

AVD Golden Image from another  
point of view.

Raúl Carvajal Bustos

## Contenido

<b>REFERENCIAS.....</b>	<b>3</b>
<b>TOM HICKLING VISITA SU BLOG. ....</b>	<b>3</b>
<b>PARVEEN SINGH VISITA SU BLOG.....</b>	<b>3</b>
<b>ROBERT PRZYBYLSKI VISISTA SU BLOG.....</b>	<b>3</b>
<b>JAKE WALSH VISITA SU BLOG. ....</b>	<b>3</b>
<b>ADIN ERMIE VISITA SU BLOG. ....</b>	<b>3</b>
¿Qué vamos a desplegar? .....	4
Contenido del despliegue. ....	6
<b>Explicación de cada archivo.....</b>	<b>7</b>
Providers.tf .....	7
Main.tf .....	8
Ad_sp.tf .....	9
Az_keyvault.tf.....	10
Az_networking.tf .....	13
Az_vars.tf.....	14
ADO_project.tf.....	14
ADO_variables.tf .....	16
Outputs.tf.....	18
Terraform init, plan y apply.....	18
Archivos para desplegar la Golden Image en Azure DevOps.....	19
Packer.json .....	19
Azure-pipelines.yml .....	19
Trigger .....	19
Name.....	19
Stage .....	19
displayname .....	19
Jobs.....	20
Steps .....	20
Subir los scripts que usaremos al File Share. ....	23
Os adjunto los enlaces de donde sacar los scripts de automatización: .....	23
Azure DevOps .....	24

Consideraciones. ....	24
Visualizando lo desplegado. ....	24
Creando el pipeline. ....	29

Microsoft Azure © and Azure Devops © are trademarks of Microsoft ©.

Terraform © and Packer © are trademarks of Hashicorp ©.

# Terraform + Packer + Azure DevOps = AVD Golden Image from another point of view.

## REFERENCIAS.

Este tutorial se ha realizado gracias en base al trabajo de varias personas en internet que han compartido sus conocimientos. Es por ello por lo que quisiera dejar constancia de la gente a la cual me ha inspirado:

**[TOM HICKLING VISITA SU BLOG.](#)**

**[PARVEEN SINGH VISITA SU BLOG.](#)**

**[ROBERT PRZYBYLSKI VISISTA SU BLOG.](#)**

**[JAKE WALSH VISITA SU BLOG.](#)**

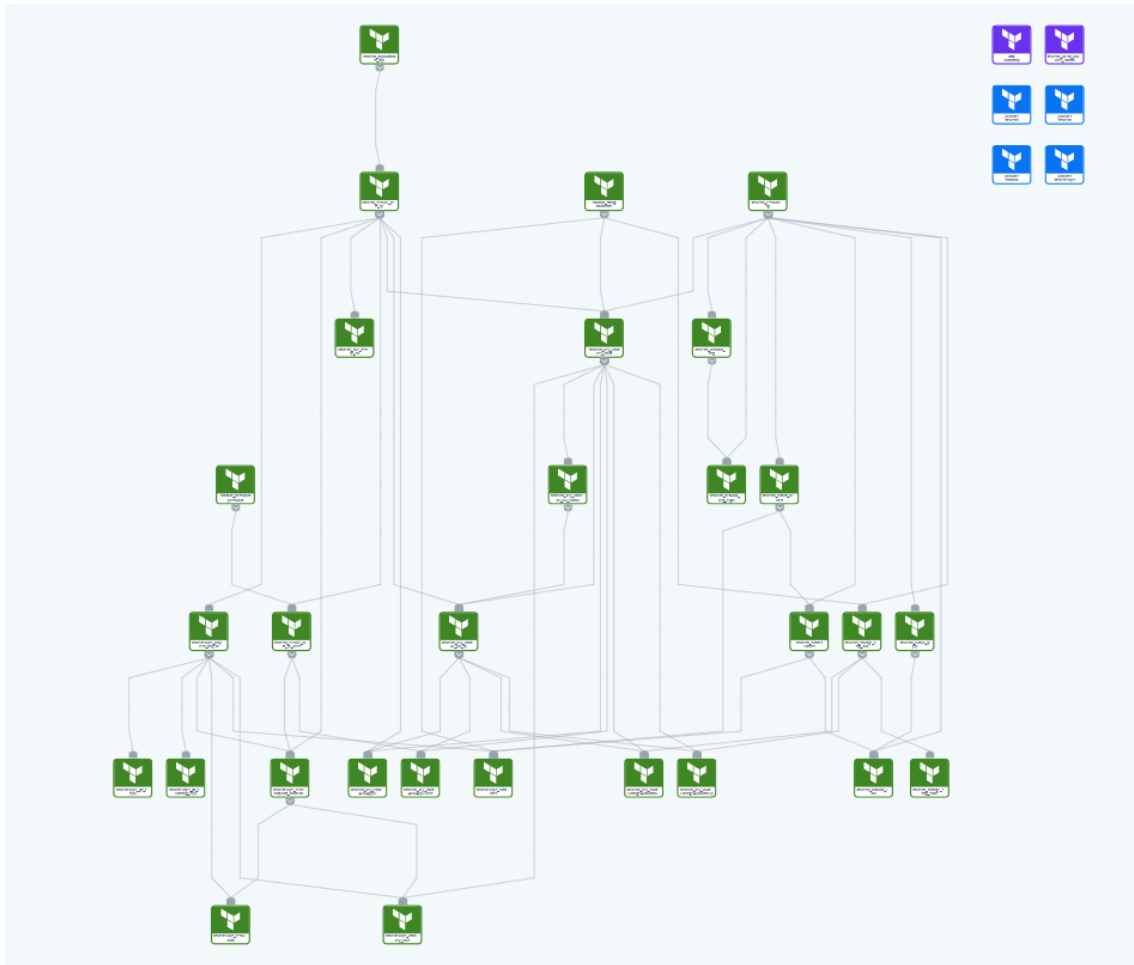
**[ADIN ERMIE VISITA SU BLOG.](#)**

## ¿Qué vamos a desplegar?

Dentro del mundo DevOps, a menudo se requiere del conocimiento de uso de diferentes herramientas, en esta ocasión vamos a generar una *Golden Image* de Windows 10 multisesión + office 365 mediante Packer, pero con un toque diferente.

