EBook Gratis

APRENDIZAJE azure

Free unaffiliated eBook created from
Stack Overflow contributors.

#azure

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: azure

It is an unofficial and free azure ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official azure.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con el azul

Observaciones

Azure es el nombre comercial con el que Microsoft ofrece sus servicios de computación en la nube. Algunos de los principales servicios ofrecidos dentro de la plataforma Microsoft Azure son:

- Infraestructura como servicio (IaaS): máquinas virtuales de Linux y Windows Azure
- Platform as a Service (PaaS): App Service proporciona una plataforma completa para el desarrollo de aplicaciones (web y móvil),
- Almacenamiento en la nube: servicios de almacenamiento SQL y noSQL
- Software como servicio (SaaS): planificador, copia de seguridad, análisis, aprendizaje automático, seguridad y autenticación

Aquí hay una infografía para ver las principales ofertas de Azure de un vistazo: https://azure.microsoft.com/en-us/resources/infographics/azure/ . Y aquí puede examinar y filtrar todos los productos de Azure por categoría.

Examples

Serie N de Azure (GPU): instale CUDA, cudnn, Tensorflow en UBUNTU 16.04 LTS

Después de pasar más de 5 horas, encontré esta solución fácil:

-Para verificar que el sistema tenga una GPU compatible con CUDA, ejecute el siguiente comando:

```
lspci | grep -i NVIDIA
```

Verá una salida similar a la del siguiente ejemplo (que muestra una tarjeta NVIDIA Tesla K80 / M60):

```
af8a:00:00.0 3D controller: NVIDIA Corporation GK210GL [Tesla K80] (rev a1)
```

-Desactivar el controlador nouveau:

```
sudo -i
rmmod nouveau
```

-Después de *reiniciar* : *reiniciar* `sudo reboot` , verifique que el controlador esté instalado correctamente emitiendo:

```
lsmod | grep -i nvidia
```

-A continuación, descargue el paquete **CUDA** de Nvidia, ...

```
wget https://developer.nvidia.com/compute/cuda/8.0/prod/local_installers/cuda-repo-ubuntu1604-8-0-local_8.0.44-1_amd64-deb
```

-... dar a conocer apt-get e instalar el kit de herramientas CUDA:

```
sudo dpkg -i cuda-repo-ubuntu1604-8-0-local_8.0.44-1_amd64-deb
sudo apt-get update
sudo apt-get install -y cuda
```

-Ahora podemos verificar el estado de la GPU (s) ejecutando:

```
nvidia-smi
```

A continuación, descargamos **cuDNN** ...

```
wget http://developer.download.nvidia.com/compute/redist/cudnn/v5.1/cudnn-8.0-linux-x64-v5.1.tgz
```

-... descomprime, copia el lib64 e incluye las carpetas:

```
tar -zxf cudnn-8.0-linux-x64-v5.1.tgz
sudo cp cuda/lib64/* /usr/local/cuda/lib64/
sudo cp cuda/include/* /usr/local/cuda/include/
sudo rm -R cuda
```

-Tiempo para hacer un poco de limpieza y eliminar los archivos descargados:

```
rm cuda-repo-ubuntu1604-8-0-local_8.0.44-1_amd64-deb
rm cudnn-8.0-linux-x64-v5.1.tgz
```

Para instalar **Tensorflow** con **CPU / GPU** , vaya aquí:

https://www.tensorflow.org/install/install_linux#installing_with_anaconda

Referencia:

1. https://www.lutzroeder.com/blog/2016-12-27-tensorflow-azure 2. https://www.tensorflow.org/install/install_linux#installing_with_anaconda

Lea Empezando con el azul en línea: https://riptutorial.com/es/azure/topic/1060/empezando-con-el-azul

Capítulo 2: Azulejo de poder azul

Examples

Modo clásico vs modo ARM

Cuando trabaje con Azure con PowerShell, hay 2 formas diferentes de las que debe estar consciente y verá una gran cantidad de documentación de Microsoft que se refiere a ambas:

"Modo clásico (Gestión de servicios)"

Esta es la antigua forma de operar Azure y administrar Azure. Todavía hay algunos servicios en Azure que solo pueden administrarse con el modo clásico, aunque cada vez más servicios se están moviendo hacia el nuevo modo ARM.

Para listar los módulos instalados en sus máquinas que funcionan en modo clásico, puede hacer lo siguiente:

```
Get-Module -ListAvailable Azure.*
```

"Administrador de recursos (ARM)"

Esta es la nueva forma de administrar Azure (basada en las API REST proporcionadas). La mayoría de los servicios que podría administrar en Azure desde el modo clásico de Powershell ahora se pueden administrar usando el nuevo modo con algunas excepciones. Esta debería ser la forma preferida de administrar sus recursos de Azure a menos que tenga ciertos servicios que aún no son compatibles con el modo ARM.

Para enumerar los módulos instalados en su máquina que contienen comandos para operar en modo ARM, puede hacer lo siguiente:

```
Get-Module -ListAvailable AzureRM*
```

Inicia sesión en Azure

Modo clásico (gestión de servicios):

```
Add-AzureAccount
```

Esto lo autenticará mediante el Directorio activo de Azure, y PowerShell obtiene un token de acceso que caduca después de aproximadamente 12 horas. Así que debes repetir la autenticación después de 12 horas.

Una alternativa es ejecutar el siguiente cmdlet:

```
Get-AzurePublishSettingsFile
```

Esto abre una ventana del navegador donde puede descargar un archivo de configuración de publicación. Este archivo contiene un certificado que le permite a PowerShell autenticarse. Luego puede importar su archivo de configuración de publicación utilizando lo siguiente:

```
Import-AzurePublishSettingsFile
```

Recuerde que el archivo de configuración de publicación contiene un certificado con privilegios de administrador efectivos en su suscripción. Manténgalo seguro o elimínelo después de usarlo.

Administrador de recursos

En el lado del Administrador de recursos, solo podemos usar la autenticación de Active Directory de Azure con los tokens de acceso de 12 horas. Hay dos comandos alternativos actualmente que puedes usar:

```
Login-AzureRmAccount
Add-AzureRmAccount
```

Seleccionando suscripcion

Cuando tiene múltiples suscripciones bajo su cuenta de Azure; es importante que esté seleccionando el que desea operar (y use esto como predeterminado); Para evitar accidentes que suceden a los recursos en la suscripción incorrecta.

