

Microsoft® SQL® Backup to Azure

Prepared for

Customer Name

Thursday, 6 February 2020

Version 0.9

Prepared by

Petar Trboš

Principal Premier Field Engineer

petar.trbos@microsoft.com

Revision and Signoff Sheet

Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change reference |
| 10 Nov 2019 | Petar Trboš | 0.1 | Create document |
| 05 Feb 2020 | Petar Trboš | 0.9 | Initial draft |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Reviewers

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Version reviewed | Position | Date |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Referenced Documents

|  |  |  |  |
| --- | --- | --- | --- |
| Document | Document Name | Version | Author |
|  | [SQL Server backup to URL best practices and troubleshooting](https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/sql-server-backup-to-url-best-practices-and-troubleshooting?view=sql-server-ver15) |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Content

[SQL Server backup to URL best practices and troubleshooting 3](#_Toc31885865)

[Introduction 6](#_Toc31885866)

[Benefits of Using the Azure Blob Service for SQL Server Backups 7](#_Toc31885867)

[Intended Audience 8](#_Toc31885868)

[Azure Blob Storage Service Components 8](#_Toc31885869)

[SQL Server Components 8](#_Toc31885870)

[Security 9](#_Toc31885871)

[Microsoft Azure Blob storage service 10](#_Toc31885872)

[Limitations 11](#_Toc31885873)

[Create storage account 12](#_Toc31885874)

[Practice 1-1 : Create storage account and container(Portal) 13](#_Toc31885875)

[Practice 1-2 : Create storage account (PowerShell) 15](#_Toc31885876)

[Configure Azure Storage firewalls and virtual networks 16](#_Toc31885877)

[Change the default network access rule 17](#_Toc31885878)

[Manage storage account access keys 18](#_Toc31885879)

[Protect your access keys 18](#_Toc31885880)

[View access keys and connection string 18](#_Toc31885881)

[View access keys by PowerShell 19](#_Toc31885882)

[Use Azure Key Vault to manage your access keys 19](#_Toc31885883)

[Manually rotate access keys 19](#_Toc31885884)

[Rotate access keys using PowerShell 20](#_Toc31885885)

[Grant access to Azure Storage resources using shared access signatures (SAS) 21](#_Toc31885886)

[Types of shared access signatures 21](#_Toc31885887)

[How a shared access signature works 22](#_Toc31885888)

[1. SAS signature 22](#_Toc31885889)

[2. SAS token 22](#_Toc31885890)

[When to use a shared access signature 23](#_Toc31885891)

[Best practices when using SAS 24](#_Toc31885892)

[Practice 2-1 Create SAS Token using portal 26](#_Toc31885893)

[Practice 2-2 Create SAS Token using PowerShell 27](#_Toc31885894)

[practice 3 Create SQL credential using SAS or Access Keys 28](#_Toc31885895)

[SQL Server Backup 29](#_Toc31885896)

[Practice 2-1 : Backup database to Azure Blob 30](#_Toc31885897)

[Practice 2-2 : Backup database to Azure Blob using PowerShell 32](#_Toc31885898)

[Blob Cleanup 34](#_Toc31885899)

[Blob Container usage 35](#_Toc31885900)

Introduction

Starting with SQL Server 2012 SP1 CU2, you can now write SQL Server backups directly to the Azure Blob storage service. You can use this functionality to back up to and restore from the Azure Blob service with an on-premises SQL Server database or a SQL Server database in an Azure virtual machine. Backup to cloud offers benefits of availability, limitless geo-replicated off-site storage, and ease of migration of data to and from the cloud. You can issue BACKUP or RESTORE statements by using Transact-SQL, or SMO, or PowerShell command.

SQL Server 2016 introduces new capabilities; you can use file-snapshot backup to perform nearly instantaneous backups and incredibly quick restores.

Starting in SQL Server 2014 (12.x), SQL Server has the ability to **encrypt the data** while creating a backup. By specifying the encryption algorithm and the encryptor (a Certificate or Asymmetric Key) when creating a backup, you can create an encrypted backup file. All storage destinations: on-premises and Window Azure storage are supported. In addition, encryption options can be configured for SQL Server Managed Backup to Microsoft Azure operations, a new feature introduced in SQL Server 2014 (12.x).

This topic explains why you might choose to use Azure storage for SQL backups and then describes the components involved. You can use the resources provided at the end of the article to access walkthroughs and additional information to start using this service with your SQL Server backups.

Benefits of Using the Azure Blob Service for SQL Server Backups

There are several challenges that you face when backing up SQL Server. These challenges include storage management, risk of storage failure, access to off-site storage, and hardware configuration. Many of these challenges are addressed by using the Azure Blob store service for SQL Server backups. Consider the following benefits:

**Ease of use**: Storing your backups in Azure blobs can be a convenient, flexible, and easy to access off-site option. Creating off-site storage for your SQL Server backups can be as easy as modifying your existing scripts/jobs to use the BACKUP TO URL syntax. Off-site storage should typically be far enough from the production database location to prevent a single disaster that might impact both the off-site and production database locations. By choosing to geo-replicate your Azure blobs, you have an extra layer of protection in the event of a disaster that could affect the whole region.

**Added security**: By adding Encryption option to backup operation, additional level of protection of backup files are enabled.

**Backup archive**: The Azure Blob Storage service offers a better alternative to the often used tape option to archive backups. Tape storage might require physical transportation to an off-site facility and measures to protect the media. Storing your backups in Azure Blob Storage provides an instant, highly available, and a durable archiving option.

**Managed hardware**: There is no overhead of hardware management with Azure services. Azure services manage the hardware and provide geo-replication for redundancy and protection against hardware failures.

**Unlimited storage**: By enabling a direct backup to Azure blobs, you have access to virtually unlimited storage. Alternatively, backing up to an Azure virtual machine disk has limits based on machine size. There is a limit to the number of disks you can attach to an Azure virtual machine for backups. This limit is 16 disks for an extra large instance and fewer for smaller instances.

**Backup availability**: Backups stored in Azure blobs are available from anywhere and at any time and can easily be accessed for restores to either an on-premises SQL Server or another SQL Server running in an Azure Virtual Machine, without the need for database attach/detach or downloading and attaching the VHD.

**Cost**: Pay only for the service that is used. Can be cost-effective as an off-site and backup archive option. See the Azure pricing calculator, and the Azure Pricing article for more information.

**Storage snapshots**: When database files are stored in an Azure blob and you are using SQL Server 2016, you can use file-snapshot backup to perform nearly instantaneous backups and incredibly quick restores.

For more details, see [SQL Server Backup and Restore with Azure Blob Storage Service](https://go.microsoft.com/fwlink/?LinkId=271617)

following sections introduce the Azure Blob storage service, including the required SQL Server components. It is important to understand the components and their interaction to successfully use backup and restore from the Azure Blob storage service.

## 

## Intended Audience

This document is intended for architects, managers, server engineers, developers, and database administrators involved with the planning, deployment, management and operation of the SQL Server backup operations and disaster recovery solutions. This document assumes a working knowledge of SQL Server database management concepts.