Vamos a utilizar Terraform para desplegar la infraestructura necesaria, usaremos los pipelines de Azure DevOps para generar una Golden Image mediante Packer:

- Grupo de recursos:
  - Donde se desplegarán los recursos y la imagen final de Windows.
- Azure DevOps:
  - Repositorio y pipelines para generar la imagen de Windows.
- key Vault:
  - Para el almacenamiento de claves.
- Storage account:
  - Donde subiremos los scripts para la configuración de la *Golden image*.
- Service Principal:
  - Cuenta para realizar las tareas para generar la *Golden image*.
- Proyecto Azure DevOps:
  - Donde alojaremos el repositorio.
- Repositorio en Azure DevOps:
  - Donde subiremos los archivos: Packer.json y azure-pipelines.yml
- 2 grupos de variables para los pipelines:
  - Uno de ellos importando las variables desde el key vault.






*Cascada de dependencias (codeherent.tech) 1*

Tal y como ilustra la imagen de arriba, podemos ver arriba a la izquierda el Service Principal y sus dependencias que desplegaremos más adelante.

## Contenido del despliegue.

He creado los archivos de Terraform para representar los recursos que vamos a desplegar:

Nombre	Fecha de modificación	Tipo	Tamaño
 .terraform	26/05/2021 13:09	Carpeta de archivos	
 scripts	03/06/2021 16:23	Carpeta de archivos	
 ad_sp.tf	04/06/2021 12:25	Archivo TF	2 KB
 ADO_project.tf	04/06/2021 14:52	Archivo TF	3 KB
 ADO_variables.tf	04/06/2021 14:20	Archivo TF	2 KB
 az_keyvault.tf	04/06/2021 12:21	Archivo TF	4 KB
 az_networking.tf	04/06/2021 14:20	Archivo TF	2 KB
 az_vars.tf	04/06/2021 14:20	Archivo TF	1 KB
 azure-pipelines.yml	04/06/2021 14:09	Archivo YML	3 KB
 main.tf	04/06/2021 14:20	Archivo TF	2 KB
 outputs.tf	28/05/2021 9:02	Archivo TF	1 KB
 packer.json	02/06/2021 16:08	Archivo JSON	5 KB
 providers.tf	04/06/2021 12:25	Archivo TF	1 KB

*Contenido carpeta despliegue 1*

## Explicación de cada archivo.

Providers.tf

```
provider "azurerm" {  
  features {  
    key_vault {  
      purge_soft_delete_on_destroy = true  
    }  
  }  
}  
  
provider "azuread" {  
}  
  
provider "random" {  
}  
  
terraform {  
  required_providers {  
    azuredevops = {  
      source = "microsoft/azuredevops"  
      version = "0.1.4"  
    }  
  }  
}  
  
provider "azuredevops" {  
  org_service_url = var.org_url  
  personal_access_token = var.pat  
}
```

*archivo providers.tf*

Archivo providers.tf indicaremos los providers que utilizaremos. En el caso del key vault, habilitamos que elimine el soft delete antes de eliminar el recurso. En el provider de azuredevops introduciremos la url de nuestra organización y el Personal Token Access:



```

80 variable "org_url" {
81     description = "Organization url"
82     default     = "https://dev.azure.com, [REDACTED]"
83 }
84
85 variable "pat" {}
86     description = "Personal Access Token"
87     default     = [REDACTED]
88 }

```

*variables providers.tf*

Main.tf

```

# Necesitamos un resource group para poder desplegar
# los recursos básicos para poder desplegar los recursos mediante packer.
####
resource "azurerm_resource_group" "rg" {
    name      = var.rgname
    location = var.location
}

## Obtendremos los datos del usuario conectado:
data "azurerm_client_config" "user_extract" {
}

```

*RG y extracción de datos del usuario*

```

# Creación de un Storage Account
resource "azurerm_storage_account" "stg_acc" {}
    account_replication_type = "LRS"
    account_tier              = "Standard"
    location                  = var.location
    name                      = "packerinstapps${random_string.randomize.result}"
    resource_group_name       = azurerm_resource_group.rg.name
    access_tier                = "Hot"
    account_kind              = "StorageV2"
    enable_https_traffic_only = true
    allow_blob_public_access  = false
}

# Necesitamos un file share donde poner los instaladores del software para la image
resource "azurerm_storage_share" "strg_share" {
    depends_on = [
        azurerm_storage_account.stg_acc
    ]
    name                = "installers"
    storage_account_name = azurerm_storage_account.stg_acc.name
    quota               = 2
}

```

*Storage Account y un file share*

En este archivo, estamos creando un Storage Account con un nombre packerinstapps+5 letras random.

```
# Generador de 5 caracteres para el account storage.
resource "random_string" "randomize" {
  length  = 5
  lower   = true
  upper   = false
  number  = false
  special = false
}
```

*Randomize*

Con el random string randomize generamos un string de 5 letras para hacer único el nombre del storage account.

Creemos un file share con un nombre “installers” donde crearemos una carpeta llamada scripts y alojaremos los archivos de configuración y scripts para maquetar nuestra imagen.

