eBook Gratuit

APPRENEZ azure

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#azure

Table des matières

À propos

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

Chapitre 1: Se lancer dans l'azur

Remarques

Azure est le nom de marque sous lequel Microsoft propose ses services de cloud computing. Certains des principaux services proposés dans la plate-forme Microsoft Azure sont les suivants:

- Infrastructure en tant que service (IaaS): machines virtuelles Linux et Windows Azure
- Plate-forme en tant que service (PaaS): App Service fournit une plate-forme complète pour le développement d'applications (Web et mobile),
- Cloud Storage: services de stockage SQL et noSQL
- Logiciel en tant que service (SaaS): planificateur, sauvegarde, analyse, apprentissage automatique, sécurité et authentification

Voici une infographie pour afficher les principales offres Azure en un coup d'œil: https://azure.microsoft.com/en-us/resources/infographics/azure/ . Et ici, vous pouvez parcourir et filtrer tous les produits Azure par catégorie.

Examples

Azure N-series (GPU): installez CUDA, cudnn, Tensorflow sur UBUNTU 16.04 LTS

Après avoir passé plus de 5 heures, j'ai trouvé cette solution simple:

-Pour vérifier que le système dispose d'un GPU compatible CUDA, exécutez la commande suivante:

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

Vous verrez une sortie similaire à l'exemple suivant (montrant une carte NVIDIA Tesla K80 / M60):

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

-Désactiver le pilote nouveau:

```
sudo -i
rmmod nouveau
```

-Après un *redémarrage* : `sudo reboot` , vérifiez que le pilote est installé correctement en émettant:

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

-Suivant, téléchargez le package **CUDA** de Nvidia, ...

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

-... faire savoir à apt-get et installer la boîte à outils CUDA:

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

-Maintenant, nous pouvons vérifier l'état du ou des GPU en exécutant:

```
nvidia-smi
```

Ensuite, nous téléchargeons **cuDNN** ...

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

-... dézippez, copiez la lib64 et incluez les dossiers:

```
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
```

-Le temps de faire un peu de nettoyage et de supprimer les archives téléchargées:

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

Pour installer **Tensorflow** avec **CPU / GPU** , cliquez ici:

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

Référence:

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

Lire Se lancer dans l'azur en ligne: https://riptutorial.com/fr/azure/topic/1060/se-lancer-dans-l-azur

Chapitre 2: Azure DocumentDB

Examples

Se connecter à un compte (.NET)

Pour vous connecter à votre base de données DocumentDB, vous devez créer un `DocumentClient` avec votre **URI de point de terminaison** et la **clé de service** (vous pouvez obtenir les deux à partir du portail).

Tout d'abord, vous aurez besoin des clauses suivantes:

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

Ensuite, vous pouvez créer le client avec:

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

Créer une base de données (.NET)

Votre **base de données** DocumentDB peut être créée à l'aide de la méthode `CreateDatabaseAsync` de la classe `DocumentClient` . Une base de données est le conteneur logique du stockage de documents JSON partitionné entre les collections.

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

Pour créer votre base de données:

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

Vous pouvez également vérifier si la base de données existe déjà et la créer si nécessaire:

```
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;
        }
    }
}
```

Créer une collection (.NET)

Une **collection** peut être créée à l'aide de la méthode `CreateDocumentCollectionAsync` de la classe `DocumentClient` . Une collection est un conteneur de documents JSON et de la logique d'application JavaScript associée.

```
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 });
}
```

Ou vous pouvez vérifier si la collection existe et la créer si nécessaire:

```
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;
        }
    }
}
```

Créer des documents JSON (.NET)

Un **document** peut être créé à l'aide de la méthode `CreateDocumentAsync` de la classe `DocumentClient` . Les documents sont du contenu JSON défini (arbitraire).

```
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;
        }
    }
}
```

Ayant les classes suivantes qui représentent une famille (simplifiée):

```
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; }
}
```

Requête pour les documents (.NET)

DocumentDB prend en charge les **requêtes** enrichies sur **les documents JSON** stockés dans chaque collection.