## Azure Blob Storage Service Components

The following Azure components are used when backing up to the Azure Blob storage service.

| **Component** | **Description** |
| --- | --- |
| **Storage Account** | The storage account is the starting point for all storage services. To access an Azure Blob Storage service, first create an Azure Storage account. For more information about Azure Blob storage service, see [How to use the Azure Blob Storage Service](https://azure.microsoft.com/develop/net/how-to-guides/blob-storage/) |
| **Container** | A container provides a grouping of a set of blobs, and can store an unlimited number of Blobs. To write a SQL Server backup to an Azure Blob service, you must have at least the root container created. |
| **Blob** | A file of any type and size. Blobs are addressable using the following URL format: **https://[storage account].blob.core.windows.net/[container]/[blob]**. For more information about page Blobs, see [Understanding Block and Page Blobs](https://msdn.microsoft.com/library/azure/ee691964.aspx) |

## SQL Server Components

The following SQL Server components are used when backing up to the Azure Blob storage service.

| **Component** | **Description** |
| --- | --- |
| **URL** | A URL specifies a Uniform Resource Identifier (URI) to a unique backup file. The URL is used to provide the location and name of the SQL Server backup file. The URL must point to an actual blob, not just a container. If the blob does not exist, it is created. If an existing blob is specified, BACKUP fails, unless the > WITH FORMAT option is specified. The following is an example of the URL you would specify in the BACKUP command: **http[s]://[storageaccount].blob.core.windows.net/[container]/[FILENAME.bak]**. HTTPS is recommended but not required. |
| **Credential** | The information that is required to connect and authenticate to Azure Blob storage service is stored as a Credential. In order for SQL Server to write backups to an Azure Blob or restore from it, a SQL Server credential must be created. For more information, see [SQL Server Credential](https://msdn.microsoft.com/library/ms189522.aspx). |

## Security

The following are security considerations and requirements when backing up to or restoring from the Microsoft Azure Blob storage service.

When creating a container for the Microsoft Azure Blob storage service, we recommend that you set the access to private. Setting the access to private restricts the access to users or accounts able to provide the necessary information to authenticate to the Azure account.

***Important :*** *SQL Server requires that either an Azure account name and access key authentication or a Shared Access Signature and access token be stored in a SQL Server Credential. This information is used to authenticate to the Azure account when performing backup or restore operations.*

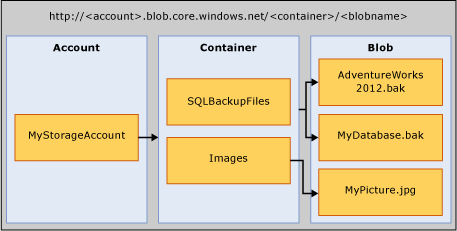
The user account that is used to issue BACKUP or RESTORE commands should be in the **db\_backup operator** database role with **Alter any credential** permissions.

Microsoft Azure Blob storage service

**Storage Account**: The storage account is the starting point for all storage services. To access the Microsoft Azure Blob storage service, first create an Azure storage account. For more information, see Create a Storage Account

**Container**: A container provides a grouping of a set of blobs, and can store an unlimited number of blobs. To write a SQL Server backup to the Microsoft Azure Blob storage service, you must have at least the root container created. You can generate a Shared Access Signature token on a container and grant access to objects on a specific container only.

**Blob**: A file of any type and size. There are two types of blobs that can be stored in the Microsoft Azure Blob storage service: block and page blobs. SQL Server backup can use either blob type depending upon the Transact-SQL syntax used. Blobs are addressable using the following URL format: https://<storage account>.blob.core.windows.net/<container>/<blob>. For more information about the Microsoft Azure Blob storage service, see How to use the Blob Storage from .NET. For more information about page and block blobs, see Understanding Block and Page Blobs.



**Azure Snapshot**: A snapshot of an Azure blob taken at a point in time. For more information, see Creating a Snapshot of a Blob. SQL Server backup now supports Azure snapshot backups of database files stored in the Microsoft Azure Blob storage service. For more information, see File-Snapshot Backups for Database Files in Azure.

## Limitations

* Backup to premium storage is not supported.
* SQL Server limits the maximum backup size supported using a page blob to 1 TB. The maximum backup size supported using block blobs is limited to approximately 200 GB (50,000 blocks \* 4MB MAXTRANSFERSIZE). Block blobs support striping to support substantially larger backup sizes.
* You can issue backup or restore statements by using TSQL, SMO, PowerShell cmdlets, SQL Server Management Studio Backup or Restore wizard.
* Creating a logical device name is not supported. So adding URL as a backup device using sp\_dumpdevice or through SQL Server Management Studio is not supported.
* Appending to existing backup blobs is not supported. Backups to an existing blob can only be overwritten by using the **WITH FORMAT** option. However, when using file-snapshot backups (using the **WITH FILE\_SNAPSHOT** argument), the **WITH FORMAT** argument is not permitted to avoid leaving orphaned file-snapshots that were created with the original file-snapshot backup.
* Backup to multiple blobs in a single backup operation is only supported using block blobs and using a Shared Access Signature (SAS) token rather than the storage account key for the SQL Credential.
* Specifying **BLOCKSIZE** is not supported for page blobs. (but block blob is supported)
* Specifying **MAXTRANSFERSIZE** is not supported page blobs. (but block blob is supported)
* Specifying backupset options - **RETAINDAYS** and **EXPIREDATE** are not supported.
* SQL Server has a maximum limit of 259 characters for a backup device name. The BACKUP TO URL consumes 36 characters for the required elements used to specify the URL - 'https://.blob.core.windows.net//.bak', leaving 223 characters for account, container, and blob names put together.

Please note: SQL Server integration support for the Microsoft Azure Blob storage service began as a SQL Server 2012 Service Pack 1 CU2 enhancement, and has been enhanced further with SQL Server 2014 and SQL Server 2016. For an overview of the functionality and benefits of using this feature, see [SQL Server Data Files in Microsoft Azure](https://docs.microsoft.com/en-us/sql/relational-databases/databases/sql-server-data-files-in-microsoft-azure?view=sql-server-ver15), but main benefit is support for backups larger than 1TB. For a live demo, see [Demo of Point in Time Restore](https://channel9.msdn.com/Blogs/Windows-Azure/File-Snapshot-Backups-Demo).

To find out more on Azure Blob storage service with SQL server read [this article](https://docs.microsoft.com/en-us/sql/relational-databases/tutorial-use-azure-blob-storage-service-with-sql-server-2016?view=sql-server-ver15)

Create storage account

An Azure storage account contains all of your Azure Storage data objects: blobs, files, queues, tables, and disks. The storage account provides a unique namespace for your Azure Storage data that is accessible from anywhere in the world over HTTP or HTTPS. Data in your Azure storage account is durable and highly available, secure, and massively scalable.