[Ad\\_sp.tf](#)

Aquí vamos a crear nuestro Service Principal, darle el rol de contributor y la contraseña con una duración:

```
1  #Necesitaremos un SP para ejecutar los pipelines
2  resource "azuread_application" "tf_app" {
3    display_name = var.tf_app_name
4  }
5  resource "azuread_service_principal" "tf_sp" {
6    application_id = azuread_application.tf_app.application_id
7  }
8  # Añadimos el rol al SP.
9  resource "azurerms_role_assignment" "sp_role" {
10   principal_id      = azuread_service_principal.tf_sp.object_id
11   scope             = var.scope
12   role_definition_name = "Contributor"
13 }
```

*SP y rol contributor*

```

15 # Generador de contraseña para el SP con una longitud de 32 caracteres.
16 resource "random_password" "password" {
17     length = 32
18     special = true
19 }
20
21 # Aplicación de la contraseña al SP y su validez.
22 resource "azuread_service_principal_password" "tf_sp_pass" {
23     service_principal_id = azuread_service_principal.tf_sp.id
24     value                = random_password.password.result
25     end_date_relative    = "8760h"
26 }

```

*password para el SP*

Con random\_password generamos una clave de longitud de 32 caracteres. También pondremos la contraseña una caducidad, en este caso, de 8760 horas.

Az\_keyvault.tf

```

1 resource "azurerm_key_vault" "key_vault" {
2     depends_on = [
3         azuread_service_principal.tf_sp
4     ]
5     name                = "packer-secrets-${random_string.randomize.result}"
6     location            = var.location
7     resource_group_name = azurerm_resource_group.rg.name
8     tenant_id           = data.azurerm_client_config.user_extract.tenant_id
9     soft_delete_retention_days = 7
10    purge_protection_enabled = false
11    sku_name              = "standard"
12 }
13

```

*Declaración del key vault 1*

Generaremos un Key Vault con un nombre único (usaremos otra vez el randomize)

```

resource "azurerm_key_vault_access_policy" "kv_pol_admin" {
  depends_on = [
    azurerm_key_vault.key_vault
  ]
  key_vault_id = azurerm_key_vault.key_vault.id
  tenant_id    = data.azurem_client_config.user_extract.tenant_id
  object_id    = data.azurem_client_config.user_extract.object_id
  key_permissions = [
    "get", "list", "update", "create", "import", "delete", "recover", "backup", "re
  ]
  secret_permissions = [
    "get", "list", "delete", "recover", "backup", "restore", "set", "purge"
  ]
  certificate_permissions = [
    "get", "list", "update", "create", "import", "delete", "recover", "backup", "re
  ]
}

```

*key\_vault access policy 1*

En este campo, añadiremos el administrador del key vault con los permisos necesarios. Le pasaremos el tenant\_id y el object\_id mediante:

```

10  ## Obtendremos los datos del usuario conectado:
11  data "azurerm_client_config" "user_extract" {
12  }

```

*data extraction 1*

Si quisiéramos poner a otro usuario del tenant como administrador, tendríamos que conocer su object\_id.

```

33  resource "azurerm_key_vault_access_policy" "kv_pol_sp" {
34    depends_on = [
35      azurerm_key_vault_access_policy.kv_pol_admin
36    ]
37    key_vault_id = azurerm_key_vault.key_vault.id
38    tenant_id    = data.azurem_client_config.user_extract.tenant_id
39    object_id    = azuread_service_principal.tf_sp.object_id
40    key_permissions = [
41      "get", "list", "update", "create", "import", "delete", "recover", "backup", "re
42    ]
43    secret_permissions = [
44      "get", "list", "delete", "recover", "backup", "restore", "set", "purge"
45    ]
46    certificate_permissions = [
47      "get", "list", "update", "create", "import", "delete", "recover", "backup", "re
48    ]
49  }

```

*key\_vault access policy 2*

Aquí daremos al SP acceso al key vault (los permisos hay que adecuarlos a las necesidades, esto es un entorno privado). Como antes, extraeremos el tenant

del data->user\_extraction. El object\_id del SP mediante las redirecciones de datos Terraform.

```
# Creación de los secretos para los pipelines:
resource "azurerm_key_vault_secret" "ADOAppID" {
  depends_on = [
    azurerm_key_vault_access_policy.kv_pol_sp
  ]
  key_vault_id = azurerm_key_vault.key_vault.id
  name         = "ADOAppID"
  value        = azuread_service_principal.tf_sp.application_id
}

resource "azurerm_key_vault_secret" "ADOAppSecret" {
  depends_on = [
    azurerm_key_vault_access_policy.kv_pol_sp
  ]
  key_vault_id = azurerm_key_vault.key_vault.id
  name         = "ADOAppSecret"
  value        = azuread_service_principal_password.tf_sp_pass.value
}

resource "azurerm_key_vault_secret" "StorageAccountName" {
  depends_on = [
    azurerm_key_vault_access_policy.kv_pol_sp
  ]
  key_vault_id = azurerm_key_vault.key_vault.id
  name         = "StorageAccountName"
  value        = azurerm_storage_account.stg_acc.name
}

resource "azurerm_key_vault_secret" "StorageAccountKey" {
  depends_on = [
    azurerm_key_vault_access_policy.kv_pol_sp
  ]
  key_vault_id = azurerm_key_vault.key_vault.id
  name         = "StorageAccountKey"
  value        = azurerm_storage_account.stg_acc.primary_access_key
}
```

*key\_vault secrets 1*

Definiremos los secrets en el key vault como hemos utilizado antes, mediante redirección de los datos de Terraform.

El depends\_on es una manera de forzar a Terraform que despliegue solamente cuando la policy de acceso del SP se haya creado.

[Az\\_networking.tf](#)

Generaremos una virtual network, subnet y un NSG para tenerlo preparado para cuando tengamos la imagen y poder hacer pruebas. [Parveen](#) indicaba que era necesario para la creación de la imagen, pero no es necesario. Tengo creada una public ip para mis pruebas, no afecta al tutorial.

