ARM Templates

Arm templates – Infraestructura como código

https://docs.microsoft.com/es-es/azure/azure-resource-manager/templates

Los ARM Templates son documentos declarativos para describir los recursos de Azure.

Para implementar la infraestructura como código para las soluciones de Azure, use las plantillas de Azure Resource Manager (ARM). La plantilla es un archivo de notación de objetos JavaScript (JSON) que contiene la infraestructura y la configuración del proyecto. La plantilla usa sintaxis declarativa, lo que permite establecer lo que pretende implementar sin tener que escribir la secuencia de comandos de programación para crearla. En la plantilla se especifican los recursos que se van a implementar y las propiedades de esos recursos.

¿Por qué usarlos?

Sintaxis declarativa:

Las plantillas de Resource Manager permiten crear e implementar una infraestructura de Azure completa de forma declarativa.

Resultados repetibles:

Implemente repetidamente la infraestructura a lo largo del ciclo de vida del desarrollo y tenga la seguridad de que los recursos se implementan de forma coherente. Las plantillas son idempotentes, lo que significa que puede implementar la misma plantilla varias veces y obtener los mismos tipos de recursos en el mismo estado. Puede desarrollar una plantilla que represente el estado deseado, en lugar de desarrollar muchas plantillas independientes para representar las actualizaciones.

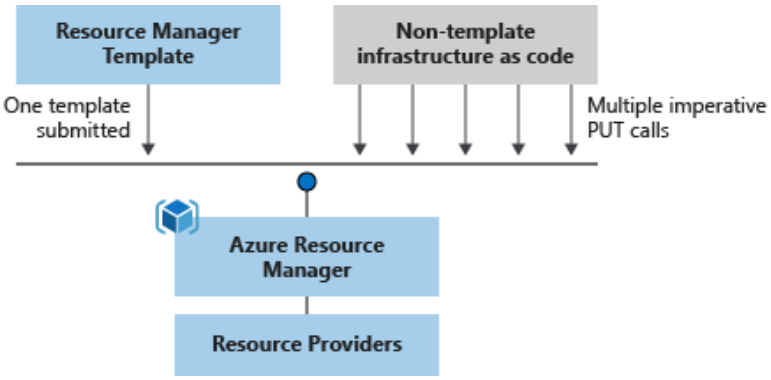
Parametrizar y usar un parameter template

Orquestación:

Resource Manager se encarga de gestionar la implementación de recursos interdependientes para que se creen en el orden correcto.

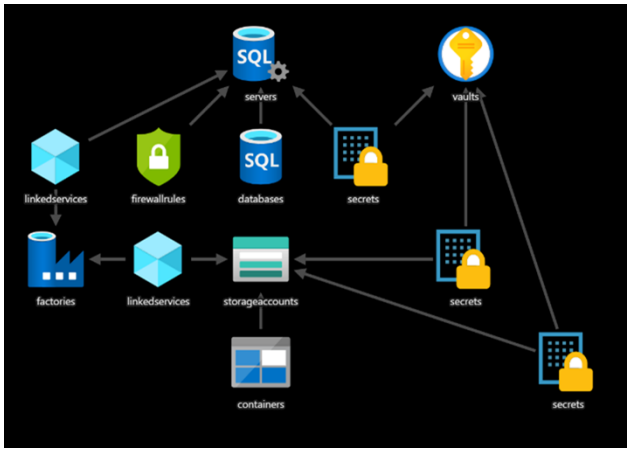
Cuando es posible, Resource Manager implementa los recursos en paralelo para que las implementaciones finalicen más rápido que las implementaciones en serie.

La plantilla se implementa mediante un comando, en lugar de hacerlo con varios comandos imperativos.



Se pueden crear **dependencias** entre los recursos para determinar el orden del deployment. Las dependencias pueden crearse dentro del mismo recurso como "child resources" o por fuera.

Las copy iterations no pueden ser parte de un child resource. Por eso, si queremos usar el copy, crear los recursos y determinar las dependencias por fuera del recurso padre (necesita ser un top level resource).



Con la extensión ARM Viewer, presionando F1 se pueden observar gráficamente los recursos.

Algunos de los recursos no se pueden implementar hasta que existe otro recurso. Por ejemplo, necesitará un servidor SQL Server antes de implementar una base de datos SQL. Esta relación se define al marcar un recurso como dependiente de los otros, y se define con el elemento *dependsOn* o mediante la función *reference*. El administrador de recursos evalúa las dependencias entre recursos y los implementa en su orden dependiente. Cuando no hay recursos dependientes entre sí,

Resource Manager los implementa en paralelo. Solo tiene que definir las dependencias de recursos que se implementan en la misma plantilla.

Al definir las dependencias, puede incluir el espacio de nombres de proveedor de recursos y el tipo de recurso para evitar la ambigüedad. Por ejemplo, para aclarar un equilibrador de carga y la red virtual que puede tener los mismos nombres que otros recursos, utilice el formato siguiente:

```
"dependsOn": [
  "[resourceId('Microsoft.Network/loadBalancers', variables('loadBalancerName'))]",
  "[resourceId('Microsoft.Network/virtualNetworks',variables('virtualNetworkName'))]"
]
```

Si bien puede que se sienta tentado a usar *dependsOn* para asignar relaciones entre los recursos, es importante comprender por qué lo hace. Por ejemplo, para documentar cómo están interconectados los recursos, *dependsOn* no es el enfoque correcto. No se pueden consultar los recursos que se definieron en el elemento *dependsOn* después de realizar la implementación. El uso de *dependsOn* podría llegar a repercutir en el tiempo de implementación, ya que Resource Manager no implementa dos recursos en paralelo que tengan una dependencia.

Recursos secundarios

La propiedad *resources* permite especificar los recursos secundarios (child resources) que están relacionados con el recurso que se está definiendo. Los recursos secundarios solo se pueden definir en cinco niveles de profundidad. Es importante tener en cuenta que no se crea una dependencia de implementación implícita entre un recurso secundario y el recurso primario. Si necesita que el recurso secundario se implemente después del recurso primario, debe declarar explícitamente esa dependencia con la propiedad *dependsOn*.