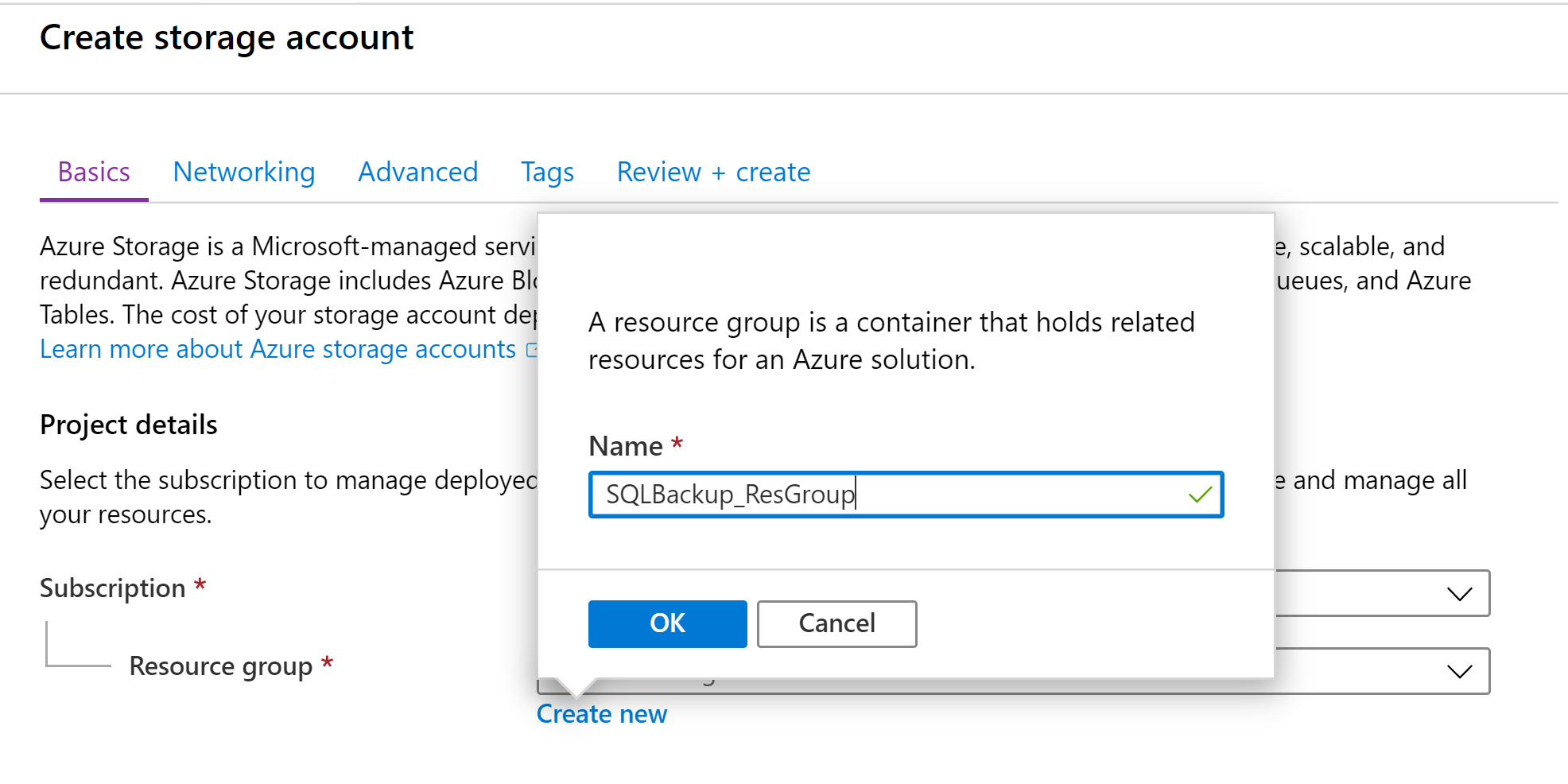
Every storage account must belong to an Azure resource group. A resource group is a logical container for grouping your Azure services. When you create a storage account, you have the option to either create a new resource group, or use an existing resource group. This article shows how to create a new resource group.

A general-purpose v2 storage account provides access to all of the Azure Storage services: blobs, files, queues, tables, and disks. The steps outlined here create a general-purpose v2 storage account, but the steps to create any type of storage account are similar.

### Practice 1-1 : Create storage account and container(Portal)

To create a general-purpose v2 storage account in the Azure portal, follow these steps:

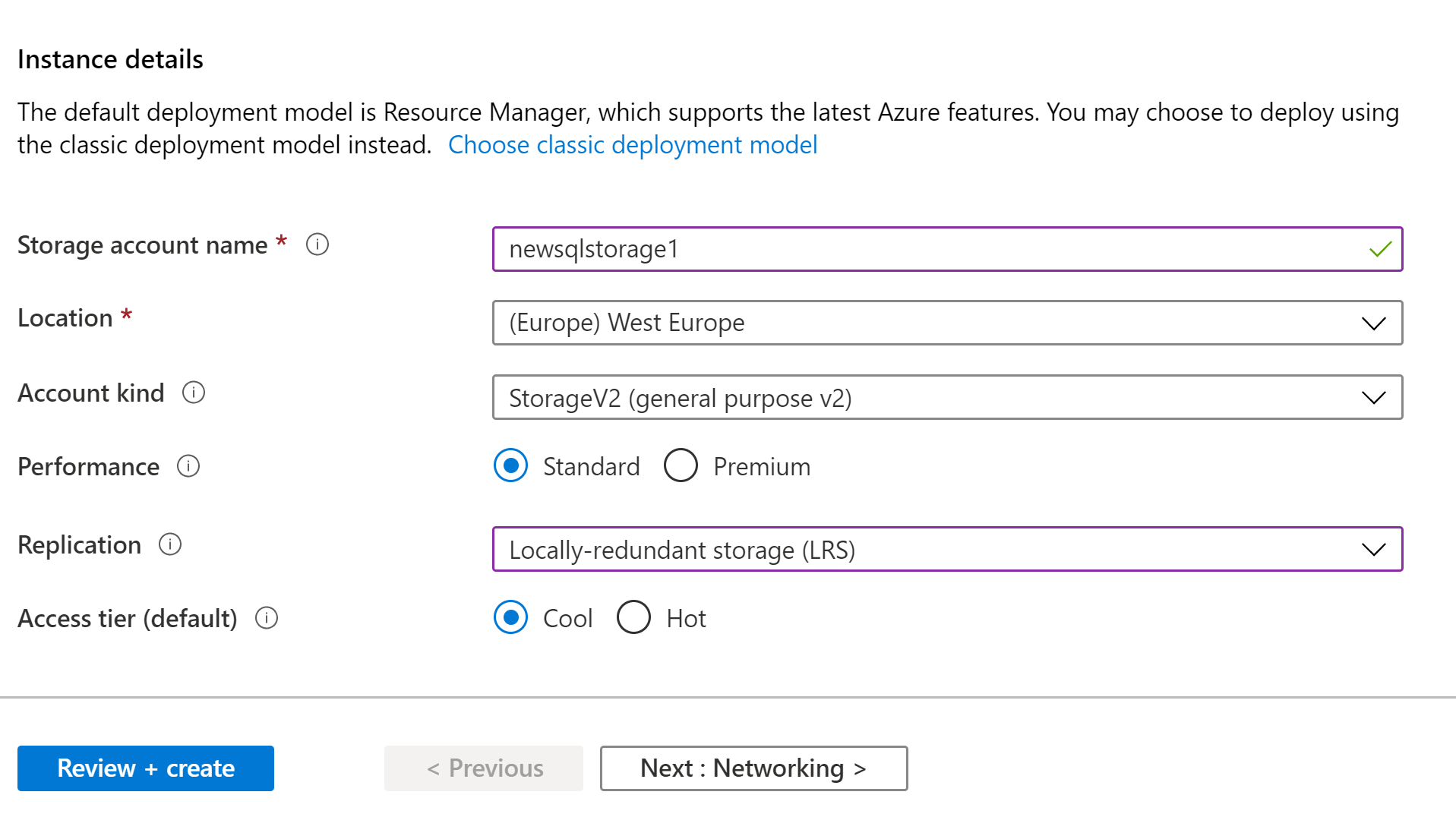
1. On the Azure portal menu, select **All services**. In the list of resources, type **Storage Accounts**. As you begin typing, the list filters based on your input. Select **Storage Accounts.**
2. On the **Storage Accounts** window that appears, choose **Add**.
3. Select the subscription in which to create the storage account.
4. Under the **Resource group** field, select **Create new**. Enter a name for your new resource group, as shown in the following image.



