FREE eBook

LEARNING azure

Free unaffiliated eBook created from
Stack Overflow contributors.

#azure

Table of Contents

About

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

Chapter 1: Getting started with azure

Remarks

Azure is the brand name under which Microsoft is offering its cloud computing services. Some of the main services offered within the Microsoft Azure platform are:

- Infrastructure as a Service (IaaS): Linux and Windows Azure Virtual Machines
- Platform as a Service (PaaS): App Service provides a complete platform for app (Web and mobile) development,
- Cloud Storage: SQL and noSQL storage services
- Software as a Service (SaaS): scheduler, backup, analytics, Machine Learning, security and authentication

Here's an infographic to view the main Azure offerings at-a-glance: https://azure.microsoft.com/en-us/resources/infographics/azure/. And here you can browse through and filter all Azure products by category.

Examples

Azure N-series(GPU) : install CUDA, cudnn, Tensorflow on UBUNTU 16.04 LTS

After spending more than 5 hours, i found this easy solution:

-To verify that the system has a CUDA-capable GPU, run the following command:

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

You will see output similar to the following example (showing an NVIDIA Tesla K80/M60 card):

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

-Disabling the nouveau driver:

```
sudo -i
rmmod nouveau
```

-After a *reboot*: `sudo reboot`, verify the driver is installed properly by issuing:

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

-Next, download the **CUDA** package from 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
```

-... make it known to apt-get and install the CUDA Toolkit:

```
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
```

-Now we can check the status of the GPU(s) by running:

```
nvidia-smi
```

Next, we download **cuDNN**...

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

-... unzip, copy the lib64 and include folders:

```
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
```

-Time to do some clean up and remove the downloaded archives:

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

To install **Tensorflow** with **CPU/GPU** , go here :

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

Reference:

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

Read Getting started with azure online: https://riptutorial.com/azure/topic/1060/getting-started-with-azure

Chapter 2: Azure DocumentDB

Examples

Connect to an account (.NET)

To connect to your DocumentDB database you will need to create a `DocumentClient` with your **Endpoint URI** and the **Service Key** (you can get both from the portal).

First of all, you will need the following using clauses:

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

Then you can create the client with:

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

Create a database (.NET)

Your DocumentDB **database** can be created by using the `CreateDatabaseAsync` method of the `DocumentClient` class. A database is the logical container of JSON document storage partitioned across collections.

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

To create your database:

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

You can also check if the database already exists and create it if needed:

```
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;
        }
    }
}
```

Create a collection (.NET)

A **collection** can be created by using the `CreateDocumentCollectionAsync` method of the `DocumentClient` class. A collection is a container of JSON documents and associated JavaScript application logic.

```
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 });
}
```

Or you can check if the collection exists and create it if necessary:

```
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;
        }
    }
}
```

Create JSON documents (.NET)

A **document** can be created by using the `CreateDocumentAsync` method of the `DocumentClient` class. Documents are user defined (arbitrary) JSON content.

```
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;
        }
    }
}
```

Having the following classes that represent a (simplified) family:

```
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; }
}
```

Query for documents (.NET)

DocumentDB supports rich **queries** against **JSON documents** stored in each collection.

With a LINQ query

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

With a SQL query

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

Pagination on a LINQ query

The FeedOptions is used to set the RequestContinuation property obtained on the first 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));
}
```

You can always call once and return the Continuation Token to the client, so the paginated request is sent when the client wants the next page. Using a helper class and an extension:

```
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;
        }
}
```

Update a document (.NET)

DocumentDB supports replacing JSON documents using the `ReplaceDocumentAsync` method of the `DocumentClient` class.

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

Delete a document (.NET)

DocumentDB supports deleting JSON documents using the `DeleteDocumentAsync` method of the `DocumentClient` class.

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

Delete a database (.NET)

Deleting a database will remove the database and all children resources (collections, documents, etc.).

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

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

Chapter 3: Azure Media Service Account

Remarks

A media service account is a Azure based account which gives you access to cloud based media services in Azure. Stores metadata of the media files you create, instead saving the actual media content. To work with media service account, you must have an associated storage account. While creating a media service account, you can either select the storage account you already have or you can create a new one. Since the media service account and storage account is treated separately, the content will be available in your storage account even if you delete your media service account Please be noted that your storage account region must be same as your media service account region.