Modo clásico

```
Set-AzureSubscription
Select-AzureSubscription
```

Administrador de recursos

```
Select-AzureRmSubscription
```

Información de suscripción

Los comandos anteriores le piden que especifique información (por ejemplo, el ID de suscripción) para identificar la suscripción a la que desea cambiar. Para listar esta información para las suscripciones a las que tiene acceso, ejecute este comando:

```
Get-AzureSubscription
```

Obtenga la versión actual de Azure PowerShell

Para determinar la versión de Azure PowerShell que ha instalado, ejecute lo siguiente:

```
Get-Module -ListAvailable -Name Azure -Refresh
```

Este comando devuelve la versión instalada incluso cuando no ha cargado el módulo de Azure PowerShell en su sesión actual de PowerShell.

Manipular los activos de Azure

Los cmdlets de Azure le permiten realizar algunas de las mismas acciones en los activos de Azure a través de PowerShell que usaría con el código C # o el portal de Azure.

Por ejemplo, estos pasos le permiten descargar el contenido de un blob de Azure en un directorio local:

```
New-Item -Path .\myblob -ItemType Directory
$context = New-AzureStorageContext -StorageAccountName MyAccountName -StorageAccountKey {key
from the Azure portal}
$blob = Get-AzureStorageBlob -Container MyContainerName -Context $context
$blob | Get-AzureStorageBlobContent -Destination .\myblob\
```

Gestión de gestores de tráfico

Con Azure PowerShell puede obtener cierta funcionalidad que actualmente no está disponible en el Portal de Azure , como:

- Reconfigure todos los puntos finales de Traffic Manager a la vez
- Diríjase a otros servicios a través de Azure `ResourceId` lugar del nombre de dominio, por lo que no necesita configurar la ubicación manualmente para los puntos finales de Azure

Prerrequisitos

Para comenzar necesitas iniciar sesión y seleccionar la suscripción a RM .

Obtener el perfil de TrafficManager

Las operaciones con Traffic Managers a través de PowerShell se realizan en tres pasos:

1. Obtener perfil TM:
   `$profile = Get-AzureRmTrafficManagerProfile -ResourceGroupName my-resource-group -Name my-traffic-manager`
   O crear nuevos como en este artículo .
2. Explora y modifica el perfil de TM
   Verifique `$profile` campos de `$profile` y `$profile` `$profile.Endpoints` para ver la configuración de cada punto final.
3. Guarde los cambios a través de `Set-AzureRmTrafficManagerProfile -TrafficManagerProfile $profile` .

Cambiar puntos finales

Todos los puntos finales actuales se almacenan en `$profile.Endpoints` Lista de puntos finales, por

lo que puede modificarlos directamente por índice

```
$profile.Endpoints[0].Weight = 100
```

o por nombre

```
$profile.Endpoints | ?{ $_.Name -eq 'my-endpoint' } | %{ $_.Weight = 100 }
```

Para borrar todos los puntos finales utiliza

```
$profile.Endpoints.Clear()
```

Para eliminar el uso de un punto final en particular

```
Remove-AzureRmTrafficManagerEndpointConfig -TrafficManagerProfile $profile -EndpointName 'my-endpoint'
```

Para agregar nuevo punto final de uso

```
Add-AzureRmTrafficManagerEndpointConfig -TrafficManagerProfile $profile -EndpointName "my-endpoint" -Type AzureEndpoints -TargetResourceId "/subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/my-resource-group/providers/Microsoft.ClassicCompute/domainNames/my-azure-service" -EndpointStatus Enabled -Weight 100
```

Como puede ver, en el último caso nos hemos dirigido a nuestro servicio de azure a través de ResourceId en lugar de nombre de dominio.

Tenga en cuenta

Sus cambios en TM y sus puntos finales no se aplican hasta que invoque `Set-AzureRmTrafficManagerProfile -TrafficManagerProfile $profile` . Eso le permite reconfigurar completamente TM en una sola operación.

Traffic Manager es una implementación de DNS y la dirección IP dada a los clientes tiene algo de tiempo de vida (también conocido como TTL, puede ver su duración en segundos en el campo `$profile.Ttl`). Entonces, después de haber reconfigurado TM, algunos clientes continuarán usando los puntos finales que almacenaron en caché hasta que caduque el TTL.

Lea Azulejo de poder azul en línea: https://riptutorial.com/es/azure/topic/3961/azulejo-de-poder-azul

Capítulo 3: Azure DocumentDB

Examples

Conectarse a una cuenta (.NET)

Para conectarse a su base de datos DocumentDB, deberá crear un `DocumentClient` con su **URI de punto final** y la **clave de servicio** (puede obtener ambos del portal).

En primer lugar, necesitará las siguientes cláusulas de uso:

```
using System;
using Microsoft.Azure.Documents.Client;
```

Luego puedes crear el cliente con:

```
var endpointUri = "<your endpoint URI>";
var primaryKey = "<your key>";
var client = new DocumentClient(new Uri(endpointUri), primaryKey);
```

Crear una base de datos (.NET)

Su **base de datos** DocumentDB se puede crear utilizando el método `CreateDatabaseAsync` de la clase `DocumentClient` . Una base de datos es el contenedor lógico del almacenamiento de documentos JSON particionado en colecciones.

```
using System.Net;
using System.Threading.Tasks;
using Microsoft.Azure.Documents;
using Microsoft.Azure.Documents.Client;
```

Para crear su base de datos:

```
async Task CreateDatabase(DocumentClient client)
{
    var databaseName = "<your database name>";
    await client.CreateDatabaseAsync(new Database { Id = databaseName });
}
```

También puede verificar si la base de datos ya existe y crearla si es necesario:

```
async Task CreateDatabaseIfNotExists(DocumentClient client)
{
    var databaseName = "<your database name>";
    try
    {
        await client.ReadDatabaseAsync(UriFactory.CreateDatabaseUri(databaseName));
    }
    catch (DocumentClientException e)
```

```
    {
        // If the database does not exist, create a new database
        if (e.StatusCode == HttpStatusCode.NotFound)
        {
            await client.CreateDatabaseAsync(new Database { Id = databaseName });

        }
        else
        {
            // Rethrow
            throw;
        }
    }
}
```

Crear una colección (.NET)

Se puede crear una **colección** utilizando el método `CreateDocumentCollectionAsync` de la clase `DocumentClient` . Una colección es un contenedor de documentos JSON y la lógica de la aplicación JavaScript asociada.

```
async Task CreateCollection(DocumentClient client)
{
    var databaseName = "<your database name>";
    var collectionName = "<your collection name>";

    DocumentCollection collectionInfo = new DocumentCollection();
    collectionInfo.Id = collectionName;

    // Configure collections for maximum query flexibility including string range queries.
    collectionInfo.IndexingPolicy = new IndexingPolicy(new RangeIndex(DataType.String) {
Precision = -1 });

    // Here we create a collection with 400 RU/s.
    await client.CreateDocumentCollectionAsync(UriFactory.CreateDatabaseUri(databaseName),
        collectionInfo, new RequestOptions { OfferThroughput = 400 });
}
```