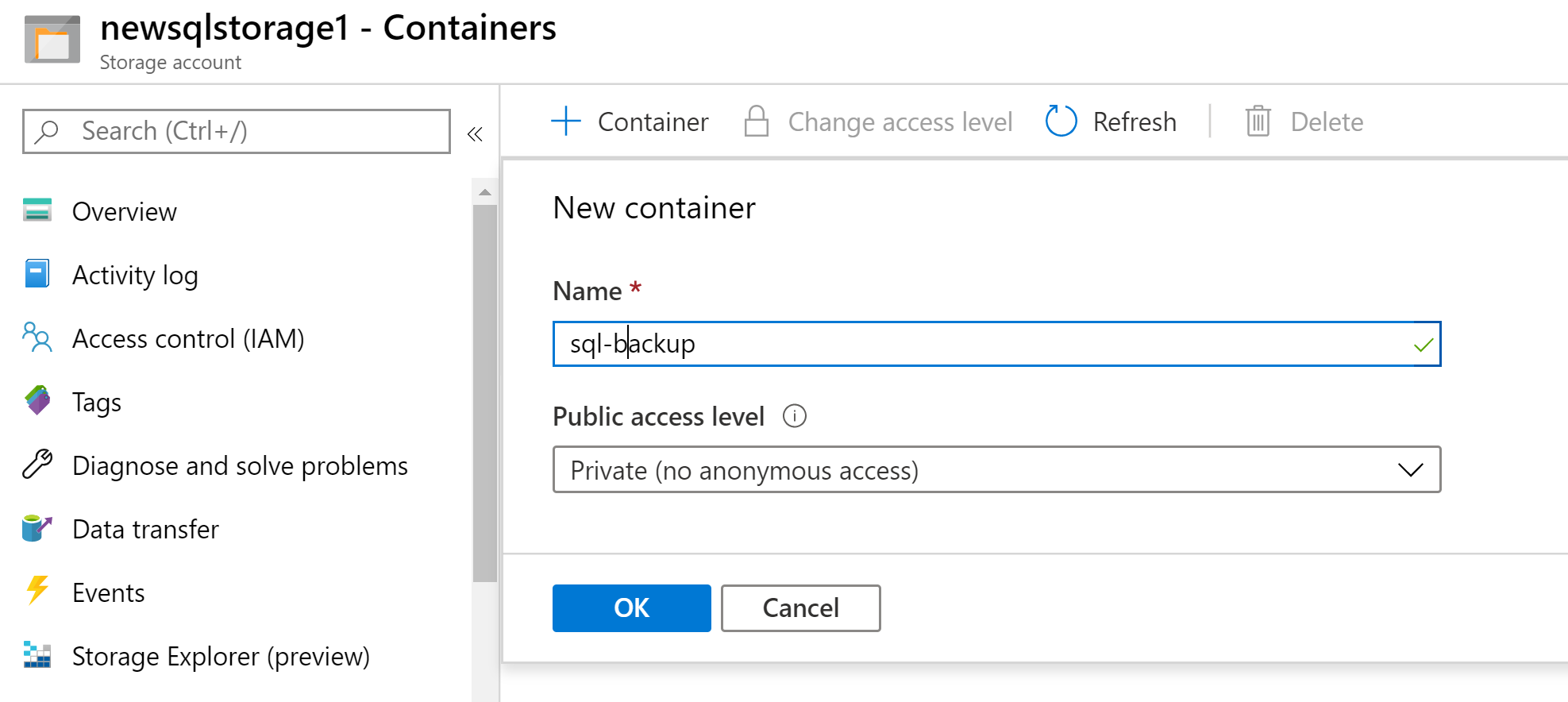
1. Next, enter a name for your storage account. The name you choose must be unique across Azure. The name also must be between 3 and 24 characters in length, and can include numbers and lowercase letters only.
2. Select a location for your storage account, or use the default location.
3. Leave these fields set to their default values:

|  |  |
| --- | --- |
| **Deployment model** | **Resource Manager** |
| Performance | Standard |
| Account kind | StorageV2 (general-purpose v2) |
| Replication | Read-access geo-redundant storage (RA-GRS) |
| Access tier | Cool |

In case you need to create block blob, change Performance to Premium and kind to BlockBlob



1. Select Review + Create to review your storage account settings and create the account.
2. Select Create.
3. When Storage account is created click on “Go to Resource”
4. Click Containers, and then choose Add (+)
5. Next, add name for your container , and Public access level



### Practice 1-2 : Create storage account (PowerShell)

First, create a new resource group with PowerShell using the New-AzResourceGroup command:

# put resource group in a variable so you can use the same group name going forward,

# without hard-coding it repeatedly

$resourceGroup = "storage-resource-group"

New-AzResourceGroup -Name $resourceGroup -Location $location

If you're not sure which region to specify for the -Location parameter, you can retrieve a list of supported regions for your subscription with the Get-AzLocation command:

Get-AzLocation | select Location

$location = "northeurope"

Next, create a general-purpose v2 storage account with read-access geo-redundant storage (RA-GRS) by using the New-AzStorageAccount command. Remember that the name of your storage account must be unique across Azure, so replace the placeholder value in brackets with your own unique value:

New-AzStorageAccount -ResourceGroupName $resourceGroup `

-Name <account-name> `

-Location $location `

-SkuName Standard\_RAGRS `

-Kind StorageV2

To create a general-purpose v2 storage account with a different replication option, substitute the desired value in the table below for the SkuName parameter.

Script with all steps would be :

$resourcegroup = "SQLBackup\_ResGroup"

$storageaccount = "newsqlstorage1"

$location = "North Europe"

$containerName = "sql-backup"

#check if Az Module is installed and

If (!(Get-module Az)) {

Install-Module -Name Az -AllowClobber}

#Connect-AzAccount -SubscriptionId $SubscriptionID

Connect-AzAccount

#Create new Resource group

New-AzResourceGroup -Name $resourceGroup -Location $location

#Create new BLOCK blob storage

$storageAccount=New-AzStorageAccount -ResourceGroupName $resourcegroup -Name $storageaccount -Location $location -Kind "BlobStorage" -SkuName "Standard\_LRS" -AccessTier Cool

<# if Block blob is required

$storageAccount = New-AzStorageAccount -ResourceGroupName $resourcegroup -Name "backup42blob" -Location $location -Kind "BlockBlobStorage" -SkuName "Premium\_LRS" -AccessTier Cool

#>

#Create Container

New-AzStorageContainer -Name $containerName -Context $storageAccount.Context -Permission blob

## Configure Azure Storage firewalls and virtual networks

Azure Storage provides a layered security model. This model enables you to secure and control the level of access to your storage accounts that your applications and enterprise environments demand, based on the type and subset of networks used. When network rules are configured, only applications requesting data over the specified set of networks can access a storage account. You can limit access to your storage account to requests originating from specified IP addresses, IP ranges or from a list of subnets in an Azure Virtual Network (VNet).

Storage accounts have a public endpoint that is accessible through the internet. You can also create Private Endpoints for your storage account, which assigns a private IP address from your VNet to the storage account, and secures all traffic between your VNet and the storage account over a private link. The Azure storage firewall provides access control access for the public endpoint of your storage account. You can also use the firewall to block all access through the public endpoint when using private endpoints. Your storage firewall configuration also enables select trusted Azure platform services to access the storage account securely.

An application that accesses a storage account when network rules are in effect still requires proper authorization for the request. Authorization is supported with Azure Active Directory (Azure AD) credentials for blobs and queues, with a valid account access key, or with an SAS token.

