Kostenloses eBook

LERNEN

azure

Free unaffiliated eBook created from
Stack Overflow contributors.

#azure

Inhaltsverzeichnis

Über ... 1

Kapitel 1: Erste Schritte mit Azure ... 2

Bemerkungen ... 2

Examples ... 2

Azure N-Serie (GPU): Installieren Sie CUDA, cudnn und Tensorflow auf UBUNTU 16.04 LTS ... 2

Kapitel 2: Azure DocumentDB ... 4

Examples ... 4

Verbindung zu einem Konto herstellen (.NET) ... 4

Erstellen Sie eine Datenbank (.NET) ... 4

Eine Sammlung erstellen (.NET) ... 5

JSON-Dokumente erstellen (.NET) ... 6

Abfrage nach Dokumenten (.NET) ... 7

Mit einer LINQ-Abfrage ... 7

Mit einer SQL-Abfrage ... 7

Paginierung bei einer LINQ-Abfrage ... 8

Aktualisieren Sie ein Dokument (.NET) ... 9

Dokument löschen (.NET) ... 9

Datenbank löschen (.NET) ... 9

Kapitel 3: Azure Media-Dienstkonto ... 11

Bemerkungen ... 11

Examples ... 11

Erstellen eines Assets in einem Mediendienstkonto ... 11

Elemente aus dem Asset abrufen ... 11

Kapitel 4: Azure Powershell ... 13

Examples ... 13

Klassischer Modus vs. ARM-Modus ... 13

Melden Sie sich bei Azure an ... 13

Abonnement auswählen ... 14

Rufen Sie die aktuelle Azure PowerShell-Version ab ... 14

Azure-Assets bearbeiten ... 15

Verkehrsmanager verwalten ... 15
Voraussetzungen ... 15
TrafficManager-Profil abrufen ... 15
Endpunkte ändern ... 15
Merken Sie sich ... 16
Kapitel 5: Azure Resource Manager-Vorlagen ... 17
Syntax ... 17
Examples ... 17
Erweiterungsressource erstellen ... 17
Kapitel 6: Azure Service Fabric ... 19
Bemerkungen ... 19
Examples ... 19
Zuverlässige Schauspieler ... 19
Kapitel 7: Azure-Automatisierung ... 21
Parameter ... 21
Bemerkungen ... 21
Examples ... 21
Löschen Sie Blobs im Blob-Speicher, die älter als eine Anzahl von Tagen sind ... 21
Indexwartung ... 25
Kapitel 8: Azure-Speicheroptionen ... 27
Examples ... 27
Umbenennen einer Blob-Datei in Azure Blob Storage ... 27
Importieren / Exportieren einer Azure Excel-Datei in / aus Azure SQL Server in ASP.NET ... 27
Unterbrechen Sie die Sperrung des Blob-Speichers in Microsoft Azure ... 31
Kapitel 9: Azure-Speicheroptionen ... 32
Examples ... 32
Herstellen einer Verbindung mit einer Azure-Speicherwarteschlange ... 32
Kapitel 10: Virtuelle Azure-Maschinen ... 34
Examples ... 34
Erstellen Sie eine Azure VM mit der klassischen ASM-API ... 34
Credits ... 35

Über

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

Kapitel 1: Erste Schritte mit Azure

Bemerkungen

Azure ist der Markenname, unter dem Microsoft seine Cloud-Computing-Dienste anbietet. Einige der wichtigsten Dienste, die auf der Microsoft Azure-Plattform angeboten werden, sind:

- Infrastruktur als Dienst (IaaS): Linux und Windows Azure Virtual Machines
- Platform as a Service (PaaS): App Service bietet eine vollständige Plattform für die Entwicklung von Apps (Web und Mobile).
- Cloud Storage: SQL- und noSQL-Speicherdienste
- Software as a Service (SaaS): Scheduler, Backup, Analyse, Maschinelles Lernen, Sicherheit und Authentifizierung

Hier eine Infografik, um die wichtigsten Azure-Angebote auf einen Blick anzuzeigen: https://azure.microsoft.com/en-us/resources/infographics/azure/ Hier können Sie alle Azure-Produkte nach Kategorien durchsuchen und filtern.

