EBook Gratuito

APPRENDIMENTO azure

Free unaffiliated eBook created from
Stack Overflow contributors.

#azure

Sommario

Di

Capitolo 1: Iniziare con l'azzurro

Osservazioni

Azure è il marchio con cui Microsoft offre i suoi servizi di cloud computing. Alcuni dei principali servizi offerti all'interno della piattaforma Microsoft Azure sono:

- Infrastruttura come servizio (IaaS): macchine virtuali Linux Azure e Windows
- Piattaforma come servizio (PaaS): il servizio app fornisce una piattaforma completa per lo sviluppo di app (Web e mobile),
- Cloud Storage: servizi di storage SQL e noSQL
- Software as a Service (SaaS): scheduler, backup, analisi, Machine Learning, sicurezza e autenticazione

Ecco un'infografica per visualizzare le principali offerte di Azure a colpo d'occhio: https://azure.microsoft.com/en-us/resources/infographics/azure/ . E qui puoi sfogliare e filtrare tutti i prodotti Azure per categoria.

Examples

Azure N-series (GPU): installa CUDA, cudnn, Tensorflow su UBUNTU 16.04 LTS

Dopo aver trascorso più di 5 ore, ho trovato questa soluzione facile:

-Per verificare che il sistema disponga di una GPU compatibile con CUDA, eseguire il seguente comando:

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

Verrà visualizzato un output simile al seguente esempio (che mostra una scheda NVIDIA Tesla K80 / M60):

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

-Disattivazione del driver nouveau:

```
sudo -i
rmmod nouveau
```

-Dopo un *riavvio* : `sudo reboot` , verifica che il driver sia installato correttamente emettendo:

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

-Successivamente, scarica il pacchetto **CUDA** da 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
```

-... rendi noto apt-get e installa il toolkit 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
```

-Ora possiamo controllare lo stato delle GPU eseguendo:

```
nvidia-smi
```

Successivamente, scarichiamo **cuDNN** ...

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

-... decomprimere, copiare lib64 e includere le cartelle:

```
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
```

-Tempo per fare un po 'di pulizia e rimuovere gli archivi scaricati:

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

Per installare **Tensorflow** con **CPU / GPU** , vai qui:

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

Riferimento:

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

Leggi Iniziare con l'azzurro online: https://riptutorial.com/it/azure/topic/1060/iniziare-con-l-azzurro

Capitolo 2: Account del servizio multimediale di Azure

Osservazioni

Un account di servizio multimediale è un account basato su Azure che consente di accedere ai servizi di media basati su cloud in Azure. Memorizza i metadati dei file multimediali creati, salvando invece il contenuto multimediale effettivo. Per lavorare con l'account del servizio multimediale, è necessario disporre di un account di archiviazione associato. Durante la creazione di un account del servizio multimediale, è possibile selezionare l'account di archiviazione già esistente o crearne uno nuovo. Poiché l'account del servizio multimediale e l'account di archiviazione sono trattati separatamente, il contenuto sarà disponibile nell'account di archiviazione anche se si elimina l'account del servizio multimediale. Si noti che l'area dell'account di archiviazione deve essere uguale all'area dell'account del servizio multimediale.

Examples

Creazione di una risorsa nell'account del servizio multimediale

```
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;
            }
        }
```

Recupero degli oggetti dal bene

```
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;
        }
    }
```

Leggi Account del servizio multimediale di Azure online:
https://riptutorial.com/it/azure/topic/4997/account-del-servizio-multimediale-di-azure

Capitolo 3: Azure DocumentDB

Examples

Connetti a un account (.NET)

Per collegarti al tuo database DocumentDB dovrai creare un `DocumentClient` con l' **URI dell'endpoint** e la **chiave di servizio** (puoi ottenerli entrambi dal portale).

Prima di tutto, avrai bisogno delle seguenti clausole:

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

Quindi puoi creare il client con:

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

Creare un database (.NET)

Il **database** DocumentDB può essere creato utilizzando il metodo `CreateDatabaseAsync` della classe `DocumentClient` . Un database è il contenitore logico della memoria di documenti JSON partizionata attraverso le raccolte.

