



Бесплатная электронная книга

УЧУСЬ

azure

Free unaffiliated eBook created from
Stack Overflow contributors.

#azure

.....	1
1:	2
.....	2
Examples.....	2
Azure N-series (GPU): CUDA, cudnn, Tensorflow UBUNTU 16.04 LTS.....	2
2: Azure DocumentDB.....	4
Examples.....	4
(.NET).....	4
(.NET).....	4
(.NET).....	5
JSON (.NET).....	6
(.NET).....	7
LINQ.....	7
SQL-.....	7
LINQ.....	8
(.NET).....	9
(.NET).....	9
(.NET).....	9
3: Azure Powershell.....	11
Examples.....	11
ARM.....	11
Azure.....	11
.....	12
Azure PowerShell.....	13
Azure Assets.....	13
.....	13
.....	13
TrafficManager.....	13
.....	14
.....	14
4: Azure Service Fabric.....	15
.....	

Examples.....	15
.....	15
5: Azure-Automation.....	17
.....	17
.....	17
Examples.....	17
Blobs Blob	17
.....	21
6: Azure Media.....	23
.....	23
Examples.....	23
.....	23
.....	23
7: Azure.....	25
Examples.....	25
Azure VM API ASM.....	25
8: Azure.....	26
Examples.....	26
blob Azure Blob.....	26
/ Azure Excel / Azure SQL Server ASP.NET.....	26
blob Microsoft Azure.....	30
9: Azure.....	31
Examples.....	31
Azure.....	31
10: Azure.....	33
.....	33
Examples.....	33
.....	33
.....	35

Около

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.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

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

глава 1: Начало работы с лазурным

замечания

Azure - это торговая марка, в которой Microsoft предлагает свои услуги облачных вычислений. Некоторые из основных услуг, предлагаемых на платформе Microsoft Azure:

- Инфраструктура как услуга (IaaS): Linux и Windows Azure Virtual Machines
- Платформа как услуга (PaaS): Служба приложений предоставляет полную платформу для разработки приложений (Web и мобильных)
- Облачное хранилище: службы хранения SQL и noSQL
- Программное обеспечение как услуга (SaaS): планировщик, резервное копирование, аналитика, машинное обучение, безопасность и аутентификация

Вот инфографика для просмотра основных предложений Azure:

<https://azure.microsoft.com/en-us/resources/infographics/azure/> . И [здесь](#) вы можете просматривать и фильтровать все продукты Azure по категориям.

Examples

Azure N-series (GPU): установить CUDA, cudnn, Tensorflow на UBUNTU 16.04 LTS

Проведя более 5 часов, я нашел это простое решение:

-Чтобы убедиться, что в системе есть графический процессор с поддержкой CUDA, выполните следующую команду:

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

Вы увидите вывод, аналогичный приведенному ниже примеру (показывающий карту NVIDIA Tesla K80 / M60):

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

-Удаление драйвера нувориша:

```
sudo -i  
rmmod nouveau
```

-После *перезагрузки* : *перезагрузка* `sudo reboot` , убедитесь, что драйвер установлен правильно, выпуская:

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

-Следуем, загрузите пакет **CUDA** из 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
```

-... сообщить ему apt-get и установить 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
```

- Теперь мы можем проверить состояние GPU (ы), запусив:

```
nvidia-smi
```

Затем мы загружаем **cuDNN** ...

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

-... разархивировать, скопировать lib64 и включить папки:

```
tar -zxvf 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
```

-Изменение времени для очистки и удаления загруженных архивов:

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

Чтобы установить **Tensorflow** с **CPU / GPU** , перейдите сюда:

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

Ссылка:

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

Прочитайте Начало работы с лазурным онлайн: <https://riptutorial.com/ru/azure/topic/1060/начало-работы-с-лазурным>

глава 2: Azure DocumentDB

Examples

Подключение к учетной записи (.NET)

Чтобы подключиться к вашей базе данных DocumentDB, вам необходимо создать `DocumentClient` с вашим **URI конечной точки** и **ключом службы** (вы можете получить оба из портала).