Cada recurso primario solo acepta determinados tipos de recursos como recursos secundarios. Los tipos de recursos que se aceptan se especifican en el esquema de plantilla del recurso principal. El nombre del tipo de recurso secundario incluye el nombre del tipo de recurso primario, por ejemplo, Microsoft.Web/sites/config y Microsoft.Web/sites/extensions son ambos recursos secundarios de Microsoft.Web/Sites.

En el ejemplo siguiente se muestran un servidor SQL y una base de datos SQL. Observe que se ha definido una dependencia explícita entre la base de datos SQL y el servidor SQL, a pesar de que la base de datos es un elemento secundario del servidor.

```
"resources": [
  {
    "name": "[variables('sqlserverName')]",
    "apiVersion": "2014-04-01-preview",
    "type": "Microsoft.Sql/servers",
    "location": "[resourceGroup().location]",
    "tags": {
      "displayName": "SqlServer"
    },
    "properties": {
      "administratorLogin": "[parameters('administratorLogin')]",
      "administratorLoginPassword": "[parameters('administratorLoginPassword')]"
    },
    "resources": [
      {
        "name": "[parameters('databaseName')]",
        "apiVersion": "2014-04-01-preview",
        "type": "databases",
        "location": "[resourceGroup().location]",
        "dependsOn": [
          "[variables('sqlserverName')]"
        ],
        "tags": {
          "displayName": "Database"
        },
        "properties": {
          "edition": "[parameters('edition')]",
          "collation": "[parameters('collation')]",
          "maxSizeBytes": "[parameters('maxSizeBytes')]",
          "requestedServiceObjectiveName": "[parameters('requestedServiceObjectiveName')]"
        }
      }
    ]
  }
]
```

Si el recurso se crea como child resource, no es necesario especificar el nombre completo en Type

Funciones de referencia y lista

La función *reference* permite que una expresión derive su valor de otros pares de valor y nombre JSON o de recursos en tiempo de ejecución. Las *funciones list** devuelven valores para un recurso de una operación de lista. Las expresiones de referencia y lista declaran implícitamente que un recurso depende de otro, cuando el recurso al que se hace referencia está implementado en la misma plantilla y se hace referencia a él por su nombre (no por el identificador de recurso). Si se pasa el identificador de recurso a las funciones de referencia o lista, no se crea una referencia implícita.

El formato general de la función de referencia es:

```
reference('resourceName').propertyPath
```

El formato general de la función listKeys es:

```
listKeys('resourceName', 'yyyy-mm-dd')
```

En el ejemplo siguiente, un punto de conexión de CDN depende explícitamente del perfil de CDN e implícitamente de una aplicación web.

```
{
    "name": "[variables('endpointName')]",
    "apiVersion": "2016-04-02",
    "type": "endpoints",
    "location": "[resourceGroup().location]",
    "dependsOn": [
      "[variables('profileName')]"
    ],
    "properties": {
      "originHostHeader": "[reference(variables('webAppName')).hostNames[0]]",
      ...
    }
```

Puede usar este elemento o el elemento *dependsOn* para especificar las dependencias, pero no es necesario usar ambos para el mismo recurso dependiente. Siempre que sea posible, use una referencia implícita para evitar agregar una dependencia innecesaria.

Mediante el uso de la función de referencia, se declara implícitamente que un recurso depende de otro recurso si el recurso al que se hace referencia se aprovisiona en la misma plantilla y se hace referencia a él por su nombre (y no por el identificador del recurso). No tiene que usar también la propiedad dependsOn. La función no se evalúa hasta que el recurso al que se hace referencia haya completado la implementación.

Dependencias circulares

Resource Manager identifica dependencias circulares durante la validación de plantillas. Si recibe un error que indica que existe una dependencia circular, evalúe la plantilla para ver si algunas dependencias no son necesarias y se pueden quitar. Si no es suficiente con quitar dependencias, puede evitar dependencias circulares moviendo algunas operaciones de implementación a recursos secundarios que se implementan después de los recursos que tienen la dependencia circular. Por ejemplo, suponga que va a implementar dos máquinas virtuales pero debe establecer propiedades en cada una que hagan referencia a la otra. Puede implementarlas en el orden siguiente:

1. vm1
2. vm2
3. La extensión en vm1 depende de vm1 y vm2. La extensión establece valores en vm1 que obtiene de vm2.
4. La extensión en vm2 depende de vm1 y vm2. La extensión establece valores en vm2 que obtiene de vm1.

Best Practices - Dependencias de recursos

- Use la función *reference* y pase el nombre del recurso para establecer dependencias implícitas entre los recursos que deben compartir una propiedad. No agregue un elemento *dependsOn* explícito cuando ya haya definido una dependencia implícita. Este enfoque reduce el riesgo de que se tengan dependencias innecesarias.
- Establezca un recurso secundario como dependiente de su recurso principal.
- Los recursos con el elemento *condition* establecido en *false* se quitan automáticamente de la orden de dependencia. Establezca las dependencias como si siempre se implementase el recurso.

- Permita dependencias en cascada sin establecerlas explícitamente.
- Si no se puede determinar un valor antes de la implementación, intente implementar el recurso sin una dependencia. Por ejemplo, si un valor de configuración necesita el nombre de otro recurso, quizás no necesite una dependencia. Esta guía no siempre funciona porque algunos recursos comprueban la existencia de los otros. Si recibe un error, agregue una dependencia.

Tutorial

Archivos Modulares:

Puede dividir las plantillas en componentes más pequeños y reutilizables y vincularlos en el momento de la implementación. También puede anidar una plantilla dentro de otras plantillas.

- **Linked Templates**

Es un archivo de plantilla independiente al que se hace referencia a través de un vínculo de la plantilla principal.

Para vincular una plantilla, agregue un recurso de implementaciones a la plantilla principal. En la propiedad *templateLink*, especifique el URI de la plantilla que se va a incluir.

Al hacer referencia a una plantilla vinculada, el valor de *uri* no debe ser un archivo local o un archivo que solo esté disponible en la red local. Debe proporcionar un valor de URI que se puede descargar como **http** o **https**.