Avec une requête LINQ

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

Avec une requête SQL

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

```
    queryOptions);
```

Pagination sur une requête LINQ

Le FeedOptions est utilisé pour définir la propriété RequestContinuation obtenue lors de la première requête:

```
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));
}
```

Vous pouvez toujours appeler une fois et renvoyer le jeton de continuation au client, afin que la demande paginée soit envoyée lorsque le client souhaite la page suivante. En utilisant une classe d'assistance et une 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;
        }
}
```

Mettre à jour un document (.NET)

DocumentDB prend en charge le remplacement des documents JSON à l'aide de la méthode `ReplaceDocumentAsync` de la classe `DocumentClient` .

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

Supprimer un document (.NET)

DocumentDB prend en charge la suppression des documents JSON à l'aide de la méthode `DeleteDocumentAsync` de la classe `DocumentClient` .

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

Supprimer une base de données (.NET)

La suppression d'une base de données supprime la base de données et toutes les ressources enfants (collections, documents, etc.).

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

Lire Azure DocumentDB en ligne: https://riptutorial.com/fr/azure/topic/5176/azure-documentdb

Chapitre 3: Azure Powershell

Examples

Mode classique vs mode ARM

Lorsque vous travaillez avec Azure à l'aide de PowerShell, vous devez connaître deux manières différentes et vous verrez beaucoup de documentation Microsoft les concernant:

"Mode classique (gestion des services)"

Il s'agit de l'ancienne manière d'exploiter Azure et de gérer Azure. Il existe encore des services dans Azure qui ne peuvent être gérés qu'en mode classique, même si de plus en plus de services évoluent vers le nouveau mode ARM.

Pour répertorier les modules installés sur vos machines fonctionnant en mode classique, vous pouvez effectuer les opérations suivantes:

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

"Gestionnaire de ressources (ARM)"

C'est la nouvelle méthode de gestion d'Azure (basée sur les API REST fournies). La plupart des services que vous pouvez gérer dans Azure à partir du mode Powershell Classic peuvent désormais être gérés à l'aide du nouveau mode, à quelques exceptions près. Cela devrait être la méthode préférée pour gérer vos ressources Azure, sauf si certains services ne sont pas encore pris en charge en mode ARM.

Pour répertorier les modules installés sur votre machine contenant des commandes à utiliser en mode ARM, vous pouvez effectuer les opérations suivantes:

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

Connexion à Azure

Mode classique (gestion des services):

```
Add-AzureAccount
```

Cela vous authentifie à l'aide d'Azure Active Directory et PowerShell obtient un jeton d'accès qui expire après environ 12 heures. Vous devez donc répéter l'authentification après 12 heures.

Une alternative consiste à exécuter l'applet de commande suivante:

```
Get-AzurePublishSettingsFile
```

Cela ouvre une fenêtre de navigateur où vous pouvez télécharger un fichier de paramètres de publication. Ce fichier contient un certificat permettant à PowerShell de s'authentifier. Ensuite, vous pouvez importer votre fichier de paramètres de publication en utilisant ce qui suit:

```
Import-AzurePublishSettingsFile
```

N'oubliez pas que le fichier de paramètres de publication contient un certificat avec des privilèges d'administrateur dans votre abonnement. Conservez-le en sécurité ou supprimez-le après l'avoir utilisé.

Gestionnaire de ressources

Côté Resource Manager, nous ne pouvons utiliser l'authentification Azure Active Directory qu'avec les jetons d'accès de 12 heures. Il existe actuellement deux autres commandes que vous pouvez utiliser:

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

Sélection de l'abonnement

Lorsque vous avez plusieurs abonnements sous votre compte Azure; il est important que vous sélectionniez celui sur lequel vous souhaitez opérer (et l'utilisez par défaut); pour éviter que des accidents surviennent sur des ressources sur un abonnement incorrect.

Mode classique

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

Gestionnaire de ressources

```
Select-AzureRmSubscription
```

Information d'abonnement

Les commandes ci-dessus vous demandent de spécifier des informations (par exemple, l'ID d'abonnement) pour identifier l'abonnement auquel vous souhaitez basculer. Pour répertorier ces informations pour les abonnements auxquels vous avez accès, exécutez cette commande:

```
Get-AzureSubscription
```

Obtenez la version actuelle d'Azure PowerShell

Pour déterminer la version d'Azure PowerShell que vous avez installée, exécutez les opérations suivantes:

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

Cette commande renvoie la version installée même si vous n'avez pas chargé le module Azure PowerShell dans votre session PowerShell en cours.

Manipulation des ressources Azure

Les applets de commande Azure vous permettent d'effectuer certaines des mêmes actions sur les actifs Azure via PowerShell à l'aide du code C # ou du portail Azure.

Par exemple, ces étapes vous permettent de télécharger le contenu d'un blob Azure dans un répertoire local:

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

Gestion des gestionnaires de trafic

Avec Azure PowerShell, vous pouvez obtenir certaines fonctionnalités actuellement indisponibles sur le portail Azure , telles que:

- Reconfigurez tous les points de terminaison de Traffic Manager à la fois
- Adressez d'autres services via Azure `ResourceId` au lieu du nom de domaine, vous n'avez donc pas besoin de définir manuellement l'emplacement pour les points de terminaison Azure.

Conditions préalables

Pour commencer, vous devez vous connecter et sélectionner l'abonnement RM .

Obtenir le profil TrafficManager

Les opérations avec les gestionnaires de trafic via PowerShell se font en trois étapes:

1. Obtenir le profil TM:
   `$profile = Get-AzureRmTrafficManagerProfile -ResourceGroupName my-resource-group -Name my-traffic-manager`
   Ou créer de nouvelles comme dans cet article .
2. Explorer et modifier le profil TM
   Vérifiez les champs `$profile` et `$profile.Endpoints` pour voir la configuration de chaque noeud final.
3. Enregistrez les modifications via `Set-AzureRmTrafficManagerProfile -TrafficManagerProfile $profile` .

Modifier les points de terminaison

Tous les points de terminaison actuels sont stockés dans la liste `$profile.Endpoints` , vous pouvez donc les modifier directement par index

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

ou par nom

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

Pour effacer tous les points de terminaison, utilisez

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

Pour supprimer l'utilisation d'un noeud final particulier

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

Pour ajouter une nouvelle utilisation de point de terminaison

