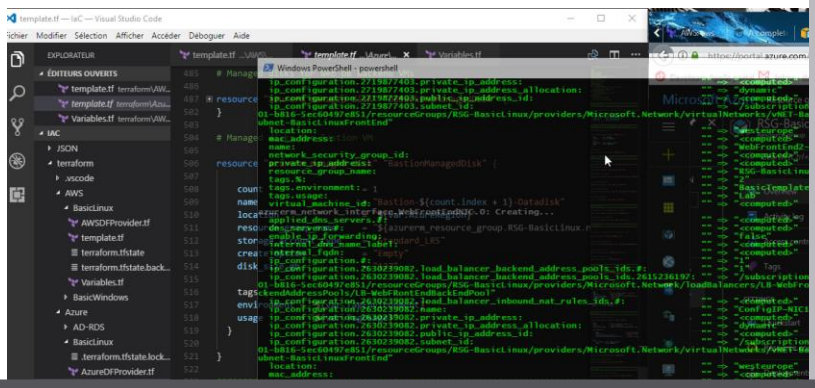


Teknews.cloud

Using Azure Key Vault with Terraform for VM Deployment

David Frappart

06/06/2018

Contributor

Name	Title	Email
David User1art	Cloud Architect	david@dfitcons.info

Version Control

Version	Date	State
1.0	2018/06/05	Final

Table of Contents

1 The security concerns when using Infrastructure as code

1.1 The issue reviewed

So far, I discussed plainly on how to create compute resource and I have avoided the question about how I stored sensitive data such as VM password.
To be clear, until now, I used variables files, obviously not stored in the Git repo, and encrypted in the remote state on Azure storage, but with no security consideration for the local files on my laptop.
Which means exactly that my configuration passwords are stored in clear text on my terraform files.
That being said, it is clearly a no go in terms of security for tools that should be used for production environment.

1.2 Azure Key Vault for storing VM password

On the other hand, Microsoft does offer a service to store sensitive value with the Azure Key Vault service. Documentation on the service is available here for details. For the purpose of this article, let's just remember that Azure Key Vault allows us to store sensitive data such as passwords and that those passwords are then retrievable with terraform for VM provisioning.
It allows some thought about roles segregation and to imagine that a Security oriented team could be managing the password provisioning in a Key Vault while the build team would only get the password through the terraform config.

1.3 Using Terraform to interact with Azure Key Vault

Terraform comes with resources and data sources to provision and interact with secrets in Azure Key Vault. The following sections gives details about those.

1.3.1 Terraform resource for Azure Key Vault

To use an Azure Key Vault, the first step is to be able to provision it. That point is addressed with the azurerm_key_vault resource which is used as follow:

```
resource "azurerm_key_vault" "TerraKeyVault" {
  name                = ""
  location            = ""
  resource_group_name = ""

  sku {
    name = ""
  }
```

```
  tenant_id = ""

  ########################
  #Access Policy 1 for user

  access_policy {
    object_id = ""
    tenant_id = ""

    certificate_permissions = [""]
    key_permissions         = [""]
    secret_permissions      = [""]
  }



  ########################
  #Others Key Vault param

  enabled_for_deployment          = ""
  enabled_for_disk_encryption     = ""
  enabled_for_template_deployment = ""
  ########################
  #Tags

  tags {
    environment = ""
    usage       = ""  }
}
```

Apart from the name, location and the associated resource group, there is a section to declare the sku which can be basic or standard, the access policy, which as the name implies, declare how access to the vault is configured and 3 others parameters which allows to use the Key Vault for deployment, disk encryption and template deployment.
Also worthy of note, the permissions parameters are expecting a list, which is only accepting a subset of specifics values.

1.3.2 Terraform resource for Azure Key Vault secret

In this article we only care about the Key Vault secrets, which are strings that need to be stored securely. Typically, a password is considered as a secret in Azure Key Vault semantic. To create a secret, the available resource is azurerm_key_vault_secret. An example is displayed below:

```
resource "azurerm_key_vault_secret" "TerraWinVMPwd" {
  name      = ""
```

```
  value     = ""
  vault_uri = ""

  tags {
    environment = ""
    usage       = ""
  }
}
```

The resource is pretty simple, and requires references to the vault uri in addition to the secret name and its desired value.

1.3.3 Terraform data sources to get Azure Key Vault related resources

To extract a secret from the vault, we can make use of data sources, which reference existing key vault related resources. Again, since we are only caring about secret here, we will limit our discussion to those data sources that are useful. The first one to use is the data source for the Key Vault itself:

```
data "azurerm_key_vault" "TerraCreatedKeyVault" {
  name                = ""
  resource_group_name = ""
}
```