Examples

Azure N-Serie (GPU): Installieren Sie CUDA, cudnn und Tensorflow auf UBUNTU 16.04 LTS

Nach mehr als 5 Stunden fand ich diese einfache Lösung:

-Um zu überprüfen, ob das System über eine CUDA-fähige GPU verfügt, führen Sie den folgenden Befehl aus:

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

Sie sehen eine Ausgabe ähnlich dem folgenden Beispiel (zeigt eine NVIDIA Tesla K80 / M60-Karte):

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

-Auswahl des Nouveau-Treibers:

```
sudo -i
rmmod nouveau
```

-Nach einem *Neustart* : `sudo reboot` überprüfen Sie, ob der Treiber ordnungsgemäß installiert ist, indem Sie Folgendes ausgeben:

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

-Nächste, laden Sie das **CUDA-** Paket von Nvidia herunter, ...

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

-... macht es bekannt, dass apt-get das CUDA Toolkit installiert und installiert:

```
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
```

- Jetzt können wir den Status der GPUs überprüfen, indem Sie Folgendes ausführen:

```
nvidia-smi
```

Als nächstes laden wir **cuDNN** ...

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

-... entpacken, lib64 kopieren und Ordner einschließen:

```
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
```

-Zeit bereinigen und die heruntergeladenen Archive entfernen:

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

Um **Tensorflow** mit **CPU / GPU** zu installieren, gehen Sie hier:

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

Referenz:

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

Erste Schritte mit Azure online lesen: https://riptutorial.com/de/azure/topic/1060/erste-schritte-mit-azure

Kapitel 2: Azure DocumentDB

Examples

Verbindung zu einem Konto herstellen (.NET)

Um eine Verbindung zu Ihrer DocumentDB-Datenbank herzustellen, müssen Sie einen `DocumentClient` mit Ihrem **Endpoint-URI** und dem **Service Key** erstellen (Sie können beide vom Portal erhalten).

Zunächst benötigen Sie folgende using-Klauseln:

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

Dann können Sie den Client erstellen mit:

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

Erstellen Sie eine Datenbank (.NET)

Ihre DocumentDB- **Datenbank** kann mithilfe der `CreateDatabaseAsync` Methode der `DocumentClient` Klasse erstellt werden. Eine Datenbank ist der logische Container des JSON-Dokumentenspeichers, der über Sammlungen hinweg partitioniert ist.

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

So erstellen Sie Ihre Datenbank:

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

Sie können auch überprüfen, ob die Datenbank bereits vorhanden ist, und sie bei Bedarf erstellen:

```
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;
        }
    }
}
```

Eine Sammlung erstellen (.NET)

Eine **Auflistung** kann mithilfe der `CreateDocumentCollectionAsync` Methode der `DocumentClient` Klasse erstellt werden. Eine Auflistung ist ein Container mit JSON-Dokumenten und zugehöriger JavaScript-Anwendungslogik.

```
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 });
}
```

Oder Sie können überprüfen, ob die Sammlung vorhanden ist, und sie ggf. erstellen:

```
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;
        }
    }
}
```

JSON-Dokumente erstellen (.NET)

Ein **Dokument** kann mithilfe der `CreateDocumentAsync` Methode der `DocumentClient` Klasse erstellt werden. Dokumente sind benutzerdefinierte (willkürliche) JSON-Inhalte.

```
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;
        }
    }
}
```

Die folgenden Klassen haben eine (vereinfachte) Familie:

```
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; }
}
```

Abfrage nach Dokumenten (.NET)

DocumentDB unterstützt umfangreiche **Abfragen** für **JSON-Dokumente,** die in jeder Sammlung gespeichert sind.

Mit einer LINQ-Abfrage

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

Mit einer SQL-Abfrage

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

```
    queryOptions);
```

Paginierung bei einer LINQ-Abfrage

Die FeedOptions wird verwendet, um die RequestContinuation- Eigenschaft festzulegen, die für die erste Abfrage abgerufen wird:

```
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));
}
```

Sie können immer ein Mal anrufen und das Fortführungs-Token an den Client zurückgeben, sodass die paginierte Anforderung gesendet wird, wenn der Client die nächste Seite wünscht. Verwenden einer Hilfsklasse und einer Erweiterung:

```
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;
        }
}
```

Aktualisieren Sie ein Dokument (.NET)

DocumentDB unterstützt das Ersetzen von JSON-Dokumenten mit der `ReplaceDocumentAsync` Methode der `DocumentClient` Klasse.

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

Dokument löschen (.NET)

DocumentDB unterstützt das Löschen von JSON-Dokumenten mit der `DeleteDocumentAsync` Methode der `DocumentClient` Klasse.