O puede verificar si la colección existe y crearla si es necesario:

```
async Task CreateDocumentCollectionIfNotExists(DocumentClient client)
{
    var databaseName = "<your database name>";
    var collectionName = "<your collection name>";
    try
    {
        await
client.ReadDocumentCollectionAsync(UriFactory.CreateDocumentCollectionUri(databaseName,
collectionName));
    }
    catch (DocumentClientException e)
    {
        // If the document collection does not exist, create a new collection
        if (e.StatusCode == HttpStatusCode.NotFound)
        {
            DocumentCollection collectionInfo = new DocumentCollection();
            collectionInfo.Id = collectionName;
```

```
            // Configure collections for maximum query flexibility including string range
queries.
            collectionInfo.IndexingPolicy = new IndexingPolicy(new RangeIndex(DataType.String)
{ Precision = -1 });

            // Here we create a collection with 400 RU/s.
            await
client.CreateDocumentCollectionAsync(UriFactory.CreateDatabaseUri(databaseName),
                collectionInfo, new RequestOptions { OfferThroughput = 400 });
        }
        else
        {
            // Rethrow
            throw;
        }
    }
}
```

Crear documentos JSON (.NET)

Se puede crear un **documento** utilizando el método `CreateDocumentAsync` de la clase `DocumentClient`. Los documentos son definidos por el usuario (arbitrario) contenido JSON.

```
async Task CreateFamilyDocumentIfNotExists(DocumentClient client, string databaseName, string
collectionName, Family family)
{
    try
    {
        await client.ReadDocumentAsync(UriFactory.CreateDocumentUri(databaseName,
collectionName, family.Id));
    }
    catch (DocumentClientException e)
    {
        if (e.StatusCode == HttpStatusCode.NotFound)
        {
            await
client.CreateDocumentAsync(UriFactory.CreateDocumentCollectionUri(databaseName,
collectionName), family);
        }
        else
        {
            // Rethrow
            throw;
        }
    }
}
```

Teniendo las siguientes clases que representan una familia (simplificada):

```
public class Family
{
    [JsonProperty(PropertyName = "id")]
    public string Id { get; set; }
    public string LastName { get; set; }
    public Parent[] Parents { get; set; }
    public Child[] Children { get; set; }
```

```
    public Address Address { get; set; }
    public bool IsRegistered { get; set; }
    public override string ToString()
    {
        return JsonConvert.SerializeObject(this);
    }
}

public class Parent
{
    public string FamilyName { get; set; }
    public string FirstName { get; set; }
}

public class Child
{
    public string FamilyName { get; set; }
    public string FirstName { get; set; }
    public string Gender { get; set; }
    public int Grade { get; set; }
    public Pet[] Pets { get; set; }
}

public class Pet
{
    public string GivenName { get; set; }
}

public class Address
{
    public string State { get; set; }
    public string County { get; set; }
    public string City { get; set; }
}
```

Consulta de documentos (.NET)

DocumentDB admite **consultas** enriquecidas contra **documentos JSON** almacenados en cada colección.

Con una consulta LINQ

```
IQueryable<Family> familyQuery = this.client.CreateDocumentQuery<Family>(
    UriFactory.CreateDocumentCollectionUri(databaseName, collectionName), queryOptions)
    .Where(f => f.LastName == "Andersen");
```

Con una consulta SQL

```
IQueryable<Family> familyQueryInSql = this.client.CreateDocumentQuery<Family>(
    UriFactory.CreateDocumentCollectionUri(databaseName, collectionName),
    "SELECT * FROM Family WHERE Family.lastName = 'Andersen'",
    queryOptions);
```

Paginación en una consulta LINQ

Las FeedOptions se utilizan para establecer la propiedad RequestContinuation obtenida en la primera consulta:

```
public async Task<IEnumerable<Family>> QueryWithPagination(int Size_of_Page)
{
    var queryOptions = new FeedOptions() { MaxItemCount = Size_of_Page };
    string continuationToken = string.Empty;
    do
    {
        if (!string.IsNullOrEmpty(continuationToken))
        {
            queryOptions.RequestContinuation = continuationToken;
        }

        IDocumentQuery<Family> familyQuery = this.client.CreateDocumentQuery<Family>(
            UriFactory.CreateDocumentCollectionUri(databaseName, collectionName),
queryOptions)
            .Where(f => f.LastName == "Andersen").AsDocumentQuery();

        var queryResult = await familyQuery.ExecuteNextAsync<Family>();
        continuationToken = queryResult.ResponseContinuation;
        yield return queryResult;

    } while (!string.IsNullOrEmpty(continuationToken));
}
```

Siempre puede llamar una vez y devolver el token de continuación al cliente, por lo que la solicitud paginada se envía cuando el cliente desea la página siguiente. Usando una clase de ayuda y una extensión:

```
public class PagedResults<T>
{
    public PagedResults()
    {
        Results = new List<T>();
    }
    public string ContinuationToken { get; set; }
    public List<T> Results { get; set; }
}

public async Task<PagedResults<Family>> QueryWithPagination(int Size_of_Page, string
continuationToken = "")
{
    var queryOptions = new FeedOptions() { MaxItemCount = Size_of_Page };
    if (!string.IsNullOrEmpty(continuationToken))
    {
        queryOptions.RequestContinuation = continuationToken;
    }

    return await familyQuery = this.client.CreateDocumentQuery<Family>(
        UriFactory.CreateDocumentCollectionUri(databaseName, collectionName), queryOptions)
        .Where(f => f.LastName == "Andersen").ToPagedResults();
}
```

```
public static class DocumentDBExtensions
{
    public static async Task<PagedResults<T>> ToPagedResults<T>(this IQueryable<T> source)
        {
            var documentQuery = source.AsDocumentQuery();
            var results = new PagedResults<T>();
            try
            {
                var queryResult = await documentQuery.ExecuteNextAsync<T>();
                if (!queryResult.Any())
                {
                    return results;
                }
                results.ContinuationToken = queryResult.ResponseContinuation;
                results.Results.AddRange(queryResult);
            }
            catch
            {
                //documentQuery.ExecuteNextAsync throws an exception on empty queries
                return results;
            }

            return results;
        }
}
```

Actualizar un documento (.NET)

DocumentDB admite la sustitución de documentos JSON utilizando el método `ReplaceDocumentAsync` de la clase `DocumentClient` .

```
await client.ReplaceDocumentAsync(UriFactory.CreateDocumentUri(databaseName, collectionName,
familyName), updatedFamily);
```