Прежде всего, вам понадобятся следующие предложения:

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

Затем вы можете создать клиент с:

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

Создать базу данных (.NET)

База данных DocumentDB может быть создана с использованием метода `CreateDatabaseAsync` класса `DocumentClient`. База данных - это логический контейнер хранения документов JSON, разделенный на коллекции.

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

Чтобы создать свою базу данных:

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

Вы также можете проверить, существует ли база данных и при необходимости ее создать:

```
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;
        }
    }
}

```

Создание коллекции (.NET)

Сбор может быть создан с помощью метода `CreateDocumentCollectionAsync` класса

`DocumentClient`. Коллекция представляет собой контейнер документов JSON и связанную с ним логику приложений JavaScript.

```

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 });
}

```

Или вы можете проверить, существует ли коллекция и создать ее, если необходимо:

```

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;
        }
    }
}

```

Создание документов JSON (.NET)

Документ может быть создан с помощью метода `CreateDocumentAsync` класса `DocumentClient`. Документы представляют собой пользовательский (произвольный) контент JSON.

```

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

```

Имея следующие классы, которые представляют (упрощенное) семейство:

```

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; }
}

```

Запрос для документов (.NET)

DocumentDB поддерживает богатые **запросы** по **документам JSON**, хранящимся в каждой коллекции.

С запросом LINQ

```

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

```

С SQL-запросом

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

Разбиение на страницы по запросу LINQ

[FeedOptions](#) используется для установки свойства [RequestContinuation](#), полученного по первому запросу:

```
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));
}
```

Вы всегда можете позвонить один раз и вернуть токен продолжения клиенту, поэтому постраничный запрос отправляется, когда клиент хочет следующую страницу.

Использование вспомогательного класса и расширения:

```
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;
    }
}

```

Обновление документа (.NET)

DocumentDB поддерживает замену документов JSON с использованием метода

ReplaceDocumentAsync класса DocumentClient .

```

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

```

Удаление документа (.NET)

DocumentDB поддерживает удаление документов JSON с использованием метода

DeleteDocumentAsync класса DocumentClient .

```

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

```

Удалить базу данных (.NET)

При удалении базы данных удаляются база данных и все дочерние ресурсы (коллекции,

документы и т. Д.).

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

Прочитайте Azure DocumentDB онлайн: <https://riptutorial.com/ru/azure/topic/5176/azure-documentdb>

глава 3: Azure Powershell

Examples

Режим классического режима против ARM

При работе с Azure с использованием PowerShell существует два разных способа, о которых вам следует знать, и вы увидите много документации Microsoft, относящуюся к обоим из них:

«Классический режим (Service Management)»

Это старый способ управлять Azure и управлять Azure. В Azure по-прежнему есть некоторые сервисы, которые могут управляться только в классическом режиме, хотя все больше и больше сервисов перемещаются в новый режим ARM.

Чтобы перечислить модули, установленные на ваших компьютерах, работающих в классическом режиме, вы можете сделать следующее:

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

«Менеджер ресурсов (ARM)»

Это новый способ управления Azure (на основе предоставленных API REST). Большинство сервисов, которыми вы могли бы управлять в Azure из классического режима Powershell, теперь можно управлять с помощью нового режима с некоторыми исключениями. Это должен быть предпочтительный способ управления ресурсами Azure, если у вас нет определенных служб, которые еще не поддерживаются в режиме ARM.

Чтобы перечислить модули, установленные на вашем компьютере, содержащие команды для работы в режиме ARM, вы можете сделать следующее:

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

Войти в Azure

Классический режим управления сервисами:

```
Add-AzureAccount
```

Это позволит вам аутентифицировать вас с помощью Azure Active Directory, а PowerShell получит токен доступа, срок действия которого истекает через 12 часов. Поэтому вы должны повторить проверку подлинности через 12 часов.

Альтернативой является запуск следующего командлета:

```
Get-AzurePublishSettingsFile
```

Откроется окно браузера, в котором вы можете загрузить файл настроек публикации. Этот файл содержит сертификат, который позволяет PowerShell аутентифицироваться. Затем вы можете импортировать файл настроек публикации, используя следующее:

```
Import-AzurePublishSettingsFile
```

Помните, что файл настроек публикации содержит сертификат с эффективными правами администратора в вашей подписке. Храните его в безопасности или удаляйте после его использования.