Del archivo muestro la parte del NSG para ver la forma en la que extraigo mi ip pública y se la paso al nsg

```
data "http" "icanhazip" {
  url = "http://icanhazip.com"
}

resource "azurerm_network_security_group" "nsg" {
  location          = var.location
  name              = "vm-NSG"
  resource_group_name = azurerm_resource_group.rg.name
}

resource "azurerm_network_security_rule" "nsg_regla" {
  access                = "Allow"
  direction             = "Inbound"
  name                 = "rdp-casa"
  network_security_group_name = azurerm_network_security_group.nsg.name
  priority              = 1100
  protocol              = "Tcp"
  resource_group_name   = azurerm_resource_group.rg.name
  source_port_range     = "*"
  destination_port_range = "3389"
  destination_address_prefix = "*"
  source_address_prefix  = chomp(data.http.icanhazip.body)
}
```

*NSG usando web para mi ip publica 1*

Con data http→http://icanhazip.com conseguimos la ip pública de nuestra vivienda y se la pasamos con `chomp(data.http.icanhazip.body)`.



## Az\_vars.tf

```
az_vars.tf > ...
1  variable "location" {
2    type    = string
3    default = "westeurope"
4  }
5  variable "rgname" {
6    type    = string
7    default = "packer-resources"
8  }
9
10 variable "app_dev" {
11   description = "Rol en el AD para crear nuestro Service Principal"
12   type        = string
13   default     = "Contributor"
14 }
15 variable "tf_app_name" {
16   description = "Nombre del service principal que usaremos para Terraform"
17   type        = string
18   default     = "packer-deploys"
19 }
20
21 variable "subscription_name" {
22   description = "The name for the target subscription"
23   default     = " "
24 }
```

*declaración de variables 1*

## ADO\_project.tf

```
ADO_project.tf > resource "azuredevops_git_repository" "existing_repo"
1  resource "azuredevops_project" "test_project" {
2    depends_on = [
3      azuread_service_principal.tf_sp
4    ]
5    name          = "TF-AzureDevOps-Packer"
6    description   = "Test para ver el funcionamiento mediante terraform"
7    visibility    = "private"
8    version_control = "Git"
9    work_item_template = "Basic"
10   features = {
11     "testplans" = "disabled"
12     "artifacts" = "enabled"
13   }
14 }
```

*declaración project en Azure DevOps 1*

Declaramos un proyecto nuevo en nuestra organización en versión de control Git, la visualización y los features que queramos usar (en este caso, testplans no lo usaremos).

```
#creamos el repo
resource "azuredevops_git_repository" "repo" {
  project_id = azuredevops_project.test_project.id
  name       = "Repositorio para implementar packer y crear una golden image"

  initialization {
    init_type = "Clean"
  }
}
```

*declaración project en Azure DevOps 2*

Aquí creamos un repositorio nuevo donde subiremos los 2 archivos necesario para hacer nuestro pipeline:

Packer.json y azure-pipelines.yml

```
resource "azuredevops_serviceendpoint_azurerm" "endpoint_connection_sp" {
  project_id           = azuredevops_project.test_project.id
  service_endpoint_name = "AzureRMConnection"
  credentials {
    serviceprincipalid = azuread_service_principal.tf_sp.application_id
    serviceprincipalkey = azuread_service_principal_password.tf_sp_pass.value
  }
  azurerm_spn_tenantid      = var.azurerm_spn_tenant_id
  azurerm_subscription_id   = var.azurerm_subscription_id
  azurerm_subscription_name = var.subscription_name
}

## Grant permission to use service connection
resource "azuredevops_resource_authorization" "auth" {
  project_id = azuredevops_project.test_project.id
  resource_id = azuredevops_serviceendpoint_azurerm.endpoint_connection_sp.id
  authorized  = true
}
```

*declaración project en Azure DevOps 3*

Aquí declaramos el endpoint para conectar el SP del tenant y Azure DevOps y le daremos autorización.



## ADO\_variables.tf

```
resource "azuredevops_variable_group" "vars" {  
  project_id = azuredevops_project.test_project.id  
  name       = "packer-image-build-variables"  
  allow_access = true  
  variable {  
    name = "ImageDestRG"  
    value = var.rgname  
  }  
  
  variable {  
    name = "Location"  
    value = var.location  
  }  
  
  variable {  
    name = "StorageAccountInstallerPath"  
    value = "\\packerinstapps${random_string.randomize.result}.file.core.windows.net\\installers"  
  }  
  
  variable {  
    name = "VirtualNetwork"  
    value = azurerm_virtual_network.vnet.name  
  }  
}
```

*Azure DevOps declaración grupo variables 1*

```
variable {  
  name = "Subnet"  
  value = azurerm_subnet.subnet.name  
}  
  
variable {  
  name = "VMSize"  
  value = "Standard_DS3_v2"  
}  
  
variable {  
  name = "SubscriptionID"  
  value = var.scope_id  
}  
  
variable {  
  name = "TenantID"  
  value = var.azurerm_spn_tenant_id  
}  
  
variable {  
  name = "TempResourceGroup"  
  value = "packer-build-images"  
}  
}
```

*Azure DevOps declaración grupo variables 2*

```

resource "azuredevops_variable_group" "key_vars" {
  name           = "keyvault-image-build-variables"
  project_id     = azuredevops_project.test_project.id
  allow_access   = true

  key_vault {
    name                = azurerm_key_vault.key_vault.name
    service_endpoint_id = azuredevops_serviceendpoint_azurerm.endpoint_connection_s
  }
  variable {
    name = "ADOAppID"
  }
  variable {
    name = "ADOAppSecret"
  }
  variable {
    name = "StorageAccountName"
  }
  variable {
    name = "StorageAccountKey"
  }
}

```

*Azure DevOps declaración grupo variables 3*

Muchas de las variables se pueden modificar como: el tamaño de la máquina que se crea provisionalmente, la zona de despliegue, etc...

AVISO. El tamaño de la máquina temporal puede afectar puesto que hay un bug con WinRM que dará timeout. Si os sale un error de WinRM timeout, cambiad de tamaño de la vm.