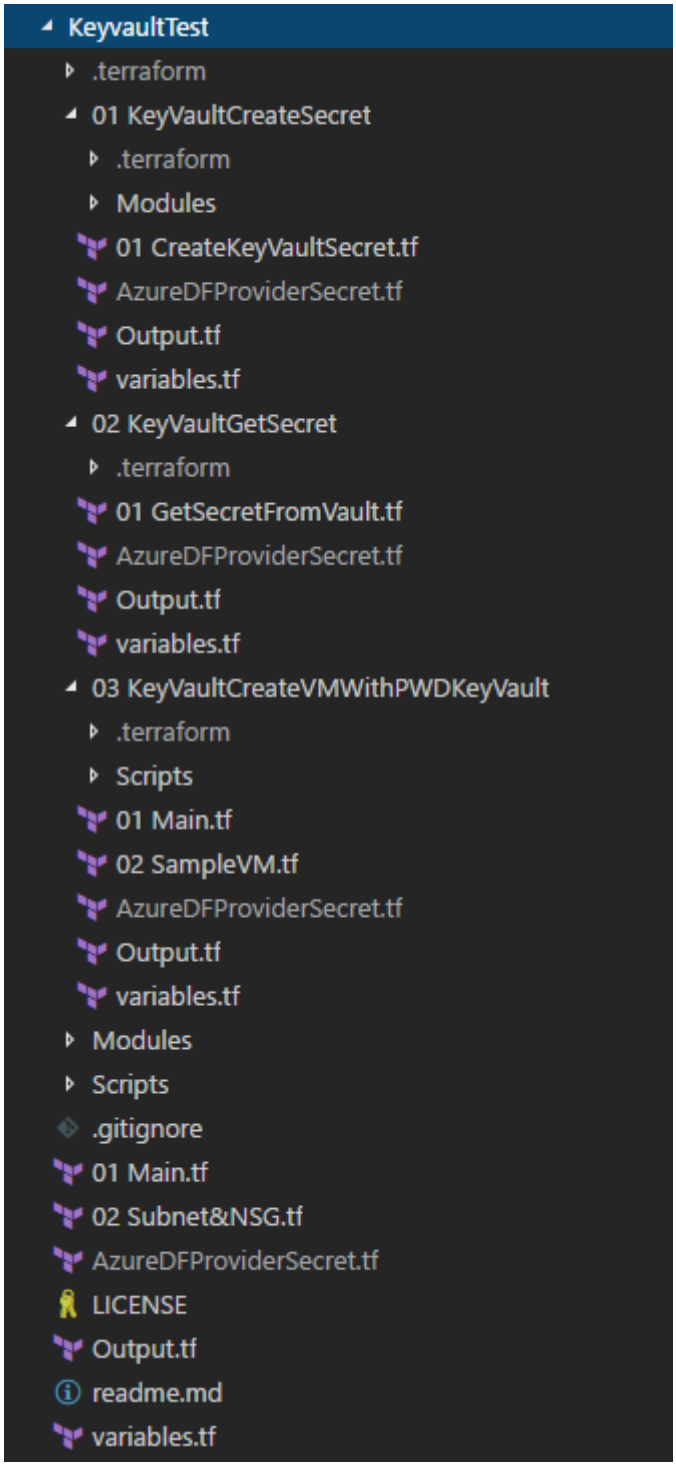
As displayed, only the code, the name of the targeted Key Vault is required. There is more than one way of referencing this resource, including its name as displayed in the Azure portal (or PowerShell…) with the Resource Group in which its resides.

The second data source that is useful is the azurerm_key_vault_secret as displayed below. The use of this data source implies that the terraform application has access to access the Azure Key Vault in the access policy

```
data "azurerm_key_vault_secret" "SecretVMPwdPortalCreatedKV" {
  name      = "DefaultWinPassword"
  vault_uri = "${data.azurerm_key_vault.PortalcreatedKeyvault.vault_uri}"
}
```

2 Terraform config

To evaluate the capabilities targeted here, we will first deploy an Azure Key Vault first, then add a secret to it, and last but not least we will create a VM (with Terraform) and get the secret value to fill in the expected password parameter of the VM. For the test purpose, i created the following folder hierarchy:



In the root folder, we will have all the Azure building blocks such as the Resource Group, the VNet... and also the Azure Key Vault. Existing module from previous config will be reused when possible. In any case, the reference to the module will either point to a GitHub repo or to a local path which would indicate that the module is new.

2.1 Sample architecture and Key Vault configuration

So in this part, the new hot stuff is the Azure Key Vault resource. We use the following code to call the newly created module:

```
module "Keyvault" {
  #Module location

  source = "./Modules/01 Keyvault"

  #Module variables
  KeyVaultName            = "keyvaultdftest"
  KeyVaultRG              = "${module.ResourceGroup.Name}"
  KeyVaultObjectIDPolicy2 = "${var.AzureObjectID}"
  KeyVaultObjectIDPolicy1 = "${var.AzureServicePrincipalInteractive}"
  KeyVaultTenantID        = "${var.AzureTenantID}"
  KeyVaultApplicationID   = "${var.AzureApplicationID}"
  EnvironmentTag          = "${var.EnvironmentTag}"
  EnvironmentUsageTag     = "${var.EnvironmentUsageTag}"
}
```

AS one can see, we have 2 parameters called KeyVaultObjectIDPolicy1 and KeyVaultObjectIDPolicy2. That is because for the test, I wrote a module which creates 2 access policy, one for my terraform application, and another one for my Azure AD Service Principal. Indeed, when first testing out the terraform config, I came across a simple but nonetheless obvious issue: since I was only created an access policy for my terraform registered app, I could not access interactively to the KeyVault with my AAD account. I avoided this issue by adding another access policy for my account.