Puede proporcionar los parámetros de la plantilla vinculada en un archivo externo o alineados. Al proporcionar un archivo de parámetros externo, use la propiedad *parametersLink.*

```
"resources": [
  {
  "type": "Microsoft.Resources/deployments",
  "apiVersion": "2019-10-01",
  "name": "linkedTemplate",
  "properties": {
    "mode": "Incremental",
    "templateLink": {
```

```
"uri":"https://mystorageaccount.blob.core.windows.net/AzureTemplates/newStorageAccount.json",
      "contentVersion":"1.0.0.0"
    },
    "parametersLink": {
"uri":"https://mystorageaccount.blob.core.windows.net/AzureTemplates/newStorageAccount.parameters.json",
      "contentVersion":"1.0.0.0"
    }
  }
  }
]
```

Para pasar los valores de parámetro alineados, use la propiedad *parameters*.

```
"resources": [
  {
   "type": "Microsoft.Resources/deployments",
   "apiVersion": "2019-10-01",
   "name": "linkedTemplate",
   "properties": {
     "mode": "Incremental",
     "templateLink": {
"uri":"https://mystorageaccount.blob.core.windows.net/AzureTemplates/newStorageAccount.json",
      "contentVersion":"1.0.0.0"
     },
     "parameters": {
      "StorageAccountName":{"value": "[parameters('StorageAccountName')]"}
    }
   }
  }
]
```

No pueden utilizarse simultáneamente la propiedad de parámetros y parametersLink. La implementación produce un error cuando ambos (*parametersLink* y *parameters*) se especifican.

Si no proporciona un *contentVersion*, se implementará la versión actual de la plantilla. Si proporciona un valor, este debe coincidir con la versión de la plantilla vinculada o, de lo contrario, se producirá un error durante la implementación.

- **Nested Templates**

Son plantillas insertadas dentro de la plantilla principal.

Para anidar una plantilla, agregue un recurso de implementaciones a la plantilla principal. En la propiedad template, especifique la sintaxis de la plantilla.

Al utilizar una plantilla anidada, puede especificar si las expresiones de plantilla se evalúan dentro del ámbito de la plantilla principal o de la plantilla anidada (Outer/Inner) . El ámbito determina cómo se resuelven los parámetros, las variables y las funciones como *resourceGroup* y *subscription*.

El ámbito se establece mediante la propiedad *expressionEvaluationOptions*. De forma predeterminada, la propiedad *expressionEvaluationOptions* está establecida en *outer*, lo que significa que usa el ámbito de plantilla principal. Establezca el valor en *inner* para que las expresiones se evalúen dentro del ámbito de la plantilla anidada.

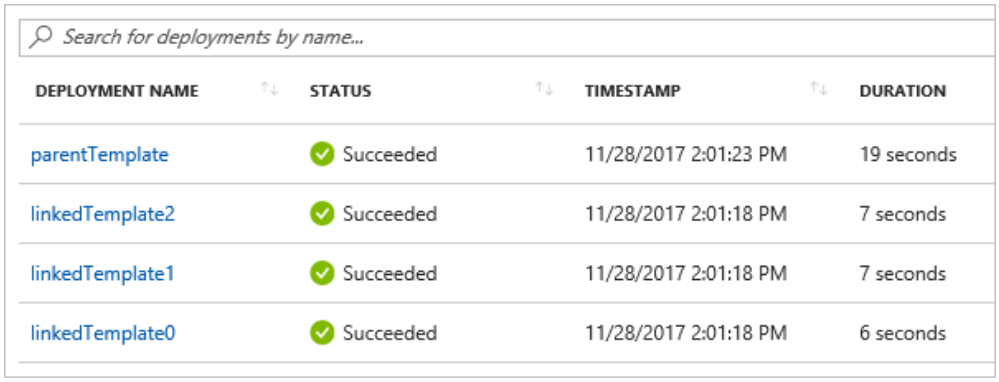
expressionEvaluationOptions scope	**Output**
inner	from nested template
outer (or default)	from parent template

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {},
  "variables": {},
  "resources": [
    {
      "name": "nestedTemplate1",
      "apiVersion": "2019-10-01",
      "type": "Microsoft.Resources/deployments",
      "properties": {
        "mode": "Incremental",
        "expressionEvaluationOptions": {
```