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

Datenbank löschen (.NET)

Durch das Löschen einer Datenbank werden die Datenbank und alle untergeordneten Ressourcen (Sammlungen, Dokumente usw.) entfernt.

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

Azure DocumentDB online lesen: https://riptutorial.com/de/azure/topic/5176/azure-documentdb

Kapitel 3: Azure Media-Dienstkonto

Bemerkungen

Ein Mediendienstkonto ist ein Azure-Konto, mit dem Sie auf Cloud-basierte Mediendienste in Azure zugreifen können. Speichert Metadaten der Mediendateien, die Sie erstellen, anstatt den tatsächlichen Medieninhalt zu speichern. Um mit dem Mediendienstkonto arbeiten zu können, müssen Sie über ein Speicherkonto verfügen. Beim Erstellen eines Mediendienstkontos können Sie entweder das bereits vorhandene Speicherkonto auswählen oder ein neues erstellen. Da das Mediendienstkonto und das Speicherkonto separat behandelt werden, sind die Inhalte auch dann in Ihrem Speicherkonto verfügbar, wenn Sie Ihr Mediendienstkonto löschen.

Examples

Erstellen eines Assets in einem Mediendienstkonto

```
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;
            }
        }
```

Elemente aus dem Asset abrufen

```
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;
        }
    }
```

Azure Media-Dienstkonto online lesen: https://riptutorial.com/de/azure/topic/4997/azure-media-dienstkonto

Kapitel 4: Azure Powershell

Examples

Klassischer Modus vs. ARM-Modus

Wenn Sie mit Azure mit PowerShell arbeiten, gibt es zwei Möglichkeiten, die Sie beachten sollten. In der Microsoft-Dokumentation finden Sie viele Informationen, die sich auf beide beziehen:

"Klassischer Modus (Service Management)"

Dies ist die alte Methode, um Azure zu betreiben und Azure zu verwalten. Es gibt immer noch einige Dienste in Azure, die nur im klassischen Modus verwaltet werden können, auch wenn sich immer mehr Dienste auf den neuen ARM-Modus zubewegen.

Um die auf Ihren Computern installierten Module im klassischen Modus aufzulisten, haben Sie folgende Möglichkeiten:

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

"Ressourcenmanager (ARM)"

Dies ist die neue Art der Verwaltung von Azure (basierend auf den bereitgestellten REST-APIs). Die meisten Dienste, die Sie in Azure im klassischen Powershell-Modus verwalten konnten, können jetzt mit einigen Ausnahmen im neuen Modus verwaltet werden. Dies sollte die bevorzugte Methode zum Verwalten Ihrer Azure-Ressourcen sein, sofern Sie nicht über bestimmte Dienste verfügen, die im ARM-Modus noch nicht unterstützt werden.

Um die auf Ihrem Rechner installierten Module mit Befehlen für den Betrieb im ARM-Modus aufzulisten, können Sie unter den folgenden Möglichkeiten vorgehen:

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

Melden Sie sich bei Azure an

Klassischer Modus (Service Management):

```
Add-AzureAccount
```

Dadurch werden Sie mit Azure Active Directory authentifiziert. PowerShell erhält ein Zugriffstoken, das nach etwa 12 Stunden abläuft. Daher müssen Sie die Authentifizierung nach 12 Stunden wiederholen.

Alternativ können Sie das folgende Cmdlet ausführen:

```
Get-AzurePublishSettingsFile
```

Dadurch wird ein Browserfenster geöffnet, in dem Sie eine Veröffentlichungseinstellungsdatei herunterladen können. Diese Datei enthält ein Zertifikat, mit dem sich PowerShell authentifizieren kann. Dann können Sie Ihre Veröffentlichungseinstellungsdatei folgendermaßen importieren:

```
Import-AzurePublishSettingsFile
```

Denken Sie daran, dass die Veröffentlichungseinstellungsdatei ein Zertifikat mit effektiven Administratorrechten in Ihrem Abonnement enthält. Bewahren Sie es sicher auf oder löschen Sie es nach der Verwendung.

Ressourcenmanager

In Resource Manager können wir die Azure Active Directory-Authentifizierung nur mit den 12-Stunden-Zugriffstoken verwenden. Derzeit können Sie zwei alternative Befehle verwenden:

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

Abonnement auswählen

Wenn Sie unter Ihrem Azure-Konto mehrere Abonnements haben; Es ist wichtig, dass Sie denjenigen auswählen, mit dem Sie arbeiten möchten (und diesen standardmäßig verwenden). um Unfälle mit Ressourcen zu vermeiden, die mit dem falschen Abonnement passieren.