Below is the module code itself:

```
#############################################################################
# This module creates a keyvault resource
#############################################################################

#Variable declaration for Module

variable "KeyVaultName" {
  type = "string"
}

variable "KeyVaultLocation" {
  type    = "string"
  default = "westeurope"
}
```

```
variable "KeyVaultRG" {
  type = "string"
}

variable "KeyVaultSKUName" {
  type    = "string"
  default = "standard"
}

variable "KeyVaultObjectIDPolicy1" {
  type = "string"
}

variable "KeyVaultObjectIDPolicy2" {
  type = "string"
}

variable "KeyVaultTenantID" {
  type = "string"
}

variable "KeyVaultApplicationID" {
  type = "string"
}

variable "KeyVaultEnabledforDeployment" {
  type    = "string"
  default = "true"
}

variable "KeyVaultEnabledforDiskEncrypt" {
  type    = "string"
  default = "true"
}

variable "KeyVaultEnabledforTempDeploy" {
  type    = "string"
  default = "true"
}

########################
#Variable for Policy 1
variable "KeyVaultCertpermlistPolicy1" {
```

```
  type    = "list"
  default = ["create", "delete", "deleteissuers", "get", "getissuers", "import",
"list", "listissuers", "managecontacts", "manageissuers", "purge", "recover",
"setissuers", "update"]
}

variable "KeyVaultKeyPermlistPolicy1" {
  type    = "list"
  default = ["backup", "create", "decrypt", "delete", "encrypt", "get", "import",
"list", "purge", "recover", "restore", "sign", "unwrapKey", "update", "verify",
"wrapKey"]
}

variable "KeyVaultSecretPermlistPolicy1" {
  type    = "list"
  default = ["backup", "delete", "get", "list", "purge", "recover", "restore",
"set"]
}

########################
#Variable for Policy 2
variable "KeyVaultCertpermlistPolicy2" {
  type    = "list"
  default = ["create", "delete", "deleteissuers", "get", "getissuers", "import",
"list", "listissuers", "managecontacts", "manageissuers", "purge", "recover",
"setissuers", "update"]
}

variable "KeyVaultKeyPermlistPolicy2" {
  type    = "list"
  default = ["backup", "create", "decrypt", "delete", "encrypt", "get", "import",
"list", "purge", "recover", "restore", "sign", "unwrapKey", "update", "verify",
"wrapKey"]
}

variable "KeyVaultSecretPermlistPolicy2" {
  type    = "list"
  default = ["backup", "delete", "get", "list", "purge", "recover", "restore",
"set"]
}

variable "EnvironmentTag" {
  type    = "string"
  default = "Poc"
```

```
}

variable "EnvironmentUsageTag" {
  type    = "string"
  default = "Poc usage only"
}

#Resource Creation

resource "azurerm_key_vault" "TerraKeyVault" {
  name                = "${lower(var.KeyVaultName)}"
  location            = "${var.KeyVaultLocation}"
  resource_group_name = "${var.KeyVaultRG}"

  sku {
    name = "${var.KeyVaultSKUName}"
  }

  tenant_id = "${var.KeyVaultTenantID}"

  ########################
  #Access Policy 1 for user

  access_policy {
    object_id = "${var.KeyVaultObjectIDPolicy1}"
    tenant_id = "${var.KeyVaultTenantID}"

    #application_id          = "${var.KeyVaultApplicationID}"
    certificate_permissions = ["${var.KeyVaultCertpermlistPolicy1}"]
    key_permissions         = ["${var.KeyVaultKeyPermlistPolicy1}"]
    secret_permissions      = ["${var.KeyVaultSecretPermlistPolicy1}"]
  }

  ########################
  #Access Policy 2 for app

  access_policy {
    object_id = "${var.KeyVaultObjectIDPolicy2}"
    tenant_id = "${var.KeyVaultTenantID}"

    application_id          = "${var.KeyVaultApplicationID}"
    certificate_permissions = ["${var.KeyVaultCertpermlistPolicy2}"]
    key_permissions         = ["${var.KeyVaultKeyPermlistPolicy2}"]
    secret_permissions      = ["${var.KeyVaultSecretPermlistPolicy2}"]
```

```
  }

  ########################
  #Others Keyvault param

  enabled_for_deployment          = "${var.KeyVaultEnabledforDeployment}"
  enabled_for_disk_encryption     = "${var.KeyVaultEnabledforDiskEncrypt}"
  enabled_for_template_deployment = "${var.KeyVaultEnabledforTempDeploy}"

  ########################
  #Tags

  tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
  }
}

#VModule Output

output "Id" {
  value = "${azurerm_key_vault.TerraKeyVault.id}"
}

output "URI" {
  value = "${azurerm_key_vault.TerraKeyVault.vault_uri}"
}
```

With this code, we are able to provision a Key Vault and then write secret in it. For this test, I was very permissive in my access policy and I granted all access to to the identity targeted in the access policies. Obviously, it is possible to be more restrictive. An access policy to only get secret could be used to reduce this list:

```
secret_permissions      = ["get", "list"]
```