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

Per creare il tuo database:

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

Puoi anche controllare se il database esiste già e crearlo se necessario:

```
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;
        }
    }
}
```

Crea una collezione (.NET)

È possibile creare una **raccolta** utilizzando il metodo `CreateDocumentCollectionAsync` della classe `DocumentClient` . Una raccolta è un contenitore di documenti JSON e la logica dell'applicazione JavaScript associata.

```
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 });
}
```

Oppure puoi verificare se la collezione esiste e crearla se necessario:

```
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;
        }
    }
}
```

Crea documenti JSON (.NET)

Un **documento** può essere creato utilizzando il metodo `CreateDocumentAsync` della classe `DocumentClient` . I documenti sono contenuti JSON (arbitrari) definiti dall'utente.

```
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;
        }
    }
}
```

Avere le seguenti classi che rappresentano una famiglia (semplificata):

```
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; }
}
```

Richiesta di documenti (.NET)

DocumentDB supporta **query avanzate su documenti JSON** archiviati in ogni raccolta.

Con una query LINQ

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

Con una query SQL

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

Impaginazione su una query LINQ

FeedOptions viene utilizzato per impostare la proprietà RequestContinuation ottenuta sulla prima query:

```
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));
}
```

È sempre possibile chiamare una volta e restituire il token di continuazione al client, quindi la richiesta impaginata viene inviata quando il cliente desidera la pagina successiva. Utilizzando una classe helper e un'estensione:

```
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;
        }
}
```

Aggiorna un documento (.NET)

DocumentDB supporta la sostituzione di documenti JSON utilizzando il metodo `ReplaceDocumentAsync` della classe `DocumentClient` .

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

Elimina un documento (.NET)

DocumentDB supporta l'eliminazione dei documenti JSON utilizzando il metodo `DeleteDocumentAsync` della classe `DocumentClient` .

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

Elimina un database (.NET)

L'eliminazione di un database rimuoverà il database e tutte le risorse dei bambini (raccolte, documenti, ecc.).

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

Leggi Azure DocumentDB online: https://riptutorial.com/it/azure/topic/5176/azure-documentdb

Capitolo 4: Azure-Automation

Parametri

Nome del parametro	Descrizione
resourceGroupName	Il gruppo di risorse di Azure in cui si trova l'account di archiviazione
connectionName	La connessione Azure Run As (servizio pricipal) che è stata creata quando è stato creato l'account di automazione
StorageAccountName	Il nome dell'account di archiviazione di Azure
ContainerName	Il nome del contenitore BLOB
DaysOld	Il numero di giorni che può essere un blob prima che venga eliminato

Osservazioni

Assicurati di avere accesso ad Azure Active Directory in modo tale che, durante la creazione dell'account di automazione, Azure crei un account RunAs per te. Questo ti farà risparmiare un sacco di problemi.

Examples

Elimina i BLOB nello storage BLOB più vecchio di un numero di giorni

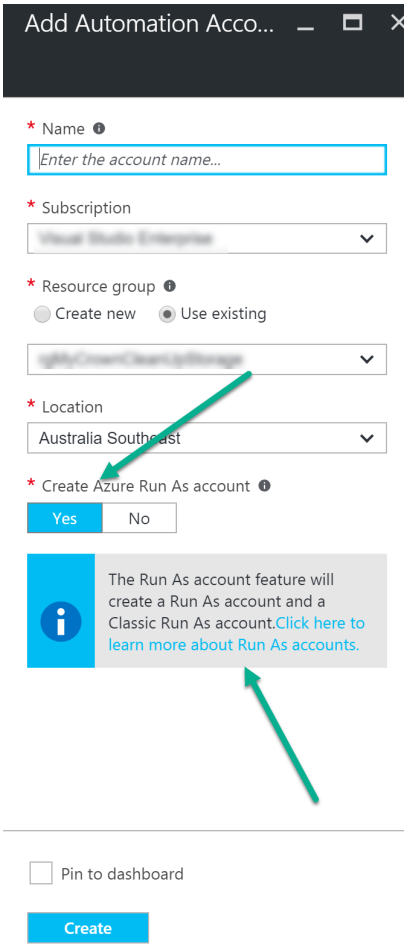
Ecco un esempio di un runbook di automazione di Azure Powershell che elimina eventuali BLOB in un contenitore di archiviazione Azure più vecchi di un numero di giorni.