En el primer grupo de variables de pipelines, hemos generado las claves y valor, pero en el segundo grupo de variables hemos obtenido la claves mediante los permisos del key vault y del SP que tiene los permisos. Si tuviésemos más claves podríamos importarlalas agregando la clave correspondiente.

## Outputs.tf

```
Outputs:

service-principal = "packer-deploys"
sp_appid = "d9d59e50-████████████████████████████████████████"
sp_password = <sensitive>
storage-account = "packerinstappsvtxgc"
storage-fileshare = {
  "acl" = toset([])
  "id" = "https://packerinstappsvtxgc.file.core.windows.net/installers"
  "metadata" = tomap({})
  "name" = "installers"
  "quota" = 2
  "resource_manager_id" = "/subscriptions/████████████████████████████████████████"
  "storage_account_name" = "packerinstappsvtxgc"
  "timeouts" = null /* object */
}
```

*outputs.tf 1*

Con este archivo, visualizamos los parámetros que declaramos y necesitamos visualizar al final el despliegue. Hay outputs que no se mostraran por tener la propiedad sensitive.

```
output "strg-acct-primary-key" {
  value      = azurerm_storage_account.stg_acc.primary_access_key
  sensitive  = true
}
```

Siempre la podemos mostrar si la necesitamos ejecutando:

```
Terraform output sp_password
```

## Terraform init, plan y apply

Aquí no hay mucho que explicar. Solamente os saldrán las variables que no hayáis declarado en los archivos. Yo creo un archivo `azure_configs.tf` con la información clave dentro y que no debe ser compartida 😊

```
Warning: Attribute is deprecated
on ad_sp.tf line 24, in resource "azuread_service_principal_password" "tf_sp_pass":
24:   value      = random_password.password.result

In version 2.0 of the AzureAD provider, this attribute will become read-only as it will no longer be possible to specify a password value.
```

*azureAD provider 2.0 aviso 1*

Para la versión 2.0 no se podrá generar “manualmente” la password del SP y se generará automáticamente.

## Archivos para desplegar la Golden Image en Azure DevOps.

### Packer.json

Con Packer.json generamos la imagen de Windows 10 Multisesión + o365 y ejecutamos los scripts que necesitamos.

Pasamos por la web de [Parveen](#) o su repositorio para los detalles:

<https://github.com/singhparveen/Packer-Image-Build>

### Azure-pipelines.yml

Con este archivo tendremos integración continua, construiremos la imagen mediante Packer (plugin de Azure DevOps), obtendremos las variables (plugin de Azure DevOps)

```
1  trigger:
2  |    batch: true
3
4  name: $(BuildID)
5
6  variables:
7  |    - group: keyvault-image-build-variables
8  |    - group: packer-image-build-variables
```

Trigger, activaremos la integración continua.

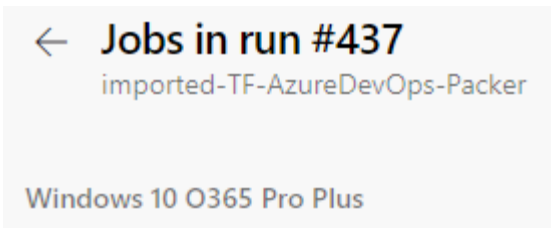
Name: Nombre del build.

Variables: Importamos los grupos de variables de la library en pipelines.

```
stages:
- stage: win10_21h1_evd_o365pp
  displayName: Windows 10 0365 Pro Plus
  jobs:
```

Stage

displayname: el nombre que se mostrará en el pipeline



## Jobs

Job: Construcción de la imagen.

Displayname: el nombre que mostrará.

Pool: quien va a ejecutar el pipeline.

```
jobs:
- job: build
  displayName: Build Image
  pool:
    name: frsteam_agent
    demands:
      - agent.name -equals agent
```

## Steps

Task: el nombre de la extensión del market place de Azure DevOps, en este caso, Packer.

Usaremos la última versión cada vez que lancemos el pipeline.

```
- task: rizebosch.Packer.PackerTool.PackerTool@0
  displayName: 'Use latest Packer'
  inputs:
    version:
      ~~~~~
  task: PackerBuild@0
```

Task: Construcción de la imagen mediante Packer.

CustomTemplateLocation: Archivo donde está la configuración para realizar la construcción de la imagen.

CustomTemplateParameters: Declaramos unas variables del json con otras que hemos declarado en el keyvault.

```
- task: PackerBuild@1
  displayName: 'Build Image'
  inputs:
    templateType: custom
    customTemplateLocation: 'packer.json'
    customTemplateParameters: '{"ADOSvcPrincipalAppId":"$(ADOAppID)","ADOSvcPrincipalSecret":"$(ADOAppSecret)","TenantID":"$(TenantID)"}'
    imageUri: BuildImage
```

Task:

Esta extensión permite usar variables y guardarlas

```
- task: nkdagility.variablehydration.variabledehydration-task.variabledehydration@0
  displayName: 'Save Build Variables: BuildImage'
  inputs:
    prefixes: BuildImage
```

✓ **Save Build Variables: BuildImage**

```
1 Starting: Save Build Variables: BuildImage
2 =====
3 Task      : Variable Save Task
4 Description : Saves the build variables as a .json file so that you can use it later.
5 Version    : 0.2.11
6 Author     : naked Agility - Martin Hinshelwood
7 Help       : v0.2.11 [More Information](https://dev.azure.com/nkdagility/_git/Azure-D
8 =====
9 Saving meta data for BuildImage
10
11 Name      Value
12 ----      -
13 BUILDIMAGE imported-TF-AzureDevOps-Packer-2021-07-05-1602-Build437
14 a
15 created
16
17
18 Finishing: Save Build Variables: BuildImage
```









Task:


Publish Build Artifacts:

Crearemos un artifact con el nombre de la máquina en formato json:

## Summary

### Rolling build triggered

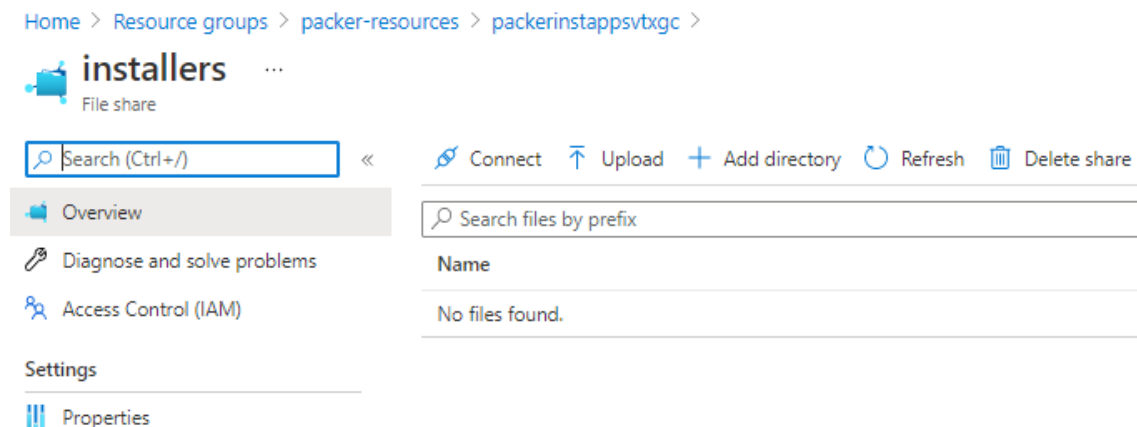
Repository and version	 imported-TF-AzureDevOps-Packer  main  a7e940a
Time started and elapsed	 Yesterday at 18:01  28m 9s
Related	 0 work items  1 published
Tests and coverage	 <a href="#">Get started</a>

 meta-buildimage.json: Bloc de notas

Archivo Edición Formato Ver Ayuda

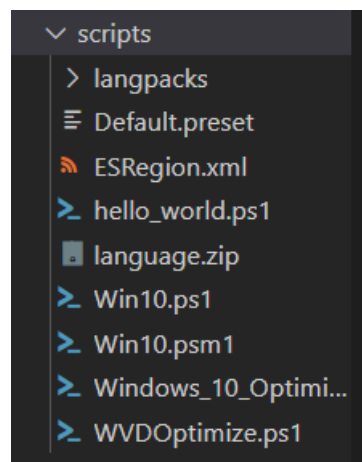
```
{
  "Name": "BUILDIMAGE",
  "Value": "imported-TF-AzureDevOps-Packer-2021-07-05-1602-Build437"
}
```

Subir los scripts que usaremos al File Share.



Dentro de la carpeta installers, crearemos una carpeta scripts o generamos la carpeta con Terraform, subiremos los scripts de powershell que usaremos:

```
resource "azurerm_storage_share_directory" "folder_scripts" {
  share_name           = azurerm_storage_share.strg_share.name
  storage_account_name = azurerm_storage_account.stg_acc.name
  name                 = "scripts"
}
```



Listado de scripts que podríamos utilizar. En la demostración voy a utilizar solamente el script hello\_world.ps1. Los demás scripts serían los de modificar el idioma de Microsoft Windows 10 y scripts para optimizar el rendimiento.

Os adjunto los enlaces de donde sacar los scripts de automatización:

[Install language packs on Windows 10 VMs in Azure Virtual Desktop - Azure | Microsoft Docs](#)

[Optimizing Windows 10, Build 2004, for a Virtual Desktop role | Microsoft Docs](#)

<https://github.com/Disassembler0/Win10-Initial-Setup-Script>



## Azure DevOps

### Consideraciones.

Yo utilizo un agente privado en una máquina virtual que se la paso al azure-pipelines.yml

```
- job: build
  displayName: Build Image
  pool:
    name: tu_agente_privado
    demands:
      - agent.name -equals agent
  steps:
```

#### Auto-provisioning

☒ Auto-provision this agent pool in new projects

#### Agent update settings

☒ Allow agents in this pool to automatically update

También, le tengo puesto que automáticamente se auto aprovisione en los proyectos que se vayan creando.

Otra consideración, actualmente estoy puliendo detalles de importar un repositorio de github pero no está al 100% aún.

### Visualizando lo desplegado.

Una vez desplegado con Terraform apply, visualizamos lo que nos ha desplegado:

**packer-resources** Resource group

Search (Ctrl+/) << + Add Edit columns Delete resource group Refresh Export to CSV Open q

**Overview**

- Activity log
- Access control (IAM)
- Tags
- Events

**Settings**

- Deployments
- Security
- Policies
- Properties
- Locks

**Cost Management**

- Cost analysis
- Cost alerts (preview)

**Essentials**

Subscription (change) : Suscripción de Plataformas de MSDN

Subscription ID :

Tags (change) : [Click here to add tags](#)

Filter for any field... Type == all Location == all Add filter

Showing 1 to 6 of 6 records. ☐ Show hidden types ⓘ

<input type="checkbox"/> Name ↑↓
<input type="checkbox"/> nic-vm
<input type="checkbox"/> packer-secrets-vbxc
<input type="checkbox"/> packerinstappsvtbxc
<input type="checkbox"/> pip
<input type="checkbox"/> vm-NSG
<input type="checkbox"/> vnet

/ TF-AzureDevOps-Packer / Repos / Files / TF-AzureDevOps-Packer

## TF-AzureDevOps-Packer is empty. Add some code!

**Clone to your computer**

HTTPS SSH  OR

ⓘ Having problems authenticating in Git? Be sure to get the latest version [Git for Windows](#) or our plugins for [IntelliJ](#), [Eclipse](#), [Android Studio](#) or [Windows comm](#)