2.2 Using the Data source to create and get Key Vault secrets

After the previous step, we are now ready to use the provisioned Key Vault to store and then get secrets. To emulate a real use case, I did stage the provisioning of the secret in a dedicated configuration. Typically, a team provisioning the secret would be security oriented while a team provisioning the Azure Key Vault would be an infra team (or not).
So in the step of storing the secret, we use data sources as below:

```
data "azurerm_resource_group" "TerracreatedRG" {
```

```
  name = "${var.RGName}-${var.EnvironmentUsageTag}${var.EnvironmentTag}"
}

data "azurerm_key_vault" "TerracreatedKeyVault" {
  name                = "keyvaultdftest"
  resource_group_name = "${data.azurerm_resource_group.TerracreatedRG.name}"
}

data "azurerm_resource_group" "PortalcreatedRG" {
  name = "rg-001"
}

data "azurerm_key_vault" "PortalcreatedKeyvault" {
  name                = "dfkeyvaulttest02"
  resource_group_name = "${data.azurerm_resource_group.PortalcreatedRG.name}"
}

#############################################################################
#Secret creation
#############################################################################

module "WinVMPassword" {
  #Module location
  source = "./Modules/02 KeyvaultSecret"

  #Module variable
  PasswordName        = "WinVMPassword"
  PasswordValue       = "${var.VMAdminPassword}"
  VaultURI            = "${data.azurerm_key_vault.PortalcreatedKeyvault.vault_uri}"
  EnvironmentTag      = "${var.EnvironmentTag}"
  EnvironmentUsageTag = "${var.EnvironmentUsageTag}"
}

module "TerraWinVMPassword" {
  #Module location
  source = "./Modules/02 KeyvaultSecret"

  #Module variable
  PasswordName        = "TerraWinVMPassword"
  PasswordValue       = "${var.VMAdminPassword}"
  VaultURI            = "${data.azurerm_key_vault.TerracreatedKeyVault.vault_uri}"
  EnvironmentTag      = "${var.EnvironmentTag}"
  EnvironmentUsageTag = "${var.EnvironmentUsageTag}"
}
```

The referenced module is displayed below:

```
#########################################################################
# This module create a keyvault resource
#########################################################################

#Variable declaration for Module

variable "PasswordName" {
  type = "string"
}

variable "PasswordValue" {
  type = "string"
}

variable "VaultURI" {
  type = "string"
}

variable "EnvironmentTag" {
  type    = "string"
  default = "Poc"
}

variable "EnvironmentUsageTag" {
  type    = "string"
  default = "Poc usage only"
}

#Resource Creation

resource "azurerm_key_vault_secret" "TerraWinVMPwd" {
  name      = "${var.PasswordName}"
  value     = "${var.PasswordValue}"
  vault_uri = "${var.VaultURI}"

  tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
  }
}
```

```
#Module Output

output "ID" {
  value = "${azurerm_key_vault_secret.TerraWinVMPwd.id}"
}

output "Version" {
  value = "${azurerm_key_vault_secret.TerraWinVMPwd.version}"
}

output "Name" {
  value = "${azurerm_key_vault_secret.TerraWinVMPwd.name}"
}
```