Eliminar un documento (.NET)

DocumentDB admite la eliminación de documentos JSON utilizando el método `DeleteDocumentAsync` de la clase `DocumentClient` .

```
await client.DeleteDocumentAsync(UriFactory.CreateDocumentUri(databaseName, collectionName,
documentName));
```

Eliminar una base de datos (.NET)

Eliminar una base de datos eliminará la base de datos y todos los recursos secundarios (colecciones, documentos, etc.).

```
await this.client.DeleteDatabaseAsync(UriFactory.CreateDatabaseUri(databaseName));
```

Lea Azure DocumentDB en línea: https://riptutorial.com/es/azure/topic/5176/azure-documentdb

Capítulo 4: Azure Service Fabric

Observaciones

Azure Service Fabric es uno de los servicios de PaaS ofrecidos por Azure. Se basa en la noción de contenedores y servicios: a diferencia de los servicios informáticos (roles web y roles de trabajo), su código no se ejecuta dentro de una (o más) máquinas virtuales, sino que se ejecutan dentro de un contenedor y lo comparten con otros servicios. .

Es importante tener en cuenta que aquí, y a lo largo de los artículos y la documentación de Microsoft sobre Service Fabric, el *contenedor* está destinado en su sentido más general, no como un contenedor de estilo "Docker".

Puede tener (y normalmente tendrá) más de un contenedor, que formará un clúster. Los servicios que se ejecutan dentro del contenedor pueden ser, en orden creciente de "conciencia"

- Ejecutables invitados
- Servicios "confiables"
    - Con estado
    - Apátrida
- Actores "confiables"

Examples

Actores confiables

Un actor dentro de Service Fabric se define por una interfaz .NET estándar / par de clase:

```
public interface IMyActor : IActor
{
    Task<string> HelloWorld();
}


internal class MyActor : Actor, IMyActor
{
    public Task<string> HelloWorld()
    {
        return Task.FromResult("Hello world!");
    }
}
```

Todos los métodos en el par de interfaz / clase deben ser asíncronos, y no pueden tener parametros fuera o ref.

Es fácil comprender por qué si piensas en el modelo de actor: objetos que interactúan entre sí a través del intercambio de mensajes. Los mensajes se entregan a una clase de actor a través de los métodos asíncronos; las respuestas son manejadas por el tiempo de ejecución de los actores

(el "contenedor" del actor) y se envían de vuelta a la persona que llama.

El SDK de Service Fabric generará un proxy en tiempo de compilación. El cliente actor utiliza este proxy para llamar a sus métodos (es decir, para entregar un mensaje al actor y `await` una respuesta).

El cliente identifica un actor a través de una identificación. El ID ya puede ser conocido (lo obtuvo de una base de datos, de otro Actor, o tal vez sea el ID de usuario vinculado a ese actor, o nuevamente el número de serie de un objeto real).

Si necesita crear un nuevo actor y solo necesita una ID, la clase ActorId (provista) tiene métodos para crear una ID de actor distribuida al azar

```
ActorId actorId = ActorId.NewId();
```

Luego, puede usar la clase `ActorProxy` para crear un objeto proxy para el actor. Esto no activa a un actor ni invoca ningún método todavía. IMyActor myActor = ActorProxy.Create (actorId, nuevo Uri ("fabric: / MyApp / MyActorService"));

Luego, puede usar el proxy para invocar un método en el actor. Si no existe un actor con el ID dado, se activará (se creará dentro de uno de los contenedores en el grupo), y luego el tiempo de ejecución publicará un mensaje al actor, ejecutando su llamada de método y completando la Tarea cuando el actor respuestas:

```
await myActor.HelloWorld();
```

Lea Azure Service Fabric en línea: https://riptutorial.com/es/azure/topic/3802/azure-service-fabric

Capítulo 5: Azure-Automation

Parámetros

Nombre del parámetro	Descripción
resourceGroupName	El grupo de recursos de Azure donde se encuentra la cuenta de almacenamiento
nombre de la conexión	La conexión de ejecución de Azure (servicio pricipal) que se creó cuando se creó la cuenta de automatización
StorageAccountName	El nombre de la cuenta de Azure Storage
Nombre del contenedor	El nombre del contenedor de blob
Viejos días	El número de días que un blob puede ser antes de que se elimine

Observaciones

Asegúrese de tener acceso a Active Directory de Azure para que, en la creación de la cuenta de automatización, Azure cree una cuenta de RunAs para usted. Esto te ahorrará muchos problemas.

Examples

Eliminar blobs en almacenamiento de blobs más antiguo que varios días

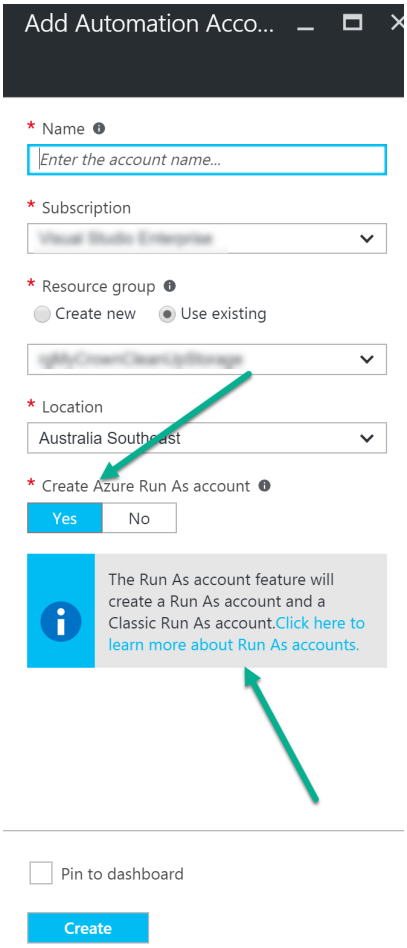
Este es un ejemplo de un runbook de automatización de Powershell de Azure que elimina cualquier blobs en un contenedor de almacenamiento de Azure que sea más antiguo que varios días.

Esto puede ser útil para eliminar las copias de seguridad de SQL antiguas para ahorrar costos y espacio.

Se necesita una serie de parámetros que se explican por sí mismos.

Nota: He dejado algún código comentado para ayudar con la depuración.