Klassischer Modus

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

Ressourcenmanager

```
Select-AzureRmSubscription
```

Informationen zum Abonnement

Bei den obigen Befehlen werden Sie aufgefordert, Informationen anzugeben (z. B. die Abonnement-ID), um das Abonnement zu identifizieren, zu dem Sie wechseln möchten. Führen Sie den folgenden Befehl aus, um diese Informationen für die Abonnements aufzulisten, auf die Sie Zugriff haben:

```
Get-AzureSubscription
```

Rufen Sie die aktuelle Azure PowerShell-Version ab

Führen Sie die folgenden Schritte aus, um die installierte Azure PowerShell-Version zu ermitteln:

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

Dieser Befehl gibt die installierte Version zurück, auch wenn Sie das Azure PowerShell-Modul in Ihrer aktuellen PowerShell-Sitzung nicht geladen haben.

Azure-Assets bearbeiten

Mit Azure-Cmdlets können Sie über PowerShell einige der gleichen Aktionen für Azure-Assets ausführen wie mit C # -Code oder dem Azure-Portal.

Mit diesen Schritten können Sie beispielsweise den Inhalt eines Azure-Blobs in ein lokales Verzeichnis herunterladen:

```
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\
```

Verkehrsmanager verwalten

Mit Azure PowerShell können Sie bestimmte Funktionen erhalten, die derzeit im Azure Portal nicht verfügbar sind, z.

- Konfigurieren Sie alle Traffic Manager-Endpunkte gleichzeitig neu
- Adressieren Sie andere Dienste über Azure `ResourceId` anstelle des Domänennamens, sodass Sie den Standort für Azure-Endpunkte nicht manuell festlegen müssen

Voraussetzungen

Um zu beginnen, müssen Sie sich anmelden und ein RM-Abonnement auswählen .

TrafficManager-Profil abrufen

Der Betrieb mit Traffic Manager über PowerShell erfolgt in drei Schritten:

1. TM-Profil abrufen:
   `$profile = Get-AzureRmTrafficManagerProfile -ResourceGroupName my-resource-group -Name my-traffic-manager`
   Oder neu erstellen wie in diesem Artikel .
2. TM-Profil erkunden und ändern
   Überprüfen Sie die Felder für `$profile` und `$profile.Endpoints` , um die Konfiguration der einzelnen Endpunkte `$profile.Endpoints` .
3. Speichern Sie die Änderungen über das `Set-AzureRmTrafficManagerProfile -TrafficManagerProfile $profile` .

Endpunkte ändern

Alle aktuellen Endpunkte werden in `$profile.Endpoints` Liste gespeichert, sodass Sie sie direkt

über den Index ändern können

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

oder nach Name

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

Um alle Endpunkte zu löschen, verwenden Sie

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

Um bestimmte Endpunkte zu löschen, verwenden Sie

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

Um einen neuen Endpunkt hinzuzufügen, verwenden Sie

```
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
```

Wie Sie sehen, haben wir unseren Azure-Service im letzten Fall über die ResourceId und nicht über den Domänennamen angesprochen.

Merken Sie sich

Ihre Änderungen an TM und seinen Endpunkten werden erst übernommen, wenn Sie das `Set-AzureRmTrafficManagerProfile -TrafficManagerProfile $profile` . So können Sie TM in einem Arbeitsgang vollständig neu konfigurieren.

Traffic Manager ist eine Implementierung von DNS- und IP-Adressen, die an Clients vergeben werden, und hat einige Zeit zu leben (TTL, die Dauer wird in Sekunden im Feld `$profile.Ttl`). Nach der Neukonfiguration von TM verwenden einige Clients daher alte Endpunkte, die zwischengespeichert wurden, bis die TTL abläuft.

Azure Powershell online lesen: https://riptutorial.com/de/azure/topic/3961/azure-powershell

Kapitel 5: Azure Resource Manager-Vorlagen

Syntax

- Die Syntax für ARM-Vorlagen ist gut dokumentiert: https://azure.microsoft.com/de-de/documentation/articles/resource-group-authoring-templates/

Examples

Erweiterungsressource erstellen

Erweiterungsressourcen in Azure sind Ressourcen, die andere Ressourcen erweitern.

Diese Vorlage erstellt einen Azure Key Vault sowie eine Erweiterung von DiagnosticSettings.