Менеджер ресурсов

В стороне Resource Manager мы можем использовать только аутентификацию Azure Active Directory с помощью 12-ти токенов доступа. В настоящее время вы можете использовать две альтернативные команды:

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

Выбор подписки

Когда у вас есть несколько подписей под вашей учетной записью Azure; важно выбрать тот, который вы хотите использовать (и использовать его по умолчанию); чтобы избежать несчастных случаев, связанных с ресурсами, при неправильной подписке.

Классический режим

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

Менеджер ресурсов

```
Select-AzureRmSubscription
```

Информация о подписке

В приведенных выше командах вам необходимо указать информацию (например, идентификатор подписки) для идентификации подписки, на которую вы хотите переключиться. Чтобы перечислить эту информацию для подписки, к которой у вас есть доступ, выполните следующую команду:

```
Get-AzureSubscription
```

Получить текущую версию Azure PowerShell

Чтобы определить версию Azure PowerShell, которую вы установили, выполните следующие действия:

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

Эта команда возвращает установленную версию, даже если вы не загрузили модуль Azure PowerShell в текущий сеанс PowerShell.

Манипулировать Azure Assets

Командлеты Azure позволяют выполнять некоторые из одних и тех же действий с активами Azure через PowerShell, чтобы использовать код C # или портал Azure.

Например, эти шаги позволяют загружать содержимое лабиринта Azure в локальный каталог:

```
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\
```

Управление менеджерами трафика

С Azure PowerShell вы можете получить определенную функциональность, недоступную в настоящее время на [Azure Portal](#) , например:

- Переконфигурировать конечные точки всех диспетчеров одновременно
- Адреса других служб через Azure `ResourceId` вместо имени домена, поэтому вам не нужно вручную устанавливать местоположение для конечных точек Azure

Предпосылки

Для начала вам нужно [войти в систему](#) и [выбрать подписку RM](#) .

Получить профиль TrafficManager

Операции с диспетчерами трафика через PowerShell выполняются в три этапа:

1. Получить профиль ТМ:

```
$profile = Get-AzureRmTrafficManagerProfile -ResourceGroupName my-resource-group -Name my-traffic-manager
```

Или создайте новый, как [в этой статье](#) .

2. Изучить и изменить профиль ТМ

Проверьте поля `$profile` и `$profile.Endpoints` чтобы увидеть конфигурацию каждой конечной точки.

3. Сохранить изменения через `Set-AzureRmTrafficManagerProfile -TrafficManagerProfile $profile .`

Изменение конечных точек

Все текущие конечные точки хранятся в списке `$profile.Endpoints` , поэтому вы можете изменять их напрямую по индексу

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

или по имени

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

Чтобы очистить все конечные точки, используйте

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

Чтобы удалить конкретную конечную точку

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

Чтобы добавить новую конечную точку

```
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
```

Как вы можете видеть, в последнем случае мы обратились к нашей службе azure через `ResourceId`, а не к доменному имени.

Иметь ввиду

Ваши изменения в ТМ и ее конечных точках не применяются до тех пор, пока вы не вызовете `Set-AzureRmTrafficManagerProfile -TrafficManagerProfile $profile .` Это позволяет полностью переконфигурировать ТМ за одну операцию.

Traffic Manager - это реализация DNS и IP-адреса, предоставляемых клиентам, имеет некоторое время для жизни (например, TTL, вы можете видеть, что это длительность в секундах в поле `$profile.Ttl`). Таким образом, после того, как вы переконфигурировали ТМ, некоторые клиенты будут продолжать использовать старые конечные точки, которые они кэшировали до истечения срока действия TTL.