Questo può essere utile per rimuovere vecchi backup SQL per risparmiare costi e spazio.

Ci vuole un certo numero di parametri che sono auto esplicativi.

Nota: ho lasciato qualche codice commentato per aiutare con il debug.

Utilizza un'entità servizio che Azure può configurare automaticamente quando crei il tuo account di automazione. È necessario disporre dell'accesso ad Azure Active Directory. Vedi foto:

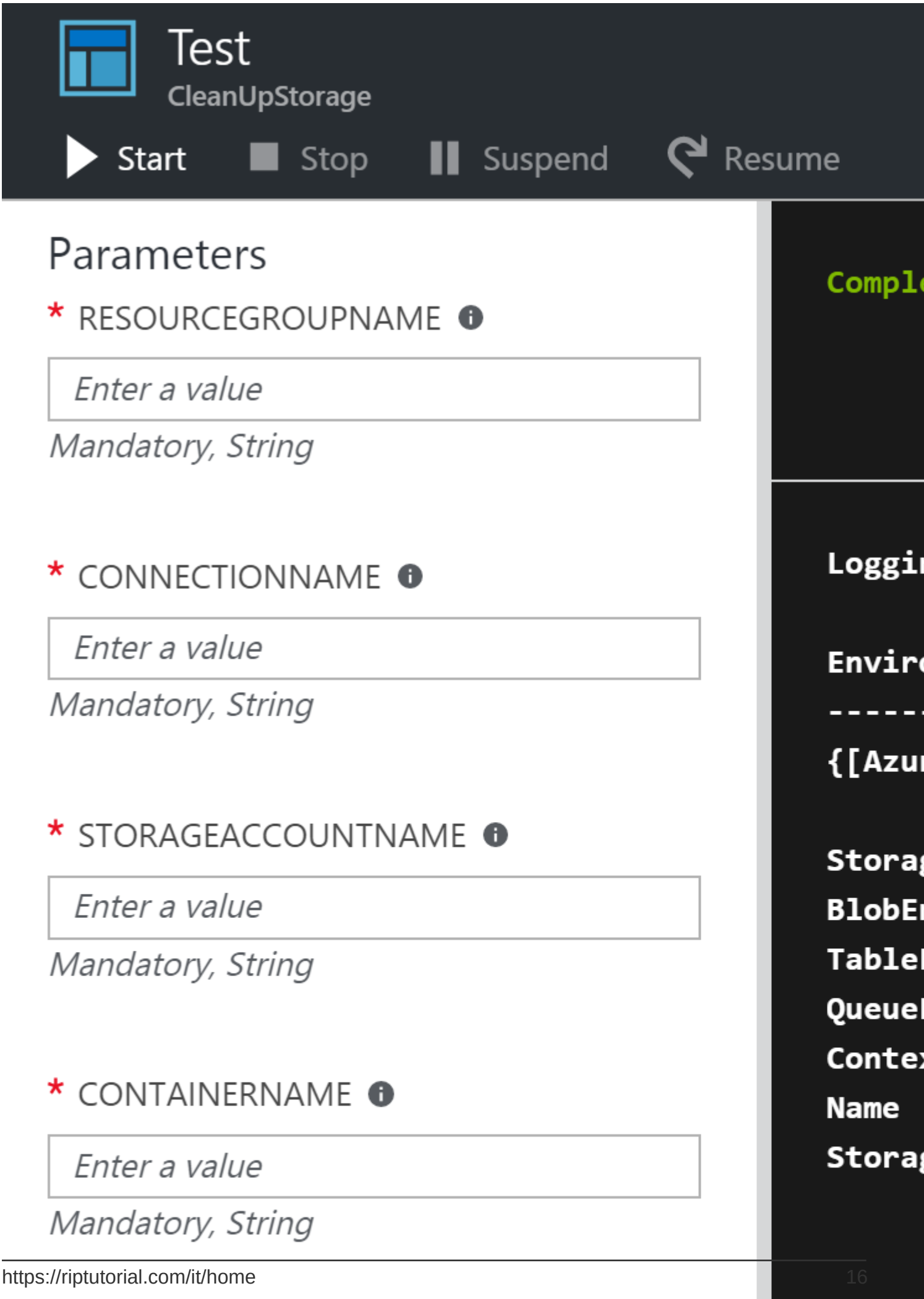



```
<#
.DESCRIPTION
    Removes all blobs older than a number of days back using the Run As Account (Service
Principal)

.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 utilizza il riquadro di test è possibile inserire i parametri richiesti ed eseguirlo.


Test
CleanUpStorage
Start
Stop
Suspend
Resume
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
Compl
Loggi
Envir
{[Azu
Storag
BlobE
Table
Queue
Conte
Name
Storag

Capitolo 5: Macchine virtuali di Azure

Examples

Crea VM di Azure tramite la classica API 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
```