Utiliza un principal de servicio que Azure puede configurar automáticamente cuando crea su cuenta de automatización. Necesita tener acceso a Active Directory de Azure. Ver foto:

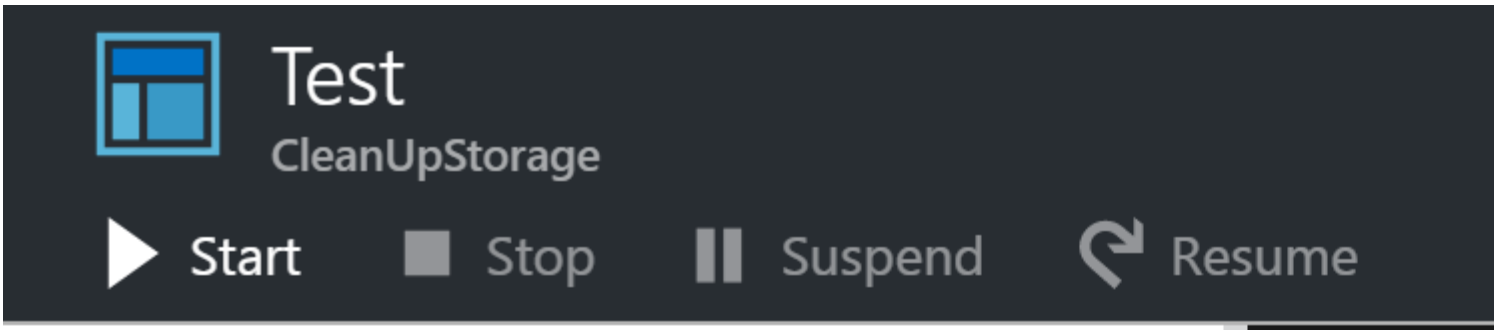



```
<#
.DESCRIPTION
    Removes all blobs older than a number of days back using the Run As Account (Service
Principal)

.NOTES
    AUTHOR: Russ
    LASTEDIT: Oct 03, 2016   #>

param(
    [parameter(Mandatory=$true)]
    [String]$resourceGroupName,

    [parameter(Mandatory=$true)]
    [String]$connectionName,

    # StorageAccount name for content deletion.
    [Parameter(Mandatory = $true)]
    [String]$StorageAccountName,

    # StorageContainer name for content deletion.
    [Parameter(Mandatory = $true)]
    [String]$ContainerName,

    [Parameter(Mandatory = $true)]
    [Int32]$DaysOld

)
$VerbosePreference = "Continue";
try
{
# Get the connection "AzureRunAsConnection "
```

```
$servicePrincipalConnection=Get-AutomationConnection -Name $connectionName

"Logging in to Azure..."
Add-AzureRmAccount `
    -ServicePrincipal `
    -TenantId $servicePrincipalConnection.TenantId `
    -ApplicationId $servicePrincipalConnection.ApplicationId `
    -CertificateThumbprint $servicePrincipalConnection.CertificateThumbprint
catch {
if (!$servicePrincipalConnection)
{
    $ErrorMessage = "Connection $connectionName not found."
    throw $ErrorMessage
} else{
    Write-Error -Message $_.Exception
    throw $_.Exception
}
$keys = Get-AzureRMStorageAccountKey -ResourceGroupName $resourceGroupName -AccountName
$StorageAccountName
# get the storage account key
Write-Host "The storage key is: "$StorageAccountKey;
# get the context
$StorageAccountContext = New-AzureStorageContext -storageAccountName $StorageAccountName -
StorageAccountKey $keys.Key1 #.Value;
$StorageAccountContext;
$existingContainer = Get-AzureStorageContainer -Context $StorageAccountContext -Name
$ContainerName;
#$existingContainer;
if (!$existingContainer)
{
 "Could not find storage container";
}
else
{
$containerName = $existingContainer.Name;
Write-Verbose ("Found {0} storage container" -f $containerName);
$blobs = Get-AzureStorageBlob -Container $containerName -Context $StorageAccountContext;
$blobsremoved = 0;

if ($blobs -ne $null)
{
    foreach ($blob in $blobs)
    {
        $lastModified = $blob.LastModified
        if ($lastModified -ne $null)
        {
            #Write-Verbose ("Now is: {0} and LastModified is:{1}" -f [DateTime]::Now,
[DateTime]$lastModified);
            #Write-Verbose ("lastModified: {0}" -f $lastModified);
            #Write-Verbose ("Now: {0}" -f [DateTime]::Now);
            $blobDays = ([DateTime]::Now - $lastModified.DateTime)  #[DateTime]

            Write-Verbose ("Blob {0} has been in storage for {1} days" -f $blob.Name,
$blobDays);

            Write-Verbose ("blobDays.Days: {0}" -f $blobDays.Hours);
            Write-Verbose ("DaysOld: {0}" -f $DaysOld);

            if ($blobDays.Days -ge $DaysOld)
            {
                Write-Verbose ("Removing Blob: {0}" -f $blob.Name);
```

```
                Remove-AzureStorageBlob -Blob $blob.Name -Container $containerName -Context
$StorageAccountContext;
                $blobsremoved += 1;
            }
            else {
                Write-Verbose ("Not removing blob as it is not old enough.");
            }
        }
    }
}

Write-Verbose ("{0} blobs removed from container {1}." -f $blobsremoved, $containerName);
}
```

Si usa el panel de prueba, puede ingresar los parámetros requeridos y ejecutarlo.



Parameters

* RESOURCEGROUPNAME

Enter a value

Mandatory, String

* CONNECTIONNAME

Enter a value

Mandatory, String

* STORAGEACCOUNTNAME

Enter a value

Mandatory, String

* CONTAINERNAME

Enter a value

Mandatory, String

Capítulo 6: Cuenta de servicio de Azure Media

Observaciones

Una cuenta de servicio de medios es una cuenta basada en Azure que le brinda acceso a servicios de medios basados en la nube en Azure. Almacena metadatos de los archivos de medios que crea, en lugar de guardar el contenido de medios real. Para trabajar con una cuenta de servicio de medios, debe tener una cuenta de almacenamiento asociada. Al crear una cuenta de servicio de medios, puede seleccionar la cuenta de almacenamiento que ya tiene o puede crear una nueva. Dado que la cuenta del servicio de medios y la cuenta de almacenamiento se tratan por separado, el contenido estará disponible en su cuenta de almacenamiento incluso si elimina su cuenta del servicio de medios Tenga en cuenta que la región de su cuenta de almacenamiento debe ser la misma que la región de su cuenta de servicios de medios.