```
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
```

Comme vous pouvez le voir, dans le dernier cas, nous avons traité notre service azure via ResourceId plutôt que le nom de domaine.

Garder en tete

Vos modifications apportées à TM et à ses points de terminaison ne sont appliquées que lorsque vous `Set-AzureRmTrafficManagerProfile -TrafficManagerProfile $profile` . Cela vous permet de reconfigurer complètement TM en une seule opération.

Traffic Manager est une implémentation de DNS et l'adresse IP donnée aux clients a du temps à vivre (aka TTL, vous pouvez voir sa durée en secondes dans le champ `$profile.Ttl`). Ainsi, après avoir reconfiguré TM, certains clients continueront d'utiliser les anciens points de terminaison qu'ils ont mis en cache jusqu'à l'expiration de cette durée de vie.

Lire Azure Powershell en ligne: https://riptutorial.com/fr/azure/topic/3961/azure-powershell

Chapitre 4: Azure Service Fabric

Remarques

Azure Service Fabric est l'un des services PaaS proposés par Azure. Il repose sur la notion de conteneurs et de services: contrairement aux services de calcul (rôles Web et rôles de travail), votre code ne s'exécute pas dans une (ou plusieurs) machine virtuelle, mais est exécuté dans un conteneur et partagé avec d'autres services. .

Il est important de noter qu'ici, et à travers les articles et la documentation Microsoft sur Service Fabric, le *conteneur* est conçu dans son sens plus général, et non comme un conteneur de style "Docker".

Vous pouvez avoir (et vous aurez généralement) plusieurs conteneurs, qui formeront un cluster. Les services exécutés à l'intérieur du conteneur peuvent être, par ordre croissant de "prise de conscience"

- Exécutables invités
- Services "fiables"
    - Stateful
    - Apatride
- Acteurs "fiables"

Examples

Acteurs fiables

Un acteur de Service Fabric est défini par une paire standard d'interface / classe .NET:

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


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

Chaque méthode de la paire interface / classe doit être asynchrone et ne peut pas avoir de paramètre de sortie ou de référence.

Il est facile de comprendre pourquoi si vous pensez au modèle de l’acteur: les objets interagissent les uns avec les autres par l’échange de messages. Les messages sont livrés à une classe

d'acteurs via les méthodes asynchrones; les réponses sont gérées par le runtime des acteurs (l'acteur "container") et renvoyées à l'appelant.

Le kit de développement de Service Fabric génère un proxy au moment de la compilation. Ce proxy est utilisé par le client acteur pour appeler ses méthodes (c'est-à-dire pour envoyer un message à l'acteur et `await` une réponse).

Le client identifie un acteur via un identifiant. L'ID peut être déjà connu (vous l'avez obtenu à partir d'une base de données, d'un autre acteur, ou bien de l'ID utilisateur lié à cet acteur, ou encore du numéro de série d'un objet réel).

Si vous avez besoin de créer un nouvel acteur et que vous avez juste besoin d'un identifiant, la classe ActorId (fournie) a des méthodes pour créer un identifiant d'acteur distribué aléatoirement

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

Ensuite, vous pouvez utiliser la classe `ActorProxy` pour créer un objet proxy pour l'acteur. Cela n'active pas un acteur ou n'invoque aucune méthode pour le moment. IMyActor myActor = ActorProxy.Create (actorId, new Uri ("fabric: / MyApp / MyActorService"));

Ensuite, vous pouvez utiliser le proxy pour appeler une méthode sur l'acteur. Si un acteur avec l'ID donné n'existe pas, il sera activé (créé dans l'un des conteneurs du cluster), puis le moteur d'exécution enverra un message à l'acteur, exécutant son appel de méthode et complétant la tâche lorsque l'acteur réponses:

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

Lire Azure Service Fabric en ligne: https://riptutorial.com/fr/azure/topic/3802/azure-service-fabric

Chapitre 5: Azure Virtual Machines

Examples

Créer une VM Azure par une API ASM classique