Examples

Creating an asset in media service account

```
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;
            }
        }
```

Retrieving the items from the Asset

```
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;
        }
    }
```

Read Azure Media Service Account online: https://riptutorial.com/azure/topic/4997/azure-media-service-account

Chapter 4: Azure Powershell

Examples

Classic mode vs ARM mode

When working with Azure using PowerShell there are 2 different ways you should be aware of and you will see a lot of the Microsoft documentation referring to both of them:

"Classic mode (Service Management)"

This is the old way of operating Azure and managing Azure. There is still some services in Azure that can only be managed using the classic mode, even though more and more services are moving towards the new ARM mode.

To list modules installed on your machines operating under classic mode you can do the following:

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

"Resource manager (ARM) "

This is the new way of managing Azure (Based on the REST API's provided). Most of the services you could manage in Azure from Powershell classic mode can now be managed using the new mode with some exceptions. This should be the preferred way of managing your Azure resources unless you have certain services that are not supported under ARM mode yet.

To list the modules installed on your machine containing commands to operate in ARM mode are under you can do the following :

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

Login to Azure

Classic (Service Management) mode:

```
Add-AzureAccount
```

This will authenticate you using Azure Active Directory, and PowerShell gets an access token that expires after about 12 hours. So you must repeat the authentication after 12 hours.

An alternative is to run the following cmdlet:

```
Get-AzurePublishSettingsFile
```

This opens a browser window where you can download a publish settings file. This file contains a certificate that allows PowerShell to authenticate. Then you can import your publish settings file

using the following:

```
Import-AzurePublishSettingsFile
```

Remember that the publish settings file contains a certificate with effectively admin privileges in your subscription. Keep it secure or delete it after using it.

Resource manager

In Resource Manager side, we can only use Azure Active Directory authentication with the 12 hour access tokens. There are two alternative commands currently that you can use:

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

Selecting subscription

When you have multiple subscriptions under your Azure account; it's important that you are selecting the one you wish to operate on (and use this as default); to avoid accidents happening to resources on the wrong subscription.

Classic mode

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

Resource manager

```
Select-AzureRmSubscription
```

Subscription information

The commands above ask you to specify information (e.g., the Subscription ID) to identify the subscription you want to switch to. To list this information for the subscriptions you have access to, run this command:

```
Get-AzureSubscription
```

Get the Current Azure PowerShell Version

To determine the version of Azure PowerShell that you have installed, run the following:

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

This command returns the installed version even when you haven't loaded the Azure PowerShell module in your current PowerShell session.

Manipulate Azure Assets

Azure Cmdlets let you perform some of the same actions on Azure assets through PowerShell that you would using C# code or the Azure portal.

For example, these steps let you download the contents of an Azure blob into a local directory:

```
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\
```

Managing Traffic Managers

With Azure PowerShell you can get certain functionality currently unavailable on Azure Portal, like:

- Reconfigure all Traffic Manager's endpoints at once
- Address other services via Azure `ResourceId` instead of domain name, so you don't need to set Location manually for Azure Endpoints

Prerequisites

To start you need to login and select RM subscription.

Get TrafficManager profile

Operations with Traffic Managers via PowerShell are done in three steps:

1. Get TM profile:
   `$profile = Get-AzureRmTrafficManagerProfile -ResourceGroupName my-resource-group -Name my-traffic-manager`
   Or create new as in this article.
2. Explore and modify TM profile
   Check `$profile` fields and `$profile.Endpoints` to see each endpoint's configuration.
3. Save changes via `Set-AzureRmTrafficManagerProfile -TrafficManagerProfile $profile`.

Change endpoints

All current endpoints are stored in `$profile.Endpoints` list, so you can alter them directly by index

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

or by name

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

To clear all endpoints use

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

To delete particular endpoint use

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

```
endpoint'
```

To add new endpoint use

```
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
```

As you can see, in the last case we've addressed our azure service via ResourceId rather than domain name.

Keep in mind

Your changes to TM and it's endpoints are not applied until you'll invoke `Set-AzureRmTrafficManagerProfile -TrafficManagerProfile $profile`. That allows you to fully reconfigure TM in one operation.

Traffic Manager is an implementation of DNS and IP address given to clients has some time to live (aka TTL, you can see it's duration in seconds in the `$profile.Ttl` field). So, after you've reconfigured TM some clients will continue to use old endpoints they cached until that TTL expire.