```
            "scope": "inner"
            },
        "template": {
          <nested-template-syntax>
        }
      }
    }
  ],
  "outputs": {
  }
}
```

Historial de implementación

Resource Manager procesa cada plantilla como una implementación independiente en el historial de implementación. Una plantilla principal con tres plantillas vinculadas o anidadas aparece en el historial de implementación del modo siguiente:



Cree cualquier recurso de Azure:

Puede usar inmediatamente los nuevos servicios y características de Azure en las plantillas. En cuanto un proveedor de recursos introduce nuevos recursos, puede implementarlos a través de plantillas. No tiene que esperar a que se actualicen las herramientas o los módulos antes de usar los nuevos servicios.

Extensibilidad:

Con los scripts de implementación (deployment), puede agregar scripts de PowerShell o Bash a las plantillas. Los scripts de implementación amplían su capacidad para configurar recursos durante la

implementación. Un script se puede incluir en la plantilla o almacenar en un origen externo y hacer referencia a él en la plantilla. Los scripts de implementación le ofrecen la posibilidad de completar la configuración del entorno integral en una sola plantilla de ARM.

Con un nuevo tipo de recurso denominado Microsoft.Resources/deploymentScripts, los usuarios pueden ejecutar scripts de implementación en implementaciones de plantilla y revisar los resultados de la ejecución. Estos scripts se pueden usar para realizar pasos personalizados tales como:

- Adición de usuarios a un directorio
- Realización de operaciones de plano de datos como, por ejemplo, copiar blobs o una base de datos de inicialización
- Búsqueda y validación de una clave de licencia
- Creación de un certificado autofirmado.
- Creación de un objeto en Azure AD
- Búsqueda de bloques de direcciones IP en el sistema personalizado

Ventajas del script de implementación:

- Fácil de programar, usar y depurar. Puede desarrollar scripts de implementación en sus entornos de desarrollo favoritos. Los scripts se pueden insertar en plantillas o en archivos de script externos.
- Puede especificar el lenguaje y la plataforma del script. Actualmente, se admiten scripts de implementación de Azure PowerShell y la CLI de Azure en el entorno de Linux.
- Permite especificar las identidades que se usan para ejecutar los scripts. Actualmente, solo se admiten identidades asignadas por el usuario de Azure.
- Permite pasar cuatro argumentos de la línea de comandos al script.
- Puede especificar salidas de script y pasarlas de nuevo a la implementación.
- El recurso de script de implementación solo está disponible en las regiones donde Azure Container Instances está disponible.

Pruebas:

Puede asegurarse de que la plantilla sigue las instrucciones recomendadas si la prueba con el kit de herramientas de la plantilla ARM (arm-ttk). Este kit de pruebas es un script de PowerShell que puede

descargar de GitHub. El kit de herramientas facilita el desarrollo de conocimientos con el lenguaje de plantilla.

Vista previa de los cambios:

Puede usar la operación hipotética (What-if) para obtener una vista previa de los cambios antes de implementar la plantilla. Con la operación hipotética puede ver qué recursos se crearán, actualizarán o eliminarán, así como las propiedades de los recursos que cambiarán. La operación hipotética comprueba el estado actual del entorno y elimina la necesidad de administrar el estado.

Para hacer un What-If se necesita conocer todo sobre el template, el grupo de recursos, la suscripción, el tenant y todos los datos referidos a donde se implementará el template, un archivo de parámetros o que estén listados los parámetros, el modo de implementación (completo o incremental). Luego se envia a Azure, donde se realizan dos tareas: se predice el estado deseado y luego se hace un GET request de todos los recursos en el scope para obtener el estado actual. Si los recursos ya existen se realiza una comparación y se obtiene un “calculated diff” obteniendo una lista de los recursos y que acción se implementara sobre cada una en caso de que se realice la implementación del template.

Validación integrada:

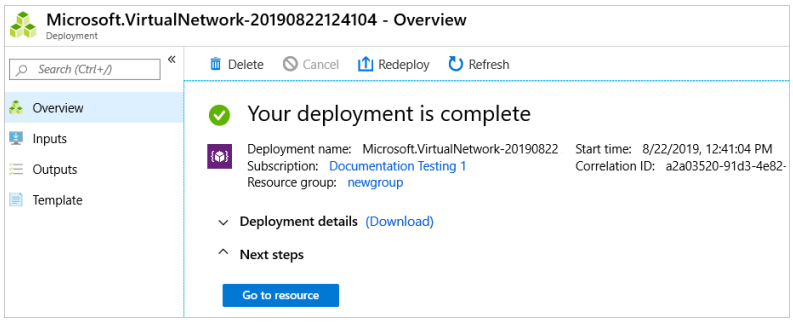
La plantilla solo se implementa después de pasar la validación. Resource Manager se encarga de comprobar la plantilla antes de iniciar la implementación para asegurarse de que esta se realizará correctamente. Es menos probable que la implementación se detenga a medio acabar.

En el caso de hacer la implementación por medio de PowerShell, tener en cuenta si se determina un parámetro como SecureString porque no podrá hacer las validaciones correspondientes (por ejemplo, en el caso de Linked Services)

Implementaciones con seguimiento:

En Azure Portal puede revisar el historial de implementación y obtener información sobre la implementación de la plantilla. También puede ver la plantilla que se implementó, los valores de parámetro agregados y los valores de salida. Recuerde que no se realiza el seguimiento de otras infraestructuras como servicios de código a través del portal.

Si la implementación se realiza mediante el portal, el nombre de la misma será el otorgado por defecto por Azure, si se realiza otra implementación se reemplazará y no se podrá hacer el seguimiento de la anterior. Si la implementación se realiza por Bash o PowerShell, es posible



determinar el nombre y realizar el seguimiento por el Azure Portal.

Directiva como código:

Azure Policy es un marco de directiva como código para automatizar la gobernanza. Si usa directivas de Azure, la corrección de la directiva se realiza en recursos no compatibles cuando se implementa mediante plantillas.

Planos técnicos de implementación:

Puede aprovechar las ventajas de los Planos técnicos (Azure Bluepints) proporcionados por Microsoft para cumplir los estándares de cumplimiento normativo. Estos planos técnicos incluyen plantillas precompiladas para distintas arquitecturas.

Código exportable:

Puede recuperar una plantilla de un grupo de recursos existente mediante la exportación del estado actual del grupo de recursos o la visualización de la plantilla de una implementación determinada. Una buena estrategia para aprender sobre la sintaxis de una plantilla es consultar la plantilla exportada.

Integración de CI/CD:

Puede integrar plantillas en sus herramientas de integración e implementación continuas (CI/CD), que pueden automatizar las canalizaciones (pipelines) de versión para llevar a cabo actualizaciones de infraestructura y aplicaciones rápidas y confiables. Mediante la tarea de plantilla de Resource Manager y Azure DevOps puede usar Azure Pipelines para compilar e implementar proyectos de plantillas de Resource Manager de manera continua.

Visual Studio proporciona el proyecto del grupo de recursos de Azure para crear plantillas de Azure Resource Manager (ARM) e implementarlas en su suscripción a Azure. Este proyecto se puede integrar con Azure Pipelines para la integración e implementación continuas (CI/CD).

Hay dos maneras de implementar plantillas con Azure Pipelines:

- **Agregar una tarea que ejecute un script de Azure PowerShell**. Esta opción tiene la ventaja de ofrecer coherencia en todo el ciclo de vida de desarrollo debido a que usa el mismo script que se incluye en el proyecto de Visual Studio (Deploy-AzureResourceGroup.ps1). El script agrega los artefactos del proyecto al "stage" en una cuenta de almacenamiento a la que Resource Manager puede acceder. Los artefactos son elementos del proyecto tales como plantillas vinculadas, scripts y archivos binarios de aplicación. Luego, el script implementa la plantilla.
- **Agregar tareas para copiar e implementar tareas**. Esta opción ofrece una alternativa conveniente al script del proyecto. Configure dos tareas en la canalización. Una tarea agrega los artefactos al "stage" y la otra tarea implementa la plantilla.

Tutorial

Más info: Integración de plantillas de ARM con Azure Pipelines

Herramientas de creación:

Puede crear plantillas con Visual Studio Code y la extensión de la herramienta de plantillas. Podrá utilizar IntelliSense, el resaltado de sintaxis, la ayuda en línea y muchas otras funciones de lenguaje. Además de Visual Studio Code, también puede usar Visual Studio.

Descargar las extensiones:

- Azure Resource Manager (ARM) Tools
    - Comentarios // y /* */
    - Multi-lines strings
    - Case Insensitivity
    - Open-source snippets
    - IntelliSense
- ARM Template Viewer

Template – Json

Secciones:

- Parameters
  - Valores proporcionados durante la implementación que permitan usar la misma plantilla con entornos diferentes
  - Se pueden determinar mediante un parameter template
- Variables
  - Valores que se reutilizan en las plantillas. Se pueden crear a partir de parámetros
- User-defined functions
  - Funciones personalizadas que simplifican la plantilla
- Resources
  - Recursos que se van a implementar
- Outputs
  - Valores devueltos de los recursos implementados

Ctrl + espacio para IntelliSense

Best practices - Límites de plantilla

Limite el tamaño de la plantilla a 4 MB y cada archivo de parámetros a 64 KB. El límite de 4 MB se aplica al estado final de la plantilla una vez se ha ampliado con definiciones de recursos iterativas y los valores de variables y parámetros.

También está limitado a:

- 256 parámetros
- 256 variables
- 800 recursos (incluido el recuento de copia)
- 64 valores de salida
- 24 576 caracteres en una expresión de plantilla

Usar **Naming conventions!!**

Parámetros

Componentes de los parámetros

- Name: nombre del parámetro. Se recomienda utilizar Camel Case.
- Type: puede ser array, bool, int, object, secureObject, secureString, string.
- MinLength / MaxLength: no utilizar condiciones de largo para parámetros seguros ya que no se podrá realizar la validación en la implementación.
- DefaultValue: Valor que se utilizará por defecto en la implementación si no se le otorga uno distinto.
- AllowedValues: array de valores permitidos.
- MinValue / MaxValue: valor mínimo o máximo que puede tomar el parámetro.