```
# 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
```

Pour plus d'informations, consultez Comment créer une machine virtuelle Azure (VM) par Powershell à l'aide de l'API ASM classique.

Lire Azure Virtual Machines en ligne: https://riptutorial.com/fr/azure/topic/6350/azure-virtual-machines

Chapitre 6: Azure-Automation

Paramètres

Le nom du paramètre	La description
resourceGroupName	Le groupe de ressources Azure sur lequel se trouve le compte de stockage
connectionName	La connexion Azure Run As (pricipal de service) créée lors de la création du compte Automation
StorageAccountName	Le nom du compte Azure Storage
Nom du conteneur	Le nom du conteneur blob
DaysOld	Nombre de jours pendant lesquels un objet blob peut être supprimé

Remarques

Assurez-vous d'avoir accès à Azure Active Directory afin que, lors de la création du compte Automation, Azure crée un compte RunAs pour vous. Cela vous évitera beaucoup de problèmes.

Examples

Supprimer Blobs dans le stockage Blob plus ancien que le nombre de jours

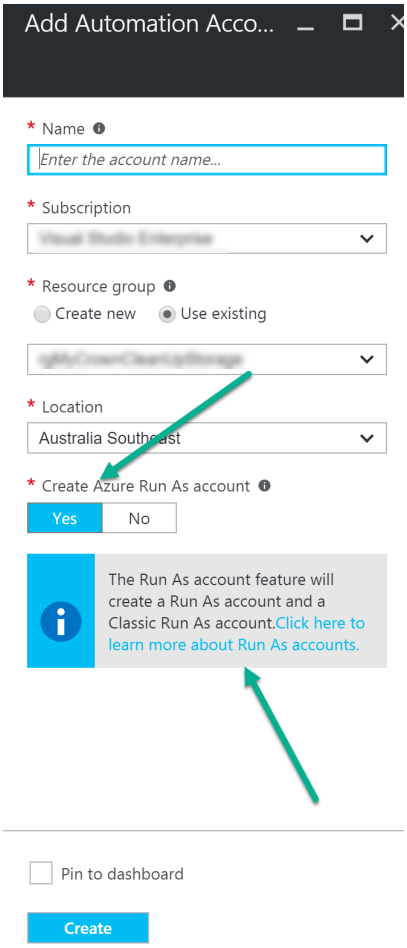
Voici un exemple de runbook d'automatisation Azure Powershell qui supprime les blobs d'un conteneur de stockage Azure datant de plusieurs jours.

Cela peut être utile pour supprimer les anciennes sauvegardes SQL afin de réduire les coûts et l'espace.

Il faut un certain nombre de paramètres qui sont explicites.

Note: J'ai laissé du code commenté pour aider au débogage.

Il utilise un principal de service que Azure peut configurer automatiquement lorsque vous créez votre compte d'automatisation. Vous devez disposer d'un accès à Azure Active Directory. Voir photo:

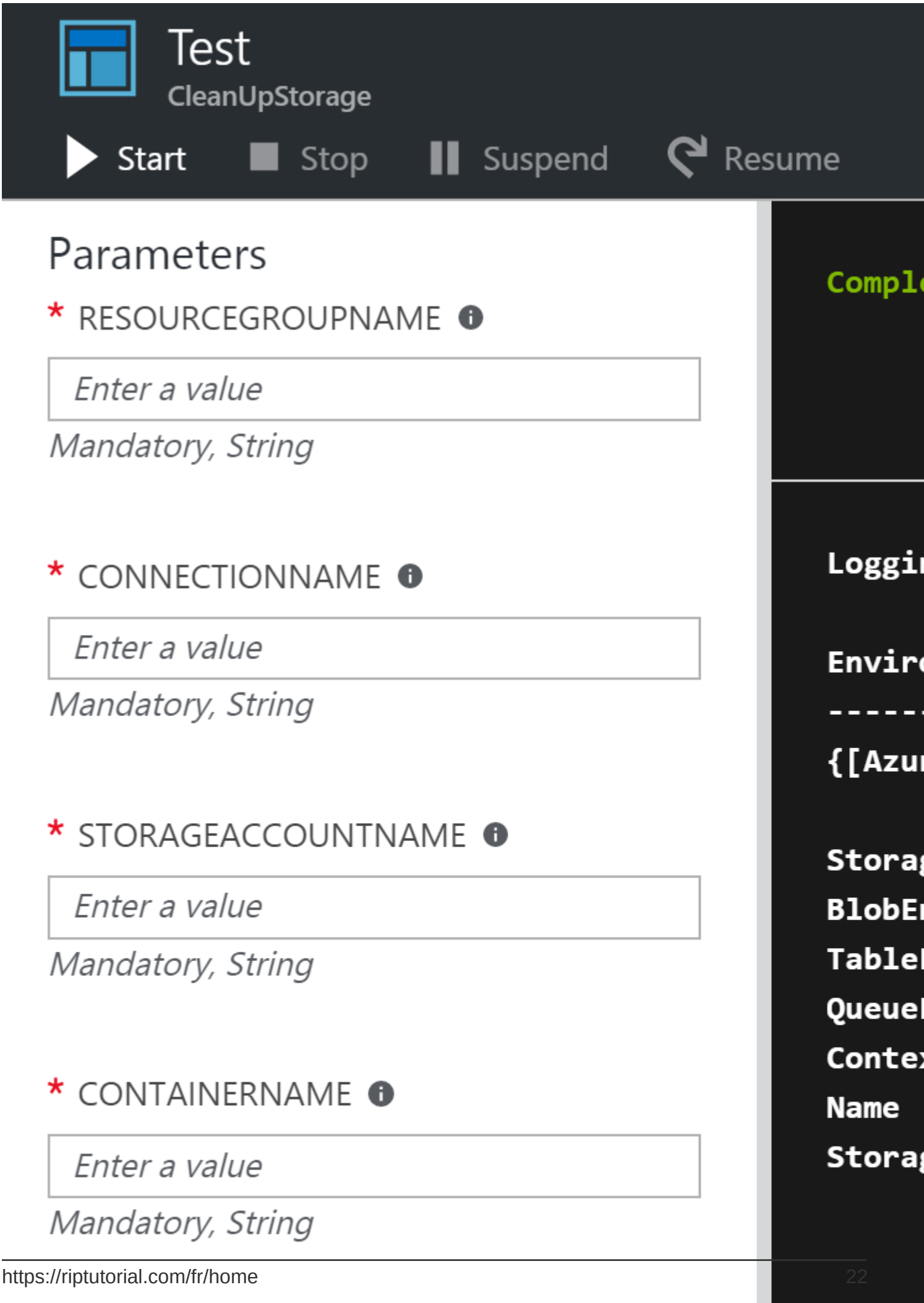