Dinge zu beachten:

- Die Erweiterungsressource wird unter dem `resources` der übergeordneten Ressource erstellt
- Es muss über `dependsOn` Attribut „ `dependsOn` verfügen, `dependsOn` auf die übergeordnete Ressource verweist (damit ARM nicht versucht, die Erweiterung parallel zur übergeordneten Ressource zu erstellen)

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

Azure Resource Manager-Vorlagen online lesen: https://riptutorial.com/de/azure/topic/3923/azure-resource-manager-vorlagen

Kapitel 6: Azure Service Fabric

Bemerkungen

Azure Service Fabric ist einer der von Azure angebotenen PaaS-Services. Es basiert auf der Vorstellung von Containern und Diensten: Im Gegensatz zu Compute-Diensten (Webrollen und Worker-Rollen) wird Ihr Code nicht in einer (oder mehreren) virtuellen Maschinen ausgeführt, sondern in einem Container und wird mit anderen Diensten geteilt .

Es ist wichtig zu beachten, dass *Container* hier und in den gesamten Microsoft-Artikeln und der Dokumentation zu Service Fabric in seiner allgemeineren Bedeutung nicht als "Docker" -Container gedacht ist.

Sie können (und Sie haben normalerweise) mehr als einen Container, der einen Cluster bilden wird. Dienste, die innerhalb des Containers ausgeführt werden, können in aufsteigender Reihenfolge "Bewusstsein" sein.

- Ausführbare Dateien für Gäste
- "Zuverlässige" Dienste
    - Staatsbürgerlich
    - Staatenlos
- "Zuverlässige" Schauspieler

Examples

Zuverlässige Schauspieler

Ein Akteur in Service Fabric wird durch ein standardmäßiges Paar aus .NET-Schnittstelle / Klasse definiert:

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


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

Jede Methode im Interface / Class-Paar muss asynchron sein, und sie können keine Parameter für paramenters verwenden.

Es ist leicht zu verstehen, warum, wenn Sie über das Akteurmodell nachdenken: Objekte, die über

den Nachrichtenaustausch miteinander interagieren. Die Nachrichten werden über die asynchronen Methoden an eine Darstellerklasse übermittelt. Antworten werden von der Darstellerlaufzeit (dem Darsteller "Container") verarbeitet und an den Aufrufer zurückgeleitet.

Das Service Fabric-SDK generiert zur Kompilierungszeit einen Proxy. Dieser Proxy wird vom Actor-Client verwendet, um seine Methoden aufzurufen (dh dem Actor eine Nachricht zuzustellen und `await` eine Antwort zu `await`).

Der Client identifiziert einen Akteur anhand einer ID. Die ID kann bereits bekannt sein (Sie haben sie von einer Datenbank, von einem anderen Actor erhalten, oder es handelt sich möglicherweise um die mit diesem Actor verknüpfte Benutzer-ID oder wiederum die Seriennummer eines realen Objekts).

Wenn Sie einen neuen Akteur erstellen müssen und nur eine ID benötigen, verfügt die (bereitgestellte) ActorId-Klasse über Methoden zum Erstellen einer zufällig verteilten Aktor-ID

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

Anschließend können Sie mit der `ActorProxy` Klasse ein Proxy-Objekt für den Schauspieler `ActorProxy` . Dies aktiviert noch keinen Akteur oder ruft noch keine Methoden auf. IMyActor myActor = ActorProxy.Create (ActorId, neue Uri ("Fabric: / MyApp / MyActorService"));

Anschließend können Sie den Proxy verwenden, um eine Methode für den Akteur aufzurufen. Wenn ein Akteur mit der angegebenen ID nicht vorhanden ist, wird er aktiviert (in einem der Container im Cluster erstellt), und die Laufzeitumgebung sendet eine Nachricht an den Akteur, führt ihren Methodenaufruf aus und beendet die Task des Aktors Antworten:

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

Azure Service Fabric online lesen: https://riptutorial.com/de/azure/topic/3802/azure-service-fabric

Kapitel 7: Azure-Automatisierung

Parameter

Parametername	Beschreibung
resourceGroupName	Die Azure-Ressourcengruppe, in der sich das Speicherkonto befindet
Verbindungsname	Die Azure Run As-Verbindung (Servicepricipal), die beim Erstellen des Automatisierungskontos erstellt wurde
StorageAccountName	Der Name des Azure Storage-Kontos
Containername	Der Blobcontainername
Tage alt	Die Anzahl der Tage, die ein Blob sein darf, bevor er gelöscht wird

Bemerkungen

Stellen Sie sicher, dass Sie Zugriff auf Azure Active Directory haben. Bei der Erstellung eines Automatisierungskontos erstellt Azure ein RunAs-Konto für Sie. Dies erspart Ihnen viel Ärger.