```
"storageName": {
   "type": "string",
   "minLength":3,
   "maxLength":24,
   "defaultValue": "[resourceGroup().name]",
   "metadata": {
         "description": "Nombre de l
cuenta de almacenamiento.
Si no brinda un nombre se asignará eldel grupo de
 recursos "
```

resourceGroup() es una función que permite obtener información del grupo de recursos

Objetos como parámetros

– Puede ser más fácil organizar los valores relacionados pasándolos como objetos. Con este enfoque también se reduce el número de parámetros de la plantilla. Para hacer referencia a las propiedades del objeto, utilizar el operador punto.

Ejemplo:

```
"parameters": {
```

```
  "VNetSettings": {
    "type": "object",
    "defaultValue": {
      "name": "VNet1",
      "location": "eastus",
      "addressPrefixes": [
        {
          "name": "firstPrefix",
          "addressPrefix": "10.0.0.0/22"
        }
      ],
      "subnets": [
        {
          "name": "firstSubnet",
          "addressPrefix": "10.0.0.0/24"
        },
        {
          "name": "secondSubnet",
          "addressPrefix": "10.0.1.0/24"
        }
      ]
    }
  }
},
```

Haga referencia a las propiedades del objeto con el operador punto.

```
"resources": [
  {
    "type": "Microsoft.Network/virtualNetworks",
    "apiVersion": "2015-06-15",
    "name": "[parameters('VNetSettings').name]",
    "location": "[parameters('VNetSettings').location]",
    "properties": {
      "addressSpace":{
        "addressPrefixes": [
          "[parameters('VNetSettings').addressPrefixes[0].addressPrefix]"
```

```
        ]
      },
      "subnets":[
        {
          "name":"[parameters('VNetSettings').subnets[0].name]",
          "properties": {
            "addressPrefix": "[parameters('VNetSettings').subnets[0].addressPrefix]"
          }
        },
        {
          "name":"[parameters('VNetSettings').subnets[1].name]",
          "properties": {
            "addressPrefix": "[parameters('VNetSettings').subnets[1].addressPrefix]"
          }
        }
      ]
    }
  }
]
```

Funciones en parámetros

- Cuando se especifica el valor predeterminado de un parámetro, puede usar la mayoría de las funciones de plantilla. No puede usar la función *reference* ni ninguna de las funciones *list* de la sección de parámetros. Estas funciones obtienen el estado de tiempo de ejecución de un recurso y no se pueden ejecutar antes de la implementación cuando se resuelven parámetros.

Beneficios de parametrizar:

- Reusable
- No errores humanos
- Modularidad
- Facilidad de mantenimiento
- Código limpio

Best practices – Parámetros

- Minimice el uso de los parámetros. En su lugar, use las variables o valores literales de propiedades que no deben especificarse durante la implementación.
- Use una mezcla de mayúsculas y minúsculas para los nombres de parámetro (Camel Case)
- Use parámetros para configuraciones que varían según el entorno, como la SKU, el tamaño o la capacidad.
- Use parámetros para nombres de recurso que quiera especificar para facilitar la identificación.
- Proporcione una descripción de cada parámetro en los metadatos
- Defina valores predeterminados para los parámetros que no son confidenciales. Al hacerlo, resulta más fácil implementar la plantilla y los usuarios de esta ven un ejemplo de un valor adecuado.
- Para especificar un parámetro opcional, no use una cadena vacía como valor predeterminado. En su lugar, use un valor literal o una expresión de lenguaje para construir un valor.
- Evite usar un parámetro en la versión de API de un tipo de recurso. Las propiedades y los valores de los recursos pueden variar en función del número de la versión. La función IntelliSense en un editor de código no puede determinar el esquema correcto si la versión de API se establece como un parámetro. En lugar de ello, debe codificar de forma rígida la versión de la API en la plantilla.
- Use *allowedValues* con moderación. Úselo solo cuando deba asegurarse de que algunos valores no están incluidos en las opciones permitidas. Si usa *allowedValues* de forma demasiado amplia, podría bloquear implementaciones válidas al no mantener actualizada la lista.
- Cuando un nombre de parámetro en la plantilla coincide con un parámetro en el comando de PowerShell de implementación, Resource Manager resuelve este conflicto de nomenclatura agregando el postfijo FromTemplate al parámetro de plantilla. Por ejemplo, si incluye un parámetro llamado ResourceGroupName en la plantilla, entra en conflicto con el

parámetro ResourceGroupName del cmdlet *New-AzResourceGroupDeployment*. Durante la implementación, se le pide que proporcione un valor para *ResourceGroupNameFromTemplate*.

- Use siempre los parámetros para los nombres de usuario y contraseñas (o secretos).
- Use *securestring* para todas las contraseñas y secretos. Si pasa datos confidenciales en un objeto JSON, use el tipo *secureObject*. No se pueden leer los parámetros con los tipos *secureString* o *secureObject* después de la implementación de recursos.
- No proporcione valores predeterminados para los nombres de usuario, contraseñas o cualquier valor que requiera un tipo *secureString*.
- No proporcione valores predeterminados para las propiedades que aumenten la superficie de ataque de la aplicación.
- Use un parámetro para especificar la ubicación de recursos y establecer el valor predeterminado en *resourceGroup().location*. Proporcionar un parámetro de ubicación permite a los usuarios de la plantilla especificar una ubicación en la que tienen permiso para implementar.
- No especifique *allowedValues* para el parámetro de ubicación. Las ubicaciones que especifique pueden no estar disponibles en todas las nubes.
- Utilice el valor del parámetro de ubicación para los recursos que pueden estar en la misma ubicación. Con este enfoque se minimiza la cantidad de veces que se les pide a los usuarios que proporcionen información de ubicación.
- Para los recursos que no están disponibles en todas las ubicaciones, utilice un parámetro independiente o especifique un valor de ubicación literal.

Variables

Las variables se utilizan para simplificar la plantilla. En lugar de repetir expresiones complicadas en toda la plantilla, defina una variable que contenga la expresión complicada.

Resource Manager resuelve las variables antes de iniciar las operaciones de implementación. Siempre que la variable se use en la plantilla, Resource Manager la reemplaza por el valor resuelto.

El ejemplo siguiente muestra una variable de definición. Crea un valor de cadena para un nombre de cuenta de almacenamiento. Usa varias funciones de plantilla para obtener un valor de parámetro y lo concatena a una cadena única.