Прочитайте Azure Powershell онлайн: <https://riptutorial.com/ru/azure/topic/3961/azure-powershell>

глава 4: Azure Service Fabric

замечания

Azure Service Fabric - одна из услуг PaaS, предлагаемых Azure. Он основан на понятии контейнеров и сервисов: в отличие от служб вычислений (роли веб-роли и рабочих), ваш код не запускается внутри одной (или более) виртуальных машин, но вместо этого они запускаются внутри контейнера, деля его с другими службами ,

Важно отметить, что здесь и во всех статьях и документах Microsoft по сервису Fabric *контейнер* предназначен в более общем смысле, а не как контейнер стиля «Docker».

Вы можете иметь (и, как правило, у вас) несколько контейнеров, которые образуют кластер. Услуги, выполняемые внутри контейнера, могут быть в порядке возрастания «осведомленности»,

- Гостевой исполняемый файл
- «Надежные» услуги
 - Stateful
 - Stateless
- «Надежные» актеры

Examples

Надежные актеры

Актер внутри Service Fabric определяется стандартной парой .NET interface / class:

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

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

Каждый метод в паре интерфейса / класса должен быть асинхронным, и они не могут иметь или обновлять паразитные элементы.

Легко понять, почему, если вы думаете о модели актера: объекты взаимодействуют друг с

другом посредством обмена сообщениями. Сообщения доставляются в класс актера через асинхронные методы; ответы обрабатываются участниками времени выполнения (актер «контейнер») и перенаправляются обратно вызывающему абоненту.

SDK Service Fabric генерирует прокси во время компиляции. Этот прокси используется клиентом-актером для вызова его методов (т. Е. Доставлять сообщение актеру и `await` ответа).

Клиент идентифицирует актера через идентификатор. Идентификатор может быть уже известен (вы получили его от БД, от другого Актера, или, возможно, это идентификатор пользователя, связанный с этим актером, или снова серийный номер реального объекта).

Если вам нужно создать нового актера, и вам просто нужен идентификатор, класс (предоставленный) `ActorId` имеет методы для создания случайно распределенного идентификатора игрока

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

Затем вы можете использовать класс `ActorProxy` для создания прокси-объекта для актера. Это не активирует актера или еще не вызывает какие-либо методы. `IMyActor myActor = ActorProxy.Create(actorId, новый Uri («fabric: / MyApp / MyActorService»));`

Затем вы можете использовать прокси для вызова метода на актера. Если актер с данным идентификатором не существует, он будет активирован (создается внутри одного из контейнеров в кластере), а затем среда выполнения отправит сообщение актеру, выполнив вызов метода и завершив задачу, когда актер ответит:

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

Прочитайте [Azure Service Fabric онлайн](https://riptutorial.com/ru/azure/topic/3802/azure-service-fabric): <https://riptutorial.com/ru/azure/topic/3802/azure-service-fabric>

глава 5: Azure-Automation

параметры

Имя параметра	Описание
resourceGroupName	Группа ресурсов Azure, в которой находится учетная запись хранилища
название соединения	Соединение Azure Run As (service principal), созданное при создании учетной записи автоматизации
StorageAccountName	Имя учетной записи Azure Storage
ИмяКонтейнера	Название контейнера blob
DaysOld	Количество дней, в течение которых может быть blob, до его удаления.

замечания

Убедитесь, что у вас есть доступ к Azure Active Directory, поэтому при создании учетной записи автоматизации Azure создает для вас учетную запись RunAs. Это сэкономит вам массу неприятностей.

Examples

Удалить Blobs в хранилище Blob старше нескольких дней

Ниже приведен пример рабочей книги автоматизации Azure Powershell, которая удаляет любые капли в контейнере хранения Azure, которые старше, чем несколько дней.

Это может быть полезно для удаления старых резервных копий SQL для экономии затрат и пространства.

Он требует нескольких параметров, которые самоочевидны.

Примечание. Я оставил некоторый код с комментариями, чтобы помочь в отладке.

Он использует принципала службы, который Azure может настроить для вас автоматически при создании учетной записи автоматизации. Вам необходимо иметь доступ к Azure Active Directory. См. Рис.

Add Automation Acco... — □ ×

* Name ?


* Subscription

* Resource group ?

Create new Use existing

* Location

* Create Azure Run As account ?