**Push an existing repository from command line**

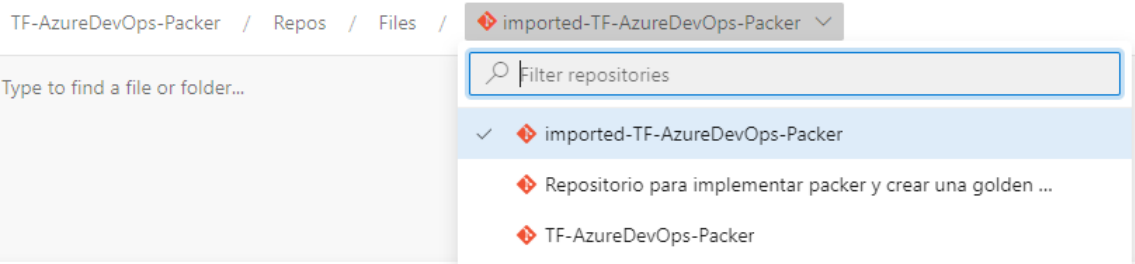
HTTPS SSH

**Import a repository**

**Initialize main branch with a README or gitignore**





☒ Add a README

Para mostrar el repositorio importado de Github, en files, pinchamos en la parte de los repositorios y seleccionamos el importado:



## Library

Variable groups | Secure files | + Variable group | Security | Help

Name ↑	Date modified	Modified by
 keyvault-image-build-variables	8 minutes ago	 Raúl Carvajal Bustos
 packer-image-build-variables	17 minutes ago	 Raúl Carvajal Bustos

Properties

Variable group name

keyvault-image-build-variables

Description

☒ Link secrets from an Azure key vault as variables

Azure subscription \* | [Manage](#)



Key vault name \* | [Manage](#)

packer-secrets-vtxgc



Variables

Last refreshed: Invalid Date

Delete	Secret name	Content type	Status	Expiration date
	ADOAppID		Enabled	Never
	ADOAppSecret		Enabled	Never
	StorageAccountKey		Enabled	Never
	StorageAccountName		Enabled	Never

[+ Add](#)

Variable group name

packer-image-build-variables

Description

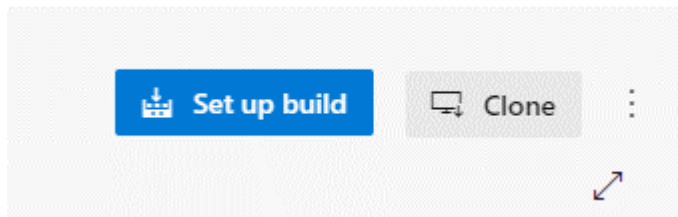
☐ Link secrets from an Azure key vault as variables ⓘ

Variables

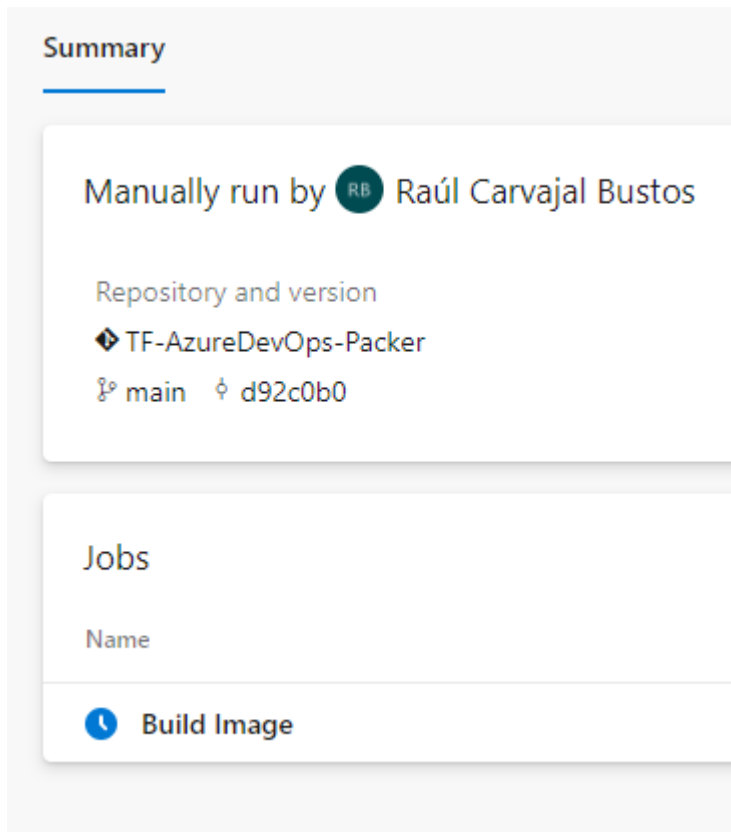
Name ↑	Value
ImageDestRG	packer-resources
Location	westeurope
StorageAccountInstallerPath	\\packerinstappsvtxgc.blob.core.windows.net\installers
Subnet	subnet
SubscriptionID	65770b1a-194b-477d-b6f9-f766e8792a6b
TempResourceGroup	packer-build-images
TenantID	41ebac77-4ea4-4e10-80b3-ee28b24f9ef5
VirtualNetwork	vnet
VirtualNetworkRG	packer-resources
VMSize	Standard_DS3_v2

## Creando el pipeline.

Con el repositorio importado y seleccionado, podemos ejecutar nuestra primera build:



Ejecutamos el pipeline



Hay que aceptar el consentimiento del agente:

## Waiting for review



Windows 10 O365 Pro Plus



Permission  
Permission needed



Queue

Permit

El proceso tardará dependiendo de los scripts que le pongamos, pero de 15 minutos no baja.