```
"variables": {
  "storageName": "[concat(toLower(parameters('storageNamePrefix')),
uniqueString(resourceGroup().id))]"
},
```

La función no puede usar la función *reference* ni ninguna de las funciones *list* de la sección de variables. Estas funciones obtienen el estado de tiempo de ejecución de un recurso y no se pueden ejecutar antes de la implementación cuando se resuelven variables.

Variables de configuración

Puede definir variables que contengan valores relacionados para configurar un entorno. La variable se define como un objeto con los valores. En el ejemplo siguiente se muestra un objeto que contiene valores para dos entornos: *test* y *prod*.

```
"variables": {
  "environmentSettings": {
    "test": {
      "instanceSize": "Small",
      "instanceCount": 1
    },
    "prod": {
      "instanceSize": "Large",
      "instanceCount": 4
    }
  }
},
```

En parámetros, se crea un valor que indica que valores de configuración usar.

```
"parameters": {
  "environmentName": {
    "type": "string",
    "allowedValues": [
      "test",
      "prod"
    ]
  }
```

```
},
```

Para recuperar la configuración del entorno especificado, use la variable y el parámetro conjuntamente.

```
"[variables('environmentSettings')[parameters('environmentName')].instanceSize]"
```

Best Practices – Variables

- Use mayúsculas y minúsculas combinadas para los nombres de variables.
- Use las variables para los valores que deba utilizar más de una vez en una plantilla. Si un valor se usa solo una vez, codificarlo de forma rígida hace que la plantilla resulte más fácil de leer.
- Use variables para los valores que construya a partir de una organización compleja de funciones de plantilla. La plantilla es más fácil de leer cuando la expresión compleja aparece solo en las variables.
- No use variables para apiVersion en un recurso. La versión de API determina el esquema del recurso. A menudo, no se puede cambiar la versión sin cambiar las propiedades del recurso.
- No se puede usar la función *reference* en la sección variables de la plantilla. La función *reference* deriva su valor desde el estado de tiempo de ejecución del recurso. Sin embargo, las variables se resuelven durante el análisis inicial de la plantilla. Construya valores que requieran la función *reference* directamente en las secciones resources u outputs de la plantilla.
- Incluya variables para los nombres de recursos que deben ser únicos.
- Use un bucle de copia en variables para crear un patrón repetido de objetos JSON.
- Quite las variables no utilizadas.

Funciones definidas por el usuario

Las funciones definidas por el usuario son independientes de las funciones de plantilla estándares que están disponibles automáticamente dentro de la plantilla. Cree sus propias funciones cuando tenga expresiones complicadas que se usen repetidamente en la plantilla.

Las funciones requieren un valor de espacio de nombres para evitar conflictos de nomenclatura con las funciones de plantilla.

Uso de la función

En el ejemplo siguiente se muestra una plantilla que incluye una función definida por el usuario. Esa función se utiliza para obtener un nombre único para una cuenta de almacenamiento. La plantilla tiene un parámetro denominado storageNamePrefix que se pasa como un parámetro a la función.

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
  "storageNamePrefix": {
   "type": "string",
   "maxLength": 11
  }
 },
 "functions": [
  {
   "namespace": "contoso",
   "members": {
    "uniqueName": {
     "parameters": [
      {
       "name": "namePrefix",
       "type": "string"
      }
     ],
     "output": {
      "type": "string",
      "value": "[concat(toLower(parameters('namePrefix')), uniqueString(resourceGroup().id))]"
     }
    }
   }
  }
```

```
],
 "resources": [
  {
    "type": "Microsoft.Storage/storageAccounts",
    "apiVersion": "2019-04-01",
    "name": "[contoso.uniqueName(parameters('storageNamePrefix'))]",
    "location": "South Central US",
    "sku": {
      "name": "Standard_LRS"
    },
    "kind": "StorageV2",
    "properties": {
      "supportsHttpsTrafficOnly": true
    }
  }
 ]
}
```

Limitaciones

Al definir una función de usuario, hay algunas restricciones:

- La función no puede acceder a las variables.
- La función solo puede usar los parámetros que se definen en la función. Cuando usa la función *parameters* dentro de una función definida por el usuario, los parámetros de esa función serán los que limiten sus acciones.
- La función no puede llamar a otras funciones definidas por el usuario.
- La función no puede usar la función *reference* ni ninguna de las funciones *list*.
- Los parámetros de la función no pueden tener valores predeterminados.

Recursos

En la sección de recursos, se define que los recursos se implementan o se actualizan.