Change the default network access rule

By default, storage accounts accept connections from clients on any network. To limit access to selected networks, you must first change the default action.

#### Azure portal

1. Go to the storage account you want to secure.
2. Click on the settings menu called **Firewalls and virtual networks**.
3. To deny access by default, choose to allow access from **Selected networks**. To allow traffic from all networks, choose to allow access from **All networks**.
4. Click **Save** to apply your changes.

#### PowerShell

#Display the status of the default rule for the storage account.

(Get-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -AccountName "mystorageaccount").DefaultAction

#Set the default rule to deny network access by default.

Update-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -Name "mystorageaccount" -DefaultAction Deny

#Set the default rule to allow network access by default.

Update-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -Name "mystorageaccount" -DefaultAction Allow

Manage storage account access keys

When you create a storage account, Azure generates two 512-bit storage account access keys. These keys can be used to authorize access to data in your storage account via Shared Key authorization.

Microsoft recommends that you use Azure Key Vault to manage your access keys, and that you regularly rotate and regenerate your keys. Using Azure Key Vault makes it easy to rotate your keys without interruption to your applications. You can also manually rotate your keys.

## Protect your access keys

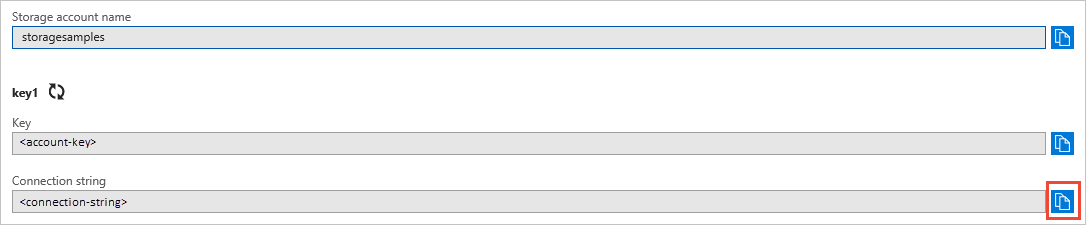
Your storage account access keys are similar to a root password for your storage account. Always be careful to protect your access keys. Use Azure Key Vault to manage and rotate your keys securely. Avoid distributing access keys to other users, hard-coding them, or saving them anywhere in plain text that is accessible to others. Rotate your keys if you believe they may have been compromised.

If possible, use Azure Active Directory (Azure AD) to authorize requests to Blob and Queue storage instead of Shared Key. Azure AD provides superior security and ease of use over Shared Key. For more information about authorizing access to data with Azure AD, see [Authorize access to Azure blobs and queues using Azure Active Directory](https://docs.microsoft.com/en-us/azure/storage/common/storage-auth-aad).

## View access keys and connection string

To view and copy your storage account access keys or connection string from the Azure portal:

1. Navigate to the [Azure portal](https://portal.azure.com/).
2. Locate your storage account.
3. In the **Settings** section of the storage account overview, select **Access keys**. Your account access keys appear, as well as the complete connection string for each key.
4. Find the **Key** value under **key1**, and click the **Copy** button to copy the account key.
5. Alternately, you can copy the entire connection string. Find the **Connection string** value under **key1**, and click the **Copy** button to copy the connection string.



You can use either key to access Azure Storage, but in general it's a good practice to use the first key, and reserve the use of the second key for when you are rotating keys.

## View access keys by PowerShell

To view and copy your storage account access keys or connection string PowerShell ISE:

$resourcegroup = "SQLBackup\_ResGroup"

$storageaccountName = "newsqlstorage1"

# retrieve key #2 and display it

$storageAccountKey = (Get-AzStorageAccountKey -ResourceGroupName $resourceGroup -Name $storageAccountName ).Value[1]

Write-Host "OLD storage account key 2 = " $storageAccountKey -ForegroundColor Yellow

## Use Azure Key Vault to manage your access keys

Microsoft recommends using Azure Key Vault to manage and rotate your access keys. Your application can securely access your keys in Key Vault, so that you can avoid storing them with your application code. For more information about using Key Vault for key management, see the following articles:

[Manage storage account keys with Azure Key Vault and PowerShell](https://docs.microsoft.com/en-us/azure/key-vault/key-vault-overview-storage-keys-powershell)

[Manage storage account keys with Azure Key Vault and the Azure CLI](https://docs.microsoft.com/en-us/azure/key-vault/key-vault-ovw-storage-keys)

## Manually rotate access keys

Microsoft recommends that you rotate your access keys periodically to help keep your storage account secure. If possible, use Azure Key Vault to manage your access keys. If you are not using Key Vault, you will need to rotate your keys manually.

Two access keys are assigned so that you can rotate your keys. Having two keys ensures that your application maintains access to Azure Storage throughout the process.

**Warning:** Regenerating your access keys can affect any applications or Azure services that are dependent on the storage account key. Any clients that use the account key to access the storage account must be updated to use the new key, including media services, cloud, desktop and mobile applications, and graphical user interface applications for Azure Storage, such as Azure Storage Explorer.

Follow this process to rotate your storage account keys:

1. Update the connection strings in your application code to use the secondary key.
2. Regenerate the primary access key for your storage account. On the **Access Keys** blade in the Azure portal, click **Regenerate Key1**, and then click **Yes** to confirm that you want to generate a new key.
3. Update the connection strings in your code to reference the new primary access key.
4. Regenerate the secondary access key in the same manner.

Note : Microsoft recommends using only one of the keys in all of your applications at the same time. If you use Key 1 in some places and Key 2 in others, you will not be able to rotate your keys without some application losing access.

## Rotate access keys using PowerShell

To rotate your storage account access keys or connection string PowerShell ISE:

Example is rotating Key2

$resourcegroup = "SQLBackup\_ResGroup"

$storageaccountName = "newsqlstorage1"

# retrieve key #2 and display it

$storageAccountKey = (Get-AzStorageAccountKey -ResourceGroupName $resourceGroup -Name $storageAccountName ).Value[1]

Write-Host "OLD storage account key 2 = " $storageAccountKey -ForegroundColor Yellow

# re-generate the key

New-AzStorageAccountKey -ResourceGroupName $resourceGroup -Name $storageAccountName -KeyName key2

# retrieve it again and display it

$storageAccountKey = (Get-AzStorageAccountKey -ResourceGroupName $resourceGroup -Name $storageAccountName).Value[1]

Write-Host "NEW storage account key 2 = " $storageAccountKey -ForegroundColor Red

Grant access to Azure Storage resources using shared access signatures (SAS)

A shared access signature (SAS) provides secure delegated access to resources in your storage account without compromising the security of your data. With a SAS, you have granular control over how a client can access your data. You can control what resources the client may access, what permissions they have on those resources, and how long the SAS is valid, among other parameters.

## Types of shared access signatures

Azure Storage supports three types of shared access signatures:

* **User delegation SAS.** A user delegation SAS is secured with Azure Active Directory (Azure AD) credentials and also by the permissions specified for the SAS. A user delegation SAS applies to Blob storage only.

For more information about the user delegation SAS, see [Create a user delegation SAS (REST API)](https://docs.microsoft.com/en-us/rest/api/storageservices/create-user-delegation-sas).

* **Service SAS.** A service SAS is secured with the storage account key. A service SAS delegates access to a resource in only one of the Azure Storage services: Blob storage, Queue storage, Table storage, or Azure Files.

For more information about the service SAS, see [Create a service SAS (REST API)](https://docs.microsoft.com/en-us/rest/api/storageservices/create-service-sas).