The Run As account feature will create a Run As account and a Classic Run As account. [Click here to learn more about Run As accounts.](#)

Pin to dashboard

```
<#
.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);
}
```

Вы используете тестовую панель, вы можете ввести необходимые параметры и запустить ее.



Test

CleanUpStorage



Start



Stop



Suspend



Resume

Parameters

* RESOURCEGROUPNAME ⓘ

Mandatory, String

* CONNECTIONNAME ⓘ

Mandatory, String

* STORAGEACCOUNTNAME ⓘ

Mandatory, String

* CONTAINERNAME ⓘ

Mandatory, String

Comple

Loggin

Enviro

{[Azur

Storag

BlobE

Table

Queue

Contex

Name

Storag

на «ПОДРОБНЫЙ», вы получите гораздо лучшую оценку того, насколько фрагментированы индексы.

Я также загрузил эту версию в Github: <https://github.com/conwid/IndexRebuildScript>

Прочитайте Azure-Automation онлайн: <https://riptutorial.com/ru/azure/topic/7258/azure-automation>

глава 6: Аккаунт службы Azure Media

замечания

Учетная запись мультимедийного сервиса - это учетная запись на основе Azure, которая дает вам доступ к облачным мультимедийным службам в Azure. Сохраняет метаданные создаваемых медиафайлов, вместо этого сохраняя фактический медиаконтент. Для работы со учетной записью службы мультимедиа у вас должна быть связанная учетная запись хранилища. При создании учетной записи службы мультимедиа вы можете либо выбрать учетную запись хранилища, которую у вас уже есть, либо создать новый. Поскольку учетная запись и учетная запись мультимедийного сервиса обрабатываются отдельно, контент будет доступен на вашей учетной записи в хранилище, даже если вы удалите свою учетную запись мультимедийного сервиса. Обратите внимание, что ваша область учетной записи хранения должна быть такой же, как и ваша учетная запись службы поддержки.

Examples

Создание актива в учетной записи службы мультимедиа

```
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;
    }
}
```

Получение элементов из актива

```
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;
}
}
```

Прочитайте Аккаунт службы Azure Media онлайн: <https://riptutorial.com/ru/azure/topic/4997/аккаунт-службы-azure-media>

глава 7: Виртуальные машины Azure

Examples

Создание Azure VM по классическому API ASM

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

Для получения дополнительной информации см. [Раздел Создание виртуальной машины Azure \(VM\) от Powershell с использованием классического API ASM](#)

Прочитайте Виртуальные машины Azure онлайн: <https://riptutorial.com/ru/azure/topic/6350/виртуальные-машины-azure>

глава 8: Параметры хранения Azure

Examples

Переименование файла blob в хранилище Azure Blob

Нет API, который может переименовать файл blob на Azure. Этот фрагмент кода демонстрирует, как переименовать файл blob в 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();
    }
}
```

Дополнительные сведения см. В разделе [Как переименовать файл blob в хранилище Azure Blob](#)

Импорт / Экспорт файла Azure Excel в / из Azure SQL Server в ASP.NET

В этом примере демонстрируется, как импортировать рабочий лист Azure Excel в DB на Azure SQL Server и как его экспортировать из базы данных в Azure Excel.

Предпосылки:

- Версия Microsoft Visual Studio 2015
- [Open XML SDK 2.5 для Microsoft Office](#)
- Аккаунт хранилища Azure
- Azure SQL Server

Добавьте в проект проект DocumentFormat.OpenXml.

1. Экспорт данных из базы данных в Azure Excel blob

Сохраните Excel на сервере, а затем загрузите его в 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. Импортировать файл Excel Azure в DB

Мы не можем напрямую читать данные excel blob, поэтому нам нужно сохранить их на сервере, а затем обработать.

Мы используем [SqlBulkCopy](#) для объемной вставки данных в 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;
}

```

Для получения дополнительной информации см. <https://code.msdn.microsoft.com/How-to-ImportExport-Azure-0c858df9>.