Best Practices - Recursos

- Para ayudar a otros colaboradores a comprender el propósito del recurso, especifique comments para cada recurso de la plantilla
- Si usa un punto de conexión público en la plantilla (como un punto de conexión público de Azure Blob Storage), no codifique de forma rígida el espacio de nombres. Use la función *reference* para recuperar dinámicamente el espacio de nombres. Puede usar este enfoque para implementar la plantilla en diversos entornos de espacios de nombres públicos sin cambiar manualmente el punto de conexión de la plantilla. Establezca la versión de API en la misma que usa para la cuenta de almacenamiento de la plantilla. Si la cuenta de almacenamiento se implementa en la misma plantilla que está creando y el nombre de la cuenta de almacenamiento no se comparte con otro recurso de la plantilla, no es necesario especificar el espacio de nombres del proveedor ni apiVersion al hacer referencia al recurso. También puede hacer referencia a una cuenta de almacenamiento existente que se encuentra en un grupo de recursos distinto

Salidas

En la sección de salidas, especifique valores que se devuelven de la implementación. Normalmente, devuelve valores de los recursos implementados.

En la sección de salidas, puede devolver un valor condicionalmente. Normalmente, la condición de las salidas se usa cuando se implementa condicionalmente un recurso.

En algunos escenarios, no se conoce el número de instancias de un valor que se debe devolver al crear la plantilla. Puede devolver un número variable de valores mediante el elemento copy.

Para recuperar el valor de salida de una plantilla vinculada, use la función *reference* en la plantilla principal. Al obtener una propiedad de salida a partir de una plantilla vinculada, el nombre de propiedad no puede incluir un guion. No se puede usar la función *reference* en la sección de salidas de una plantilla anidada. Para devolver los valores de un recurso implementado en una plantilla anidada, convierta la plantilla anidada en una plantilla vinculada.

En el ejemplo siguiente se muestra la estructura de una definición de salida:

```
"outputs": {
```

```
  "<output-name>": {
    "condition": "<boolean-value-whether-to-output-value>",
    "type": "<type-of-output-value>",
    "value": "<output-value-expression>",
    "copy": {
      "count": <number-of-iterations>,
      "input": <values-for-the-variable>
    }
  }
}
```

Obtención de valores de salida

Cuando la implementación se realiza correctamente, los valores de salida se devuelven automáticamente en los resultados de la implementación.

Para obtener valores de salida del historial de implementación, puede usar un script.

PowerShell

```
(Get-AzResourceGroupDeployment `
  -ResourceGroupName <resource-group-name> `
  -Name <deployment-name>).Outputs.resourceID.value
```

Conditional Resource Deployment

El elemento *condition* se utiliza para especificar si se implementará o no un recurso. El valor de este elemento se resuelve como true o false. Cuando el valor es true, el recurso se crea. Cuando el valor es false, el recurso no se crea. El valor solo se puede aplicar a todo el recurso.

La implementación condicional no se aplica en cascada a los recursos secundarios. Si desea implementar condicionalmente un recurso y sus recursos secundarios, debe aplicar la misma condición a cada tipo de recurso.

Recurso nuevo o existente

Puede usar la implementación condicional para crear un recurso nuevo o usar uno existente. En el ejemplo siguiente se muestra cómo usar una condición para implementar una cuenta de almacenamiento nueva o usar una existente.

```
{
  "condition": "[equals(parameters('newOrExisting'),'new')]",
  "type": "Microsoft.Storage/storageAccounts",
  "apiVersion": "2017-06-01",
  "name": "[variables('storageAccountName')]",
  "location": "[parameters('location')]",
  "sku": {
    "name": "[variables('storageAccountType')]"
  },
  "kind": "Storage",
  "properties": {}
}
```

Cuando el parámetro newOrExisting está establecido en new, la condición se evalúa como true. Se implementa la cuenta de almacenamiento. Sin embargo, cuando newOrExisting está establecido existing, la condición se evalúa como false y no se implementa la cuenta de almacenamiento.

Funciones en tiempo de ejecución

Si usa una función *reference* o *list* con un recurso que se implementa de forma condicional, se puede evaluar la función incluso si el recurso no está implementado. Se genera un error si la función hace referencia a un recurso que no existe.

Use la función *if* para asegurarse de que la función se evalúa solo para las condiciones en las que se implementa el recurso.

Puede establecer un recurso como dependiente en un recurso condicional exactamente como lo haría con cualquier otro recurso. Cuando un recurso condicional no está implementado, Azure Resource Manager lo quita automáticamente de las dependencias necesarias.

Iteración de recursos

Al agregar el elemento *copy* a la sección de recursos de la plantilla, puede establecer de forma dinámica el número de elementos de los recursos que va a implementar. Asimismo, evitará tener que repetir la sintaxis de la plantilla.

También puede usar el elemento *copy* con propiedades, variables y salidas.

El elemento copy tiene el siguiente formato general:

```
"copy": {
  "name": "<name-of-loop>",
  "count": <number-of-iterations>,
  "mode": "serial" <or> "parallel",
  "batchSize": <number-to-deploy-serially>
}
```

La propiedad *name* es cualquier valor que identifique el bucle. La propiedad *count* especifica el número de iteraciones que desea realizar en el tipo de recurso.

Utilice las propiedades *mode* y *batchSize* para especificar si los recursos van a implementarse simultáneamente o por orden. Estas propiedades se describen en el artículo En serie o en paralelo.

El valor de count no puede superar 800.

En el ejemplo siguiente, se crea el número de cuentas de almacenamiento especificado en el parámetro storageCount.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storageCount": {
            "type": "int",
            "defaultValue": 2
        }
    },
    "resources": [
        {
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2019-04-01",
            "name": "[concat(copyIndex(),'storage', uniqueString(resourceGroup().id))]",
            "location": "[resourceGroup().location]",
            "sku": {
                "name": "Standard_LRS"
```

```
            },
            "kind": "Storage",
            "properties": {},
            "copy": {
                "name": "storagecopy",
                "count": "[parameters('storageCount')]"
            }
        }
    ],
    "outputs": {}
}
```

Tenga en cuenta que el nombre de cada recurso incluye la función copyIndex(), que devuelve la iteración actual del bucle. copyIndex() es de base cero. Así, en el ejemplo siguiente:

```
"name": "[concat('storage', copyIndex())]",
```

Crea estos nombres:

- storage0
- storage1
- storage2

La operación de copia es útil al trabajar con matrices, ya que puede iterar a través de cada elemento de la matriz. Use la función length en la matriz para especificar el número de iteraciones, y copyIndex para recuperar el índice actual de la matriz.

En el ejemplo siguiente, se crea una cuenta de almacenamiento para cada uno de los nombres proporcionados en el parámetro.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
      "storageNames": {
          "type": "array",
          "defaultValue": [
```

```
            "contoso",
            "fabrikam",
            "coho"
          ]
      }
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2019-04-01",
      "name": "[concat(parameters('storageNames')[copyIndex()],
uniqueString(resourceGroup().id))]",
      "location": "[resourceGroup().location]",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "Storage",
      "properties": {},
      "copy": {
        "name": "storagecopy",
        "count": "[length(parameters('storageNames'))]"
      }
    }
  ],
  "outputs": {}
}
```

Si desea devolver valores de los recursos implementados, puede usar *copy* en la sección de salidas (ouputs).

