and Terraform is now ready to use them. To verify that the credentials have been successfully set, run the following command: **Get-ChildItem Env:ARM_***

5. These environment variables will only last for the duration of the PowerShell session. Prove this by killing your terminal session, opening a new one and repeating the command. No values will be returned.
6. You could re-run the retrieve-sp-credentials script when needed if the secrets where not written out to a local file for security reasons. In our example script they were though, so navigate back to **c:\azure-tf-int\labs\03** and run **.\load-envs.ps1**. This script reads the env file values back into memory. This allows cross-session credential persistence and is an approach commonly used with CI/CD solutions. If used, access to the env file should be highly restricted.

Verifying the SP account

1. Run **az logout** to ensure you are not using your lab credentials when you interact with azure using Terraform.
2. Verify by running **az logout** again. You should be notified that you have no active accounts

3. Review the **main.tf** file in your current directory. Update the provider block with your **subscription id** and save the changes
4. Note that the file is configured to deploy an empty resource group for testing purposes.
5. Run **terraform init/plan/apply**
6. Switch to the portal and verify the existence of resource group RG2

Lab clean-up

1. Run **terraform destroy** and confirm with **yes**

2. Switch to the portal and manually delete resource group RG1

Congratulations, you have completed this lab