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

Chapter 5: Azure Resource Manager Templates

Syntax

- Syntax for ARM templates is well documented: https://azure.microsoft.com/en-us/documentation/articles/resource-group-authoring-templates/

Examples

Create extension resource

Extension Resources in Azure are resources that extend other resources.

This template creates an Azure Key Vault as well as a DiagnosticSettings extension.

Things to note:

- The extension resource is created under the `resources` attribute of the parent resource
- It needs to have a `dependsOn` attribute referencing the parent resource (to prevent ARM from attempting to create the extension in parallel with the parent resource)

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

Read Azure Resource Manager Templates online: https://riptutorial.com/azure/topic/3923/azure-resource-manager-templates

Chapter 6: Azure Service Fabric

Remarks

Azure Service Fabric is one of the PaaS services offered by Azure. It is based on the notion of containers and services: unlike Compute services (Web Roles and Worker roles), your code does not run inside one (or more) Virtual Machines, but instead they are run inside a container, sharing it with other services.

It is important to note that here, and throughout Microsoft articles and documentation on Service Fabric, *container* is intended in its more general meaning, not as a "Docker" style container.

You can have (and you'll typically have) more than one container, which will form a cluster. Services running inside the container can be, in increasing order of "awareness"

- Guest executables
- "Reliable" services
    - Stateful
    - Stateless
- "Reliable" actors

Examples

Reliable actors

An actor inside Service Fabric is defined by a standard .NET interface/class pair:

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


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

Every method in the interface/class pair must be async, and they cannot have out or ref paramenters.

It is easy to understand why if you think about the actor model: objects interacting with each other through exchange of messages. The messages are delivered to an actor class through the async methods; responses are handled by the actors runtime (the actor "container") and routed back to the caller.

The Service Fabric SDK will generate a proxy at compile time. This proxy is used by the actor client to call its methods (i.e. to deliver a message to the actor and `await` a response).

The client identifies an Actor through a ID. The ID can be already known (you got it from a DB, from another Actor, or maybe it is the userID linked to that actor, or again the serial number of a real object).

If you need to create a new actor and you just need an ID, the (provided) ActorId class has methods to create a randomly distributed actor ID

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

Then, you can use the `ActorProxy` class to creates a proxy object for the actor. This does not activate an actor or invoke any methods yet. IMyActor myActor = ActorProxy.Create(actorId, new Uri("fabric:/MyApp/MyActorService"));

Then, you can use the proxy to invoke a method on the actor. If an actor with the given ID does not exist, it will be activated (created inside one of the containers in the cluster), and then the runtime will post a message to the actor, executing its method call and completing the Task when the actor answers:

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

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

Chapter 7: Azure Storage Options

Examples

Renaming a blob file in Azure Blob Storage

There's no API that can rename the blob file on Azure. This code snippet demonstrates how to rename a blob file in Microsoft Azure Blob Storage.

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

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

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

CloudBlockBlob blobCopy = container.GetBlockBlobReference(newFileName);

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

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

For more information, see How to rename a blob file in Azure Blob Storage

Import/Export Azure Excel file to/from Azure SQL Server in ASP.NET

This sample demonstrates how to import the worksheet Azure Excel file blob to DB on the Azure SQL Server and how to export it from DB to Azure Excel blob.

Prerequisites:

- Microsoft Visual Studio 2015 version
- Open XML SDK 2.5 for Microsoft Office
- An Azure storage account
- Azure SQL Server

Add reference DocumentFormat.OpenXml to your project.

1. Export data from DB to Azure Excel blob
   Save excel to server storage then upload it to 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. Import Azure Excel file to DB
   We can't directly read excel blob data, so we have to save it to server storage, and then handle it.
   We use SqlBulkCopy to bulk insert data to 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;
}
```

For more information, see https://code.msdn.microsoft.com/How-to-ImportExport-Azure-0c858df9.

Break the locked lease of blob storage in Microsoft Azure

There's no API that can break the locked lease of blob storage in Microsoft Azure . This code snippet demonstrates break the locked lease of blob storage 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."
        }
```

For more information, see How to break the locked lease of blob storage by ARM in Microsoft Azure (PowerShell)

Read Azure Storage Options online: https://riptutorial.com/azure/topic/5405/azure-storage-options

Chapter 8: Azure Storage Options

Examples

Connecting to an Azure Storage Queue

Storage options in Azure provide a "REST" API (or, better, an HTTP API)

The Azure SDK offers clients for several languages. Let's see for example how to initialize one of the storage objects (a queue) using the C# client libraries.