Dependencia de los recursos de un bucle

Especifique que un recurso se implemente después de otro recurso mediante el elemento *dependsOn*. Para implementar un recurso que dependa de la colección de recursos de un bucle, proporcione el nombre del bucle copy en el elemento *dependsOn*.

Iteración para un recurso secundario

No puede usar un bucle copy en un recurso secundario. Para crear varias instancias de un recurso que se define normalmente como anidado dentro de otro recurso, debe crear dicho recurso como uno de nivel superior. La relación con el recurso principal se define a través de las propiedades *type* y *name*.

Deployment

Se puede hacer el deployment por PowerShell, Bash o mediante el portal de Azure.

Modo de Implementación

Al implementar los recursos, debe especificar si la implementación es una actualización incremental o una actualización completa. La diferencia entre estos dos modos es la forma en que Resource Manager controla los recursos existentes en el grupo de recursos que no están en la plantilla.

En ambos modos, Resource Manager intenta crear todos los recursos especificados en la plantilla. Si el recurso ya existe en el grupo de recursos y su configuración es igual, no se realizará ninguna operación en ese recurso. Si cambia los valores de propiedad de un recurso, el recurso se actualiza con esos nuevos valores. Si intenta actualizar la ubicación o el tipo de un recurso existente, la implementación produce un error. En su lugar, implemente un nuevo recurso con la ubicación o escriba la que necesite.

En el modo completo, Resource Manager **elimina** los recursos que existen en el grupo de recursos pero que no se especifican en la plantilla.

En el modo incremental, Resource Manager deja sin modificar los recursos que existen en el grupo de recursos pero que no se especifican en la plantilla. Los recursos de la plantilla se agregan al grupo de recursos.

El modo predeterminado es el incremental.

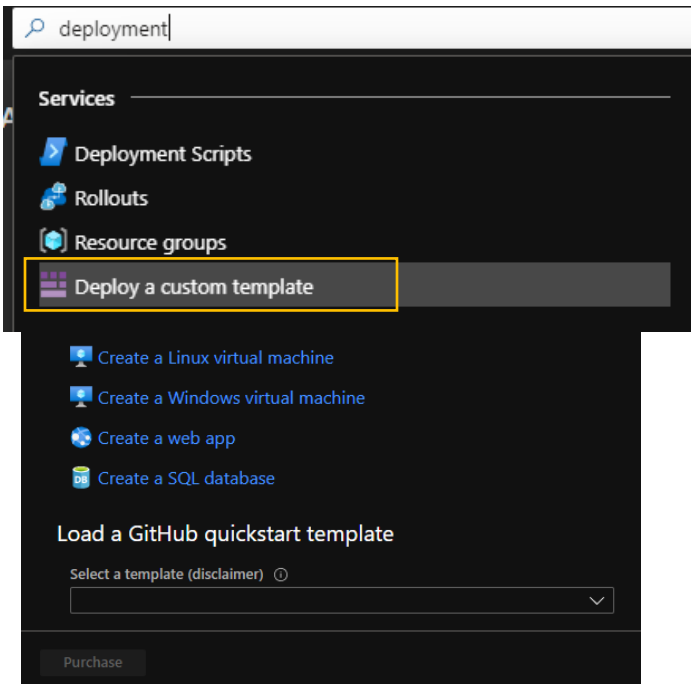
Ejemplo deployment script Powershell

Si la cuenta de Azure tiene más de una suscripción, para establecer la suscripción en la que se implementará la plantilla agregar *-SubscriptionId*. Para obtener el id, ejecutar *Connect-AzAccount* en PowerShell, después *Get-AzSubscription* y copiar el Id de la suscripción que se desea.

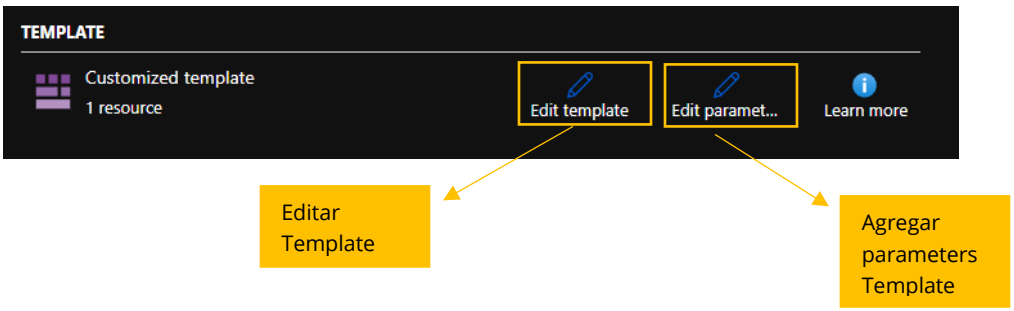
```
Connect-AzAccount  -SubscriptionId 'yyyy-yyyy-yyyy-yyyy'
```

```
$rg = Read-Host -Prompt "Ingresar nombre grupo de recursos"
$location = Read-Host -Prompt "Ingresar ubicacion de los recursos (Ejemplo: eastus o East US)"
New-AzResourceGroup `
    -Name $rg `
    -Location $location `
    -Force

$deploymentFile = 'Template.json'
$deploymentName = Read-Host -Prompt "Ingresar nombre del deployment"
$username = Read-Host -Prompt "Ingresar nombre de usuario en Azure"

New-AzResourceGroupDeployment `
    -Name $deploymentName `
    -Mode Complete `
    -TemplateFile $deploymentFile `
    -ResourceGroupName $rg `
    -objectId $(get-AzADUser -UserPrincipalName $username).Id
```

Deployment desde el portal



Importar el template o copiar el código



Presionar “purchase”