* **Account SAS.** An account SAS is secured with the storage account key. An account SAS delegates access to resources in one or more of the storage services. All of the operations available via a service or user delegation SAS are also available via an account SAS. Additionally, with the account SAS, you can delegate access to operations that apply at the level of the service, such as **Get/Set Service Properties** and **Get Service Stats** operations. You can also delegate access to read, write, and delete operations on blob containers, tables, queues, and file shares that are not permitted with a service SAS.

For more information about the account SAS, [Create an account SAS (REST API)](https://docs.microsoft.com/en-us/rest/api/storageservices/create-account-sas).

A shared access signature can take one of two forms:

* **Ad hoc SAS:** When you create an ad hoc SAS, the start time, expiry time, and permissions for the SAS are all specified in the SAS URI (or implied, if start time is omitted). Any type of SAS can be an ad hoc SAS.
* **Service SAS with stored access policy:** A stored access policy is defined on a resource container, which can be a blob container, table, queue, or file share. The stored access policy can be used to manage constraints for one or more service shared access signatures. When you associate a service SAS with a stored access policy, the SAS inherits the constraints—the start time, expiry time, and permissions—defined for the stored access policy.

## How a shared access signature works

A shared access signature is a signed URI that points to one or more storage resources and includes a token that contains a special set of query parameters. The token indicates how the resources may be accessed by the client. One of the query parameters, the signature, is constructed from the SAS parameters and signed with the key that was used to create the SAS. This signature is used by Azure Storage to authorize access to the storage resource.

1. SAS signature

You can sign a SAS in one of two ways:

* With a user delegation key that was created using Azure Active Directory (Azure AD) credentials. A user delegation SAS is signed with the user delegation key.

To get the user delegation key and create the SAS, an Azure AD security principal must be assigned a role-based access control (RBAC) role that includes the **Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey** action. For detailed information about RBAC roles with permissions to get the user delegation key, see [Create a user delegation SAS (REST API)](https://docs.microsoft.com/en-us/rest/api/storageservices/create-user-delegation-sas).

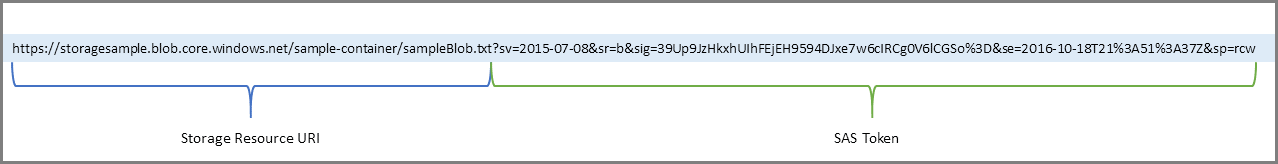
* With the storage account key. Both a service SAS and an account SAS are signed with the storage account key. To create a SAS that is signed with the account key, an application must have access to the account key.

1. SAS token

The SAS token is a string that you generate on the client side, for example by using one of the Azure Storage client libraries. The SAS token is not tracked by Azure Storage in any way. You can create an unlimited number of SAS tokens on the client side. After you create a SAS, you can distribute it to client applications that require access to resources in your storage account.

When a client application provides a SAS URI to Azure Storage as part of a request, the service checks the SAS parameters and signature to verify that it is valid for authorizing the request. If the service verifies that the signature is valid, then the request is authorized. Otherwise, the request is declined with error code 403 (Forbidden).

Here's an example of a service SAS URI, showing the resource URI and the SAS token:

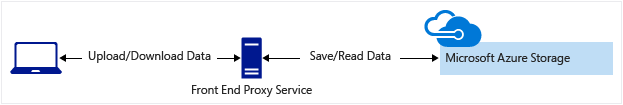


## When to use a shared access signature

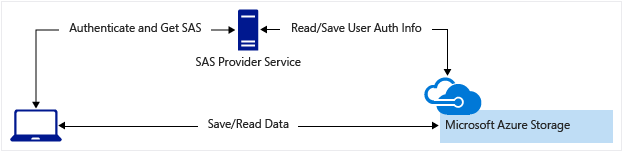
Use a SAS when you want to provide secure access to resources in your storage account to any client who does not otherwise have permissions to those resources.

A common scenario where a SAS is useful is a service where users read and write their own data to your storage account. In a scenario where a storage account stores user data, there are two typical design patterns:

1. Clients upload and download data via a front-end proxy service, which performs authentication. This front-end proxy service has the advantage of allowing validation of business rules, but for large amounts of data or high-volume transactions, creating a service that can scale to match demand may be expensive or difficult.



1. A lightweight service authenticates the client as needed and then generates a SAS. Once the client application receives the SAS, they can access storage account resources directly with the permissions defined by the SAS and for the interval allowed by the SAS. The SAS mitigates the need for routing all data through the front-end proxy service.



Many real-world services may use a hybrid of these two approaches. For example, some data might be processed and validated via the front-end proxy, while other data is saved and/or read directly using SAS.

Additionally, a SAS is required to authorize access to the source object in a copy operation in certain scenarios:

* When you copy a blob to another blob that resides in a different storage account, you must use a SAS to authorize access to the source blob. You can optionally use a SAS to authorize access to the destination blob as well.
* When you copy a file to another file that resides in a different storage account, you must use a SAS to authorize access to the source file. You can optionally use a SAS to authorize access to the destination file as well.
* When you copy a blob to a file, or a file to a blob, you must use a SAS to authorize access to the source object, even if the source and destination objects reside within the same storage account.

Best practices when using SAS

When you use shared access signatures in your applications, you need to be aware of two potential risks:

* If a SAS is leaked, it can be used by anyone who obtains it, which can potentially compromise your storage account.
* If a SAS provided to a client application expires and the application is unable to retrieve a new SAS from your service, then the application's functionality may be hindered.