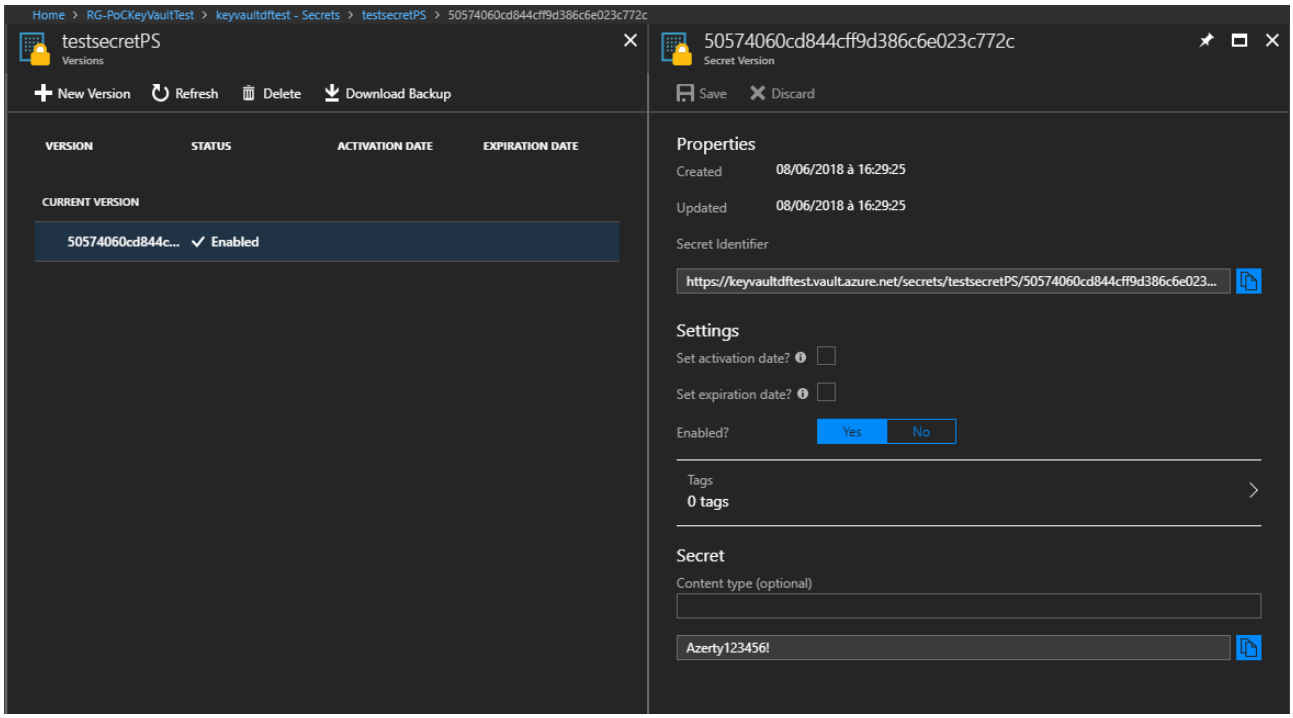
The watchful reader will notice that I reference two different KeyVault resources. One designated as Terraform provisioned and the other as portal-created. It is simply for the completeness of the test that I did this.
As a matter of fact, I came across a strange behavior while creating the KeyVault. The access policy created to allow my terraform app to access the secret vault is not working since I get a denied access when applying the configuration for the secret creation.
On the other hand, the same configuration used to create a secret in the “portal-created” Key Vault with a “portal-created” access policy for the terraform app works just fine. And if I do recreate an access policy from the portal for my terraform app, it works fine also in the Key Vault created previously with terraform.
Last, the access policy created for my Azure AD account does works and allow me to create a secret in the portal or through PowerShell:

```
PS C:\Users\user1 > Get-AzureRmKeyVault -ResourceGroupName rg-pockeyvaulttest


ResourceId        : /subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/RG-PoCKeyVaultTest/providers/Mic
                    rosoft.KeyVault/vaults/keyvaultdftest
VaultName         : keyvaultdftest
ResourceGroupName : RG-PoCKeyVaultTest
Location          : westeurope
Tags              : {environment, usage}
TagsTable         :
                    Name         Value
                    ===========  ============
                    environment  KeyVaultTest
                    usage        PoC
```

```
PS C:\Users\user1> Get-AzureKeyVaultSecret -VaultName keyvaultdftest


Enabled     : True
Expires     :
NotBefore   :
Created     : 06/06/2018 20:33:17
Updated     : 06/06/2018 20:33:17
ContentType :
Tags        : {environment, usage}
TagsTable   : Name         Value
              environment  KeyVaultTest
              usage        PoC

VaultName   : keyvaultdftest
Name        : TerraWinVMPassword
Version     :
Id          : https://keyvaultdftest.vault.azure.net:443/secrets/TerraWinVMPassword



PS C:\Users\user1> $Secret = ConvertTo-SecureString -String "Azerty123456!" -
AsPlainText -Force
PS C:\Users\user1\Documents\IaC\terraform\Azure\KeyvaultTest> Set-
AzureKeyVaultSecret -VaultName keyvaultdftest -name testsecretPS -SecretValue
$Secret


SecretValue     : System.Security.SecureString
SecretValueText : Azerty123456!
Attributes      :
Microsoft.Azure.Commands.KeyVault.Models.PSKeyVaultSecretAttributes
Enabled         : True
Expires         :
NotBefore       :
Created         : 08/06/2018 14:29:25
Updated         : 08/06/2018 14:29:25
ContentType     :
Tags            :
TagsTable       :
VaultName       : keyvaultdftest
Name            : testsecretPS
```

```
Version         : 50574060cd844cff9d386c6e023c772c
Id              :
https://keyvaultdftest.vault.azure.net:443/secrets/testsecretPS/50574060cd844cff9d3
86c6e023c772c
```

The secret then appears in the portal, thus validating the access policy is working for the Azure AD Account:



Now, creating the password as a secret in the Key Vault is one thing, another is to check if the secret is accessible. For this, we simply use data source and output to verify that we do read the secret:

```
#########################################################
# This file get a secret in a previously deployed
# keyvault
#########################################################

#############################################################################
# Access to Azure
#############################################################################

# Configure the Microsoft Azure Provider with Azure provider variable defined in
AzureDFProvider.tf

provider "azurerm" {
```

```
  subscription_id = "${var.AzureSubscriptionID1}"
  client_id       = "${var.AzureClientID}"
  client_secret   = "${var.AzureClientSecret}"
  tenant_id       = "${var.AzureTenantID}"
}

#################################################################################
# Data sources
#################################################################################

data "azurerm_resource_group" "TerraRG" {
  name = "${var.RGName}-${var.EnvironmentUsageTag}${var.EnvironmentTag}"
}

data "azurerm_key_vault" "TerraCreatedKeyVault" {
  name                = "keyvaultdftest"
  resource_group_name = "${data.azurerm_resource_group.TerraRG.name}"
}

data "azurerm_resource_group" "PortalcreatedRG" {
  name = "rg-001"
}

data "azurerm_key_vault" "PortalcreatedKeyvault" {
  name                = "dfkeyvaulttest02"
  resource_group_name = "${data.azurerm_resource_group.PortalcreatedRG.name}"
}

data "azurerm_key_vault_secret" "SecretVMPwdPortalCreatedKV" {
  name      = "DefaultWinPassword"
  vault_uri = "${data.azurerm_key_vault.PortalcreatedKeyvault.vault_uri}"
}

data "azurerm_key_vault_secret" "SecretVMPwdTerraCreatedKV" {
  name      = "TerraWinVMPassword"
  vault_uri = "${data.azurerm_key_vault.TerraCreatedKeyVault.vault_uri}"
}
```

```
#########################################################
# This file defines which value are sent to output
#########################################################

#########################################################
```

```
# KeyVault Output
#########################################################

#########################################################
#KeyvaultId
output "PortalCreatedKeyVaulId" {
  value = "${data.azurerm_key_vault.PortalcreatedKeyvault.id}"
}

output "TerraCreatedKeyVaulId" {
  value = "${data.azurerm_key_vault.TerraCreatedKeyVault.id}"
}

#########################################################
#Keyvault URI
output "PortalCreatedKeyVaultVaultUri" {
  value = "${data.azurerm_key_vault.PortalcreatedKeyvault.vault_uri}"
}

output "TerraCreatedKeyVaultVaultUri" {
  value = "${data.azurerm_key_vault.TerraCreatedKeyVault.vault_uri}"
}

#########################################################
#Keyvault Access Policies

output "PortalCreatedKeyVaultVaultAccessPolicy" {
  value = "${data.azurerm_key_vault.PortalcreatedKeyvault.access_policy}"
}

output "TerraCreatedKeyVaultVaultAccessPolicy" {
  value = "${data.azurerm_key_vault.TerraCreatedKeyVault.access_policy}"
}

#########################################################
#Keyvault Secrets

output "PortalCreatedKeyVaultPwdValue" {
  value     = "${data.azurerm_key_vault_secret.SecretVMPwdPortalCreatedKV.value}"
  sensitive = true
}

output "PortalCreatedKeyVaultPwdId" {
  value = "${data.azurerm_key_vault_secret.SecretVMPwdPortalCreatedKV.id}"
```

```
}

output "PortalCreatedKeyVaultPwdContentType" {
  value =
"${data.azurerm_key_vault_secret.SecretVMPwdPortalCreatedKV.content_type}"
}

output "TerraCreatedKeyVaultPwdValue" {
  value     = "${data.azurerm_key_vault_secret.SecretVMPwdTerraCreatedKV.value}"
  sensitive = true
}

output "TerraCreatedKeyVaultPwdId" {
  value = "${data.azurerm_key_vault_secret.SecretVMPwdTerraCreatedKV.id}"
}

output "TerraCreatedKeyVaultPwdContentType" {
  value = "${data.azurerm_key_vault_secret.SecretVMPwdTerraCreatedKV.content_type}"
}
```

After applying the configuration, we get this output:

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

Outputs:

PortalCreatedKeyVaulId = /subscriptions/f1f020d0-0fa6-4d01-b816-
5ec60497e851/resourceGroups/RG-
001/providers/Microsoft.KeyVault/vaults/dfkeyvaulttest02
PortalCreatedKeyVaultPwdContentType =
PortalCreatedKeyVaultPwdId =
https://dfkeyvaulttest02.vault.azure.net/secrets/DefaultWinPassword/0d11b32aff8f448
eba8b0e5c488a1b9c
PortalCreatedKeyVaultPwdValue = <sensitive>
PortalCreatedKeyVaultVaultAccessPolicy = [
    {
        application_id = ,
        certificate_permissions = [Get List Update Create Import Delete Recover
Backup Restore ManageContacts ManageIssuers GetIssuers ListIssuers SetIssuers
DeleteIssuers],
        key_permissions = [Get List Update Create Import Delete Recover Backup
Restore],
        object_id = c2d0f544-aa1e-454a-8daf-a99985634aa9,
```

```
        secret_permissions = [Get List Set Delete Recover Backup Restore],
        tenant_id = e0c45235-95fe-4bd6-96ca-2d529f0ebde4
    },
    {
        application_id = ,
        certificate_permissions = [],
        key_permissions = [],
        object_id = 45f1bbc1-c7cd-4490-8991-144403c0ffc5,
        secret_permissions = [Get List],
        tenant_id = e0c45235-95fe-4bd6-96ca-2d529f0ebde4
    }
]
PortalCreatedKeyVaultVaultUri = https://dfkeyvaulttest02.vault.azure.net/
TerraCreatedKeyVaulId = /subscriptions/f1f020d0-0fa6-4d01-b816-
5ec60497e851/resourceGroups/RG-
PoCKeyVaultTest/providers/Microsoft.KeyVault/vaults/keyvaultdftest
TerraCreatedKeyVaultPwdContentType =
TerraCreatedKeyVaultPwdId =
https://keyvaultdftest.vault.azure.net/secrets/TerraWinVMPassword/0d0f1aa458964d6c8
36e077741245dad
TerraCreatedKeyVaultPwdValue = <sensitive>
TerraCreatedKeyVaultVaultAccessPolicy = [
    {
        application_id = ,
        certificate_permissions = [Create Delete DeleteIssuers Get GetIssuers
Import List ListIssuers ManageContacts ManageIssuers Purge Recover SetIssuers
Update],
        key_permissions = [Backup Create Decrypt Delete Encrypt Get Import List
Purge Recover Restore Sign UnwrapKey Update Verify WrapKey],
        object_id = c2d0f544-aa1e-454a-8daf-a99985634aa9,
        secret_permissions = [Backup Delete Get List Purge Recover Restore Set],
        tenant_id = e0c45235-95fe-4bd6-96ca-2d529f0ebde4
    },
    {
        application_id = ,
        certificate_permissions = [],
        key_permissions = [],
        object_id = 45f1bbc1-c7cd-4490-8991-144403c0ffc5,
        secret_permissions = [Get List Set],
        tenant_id = e0c45235-95fe-4bd6-96ca-2d529f0ebde4
    }
]
TerraCreatedKeyVaultVaultUri = https://keyvaultdftest.vault.azure.net/
```

An interesting point is that Terraform does not display the password but show a sensitive value:

```
PortalCreatedKeyVaultPwdValue = <sensitive>
```

```
TerraCreatedKeyVaultPwdValue = <sensitive>
```

Remember, also, that the terraform state, if stored in an Azure storage account, is encrypted at rest so everything should be good. However, since the state is accessible, we can specify the output for the passwords values and then display it:

```
PS C:\Users\User1\KeyvaultTest\02 KeyVaultGetSecret> terraform output
PortalCreatedKeyVautPwdValue
Password1234!#
```

2.3 How to get a Key Vault secret as the VM password with Terraform

Ok, now we do have everything we need and we can try it out on a VM deployment. We will use the following Windows VM Module:

```
##############################################################################################
#This module allows the creation of 1 Windows VM with 1 NIC
##############################################################################################

#Variable declaration for Module

#The VM count
variable "VMCount" {
  type    = "string"
  default = "1"
}

#The VM name
variable "VMName" {
  type = "string"
}

#The VM location
variable "VMLocation" {
  type = "string"
}
```

```
#The RG in which the VMs are located
variable "VMRG" {
  type = "string"
}

#The NIC to associate to the VM
variable "VMNICid" {
  type = "list"
}

#The VM size
variable "VMSize" {
  type    = "string"
  default = "Standard_F1"
}

#The Availability set reference

variable "ASID" {
  type = "string"
}

#The Managed Disk Storage tier

variable "VMStorageTier" {
  type    = "string"
  default = "Premium_LRS"
}

#The VM Admin Name

variable "VMAdminName" {
  type    = "string"
  default = "VMAdmin"
}

#The VM Admin Password

variable "VMAdminPassword" {
  type = "string"
}

#The OS Disk Size
```

```
variable "OSDisksize" {
  type    = "string"
  default = "128"
}

# Managed Data Disk reference

variable "DataDiskId" {
  type = "list"
}

# Managed Data Disk Name

variable "DataDiskName" {
  type = "list"
}

# Managed Data Disk size

variable "DataDiskSize" {
  type = "list"
}

# VM images info
#get appropriate image info with the following command
#Get-AzureRMVMImagePublisher -location WestEurope
#Get-AzureRMVMImageOffer -location WestEurope -PublisherName <PublisherName>
#Get-AzureRmVMImageSku -Location westeurope -Offer <OfferName> -PublisherName
<PublisherName>

variable "VMPublisherName" {
  type = "string"
}

variable "VMOffer" {
  type = "string"
}

variable "VMsku" {
  type = "string"
}

#The boot diagnostic storage uri
```

```
variable "DiagnosticDiskURI" {
  type = "string"
}

#Tag info

variable "EnvironmentTag" {
  type    = "string"
  default = "Poc"
}

variable "EnvironmentUsageTag" {
  type    = "string"
  default = "Poc usage only"
}

variable "CloudinitscriptPath" {
  type = "string"
}

#VM Creation
/*
data "template_file" "cloudconfig" {
  #template = "${file("./script.sh")}"
  template = "${file("${path.root}${var.CloudinitscriptPath}")}"
}

#https://www.terraform.io/docs/providers/template/d/cloudinit_config.html
data "template_cloudinit_config" "config" {
  gzip          = true
  base64_encode = true

  part {
    content = "${data.template_file.cloudconfig.rendered}"
  }
}

*/

resource "azurerm_virtual_machine" "TerraVMwithCount" {
  count                 = "${var.VMCount}"
  name                  = "${var.VMName}${count.index+1}"
  location              = "${var.VMLocation}"
  resource_group_name   = "${var.VMRG}"
```