Examples

Löschen Sie Blobs im Blob-Speicher, die älter als eine Anzahl von Tagen sind

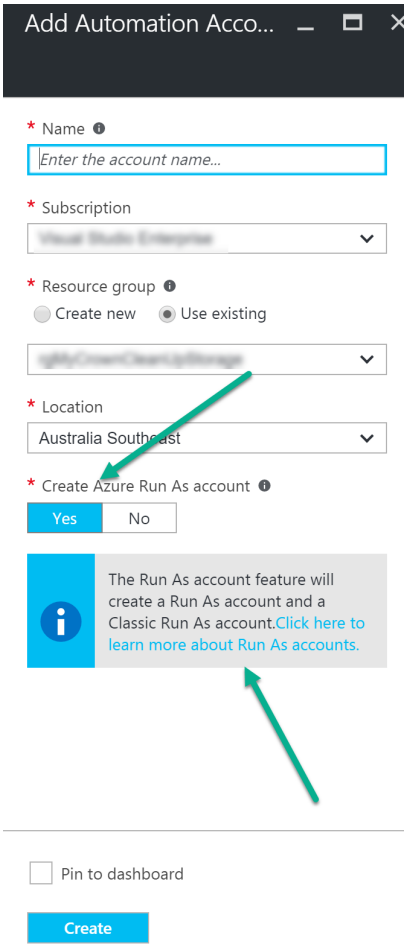
Hier ein Beispiel für ein Azure Powershell-Automatisierungs-Runbook, in dem alle Blobs in einem Azure-Speichercontainer gelöscht werden, die älter als eine Anzahl von Tagen sind.

Dies kann hilfreich sein, um alte SQL-Sicherungen zu entfernen, um Kosten und Speicherplatz zu sparen.

Es bedarf einer Reihe von Parametern, die selbsterklärend sind.

Hinweis: Ich habe etwas auskommentierten Code hinterlassen, um das Debuggen zu erleichtern.

Es verwendet einen Dienstprinzipal, den Azure automatisch einrichten kann, wenn Sie Ihr Automatisierungskonto erstellen. Sie benötigen Azure Active Directory-Zugriff. Siehe Bild:

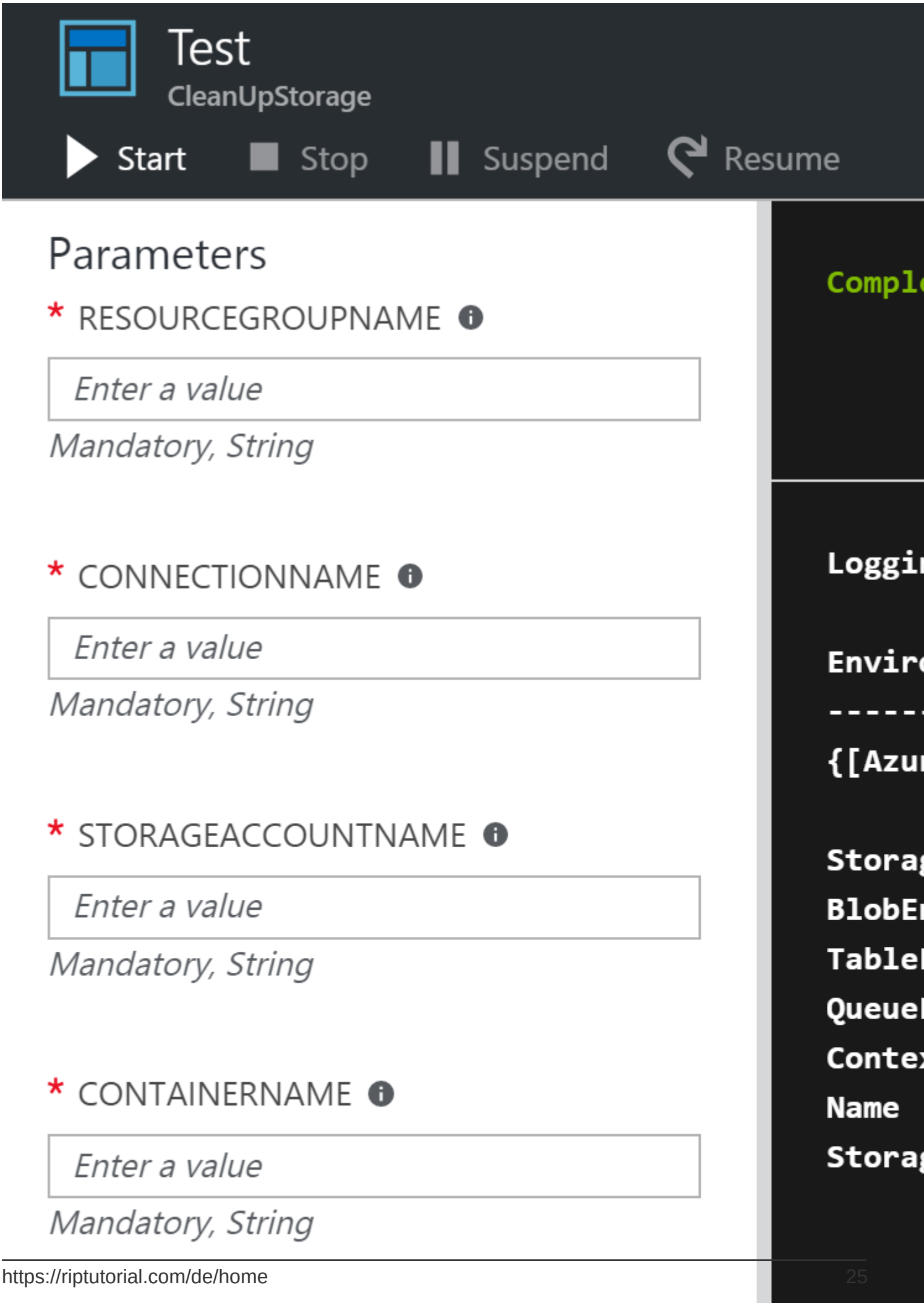



```
<#
.DESCRIPTION
    Removes all blobs older than a number of days back using the Run As Account (Service
Principal)

.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);
}
```

Wenn Sie das Testfenster verwenden, können Sie die erforderlichen Parameter eingeben und ausführen.


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

https://github.com/conwid/IndexRebuildScript

Azure-Automatisierung online lesen: https://riptutorial.com/de/azure/topic/7258/azure-automatisierung

Kapitel 8: Azure-Speicheroptionen

Examples

Umbenennen einer Blob-Datei in Azure Blob Storage

Es gibt keine API, die die Blob-Datei in Azure umbenennen kann. Dieser Code-Ausschnitt veranschaulicht, wie eine Blob-Datei in Microsoft Azure Blob Storage umbenannt wird.

```
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();
    }
}
```

Weitere Informationen finden Sie unter So benennen Sie eine Blob-Datei in Azure Blob Storage um

Importieren / Exportieren einer Azure Excel-Datei in / aus Azure SQL Server in ASP.NET

In diesem Beispiel wird veranschaulicht, wie Sie das Arbeitsblatt Azure Excel-Datei-BLOB in eine Datenbank auf dem Azure SQL Server importieren und wie Sie es aus einer Datenbank in ein Azure Excel-BLOB exportieren.

Voraussetzungen:

- Microsoft Visual Studio 2015-Version
- Open XML SDK 2.5 für Microsoft Office
- Ein Azure-Speicherkonto
- Azure SQL Server

Fügen Sie Ihrem Projekt die Referenz DocumentFormat.OpenXml hinzu.

1. Exportieren Sie Daten aus DB in Azure Excel Blob
   Speichern Sie Excel im Serverspeicher, und laden Sie es in Azure hoch.

```
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. Importieren Sie die Azure Excel-Datei in die Datenbank
   Wir können Excel-Blob-Daten nicht direkt lesen. Daher müssen wir sie im Server-Speicher speichern und dann bearbeiten.
   Wir verwenden SqlBulkCopy , um Daten massenweise in db einzufügen.

```
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;
}
```

Weitere Informationen finden Sie unter https://code.msdn.microsoft.com/How-to-ImportExport-Azure-0c858df9 .

Unterbrechen Sie die Sperrung des Blob-Speichers in Microsoft Azure

Es gibt keine API, die das Sperren von Blob-Speicher in Microsoft Azure verhindern kann. Dieses Code-Snippet demonstriert das Aufheben der Sperrung des Blob-Speichers 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."
        }
```

Weitere Informationen finden Sie unter So brechen Sie die gesperrte Lease von Blob-Speicher durch ARM in Microsoft Azure (PowerShell).