Перерыв заблокированной аренды хранилища blob в Microsoft Azure

Нет API, который может сломать заблокированную аренду хранилища blob в Microsoft Azure. Этот фрагмент кода демонстрирует разрыв заблокированной аренды хранилища blob в 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."
}

```

Дополнительные сведения см. В разделе [Как заблокировать заблокированную аренду хранилища blob с помощью ARM в Microsoft Azure \(PowerShell\)](#)

Прочитайте [Параметры хранения Azure онлайн: https://riptutorial.com/ru/azure/topic/5405/параметры-хранения-azure](https://riptutorial.com/ru/azure/topic/5405/параметры-хранения-azure)

глава 9: Параметры хранения Azure

Examples

Подключение к очереди хранения Azure

Параметры хранения в Azure предоставляют API REST (или, лучше, HTTP API)

Azure SDK предлагает клиентам несколько языков. Давайте посмотрим, например, как инициализировать один из объектов хранения (очередь) с помощью клиентских библиотек C #.

Весь доступ к хранилищу Azure осуществляется через учетную запись хранилища. Вы можете создать учетную запись хранилища несколькими способами: через портал, через Azure CLI, PowerShell, Azure Resource Manager (ARM), ...

В этом примере мы предположим, что у вас уже есть один, и вы сохранили его в файле `app.config`.

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

Очереди достижимы по следующему URL-адресу: `http://<storage account>.queue.core.windows.net/<queue>`

Библиотеки клиентов будут генерировать этот URL-адрес для вас; вам просто нужно указать имя очереди (которое должно быть строчным). Первым шагом является получение ссылки на клиента очереди, который будет использоваться для управления очередями (очереди содержатся в указанной учетной записи хранилища).

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

Вы используете клиент для получения ссылки на очередь.

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

Теперь, используя этот прокси-сервер `queue`, вы можете направить любую операцию в свою очередь.

Как правило, первая операция заключается в создании очереди, если она еще не существует

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

Обратите внимание на название операции. Почему «если не существует»? Существует несколько причин:

- вы можете развертывать несколько экземпляров «чего-то», которые будут запускать этот код («что-то», как правило, представляет собой [вычислительную службу](#), такую как роль Web или роль «рабочий», но это может быть веб-приложение, служба Fabric, какой-то пользовательский код в виртуальная машина ...)
- ваше приложение может перезагрузиться в любое время. Помните, что это облачная среда, где, особенно для служб PaaS, экземпляры являются эфемерными. У вас нет такой же степени контроля над вашим приложением, как и в локально развернутом приложении.

Еще лучше, вы должны использовать асинхронную версию одного и того же вызова API:

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

Мы использовали очередь в этом примере, но этот пример можно легко применить к другим объектам хранения (блокам, таблицам и файлам).

Создав объект хранения, вы можете начать его использовать.

Прочитайте [Параметры хранения Azure онлайн](https://riptutorial.com/ru/azure/topic/6008/параметры-хранения-azure): <https://riptutorial.com/ru/azure/topic/6008/параметры-хранения-azure>

глава 10: Шаблоны ресурсов Azure

Синтаксис

- Синтаксис шаблонов ARM хорошо документирован: <https://azure.microsoft.com/en-us/documentation/articles/resource-group-authoring-templates/>

Examples

Создать ресурс расширения

Ресурсы расширения в Azure - это ресурсы, которые расширяют другие ресурсы.

Этот шаблон создает хранилище ключей Azure, а также расширение DiagnosticSettings.

Что нужно отметить:

- Ресурс расширения создается под `resources` атрибут родительского ресурса
- Он должен иметь атрибут `dependsOn` ссылающийся на родительский ресурс (чтобы предотвратить попытку ARM создать расширение параллельно с родительским ресурсом)

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

```

Прочитайте Шаблоны ресурсов Azure онлайн: <https://riptutorial.com/ru/azure/topic/3923/шаблоны-ресурсов-azure>

кредиты

S. No	Главы	Contributors
1	Начало работы с лазурным	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-Automation	Akos Nagy , RuSs
6	Аккаунт службы Azure Media	Sibeesh Venu
7	Виртуальные машины Azure	Dale Chen
8	Параметры хранения Azure	Dale Chen , Gaurav Mantri
9	Шаблоны ресурсов Azure	BenV