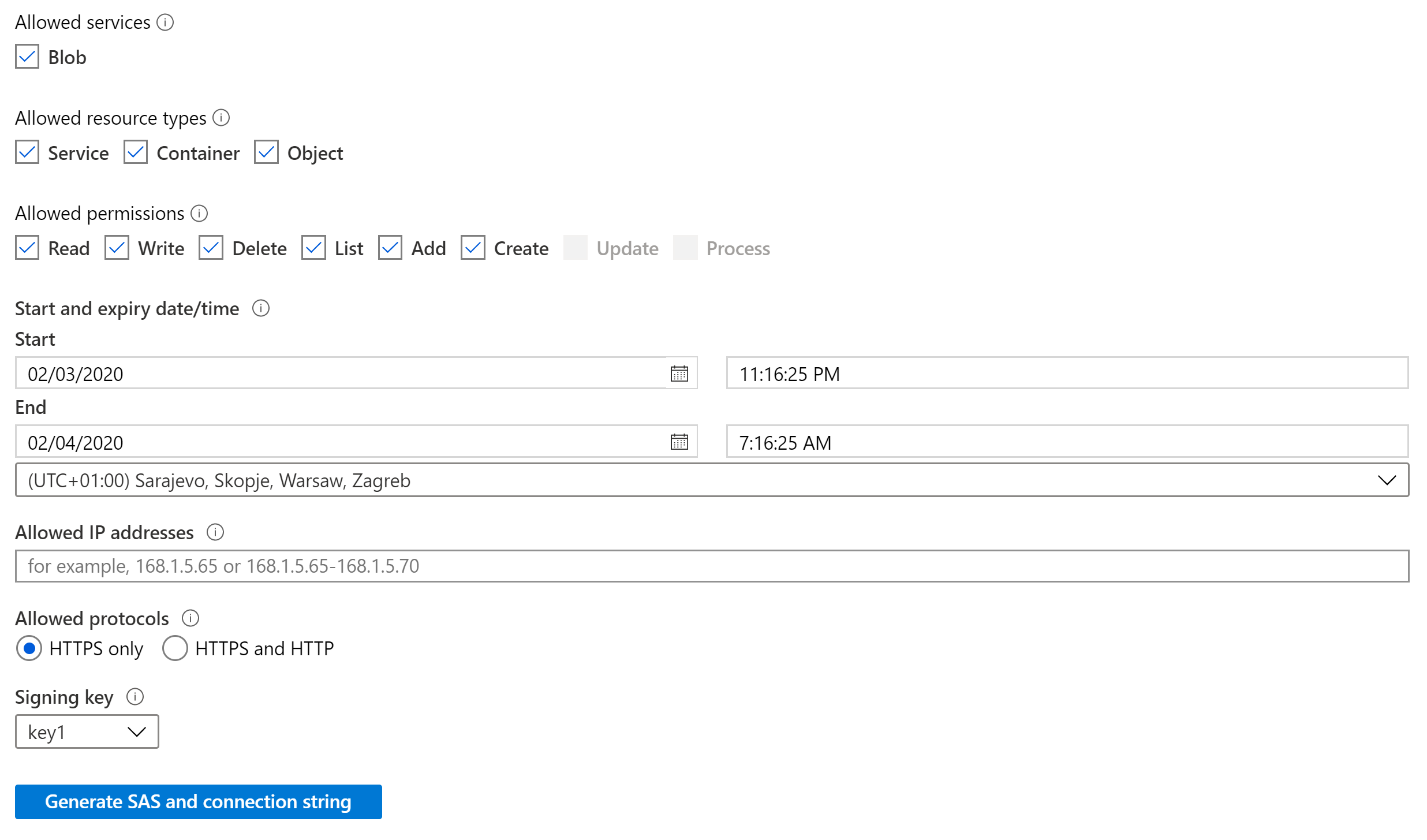
The following recommendations for using shared access signatures can help mitigate these risks:

* **Always use HTTPS** to create or distribute a SAS. If a SAS is passed over HTTP and intercepted, an attacker performing a man-in-the-middle attack is able to read the SAS and then use it just as the intended user could have, potentially compromising sensitive data or allowing for data corruption by the malicious user.
* **Use a user delegation SAS when possible.** A user delegation SAS provides superior security to a service SAS or an account SAS. A user delegation SAS is secured with Azure AD credentials, so that you do not need to store your account key with your code.
* **Have a revocation plan in place for a SAS.** Make sure you are prepared to respond if a SAS is compromised.
* **Define a stored access policy for a service SAS.** Stored access policies give you the option to revoke permissions for a service SAS without having to regenerate the storage account keys. Set the expiration on these very far in the future (or infinite) and make sure it's regularly updated to move it farther into the future.
* **Use near-term expiration times on an ad hoc SAS service SAS or account SAS.** In this way, even if a SAS is compromised, it's valid only for a short time. This practice is especially important if you cannot reference a stored access policy. Near-term expiration times also limit the amount of data that can be written to a blob by limiting the time available to upload to it.
* **Have clients automatically renew the SAS if necessary.** Clients should renew the SAS well before the expiration, in order to allow time for retries if the service providing the SAS is unavailable. If your SAS is meant to be used for a small number of immediate, short-lived operations that are expected to be completed within the expiration period, then this may be unnecessary as the SAS is not expected to be renewed. However, if you have client that is routinely making requests via SAS, then the possibility of expiration comes into play. The key consideration is to balance the need for the SAS to be short-lived (as previously stated) with the need to ensure that the client is requesting renewal early enough (to avoid disruption due to the SAS expiring prior to successful renewal).
* **Be careful with SAS start time.** If you set the start time for a SAS to **now**, then due to clock skew (differences in current time according to different machines), failures may be observed intermittently for the first few minutes. In general, set the start time to be at least 15 minutes in the past. Or, don't set it at all, which will make it valid immediately in all cases. The same generally applies to expiry time as well--remember that you may observe up to 15 minutes of clock skew in either direction on any request. For clients using a REST version prior to 2012-02-12, the maximum duration for a SAS that does not reference a stored access policy is 1 hour, and any policies specifying longer term than that will fail.
* **Be specific with the resource to be accessed.** A security best practice is to provide a user with the minimum required privileges. If a user only needs read access to a single entity, then grant them read access to that single entity, and not read/write/delete access to all entities. This also helps lessen the damage if a SAS is compromised because the SAS has less power in the hands of an attacker.
* **Understand that your account will be billed for any usage, including via a SAS.** If you provide write access to a blob, a user may choose to upload a 200 GB blob. If you've given them read access as well, they may choose to download it 10 times, incurring 2 TB in egress costs for you. Again, provide limited permissions to help mitigate the potential actions of malicious users. Use short-lived SAS to reduce this threat (but be mindful of clock skew on the end time).
* **Validate data written using a SAS.** When a client application writes data to your storage account, keep in mind that there can be problems with that data. If your application requires that data be validated or authorized before it is ready to use, you should perform this validation after the data is written and before it is used by your application. This practice also protects against corrupt or malicious data being written to your account, either by a user who properly acquired the SAS, or by a user exploiting a leaked SAS.
* **Know when not to use a SAS.** Sometimes the risks associated with a particular operation against your storage account outweigh the benefits of using a SAS. For such operations, create a middle-tier service that writes to your storage account after performing business rule validation, authentication, and auditing. Also, sometimes it's simpler to manage access in other ways. For example, if you want to make all blobs in a container publicly readable, you can make the container Public, rather than providing a SAS to every client for access.
* **Use Azure Monitor and Azure Storage logs to monitor your application.** You can use Azure Monitor and storage analytics logging to observe any spike in authorization failures due to an outage in your SAS provider service or to the inadvertent removal of a stored access policy. For more information, see [Azure Storage metrics in Azure Monitor](https://docs.microsoft.com/en-us/azure/storage/common/storage-metrics-in-azure-monitor?toc=%2fazure%2fstorage%2fblobs%2ftoc.json) and [Azure Storage Analytics logging](https://docs.microsoft.com/en-us/azure/storage/common/storage-analytics-logging?toc=%2fazure%2fstorage%2fblobs%2ftoc.json).