Examples

Creación de un activo en cuenta de servicio de medios

```
public static string CreateBLOBContainer(string containerName)
        {
            try
            {
                string result = string.Empty;
                CloudMediaContext mediaContext;
                mediaContext = new CloudMediaContext(mediaServicesAccountName,
mediaServicesAccountKey);
                IAsset asset = mediaContext.Assets.Create(containerName,
AssetCreationOptions.None);
                return asset.Uri.ToString();
            }
            catch (Exception ex)
            {
                return ex.Message;
            }
        }
```

Recuperando los elementos del activo

```
private static void GetAllTheAssetsAndFiles(MediaServicesCredentials _medServCredentials)
       {
           try
           {
               string result = string.Empty;
               CloudMediaContext mediaContext;
               mediaContext = new CloudMediaContext(_medServCredentials);
               StringBuilder myBuilder = new StringBuilder();
               foreach (var item in mediaContext.Assets)
               {
```

```
                myBuilder.AppendLine(Environment.NewLine);
                myBuilder.AppendLine("--My Assets--");
                myBuilder.AppendLine("Name: " + item.Name);
                myBuilder.AppendLine("++++++++++++++++++++");

                foreach (var subItem in item.AssetFiles)
                {
                    myBuilder.AppendLine("File Name: "+subItem.Name);
                    myBuilder.AppendLine("Size: " + subItem.ContentFileSize);
                    myBuilder.AppendLine("++++++++++++++++++++++");
                }
            }
            Console.WriteLine(myBuilder);
        }
        catch (Exception)
        {
            throw;
        }
    }
```

Lea Cuenta de servicio de Azure Media en línea:
https://riptutorial.com/es/azure/topic/4997/cuenta-de-servicio-de-azure-media

Capítulo 7: Máquinas Virtuales Azure

Examples

Crear Azure VM por la API clásica de ASM

```
# 1. Login Azure by admin account
Add-AzureAccount
#
# 2. Select subscription name
$subscriptionName = Get-AzureSubscription | Select -ExpandProperty SubscriptionName
#
# 3. Create storage account
$storageAccountName = $VMName
# here we use VMName to play the storage account name and create it, you can choose your name
or use existed one to replace the storage account creation operation
New-AzureStorageAccount -StorageAccountName $storageAccountName -Location $Location | Out-Null
#
# 4. Select subscription name and storage account name for current context
Select-AzureSubscription -SubscriptionName $subscriptionName -Current | Out-Null
Set-AzureSubscription -SubscriptionName $subscriptionName -CurrentStorageAccountName
$storageAccountName | Out-Null
#
# 5. Select a VM image name
$label = $VMLabelPattern
# take care, please ensure the VM image location resides to the same location of your storage
account and service below
$imageName = Get-AzureVMImage | where { $_.Label -like $label } | sort PublishedDate -
Descending | select -ExpandProperty ImageName -First 1
#
# 6. Create cloud service
$svcName = $VMName
# here we use VMName to play the service name and create it, you can choose your name or use
existed one to replace the service creation operation
New-AzureService -ServiceName $svcName -Location $Location | Out-Null
#
# 7. Build command set
$vmConfig = New-AzureVMConfig -Name $VMName -InstanceSize $VMSize -ImageName $imageName
#
# 8. Set local admin of this vm
$cred=Get-Credential -Message "Type the name and password of the local administrator account."
$vmConfig | Add-AzureProvisioningConfig -Windows -AdminUsername $cred.Username -Password
$cred.GetNetworkCredential().Password
#
# 9. Execute the final cmdlet to create the VM
New-AzureVM -ServiceName $svcName -VMs $vmConfig | Out-Null
```

Para obtener más información, consulte Cómo crear la máquina virtual de Azure (VM) mediante Powershell mediante la API clásica de ASM

Lea Máquinas Virtuales Azure en línea: https://riptutorial.com/es/azure/topic/6350/maquinas-virtuales-azure

Capítulo 8: Opciones de almacenamiento de Azure

Examples

Cambiar el nombre de un archivo blob en Azure Blob Storage

No hay API que pueda cambiar el nombre del archivo blob en Azure. Este fragmento de código muestra cómo cambiar el nombre de un archivo blob en Microsoft Azure Blob Storage.

```
StorageCredentials cred = new StorageCredentials("[Your storage account name]", "[Your storage
account key]");

CloudBlobContainer container = new CloudBlobContainer(new Uri("http://[Your storage account
name].blob.core.windows.net/[Your container name] /"), cred);

string fileName = "OldFileName";
string newFileName = "NewFileName";

CloudBlockBlob blobCopy = container.GetBlockBlobReference(newFileName);

if (!await blobCopy.ExistsAsync())
{
    CloudBlockBlob blob = container.GetBlockBlobReference(fileName);

    if (await blob.ExistsAsync())
    {
        await blobCopy.StartCopyAsync(blob);
        await blob.DeleteIfExistsAsync();
    }
}
```

Para obtener más información, consulte Cómo cambiar el nombre de un archivo blob en Azure Blob Storage

Importe / exporte archivos de Azure Excel a / desde Azure SQL Server en ASP.NET

Este ejemplo muestra cómo importar el blob de archivo de Azure Excel de la hoja de trabajo a la base de datos en el servidor SQL de Azure y cómo exportarlo de la base de datos al blob de Excel de Azure.

Requisitos previos:

- Versión Microsoft Visual Studio 2015
- Abrir XML SDK 2.5 para Microsoft Office
- Una cuenta de almacenamiento de Azure
- Servidor SQL de Azure

Agregue DocumentFormat.OpenXml de referencia a su proyecto.