←

**Jobs in run #437**  
imported-TF-AzureDevOps-Packer

Windows 10 O365 Pro Plus

✓	Build Image	28m 2s
✓	Initialize job	<1s
⌚	Pre-job: Download secrets: pack...	<1s
✓	Download secrets: packer-secrets-...	2s
✓	Checkout imported-TF-AzureDev...	2s
✓	Use latest version Packer	1s
✓	Build Image	27m 48s
✓	Save Build Variables: BuildImage	2s
✓	Publish Artifact: Build Image and ...	2s
✓	Post-job: Checkout imported-TF-...	1s
✓	Finalize Job	<1s
✓	Report build status	<1s

✓ **Build Image**

1

Pool: frsteam\_agent

2

Agent: agent

3

Started: Yesterday at 18:01

4

Duration: 28m 2s

5

6

▶ Job preparation parameters

7

1 artifact produced

packer-build-images  
Resource group

Search (Ctrl+ /) << + Add Edit columns Delete resource group

Overview

Activity log

Access control (IAM)

Tags

Events

Settings

Deployments

Security

Policies

Properties

Locks

Cost Management

Cost analysis

Cost alerts (preview)

Budgets

Essentials

Subscription (change) : Suscripción de Plataformas de M...

Subscription ID :

Tags (change) : Image Offer : office-365

Filter for any field... Type == all Location

Showing 1 to 5 of 5 records. ☐ Show hidden types ⓘ

☐ Name ↑↓

<input type="checkbox"/>	pkripy04tzdr15u	
<input type="checkbox"/>	pkrkvy04tzdr15u	
<input type="checkbox"/>	pkrmny04tzdr15u	
<input type="checkbox"/>	pkrosy04tzdr15u	
<input type="checkbox"/>	pkrvmy04tzdr15u	

*RG paralelo temporal de la VM 1*

En este grupo paralelo se ejecuta la instancia de la máquina y se le pasan los scripts y luego creará la imagen en nuestro grupo de recursos. Una vez se pase la imagen a nuestro grupo de recursos, el temporal lo destruirá Packer.

```
543
544 ==> azure-arm: Provisioning with powershell script: Z://hello.ps1
545     azure-arm: Hello World!! I'm a script from a file-share
546 ==> azure-arm: Pausing 1m0s before the next provisioner...
```

*script ejecutado desde el file share 1*

Vemos como continúa nuestro despliegue y la ejecución de scripts. Lo último que realizaría sería el sysprep que tenemos al final del Packer y nos aseguramos de que ciertamente es lo último que ejecuta.



```

622
623 ==> azure-arm: -> ResourceGroupName : 'packer-build-images'
624
625 ==> azure-arm: -> ComputeName       : 'pkrvmy04tzdr15u'
626 ==> azure-arm: -> Managed OS Disk   : '/subscriptions/65770b1a-194b-
627
628 ==> azure-arm: Querying the machine's additional disks properties ..
629
630 ==> azure-arm: -> ResourceGroupName : 'packer-build-images'
631
632 ==> azure-arm: -> ComputeName       : 'pkrvmy04tzdr15u'
633
634 ==> azure-arm: Powering off machine ...
635
636 ==> azure-arm: -> ResourceGroupName : 'packer-build-images'
637
638 ==> azure-arm: -> ComputeName       : 'pkrvmy04tzdr15u'

```

*sysprep ejecutado y apagando la máquina 1*

```

638 ==> azure-arm: -> ComputeName       : 'pkrvmy04tzdr15u'
639 ==> azure-arm: Capturing image ...
640
641 ==> azure-arm: -> Compute ResourceGroupName : 'packer-build-images'
642
643 ==> azure-arm: -> Compute Name       : 'pkrvmy04tzdr15u'
644
645 ==> azure-arm: -> Compute Location   : 'westeurope'
646
647 ==> azure-arm: -> Image ResourceGroupName : 'packer-resources'
648
649 ==> azure-arm: -> Image Name       : 'TF-AzureDevOps-Packer-2021-06-07-1051-Build155'
650
651 ==> azure-arm: -> Image Location   : 'westeurope'

```

*Imagen creada 1*

# packer-resources

Resource group

Overview

Activity log

Access control (IAM)

Tags

Permissions

Groups

Deployments

Security

Policies

Properties

Jobs

Management

Cost analysis

Cost alerts (preview)

+ Add

≡ Edit columns

🗑 Delete resource group

🔄

^ Essentials

Subscription (change) : [Suscripción de Plataformas de MSDN](#)

Subscription ID :

Tags (change) : [Click here to add tags](#)

Filter for any field...

Type == all X

Location ==

Showing 1 to 7 of 7 records. ☐ Show hidden types ⓘ

☐ Name ↑↓

☐ nic-vm

☐ packer-secrets-vtxgc

☐ packerinstappsvtxgc

☐ pip

☐ TF-AzureDevOps-Packer-2021-06-07-1051-Build155

☐ vm-NSG

☐ vnet

imagen creada 2

Ya tenemos la imagen creada y en nuestro grupo de recursos.

← Jobs in run #437

imported-TF-AzureDevOps-Packer

Windows 10 O365 Pro Plus

✓	Build Image	28m 2s
✓	Initialize job	<1s
⌚	Pre-job: Download secrets: pack...	<1s
✓	Download secrets: packer-secrets-...	2s
✓	Checkout imported-TF-AzureDev...	2s
✓	Use latest version Packer	1s
✓	Build Image	27m 48s
✓	Save Build Variables: BuildImage	2s
✓	Publish Artifact: Build Image and ...	2s
✓	Post-job: Checkout imported-TF-...	1s
✓	Finalize Job	<1s
✓	Report build status	<1s

✓ Build Image

1 Pool: [frsteam\\_agent](#)

2 Agent: agent

3 Started: Yesterday at 18:01

4 Duration: 28m 2s

5

6 ▶ Job preparation parameters

7 📦 1 artifact produced

Azure Pipelines

succeeded

Con esto ya hemos acabado el pipeline y podíamos crear una máquina virtual con la imagen.

El tamaño de la vm que declaramos en Packer es solamente la temporal y con la que se genera la imagen, podemos poner una potente para que los scripts se ejecuten más rápido.