Per ulteriori informazioni, vedere Come creare Azure Virtual Machine (VM) di Powershell utilizzando la classica API ASM

Leggi Macchine virtuali di Azure online: https://riptutorial.com/it/azure/topic/6350/macchine-virtuali-di-azure

Capitolo 6: Modelli di Azure Resource Manager

Sintassi

- La sintassi per i modelli ARM è ben documentata: https://azure.microsoft.com/en-us/documentation/articles/resource-group-authoring-templates/

Examples

Crea una risorsa di estensione

Le risorse di estensione in Azure sono risorse che estendono altre risorse.

Questo modello crea una chiave di Azure Vault e un'estensione DiagnosticSettings.

Cose da notare:

- La risorsa estensione viene creata con l'attributo `resources` della risorsa padre
- È necessario disporre di un attributo `dependsOn` riferimento alla risorsa padre (per impedire a ARM di tentare di creare l'estensione in parallelo con la risorsa padre)

```
{
  "$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]"
      }
  }
}
```

Leggi Modelli di Azure Resource Manager online:
https://riptutorial.com/it/azure/topic/3923/modelli-di-azure-resource-manager

Capitolo 7: Opzioni di archiviazione di Azure

Examples

Rinominare un file blob in Archiviazione BLOB di Azure

Non ci sono API che possono rinominare il file blob su Azure. Questo snippet di codice dimostra come rinominare un file blob in Archiviazione BLOB di Microsoft Azure.

```
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();
    }
}
```

Per ulteriori informazioni, vedere Come rinominare un file blob in Archiviazione BLOB di Azure

Importa / Esporta file Excel di Azure su / da SQL Server di Azure in ASP.NET

Questo esempio mostra come importare il foglio di lavoro BLOB di file di Azure Excel in DB su Azure SQL Server e come esportarlo da DB a BLOB di Azure Excel.

Prerequisiti:

- Microsoft Visual Studio 2015 versione
- Open XML SDK 2.5 per Microsoft Office
- Un account di archiviazione di Azure
- SQL Server di Azure

Aggiungi riferimento DocumentFormat.OpenXml al tuo progetto.

1. Esportare i dati dal blob di DB in Azure Excel
   Salva Excel nell'archivio del server, quindi caricalo in 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. Importa il file di Azure Excel nel DB
   Non possiamo leggere direttamente i dati del blob excel, quindi dobbiamo salvarlo sull'archivio del server e poi gestirlo.
   Usiamo SqlBulkCopy per inserire in blocco i dati in 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;
}
```

Per ulteriori informazioni, consultare https://code.msdn.microsoft.com/How-to-ImportExport-Azure-0c858df9 .

Interrompere il lease bloccato dello storage BLOB in Microsoft Azure

Non ci sono API in grado di interrompere il lease bloccato dello storage BLOB in Microsoft Azure. Questo snippet di codice dimostra la rottura del lease bloccato dello storage BLOB in 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."
        }
```

Per ulteriori informazioni, vedere Come interrompere il lease bloccato dell'archiviazione BLOB da ARM in Microsoft Azure (PowerShell)

Leggi Opzioni di archiviazione di Azure online: https://riptutorial.com/it/azure/topic/5405/opzioni-di-archiviazione-di-azure

Capitolo 8: Opzioni di archiviazione di Azure

Examples

Connessione a una coda di archiviazione di Azure

Le opzioni di archiviazione in Azure forniscono un'API "REST" (o, meglio, un'API HTTP)

L'SDK di Azure offre client per diverse lingue. Vediamo ad esempio come inizializzare uno degli oggetti di memoria (una coda) usando le librerie client C #.