1. Exportar datos desde DB a Azure Excel blob
   Guarde excel en el almacenamiento del servidor y luego cárguelo en Azure.

```
public static string DBExportToExcel()
{
    string result = string.Empty;
    try
    {
        //Get datatable from db
        DataSet ds = new DataSet();
        SqlConnection connection = new SqlConnection(connectionStr);
        SqlCommand cmd = new SqlCommand($"SELECT {string.Join(",", columns)} FROM
{tableName}", connection);
        using (SqlDataAdapter adapter = new SqlDataAdapter(cmd))
        {
            adapter.Fill(ds);
        }
        //Check directory
        if (!Directory.Exists(directoryPath))
        {
            Directory.CreateDirectory(directoryPath);
        }
        // Delete the file if it exists
        string filePath = $"{directoryPath}//{excelName}";
        if (File.Exists(filePath))
        {
            File.Delete(filePath);
        }

        if (ds.Tables.Count > 0 && ds.Tables[0] != null || ds.Tables[0].Columns.Count > 0)
        {
            DataTable table = ds.Tables[0];

            using (var spreadsheetDocument = SpreadsheetDocument.Create(filePath,
SpreadsheetDocumentType.Workbook))
            {
                // Create SpreadsheetDocument
                WorkbookPart workbookPart = spreadsheetDocument.AddWorkbookPart();
                workbookPart.Workbook = new Workbook();
                var sheetPart = spreadsheetDocument.WorkbookPart.AddNewPart<WorksheetPart>();
                var sheetData = new SheetData();
                sheetPart.Worksheet = new Worksheet(sheetData);
                Sheets sheets =
spreadsheetDocument.WorkbookPart.Workbook.AppendChild<Sheets>(new Sheets());
                string relationshipId =
spreadsheetDocument.WorkbookPart.GetIdOfPart(sheetPart);
                Sheet sheet = new Sheet() { Id = relationshipId, SheetId = 1, Name =
table.TableName };
                sheets.Append(sheet);

                //Add header to sheetData
                Row headerRow = new Row();
                List<String> columns = new List<string>();
                foreach (DataColumn column in table.Columns)
                {
                    columns.Add(column.ColumnName);

                    Cell cell = new Cell();
```

```
                    cell.DataType = CellValues.String;
                    cell.CellValue = new CellValue(column.ColumnName);
                    headerRow.AppendChild(cell);
                }
                sheetData.AppendChild(headerRow);

                //Add cells to sheetData
                foreach (DataRow row in table.Rows)
                {
                    Row newRow = new Row();
                    columns.ForEach(col =>
                    {
                        Cell cell = new Cell();
                        //If value is DBNull, do not set value to cell
                        if (row[col] != System.DBNull.Value)
                        {
                            cell.DataType = CellValues.String;
                            cell.CellValue = new CellValue(row[col].ToString());
                        }
                        newRow.AppendChild(cell);
                    });
                    sheetData.AppendChild(newRow);
                }
                result = $"Export {table.Rows.Count} rows of data to excel successfully.";
            }
        }

        // Write the excel to Azure storage container
        using (FileStream fileStream = File.Open(filePath, FileMode.Open))
        {
            bool exists = container.CreateIfNotExists();
            var blob = container.GetBlockBlobReference(excelName);
            blob.DeleteIfExists();
            blob.UploadFromStream(fileStream);
        }
    }
    catch (Exception ex)
    {
        result =$"Export action failed. Error Message: {ex.Message}";
    }
    return result;
}
```

2. Importar archivo de Azure Excel a DB
   No podemos leer directamente los datos de Excel Blob, así que tenemos que guardarlos en el almacenamiento del servidor y luego manejarlos.
   Usamos SqlBulkCopy para insertar datos de forma masiva en db.

```
public static string ExcelImportToDB()
{
    string result = string.Empty;
    try
    {
        //Check directory
        if (!Directory.Exists(directoryPath))
        {
            Directory.CreateDirectory(directoryPath);
        }
        // Delete the file if it exists
```

```
        string filePath = $"{directoryPath}//{excelName}";
        if (File.Exists(filePath))
        {
            File.Delete(filePath);
        }
        // Download blob to server disk.
        container.CreateIfNotExists();
        CloudBlockBlob blob = container.GetBlockBlobReference(excelName);
        blob.DownloadToFile(filePath, FileMode.Create);

        DataTable dt = new DataTable();
        using (SpreadsheetDocument spreadSheetDocument = SpreadsheetDocument.Open(filePath,
false))
        {
            //Get sheet data
            WorkbookPart workbookPart = spreadSheetDocument.WorkbookPart;
            IEnumerable<Sheet> sheets =
spreadSheetDocument.WorkbookPart.Workbook.GetFirstChild<Sheets>().Elements<Sheet>();
            string relationshipId = sheets.First().Id.Value;
            WorksheetPart worksheetPart =
(WorksheetPart)spreadSheetDocument.WorkbookPart.GetPartById(relationshipId);
            Worksheet workSheet = worksheetPart.Worksheet;
            SheetData sheetData = workSheet.GetFirstChild<SheetData>();
            IEnumerable<Row> rows = sheetData.Descendants<Row>();

            // Set columns
            foreach (Cell cell in rows.ElementAt(0))
            {
                dt.Columns.Add(cell.CellValue.InnerXml);
            }

            //Write data to datatable
            foreach (Row row in rows.Skip(1))
            {
                DataRow newRow = dt.NewRow();
                for (int i = 0; i < row.Descendants<Cell>().Count(); i++)
                {
                    if (row.Descendants<Cell>().ElementAt(i).CellValue != null)
                    {
                        newRow[i] = row.Descendants<Cell>().ElementAt(i).CellValue.InnerXml;
                    }
                    else
                    {
                        newRow[i] = DBNull.Value;
                    }
                }
                dt.Rows.Add(newRow);
            }
        }

        //Bulk copy datatable to DB
        SqlBulkCopy bulkCopy = new SqlBulkCopy(connectionStr);
        try
        {
            columns.ForEach(col => { bulkCopy.ColumnMappings.Add(col, col); });
            bulkCopy.DestinationTableName = tableName;
            bulkCopy.WriteToServer(dt);
        }
        catch (Exception ex)
        {
            throw ex;
```

```
        }
        finally
        {
            bulkCopy.Close();
        }
        result = $"Import {dt.Rows.Count} rows of data to DB successfully.";
    }
    catch (Exception ex)
    {
        result = $"Import action failed. Error Message: {ex.Message}";
    }
    return result;
}
```

Para obtener más información, consulte https://code.msdn.microsoft.com/How-to-ImportExport-Azure-0c858df9 .

Romper la concesión bloqueada de almacenamiento de blob en Microsoft Azure

No hay API que pueda romper la concesión bloqueada del almacenamiento de blobs en Microsoft Azure. Este fragmento de código muestra cómo romper la concesión bloqueada del almacenamiento de blobs en Microsoft Azure (PowerShell).

```
 $key = (Get-AzureRmStorageAccountKey -ResourceGroupName
$selectedStorageAccount.ResourceGroupName -name $selectedStorageAccount.StorageAccountName -
ErrorAction Stop)[0].value
        $storageContext = New-AzureStorageContext -StorageAccountName
$selectedStorageAccount.StorageAccountName -StorageAccountKey $key -ErrorAction Stop
        $storageContainer = Get-AzureStorageContainer -Context $storageContext -Name
$ContainerName -ErrorAction Stop
        $blob = Get-AzureStorageBlob -Context $storageContext -Container  $ContainerName -Blob
$BlobName -ErrorAction Stop
        $leaseStatus = $blob.ICloudBlob.Properties.LeaseStatus;
        If($leaseStatus -eq "Locked")
        {
             $blob.ICloudBlob.BreakLease()
             Write-Host "Successfully broken lease on '$BlobName' blob."
        }
        Else
        {
            #$blob.ICloudBlob.AcquireLease($null, $null, $null, $null, $null)
            Write-Host "The '$BlobName' blob's lease status is unlocked."
        }
```

Para obtener más información, consulte Cómo romper la concesión bloqueada de almacenamiento de blobs por ARM en Microsoft Azure (PowerShell)

Lea Opciones de almacenamiento de Azure en línea: https://riptutorial.com/es/azure/topic/5405/opciones-de-almacenamiento-de-azure

Capítulo 9: Opciones de almacenamiento de Azure

Examples

Conectarse a una cola de almacenamiento de Azure

Las opciones de almacenamiento en Azure proporcionan una API "REST" (o, mejor, una API HTTP)

El SDK de Azure ofrece clientes para varios idiomas. Veamos, por ejemplo, cómo inicializar uno de los objetos de almacenamiento (una cola) utilizando las bibliotecas cliente de C #.