All access to Azure Storage is done through a storage account. You can create a storage account in several ways: through the portal, through the Azure CLI, PowerShell, Azure Resource Manager (ARM), ...

In this example we suppose you already have one, and you have stored it in your `app.config` file.

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

Queues are reachable at the following URL: `http://<storage account>.queue.core.windows.net/<queue>`

The client libraries will generate this URL for you; you just need to specify the queue name (which must be lowercase). The first step is to get a reference to a queue client, which will be used to manage your queues (queues are contained in the specified storage account).

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

You use the client to get a reference to a queue.

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

Now, using this `queue` proxy, you can direct any operation to your queue.

Typically, the first operation is to create the queue if it doesn't already exist

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

Notice the name of the operation. Why "if not exists"? There are several reasons:

- you may be deploying multiple instances of "something" that will run this code ("something" is typically a Compute Service, like a Web Role or a Worker role, but it can be a Web App, a Fabric Service, some custom code in a VM...)
- your App may reboot at any time. Remember, this is a cloud environment where, especially for PaaS services, instances are ephemeral. You do not have the same degree of control

over your app as you would have on your locally deployed app.

Even better, you should use the async version of the same API call:

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

We have used a queue in this example, but the example can be easily applied to the other storage objects (blobs, tables, and files).

Once you have created your storage object, you are ready to start using it.

Read Azure Storage Options online: https://riptutorial.com/azure/topic/6008/azure-storage-options