```
<#
.DESCRIPTION
    Removes all blobs older than a number of days back using the Run As Account (Service
Principal)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Si vous utilisez le volet de test, vous pouvez entrer les paramètres requis et les exécuter.


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

Chapitre 7: Compte de service Azure Media

Remarques

Un compte de service de média est un compte Azure qui vous permet d'accéder aux services de médias basés sur le cloud dans Azure. Stocke les métadonnées des fichiers multimédias que vous créez, au lieu d'enregistrer le contenu multimédia réel. Pour travailler avec un compte de service de médias, vous devez disposer d'un compte de stockage associé. Lors de la création d'un compte de service multimédia, vous pouvez sélectionner le compte de stockage que vous avez déjà ou en créer un nouveau. Le compte de service de média et le compte de stockage étant traités séparément, le contenu sera disponible dans votre compte de stockage même si vous supprimez votre compte de service de média.

Examples

Création d'un actif dans un compte de service multimédia

```
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;
            }
        }
```

Récupération des éléments de l'actif

```
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;
        }
    }
```

Lire Compte de service Azure Media en ligne: https://riptutorial.com/fr/azure/topic/4997/compte-de-service-azure-media

Chapitre 8: Modèles Azure Resource Manager

Syntaxe

- La syntaxe des modèles ARM est bien documentée: https://azure.microsoft.com/en-us/documentation/articles/resource-group-authoring-templates/

Examples

Créer une ressource d'extension

Les ressources d'extension dans Azure sont des ressources qui étendent d'autres ressources.

Ce modèle crée un Azure Key Vault ainsi qu'une extension DiagnosticSettings.

Choses à noter:

- La ressource d'extension est créée sous l'attribut `resources` de la ressource parent
- Il doit avoir un attribut `dependsOn` référençant la ressource parente (pour empêcher ARM de tenter de créer l'extension en parallèle avec la ressource parente)

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

Lire Modèles Azure Resource Manager en ligne:
https://riptutorial.com/fr/azure/topic/3923/modeles-azure-resource-manager

Chapitre 9: Options de stockage Azure

Examples

Renommer un fichier blob dans Azure Blob Storage

Il n'y a pas d'API capable de renommer le fichier blob sur Azure. Cet extrait de code montre comment renommer un fichier blob dans 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();
    }
}
```

Pour plus d'informations, consultez Comment renommer un fichier blob dans Azure Blob Storage

Importer / exporter un fichier Excel Azure vers / depuis Azure SQL Server dans ASP.NET

Cet exemple montre comment importer le fichier blob Azure Excel de la feuille de calcul vers la base de données sur le serveur SQL Azure et comment l'exporter du blob Excel vers DB Azure.

Conditions préalables:

- Version Microsoft Visual Studio 2015
- Open XML SDK 2.5 pour Microsoft Office
- Un compte de stockage Azure
- Azure SQL Server

Ajoutez la référence DocumentFormat.OpenXml à votre projet.

1. Exporter des données de la base de données vers Azure Excel blob
   Enregistrez Excel dans le stockage sur serveur, puis transférez-le sur 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. Importer le fichier Excel Azure dans la base de données
   Nous ne pouvons pas lire directement les données d'Excel Blob, nous devons donc les enregistrer sur le stockage du serveur, puis les gérer.
   Nous utilisons SqlBulkCopy pour insérer en bloc des données dans la base de données.