Tutto l'accesso ad Archiviazione di Azure avviene tramite un account di archiviazione. È possibile creare un account di archiviazione in diversi modi: attraverso il portale, tramite la CLI di Azure, PowerShell, Azure Resource Manager (ARM), ...

In questo esempio, supponiamo che ne possiedi già uno e lo hai archiviato nel tuo file `app.config` .

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

Le code sono raggiungibili al seguente URL: `http://<storage account>.queue.core.windows.net/<queue>`

Le librerie client genereranno questo URL per te; hai solo bisogno di specificare il nome della coda (che deve essere in minuscolo). Il primo passo è ottenere un riferimento a un client di coda, che verrà utilizzato per gestire le code (le code sono contenute nell'account di archiviazione specificato).

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

Si utilizza il client per ottenere un riferimento a una coda.

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

Ora, usando questo proxy `queue` , è possibile indirizzare qualsiasi operazione alla propria coda.

In genere, la prima operazione consiste nel creare la coda se non esiste già

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

Si noti il nome dell'operazione. Perché "se non esiste"? Ci sono diversi motivi:

- potresti distribuire più istanze di "qualcosa" che eseguirà questo codice ("qualcosa" è in genere un servizio di calcolo , come un ruolo Web o un ruolo di lavoratore, ma può essere un'app Web, un servizio Fabric, un codice personalizzato in una VM ...)
- la tua app può riavviarsi in qualsiasi momento. Ricorda, questo è un ambiente cloud in cui,

soprattutto per i servizi PaaS, le istanze sono effimere. Non hai lo stesso grado di controllo sulla tua app che avresti sull'app distribuita a livello locale.

Ancora meglio, dovresti usare la versione asincrona della stessa chiamata API:

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

In questo esempio abbiamo utilizzato una coda, ma l'esempio può essere facilmente applicato agli altri oggetti di archiviazione (BLOB, tabelle e file).

Una volta creato l'oggetto di archiviazione, sei pronto per iniziare a usarlo.

Leggi Opzioni di archiviazione di Azure online: https://riptutorial.com/it/azure/topic/6008/opzioni-di-archiviazione-di-azure

Capitolo 9: Powershell azzurro

Examples

Modalità classica vs modalità ARM

Quando si lavora con Azure usando PowerShell ci sono 2 modi diversi di cui si dovrebbe essere a conoscenza e si vedrà molta della documentazione Microsoft che si riferisce ad entrambi:

"Modalità classica (Gestione servizi)"

Questo è il vecchio modo di utilizzare Azure e di gestire Azure. Esistono ancora alcuni servizi in Azure che possono essere gestiti solo utilizzando la modalità classica, anche se sempre più servizi si stanno spostando verso la nuova modalità ARM.

Per elencare i moduli installati sulle tue macchine che funzionano in modalità classica puoi fare quanto segue:

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

"Gestione risorse (ARM)"

Questo è il nuovo modo di gestire Azure (basato sulle API REST fornite). La maggior parte dei servizi che è possibile gestire in Azure dalla modalità classica di PowerShell può ora essere gestita utilizzando la nuova modalità con alcune eccezioni. Questo dovrebbe essere il modo preferito di gestire le tue risorse di Azure a meno che tu non abbia determinati servizi che non sono ancora supportati in modalità ARM.

Per elencare i moduli installati sulla tua macchina contenente i comandi per operare in modalità ARM sono sotto puoi fare quanto segue:

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

Accedi ad Azure

Modalità classica (Gestione servizi):

```
Add-AzureAccount
```

Verrai autenticato utilizzando Azure Active Directory e PowerShell ottiene un token di accesso che scade dopo circa 12 ore. Quindi è necessario ripetere l'autenticazione dopo 12 ore.