Azure-Speicheroptionen online lesen: https://riptutorial.com/de/azure/topic/5405/azure-speicheroptionen

Kapitel 9: Azure-Speicheroptionen

Examples

Herstellen einer Verbindung mit einer Azure-Speicherwarteschlange

Speicheroptionen in Azure bieten eine REST-API (oder besser eine HTTP-API)

Das Azure SDK bietet Clients für mehrere Sprachen an. Sehen wir uns beispielsweise an, wie Sie eines der Speicherobjekte (eine Warteschlange) mithilfe der C # -Clientbibliotheken initialisieren.

Der gesamte Zugriff auf Azure Storage erfolgt über ein Speicherkonto. Sie können ein Speicherkonto auf verschiedene Arten erstellen: über das Portal, über die Azure-CLI, PowerShell, Azure Resource Manager (ARM), ...

In diesem Beispiel wird davon `app.config` dass Sie bereits über eines verfügen und es in Ihrer `app.config` Datei gespeichert haben.

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

Warteschlangen sind unter der folgenden URL erreichbar: `http://<storage account>.queue.core.windows.net/<queue>`

Die Client-Bibliotheken generieren diese URL für Sie. Sie müssen nur den Namen der Warteschlange angeben (der Kleinbuchstabe muss sein). Der erste Schritt besteht darin, einen Verweis auf einen Warteschlangenclient abzurufen, der zur Verwaltung Ihrer Warteschlangen verwendet wird (Warteschlangen sind im angegebenen Speicherkonto enthalten).

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

Sie verwenden den Client, um eine Referenz auf eine Warteschlange abzurufen.

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

Mit diesem `queue` Proxy können Sie nun beliebige Vorgänge an Ihre Warteschlange weiterleiten.

In der Regel wird die Warteschlange normalerweise erstellt, wenn sie noch nicht vorhanden ist

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

Beachten Sie den Namen der Operation. Warum "wenn nicht existiert"? Es gibt verschiedene Gründe:

- Sie stellen möglicherweise mehrere Instanzen von "Etwas" bereit, die diesen Code ausführen ("Etwas" ist normalerweise ein Berechnungsdienst , z. B. eine Webrolle oder eine

Worker-Rolle). Es kann sich jedoch auch um eine Webanwendung, einen Fabric Service oder einen benutzerdefinierten Code handeln eine VM ...)
- Ihre App kann jederzeit neu gestartet werden. Denken Sie daran, dass dies eine Cloud-Umgebung ist, in der Instanzen vor allem für PaaS-Services kurzlebig sind. Sie haben nicht die gleiche Kontrolle über Ihre App wie Ihre lokal bereitgestellte App.

Außerdem sollten Sie die asynchrone Version desselben API-Aufrufs verwenden:

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

In diesem Beispiel wurde eine Warteschlange verwendet. Das Beispiel kann jedoch problemlos auf die anderen Speicherobjekte (Blobs, Tabellen und Dateien) angewendet werden.

Nachdem Sie Ihr Speicherobjekt erstellt haben, können Sie es verwenden.

Azure-Speicheroptionen online lesen: https://riptutorial.com/de/azure/topic/6008/azure-speicheroptionen

Kapitel 10: Virtuelle Azure-Maschinen

Examples

Erstellen Sie eine Azure VM mit der klassischen ASM-API

```
# 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
```

Weitere Informationen finden Sie unter So erstellen Sie eine Azure Virtual Machine (VM) von Powershell mithilfe der klassischen ASM-API

Virtuelle Azure-Maschinen online lesen: https://riptutorial.com/de/azure/topic/6350/virtuelle-azure-maschinen

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Azure	awh112, Bernard Vander Beken, Community, KARANJ, lorenzo montanari, Sibeesh Venu, user2314737
2	Azure DocumentDB	gbellmann, Matias Quaranta
3	Azure Media-Dienstkonto	Sibeesh Venu
4	Azure Powershell	Anton Purin, CmdrTchort, frank tan, juunas, RedGreenCode
5	Azure Resource Manager-Vorlagen	BenV
6	Azure Service Fabric	Lorenzo Dematté, Stephen Leppik
7	Azure-Automatisierung	Akos Nagy, RuSs
8	Azure-Speicheroptionen	Dale Chen, Gaurav Mantri
9	Virtuelle Azure-Maschinen	Dale Chen