```
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;
}
```

Pour plus d'informations, consultez https://code.msdn.microsoft.com/How-to-ImportExport-Azure-0c858df9 .

Rompez le bail verrouillé du stockage d'objets blob dans Microsoft Azure

Il n'y a pas d'API capable de rompre le bail verrouillé du stockage d'objets blob dans Microsoft Azure. Cet extrait de code illustre le bail verrouillé du stockage d'objets blob dans 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."
        }
```

Pour plus d'informations, consultez Comment rompre le bail verrouillé du stockage d'objets blob par ARM dans Microsoft Azure (PowerShell).

Lire Options de stockage Azure en ligne: https://riptutorial.com/fr/azure/topic/5405/options-de-stockage-azure

Chapitre 10: Options de stockage Azure

Examples

Connexion à une file d'attente de stockage Azure

Les options de stockage dans Azure fournissent une API "REST" (ou, mieux, une API HTTP)

Azure SDK offre aux clients plusieurs langues. Voyons par exemple comment initialiser l'un des objets de stockage (une file d'attente) à l'aide des bibliothèques clientes C #.

Tous les accès à Azure Storage s'effectuent via un compte de stockage. Vous pouvez créer un compte de stockage de plusieurs manières: via le portail, via l'interface de ligne de commande Azure, PowerShell, Azure Resource Manager (ARM), ...

Dans cet exemple, nous supposons que vous en avez déjà un et que vous l'avez stocké dans votre fichier `app.config` .

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

Les files d'attente sont accessibles à l'URL suivante: `http://<storage account>.queue.core.windows.net/<queue>`

Les bibliothèques clientes généreront cette URL pour vous; il vous suffit de spécifier le nom de la file d'attente (qui doit être en minuscule). La première étape consiste à obtenir une référence à un client de file d'attente, qui sera utilisée pour gérer vos files d'attente (les files d'attente sont contenues dans le compte de stockage spécifié).

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

Vous utilisez le client pour obtenir une référence à une file d'attente.

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

Maintenant, en utilisant ce proxy de `queue` , vous pouvez diriger n'importe quelle opération vers votre file d'attente.

Généralement, la première opération consiste à créer la file d'attente si elle n'existe pas déjà.

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

Notez le nom de l'opération. Pourquoi "si n'existe pas"? Il existe plusieurs raisons:

- Vous pouvez déployer plusieurs instances de "quelque chose" qui exécutera ce code ("quelquechose" est généralement un service de calcul , comme un rôle de rôle Web ou de

travail, mais il peut s'agir d'une application Web, d'un service Fabric ou de code personnalisé). une VM ...)
- votre application peut redémarrer à tout moment. Rappelez-vous qu'il s'agit d'un environnement de cloud où, en particulier pour les services PaaS, les instances sont éphémères. Vous n'avez pas le même degré de contrôle sur votre application que sur votre application déployée localement.

Encore mieux, vous devez utiliser la version asynchrone du même appel API:

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

Nous avons utilisé une file d'attente dans cet exemple, mais l'exemple peut être facilement appliqué aux autres objets de stockage (blobs, tables et fichiers).

Une fois que vous avez créé votre objet de stockage, vous êtes prêt à l'utiliser.

Lire Options de stockage Azure en ligne: https://riptutorial.com/fr/azure/topic/6008/options-de-stockage-azure

Crédits

S. No	Chapitres	Contributeurs
1	Se lancer dans l'azur	awh112, Bernard Vander Beken, Community, KARANJ, lorenzo montanari, Sibeesh Venu, user2314737
2	Azure DocumentDB	gbellmann, Matias Quaranta
3	Azure Powershell	Anton Purin, CmdrTchort, frank tan, juunas, RedGreenCode
4	Azure Service Fabric	Lorenzo Dematté, Stephen Leppik
5	Azure Virtual Machines	Dale Chen
6	Azure-Automation	Akos Nagy, RuSs
7	Compte de service Azure Media	Sibeesh Venu
8	Modèles Azure Resource Manager	BenV
9	Options de stockage Azure	Dale Chen, Gaurav Mantri