Un'alternativa è eseguire il seguente cmdlet:

```
Get-AzurePublishSettingsFile
```

Si apre una finestra del browser in cui è possibile scaricare un file delle impostazioni di pubblicazione. Questo file contiene un certificato che consente a PowerShell di autenticarsi. Quindi è possibile importare il file delle impostazioni di pubblicazione utilizzando quanto segue:

```
Import-AzurePublishSettingsFile
```

Ricorda che il file delle impostazioni di pubblicazione contiene un certificato con i privilegi di amministratore efficaci nell'abbonamento. Mantienilo sicuro o cancellalo dopo averlo usato.

Responsabile delle risorse

In Gestione risorse, possiamo utilizzare solo l'autenticazione di Azure Active Directory con i token di accesso di 12 ore. Al momento sono disponibili due comandi alternativi che è possibile utilizzare:

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

Selezione dell'abbonamento

Quando hai più abbonamenti sotto l'account di Azure; è importante selezionare quello su cui si desidera operare (e usarlo come predefinito); per evitare incidenti che accadono a risorse sulla sottoscrizione sbagliata.

Modalità classica

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

Responsabile delle risorse

```
Select-AzureRmSubscription
```

Informazioni sull'iscrizione

I comandi sopra ti chiedono di specificare le informazioni (ad esempio, l'ID sottoscrizione) per identificare l'abbonamento a cui vuoi passare. Per elencare queste informazioni per gli abbonamenti a cui hai accesso, esegui questo comando:

```
Get-AzureSubscription
```

Ottieni la versione corrente di PowerShell di Azure

Per determinare la versione di Azure PowerShell installata, eseguire quanto segue:

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

Questo comando restituisce la versione installata anche se il modulo Azure PowerShell non è stato caricato nella sessione PowerShell corrente.

Manipolare risorse di Azure

I cmdlet di Azure consentono di eseguire alcune delle stesse azioni sulle risorse di Azure tramite PowerShell che si utilizza il codice C # o il portale di Azure.

Ad esempio, questi passaggi consentono di scaricare il contenuto di un BLOB di Azure in una directory locale:

```
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\
```

Gestione dei gestori del traffico

Con Azure PowerShell è possibile ottenere determinate funzionalità attualmente non disponibili su Azure Portal , ad esempio:

- Riconfigurare tutti gli endpoint di Traffic Manager contemporaneamente
- Indirizza altri servizi tramite Azure `ResourceId` invece del nome di dominio, quindi non è necessario impostare la posizione manualmente per gli endpoint di Azure

Prerequisiti

Per iniziare è necessario effettuare il login e selezionare l'abbonamento RM .

Ottieni il profilo di TrafficManager

Le operazioni con i gestori del traffico tramite PowerShell vengono eseguite in tre passaggi:

1. Ottieni il profilo TM:
   `$profile = Get-AzureRmTrafficManagerProfile -ResourceGroupName my-resource-group -Name my-traffic-manager`
   O creare nuovi come in questo articolo .
2. Esplora e modifica il profilo TM
   Controllare `$profile` campi `$profile` e `$profile.Endpoints` per vedere la configurazione di ciascun endpoint.
3. Salva le modifiche tramite il profilo `Set-AzureRmTrafficManagerProfile -TrafficManagerProfile $profile` .

Cambia endpoint

Tutti gli endpoint correnti sono archiviati in `$profile.Endpoints` list, quindi puoi modificarli

direttamente per indice

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

o per nome

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

Utilizzare per cancellare tutti gli endpoint

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

Per cancellare un uso particolare dell'endpoint

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

Per aggiungere un nuovo uso dell'endpoint

```
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
```

Come puoi vedere, nell'ultimo caso abbiamo indirizzato il nostro servizio azzurro tramite ResourceId piuttosto che il nome di dominio.

Tieni a mente

Le modifiche a TM e ai suoi endpoint non vengono applicate finché non viene richiamato il `Set-AzureRmTrafficManagerProfile -TrafficManagerProfile $profile` . Ciò consente di riconfigurare completamente TM in un'unica operazione.