Todo el acceso a Azure Storage se realiza a través de una cuenta de almacenamiento. Puede crear una cuenta de almacenamiento de varias maneras: a través del portal, a través de la CLI de Azure, PowerShell, el Administrador de recursos de Azure (ARM), ...

En este ejemplo, suponemos que ya tiene uno y lo ha almacenado en su archivo `app.config` .

```
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));
```

Se puede acceder a las colas en la siguiente URL: `http://<storage account>.queue.core.windows.net/<queue>`

Las bibliotecas cliente generarán esta URL para usted; solo debe especificar el nombre de la cola (que debe estar en minúsculas). El primer paso es obtener una referencia a un cliente de cola, que se utilizará para administrar sus colas (las colas están contenidas en la cuenta de almacenamiento especificada).

```
CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
```

Utiliza el cliente para obtener una referencia a una cola.

```
CloudQueue queue = queueClient.GetQueueReference("<queue>");
```

Ahora, utilizando este proxy de `queue` , puede dirigir cualquier operación a su cola.

Normalmente, la primera operación es crear la cola si aún no existe

```
queue.CreateIfNotExists();
```

Observe el nombre de la operación. ¿Por qué "si no existe"? Hay varias razones:

- es posible que esté implementando varias instancias de "algo" que ejecutará este código

("algo" suele ser un servicio informático , como un rol web o un rol de trabajador, pero puede ser una aplicación web, un servicio Fabric, algún código personalizado en Av M...)
- Su aplicación puede reiniciarse en cualquier momento. Recuerde, este es un entorno de nube donde, especialmente para los servicios de PaaS, las instancias son efímeras. No tiene el mismo grado de control sobre su aplicación que tendría en su aplicación implementada localmente.

Aún mejor, debería usar la versión asíncrona de la misma llamada a la API:

```
await queue.CreateIfNotExistsAsync();
```

Hemos utilizado una cola en este ejemplo, pero el ejemplo se puede aplicar fácilmente a los otros objetos de almacenamiento (blobs, tablas y archivos).

Una vez que haya creado su objeto de almacenamiento, estará listo para comenzar a usarlo.

Lea Opciones de almacenamiento de Azure en línea:
https://riptutorial.com/es/azure/topic/6008/opciones-de-almacenamiento-de-azure

Capítulo 10: Plantillas de Azure Resource Manager

Sintaxis

- La sintaxis de las plantillas ARM está bien documentada: https://azure.microsoft.com/en-us/documentation/articles/resource-group-authoring-templates/

Examples

Crear recurso de extensión

Los recursos de extensión en Azure son recursos que extienden otros recursos.

Esta plantilla crea una bóveda de claves de Azure, así como una extensión de configuración de diagnóstico.

Cosas a tener en cuenta:

- El recurso de extensión se crea bajo el atributo de `resources` del recurso padre
- Debe tener un atributo `dependsOn` referencia al recurso principal (para evitar que ARM intente crear la extensión en paralelo con el recurso principal)

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-
01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "keyVaultName": {
      "type": "string",
      "metadata": {
        "description": "Name of the Vault"
      }
    },
    "tenantId": {
      "type": "string",
      "metadata": {
        "description": "Tenant ID of the directory associated with this key vault"
      }
    },
    "location": {
      "type": "string",
      "metadata": {
        "description": "Key Vault location"
      }
    },
    "storageAccountResourceGroup": {
      "type": "string",
      "metadata": {
        "description": "Resource Group of the storage account where key vault activities will
```

```
be logged"
      }
    },
    "storageAccountName": {
      "type": "string",
      "metadata": {
        "description": "Name of the storage account where key vault activities will be logged.
Must be in same region as the key vault."
      }
    }
    },
  "resources": [
    {
      "type": "Microsoft.KeyVault/vaults",
      "name": "[parameters('keyVaultName')]",
      "apiVersion": "2015-06-01",
      "location": "[parameters('location')]",
      "properties": {
        "enabledForDeployment": "false",
        "enabledForDiskEncryption": "false",
        "enabledForTemplateDeployment": "false",
        "tenantId": "[variables('tenantId')]",
        "sku": {
          "name": "Standard",
          "family": "A"
        }
      },
      "resources": [
          {
      "type": "Microsoft.KeyVault/vaults/providers/diagnosticSettings",
      "name": "[concat(parameters('keyVaultName'), '/Microsoft.Insights/service')]",
      "apiVersion": "2015-07-01",
      "dependsOn": [
        "[concat('Microsoft.keyvault/vaults/', parameters('keyVaultName'))]"
      ],
      "properties": {
        "storageAccountId": "[resourceId(parameters('storageAccountResourceGroup'),
'Microsoft.Storage/storageAccounts', parameters('storageAccountName'))]",
        "logs": [{
            "category": "AuditEvent",
            "enabled": true,
            "retentionPolicy": {
                "enabled": true,
                "days": 90
            }
        }]
    }
    }]
    }
  ],
  "outputs": {
      "keyVaultUrl": {
          "type": "string",
          "value": "[reference(resourceId('Microsoft.KeyVault/vaults',
parameters('keyVaultName'))).vaultUri]"
      }
  }
}
```

Lea Plantillas de Azure Resource Manager en línea:

https://riptutorial.com/es/azure/topic/3923/plantillas-de-azure-resource-manager

Creditos

S. No	Capítulos	Contributors
1	Empezando con el azul	awh112, Bernard Vander Beken, Community, KARANJ, lorenzo montanari, Sibeesh Venu, user2314737
2	Azulejo de poder azul	Anton Purin, CmdrTchort, frank tan, juunas, RedGreenCode
3	Azure DocumentDB	gbellmann, Matias Quaranta
4	Azure Service Fabric	Lorenzo Dematté, Stephen Leppik
5	Azure-Automation	Akos Nagy, RuSs
6	Cuenta de servicio de Azure Media	Sibeesh Venu
7	Máquinas Virtuales Azure	Dale Chen
8	Opciones de almacenamiento de Azure	Dale Chen, Gaurav Mantri
9	Plantillas de Azure Resource Manager	BenV