Chapter 9: Azure Virtual Machines

Examples

Create Azure VM by classic 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
```

For more information, please see How to create Azure Virtual Machine (VM) by Powershell using classic ASM API

Read Azure Virtual Machines online: https://riptutorial.com/azure/topic/6350/azure-virtual-machines

Chapter 10: Azure-Automation

Parameters

Parameter Name	Description
resourceGroupName	The Azure Resource group where the storage account sits
connectionName	The Azure Run As connection (service pricipal) that was created when the automation account was created
StorageAccountName	The name of the Azure Storage account
ContainerName	The blob container name
DaysOld	The number of days a blob can be before it is deleted

Remarks

Ensure you have access to Azure Active Directory so, on Automation Account creation, Azure creates a RunAs account for you. This will save you a lot of trouble.

Examples

Delete Blobs in Blob storage older than a number of days

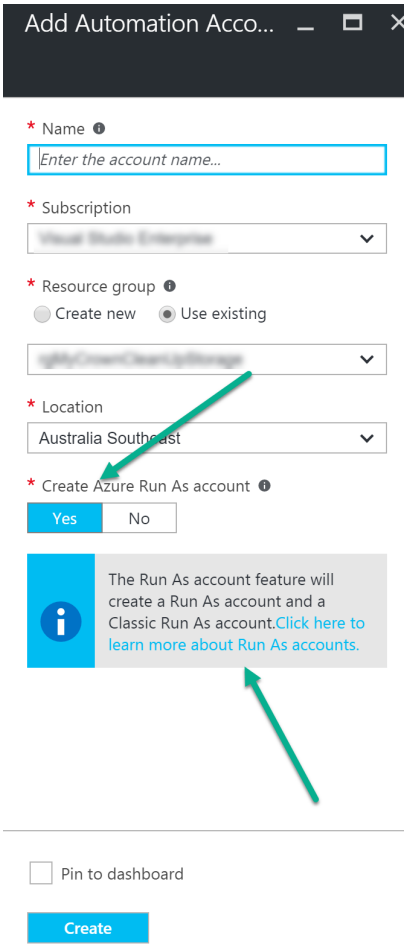
Here is an example of an Azure Powershell automation runbook that deletes any blobs in an Azure storage container that are older than a number of days.

This may be useful for removing old SQL backups to save cost and space.

It takes a number of parameters which are self explanatory.

Note: I have left some commented out code to help with debugging.

It uses a service principal that Azure can set up for you automatically when you create your automation account. You need to have Azure Active Directory access. See pic:

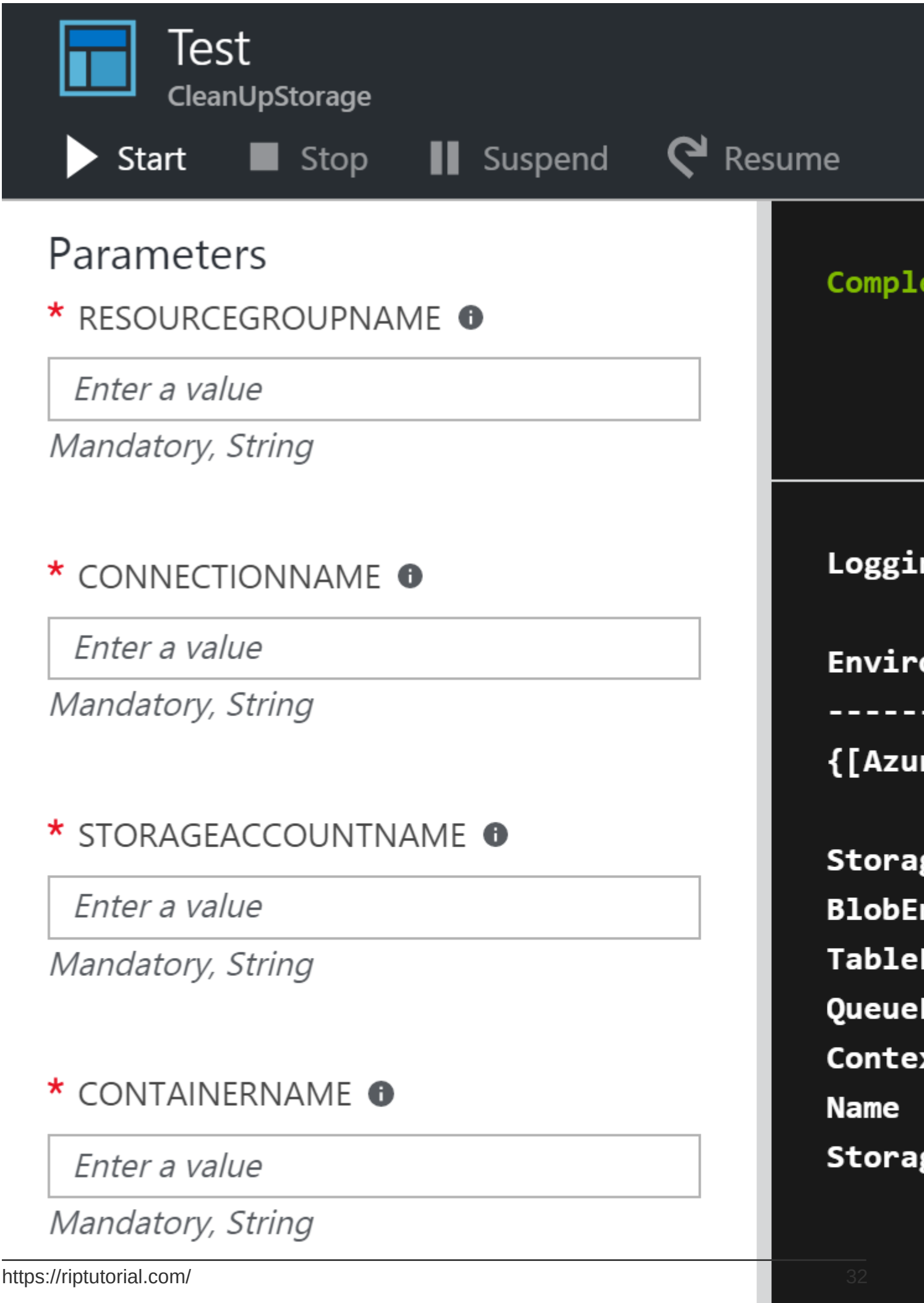



```
<#
.DESCRIPTION
    Removes all blobs older than a number of days back using the Run As Account (Service
Principal)

.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);
}
```

It you use the test pane you can enter the required parameters and run it.


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

Credits

S. No	Chapters	Contributors
1	Getting started with azure	awh112, Bernard Vander Beken, Community, KARANJ, lorenzo montanari, Sibeesh Venu, user2314737
2	Azure DocumentDB	gbellmann, Matias Quaranta
3	Azure Media Service Account	Sibeesh Venu
4	Azure Powershell	Anton Purin, CmdrTchort, frank tan, juunas, RedGreenCode
5	Azure Resource Manager Templates	BenV
6	Azure Service Fabric	Lorenzo Dematté, Stephen Leppik
7	Azure Storage Options	Dale Chen, Gaurav Mantri
8	Azure Virtual Machines	Dale Chen
9	Azure-Automation	Akos Nagy, RuSs