Traffic Manager è un'implementazione di DNS e l'indirizzo IP dato ai client ha un po 'di tempo per vivere (alias TTL, puoi vedere la sua durata in secondi nel campo `$profile.Ttl`). Quindi, dopo aver riconfigurato TM alcuni client continueranno a utilizzare i vecchi endpoint che hanno memorizzato nella cache fino alla scadenza di tale TTL.

Leggi Powershell azzurro online: https://riptutorial.com/it/azure/topic/3961/powershell-azzurro

Capitolo 10: Tessuto di servizio di Azure

Osservazioni

Azure Service Fabric è uno dei servizi PaaS offerti da Azure. Si basa sulla nozione di contenitori e servizi: a differenza dei servizi Compute (ruoli Web e ruoli dei lavoratori), il codice non viene eseguito all'interno di una (o più) macchine virtuali, ma vengono invece eseguite all'interno di un contenitore, condividendolo con altri servizi .

È importante notare che qui e in tutti gli articoli e la documentazione Microsoft su Service Fabric, il *contenitore* è inteso nel suo significato più generale, non come contenitore di stile "Docker".

Puoi avere (e avrai in genere) più di un contenitore, che formerà un cluster. I servizi in esecuzione all'interno del contenitore possono essere, in ordine crescente di "consapevolezza"

- Esecutori ospiti
- Servizi "affidabili"
    - Stateful
    - apolide
- Attori "affidabili"

Examples

Attori affidabili

Un attore all'interno di Service Fabric è definito da una coppia di interfacce / classi .NET standard:

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


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

Ogni metodo nell'interfaccia / coppia di classi deve essere asincrono e non possono avere parametri di riferimento o di ref.

È facile capire perché se pensi al modello dell'attore: oggetti che interagiscono tra loro attraverso lo scambio di messaggi. I messaggi vengono consegnati a una classe attore tramite i metodi asincroni; le risposte sono gestite dal runtime degli attori (l'attore "contenitore") e reindirizzate al chiamante.

L'SDK di Service Fabric genererà un proxy in fase di compilazione. Questo proxy è usato dal client attore per chiamare i suoi metodi (cioè per consegnare un messaggio all'attore e `await` una risposta).

Il client identifica un attore tramite un ID. L'ID può essere già noto (l'hai ottenuto da un DB, da un altro attore, o forse è l'ID utente collegato a quell'attore, o ancora il numero seriale di un oggetto reale).

Se hai bisogno di creare un nuovo attore e hai solo bisogno di un ID, la classe ActorId (fornita) ha metodi per creare un ID attore distribuito casualmente

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

Quindi, è possibile utilizzare la classe `ActorProxy` per creare un oggetto proxy per l'attore. Questo non attiva un attore o invoca ancora alcun metodo. IMyActor myActor = ActorProxy.Create (actorId, new Uri ("fabric: / MyApp / MyActorService"));

Quindi, è possibile utilizzare il proxy per richiamare un metodo sull'attore. Se un attore con l'ID specificato non esiste, verrà attivato (creato all'interno di uno dei contenitori nel cluster), quindi il runtime invierà un messaggio all'attore, eseguendo la chiamata al metodo e completando l'attività quando l'attore risposte:

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

Leggi Tessuto di servizio di Azure online: https://riptutorial.com/it/azure/topic/3802/tessuto-di-servizio-di-azure

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con l'azzurro	awh112, Bernard Vander Beken, Community, KARANJ, lorenzo montanari, Sibeesh Venu, user2314737
2	Account del servizio multimediale di Azure	Sibeesh Venu
3	Azure DocumentDB	gbellmann, Matias Quaranta
4	Azure-Automation	Akos Nagy, RuSs
5	Macchine virtuali di Azure	Dale Chen
6	Modelli di Azure Resource Manager	BenV
7	Opzioni di archiviazione di Azure	Dale Chen, Gaurav Mantri
8	Powershell azzurro	Anton Purin, CmdrTchort, frank tan, juunas, RedGreenCode
9	Tessuto di servizio di Azure	Lorenzo Dematté, Stephen Leppik