```
  network_interface_ids = ["${element(var.VMNICid,count.index)}"]
  vm_size               = "${var.VMSize}"
  availability_set_id   = "${var.ASID}"

  boot_diagnostics {
    enabled     = "true"
    storage_uri = "${var.DiagnosticDiskURI}"
  }

  storage_image_reference {
    #get appropriate image info with the following command
    #Get-AzureRmVMImageSku -Location westeurope -Offer windowsserver -PublisherName
microsoftwindowsserver
    publisher = "${var.VMPublisherName}"

    offer   = "${var.VMOffer}"
    sku     = "${var.VMsku}"
    version = "latest"
  }

  storage_os_disk {
    name              = "${var.VMName}${count.index+1}-OSDisk"
    caching           = "ReadWrite"
    create_option     = "FromImage"
    managed_disk_type = "${var.VMStorageTier}"
    disk_size_gb      = "${var.OSDisksize}"
  }

  storage_data_disk {
    name            = "${element(var.DataDiskName,count.index)}"
    managed_disk_id = "${element(var.DataDiskId,count.index)}"
    create_option   = "Attach"
    lun             = 0
    disk_size_gb    = "${element(var.DataDiskSize,count.index)}"
  }

  os_profile {
    computer_name  = "${var.VMName}${count.index+1}"
    admin_username = "${var.VMAdminName}"
    admin_password = "${var.VMAdminPassword}"
    custom_data    = "${file("${path.root}${var.CloudinitscriptPath}")}"
  }

  os_profile_windows_config {
```

```
    provision_vm_agent        = "true"
    enable_automatic_upgrades = "false"
  }

  tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
  }
}

#Adding BGInfo to VM

resource "azurerm_virtual_machine_extension" "Terra-BGInfoAgent" {
  count                = "${var.VMCount}"
  name                 = "${var.VMName}${count.index+1}BGInfo"
  location             = "${var.VMLocation}"
  resource_group_name  = "${var.VMRG}"
  virtual_machine_name =
"${element(azurerm_virtual_machine.TerraVMwithCount.*.name,count.index)}"
  publisher            = "microsoft.compute"
  type                 = "BGInfo"
  type_handler_version = "2.1"

  /*
    settings = <<SETTINGS
          {

          "commandToExecute": ""
          }
  SETTINGS
  */
  tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
  }
}

output "Name" {
  value = ["${azurerm_virtual_machine.TerraVMwithCount.*.name}"]
}

output "Id" {
  value = ["${azurerm_virtual_machine.TerraVMwithCount.*.id}"]
}
```

And we will call it as follow, specifying a data source for the VM password:

```
#Data source for VM Password in keyvault

data "azurerm_key_vault_secret" "VMPassword" {
  name      = "DefaultWinPassword"
  vault_uri = "${data.azurerm_key_vault.PortalcreatedKeyvault.vault_uri}"
}
```

```
module "VMs_SampleVM" {
  #module source

  source = "github.com/dfrappart/Terra-AZCloudInit//Modules//02 WinVMWithCount"

  #Module variables

  VMName              = "SampleVM"
  VMLocation          = "${var.AzureRegion}"
  VMRG                = "${data.azurerm_resource_group.TerraCreatedRG.name}"
  VMNICid             = ["${module.NICs_SampleVM.Ids}"]
  VMSize              = "${lookup(var.VMSize, 5)}"
  ASID                = "${module.AS_SampleVM.Id}"
  VMStorageTier       = "${lookup(var.Manageddiskstoragetier, 0)}"
  VMAdminName         = "${var.VMAdminName}"
  VMAdminPassword     = "${data.azurerm_key_vault_secret.VMPassword.value}"
  DataDiskId          = ["${module.DataDisks_SampleVM.Ids}"]
  DataDiskName        = ["${module.DataDisks_SampleVM.Names}"]
  DataDiskSize        = ["${module.DataDisks_SampleVM.Sizes}"]
  VMPublisherName     = "${lookup(var.PublisherName, 0)}"
  VMOffer             = "${lookup(var.Offer, 0)}"
  VMsku               = "${lookup(var.sku, 0)}"
  DiagnosticDiskURI   =
"${data.azurerm_storage_account.SourceSTOADiagLog.primary_blob_endpoint}"
  EnvironmentTag      = "${var.EnvironmentTag}"
  EnvironmentUsageTag = "${var.EnvironmentUsageTag}"
  CloudinitscriptPath = "./Scripts/example.ps1"
}
```

Once the deployment is completed, testing an RDP connection should validate if Terraform is indeed able to read a secret. For those who wonder, it did work but I invite ou to try it out by yourselves.

3 Security considerations and conclusion

Now let's stop for the technical and code stuff and think a little about security. In the introduction we hint that storing password in config file was not ideal. Now we are able to provision secret in an Azure KeyVault, and also to retrieve it with terraform, once a proper access policy is available for the associated Azure AD service Principal.

We do know that the state, which contains all configuration value, is thus including the password value. This state should be stored in an Azure Storage account and thus encrypted. However, it can be retrieved if we have the appropriate access.
To completely secure the Password, and any other Key Vault eligible data, a proper segregation of the roles and actions should be planned, concurrently with how to automate as much as possible those differents separate actions, and avoid going back to delay because of too many validations required.
One good thing though, since we are talking about small part to automate and in any way orchestrate, it appears that I should be easier to be kept agile.
A proper process description should take place anyway, and each step in the process then automated. Finding the orchestration in between is another step, and topics for another day

Now back to the code stuff, my code is available here, on Github, and it won't move ☺

Another Tech blog

My thoughts and experiences on Cloud related tek