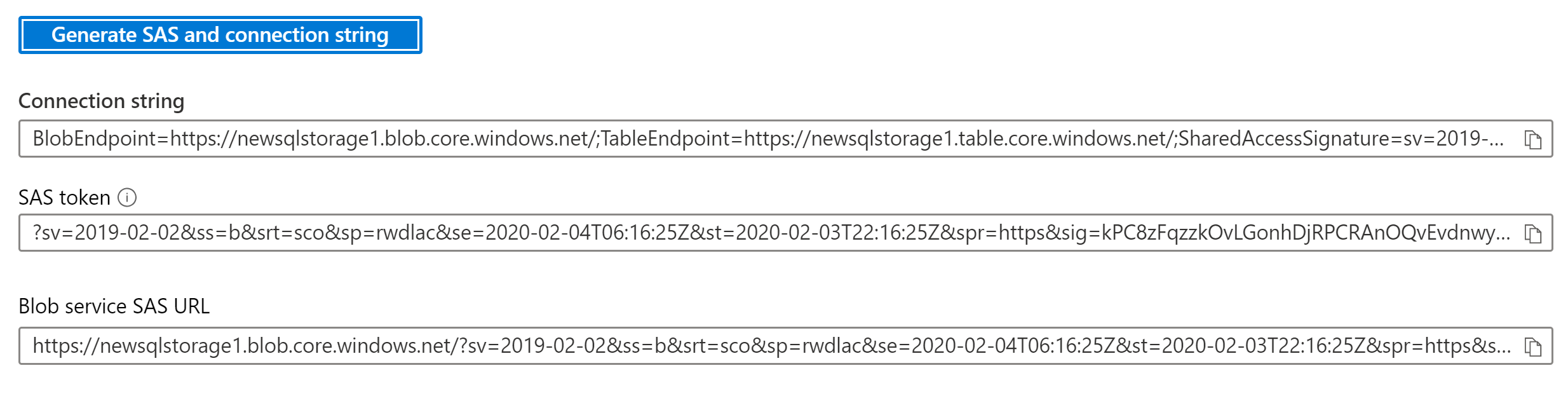
Practice 2-1 Create SAS Token using portal

To create a SAS Token for your storage account in the Azure portal, follow these steps:

1. On the Azure portal menu, browse tou your **Storage Account**
2. On the menu on the right side click “Shared Acces Signature”



1. Click Generate SAS and connection string



Practice 2-2 Create SAS Token using PowerShell

$resourcegroup = "SQLBackup\_ResGroup"

$storageaccountName = "newsqlstorage1"

$context = (Get-AzStorageAccount -ResourceGroupName $resourcegroup -AccountName $storageaccountName).context

$st=New-AzStorageAccountSASToken -Context $context -Service Blob -ResourceType Service,Container,Object -Permission racwdlu

write-host $st -ForegroundColor Yellow

practice 3 Create SQL credential using SAS or Access Keys

In SSMS connect to your instance and type commands

--CREATE Credential using SAS

IF NOT EXISTS

(SELECT \* FROM sys.credentials

WHERE name = 'https://<StorageAccesName>.blob.core.windows.net/<BlobName>')

CREATE CREDENTIAL [https://newsqlstorage1.blob.core.windows.net/sql-backup']

WITH IDENTITY = 'SHARED ACCESS SIGNATURE',

SECRET = '<secret>';

--please note ommiting first question mark in <secret>

--CREATE Credential using Access KEY

IF NOT EXISTS

(SELECT \* FROM sys.credentials

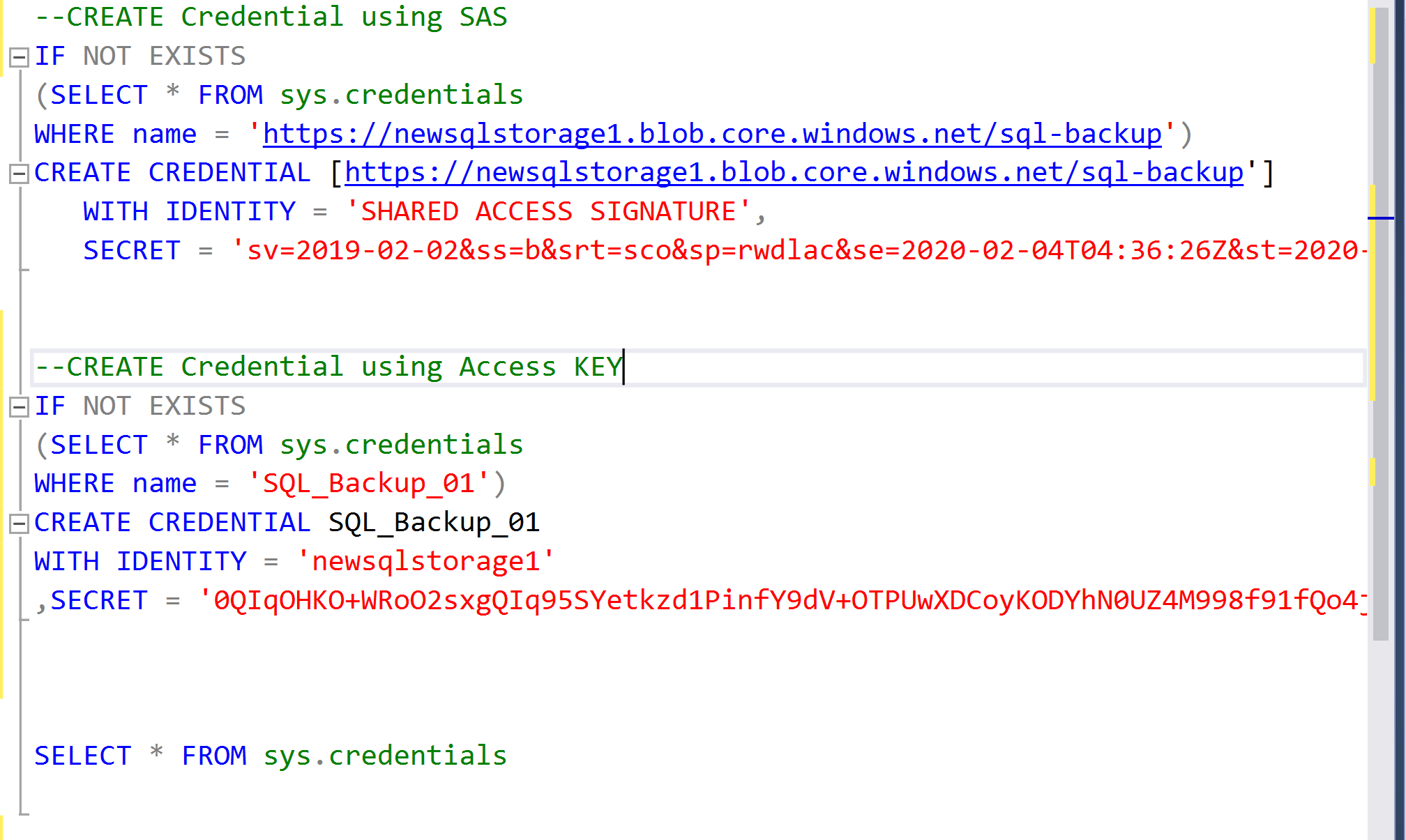
WHERE name = 'SQL\_Backup\_01')

CREATE CREDENTIAL SQL\_Backup\_01

WITH IDENTITY = '<StorageAccesName>'

,SECRET = 'AccesKey1';

SELECT \* FROM sys.credentials



SQL Server Backup

To complete this tutorial, you must be familiar with SQL Server backup and restore concepts and T-SQL syntax. To use this tutorial, you need an Azure storage account, SQL Server Management Studio (SSMS), access to an instance of SQL Server on-premises, access to an Azure virtual machine (VM) running SQL Server 2016, and an AdventureWorks2016 database. Additionally, the account used to issue the BACKUP and RESTORE commands should be in the **db\_backupoperator** database role with **alter any credential** permissions.

### Practice 2-1 : Backup database to Azure Blob

We will back up the WideWorldImporters database in your on-premises SQL Server 2016 instance to the Azure container that you created in Practice 1.

To back up a database to Blob storage, follow these steps:

1. Connect to SQL Server Management Studio.
2. Open a new query window and connect to the SQL Server 2016 instance of the database engine in your on-premises instance.
3. Copy and paste the following Transact-SQL script into the query window. Modify the URL appropriately for your storage account name and the container that you specified in section 1 and then execute this script.

-- To permit log backups, before the full database backup, modify the database to use the full recovery model.

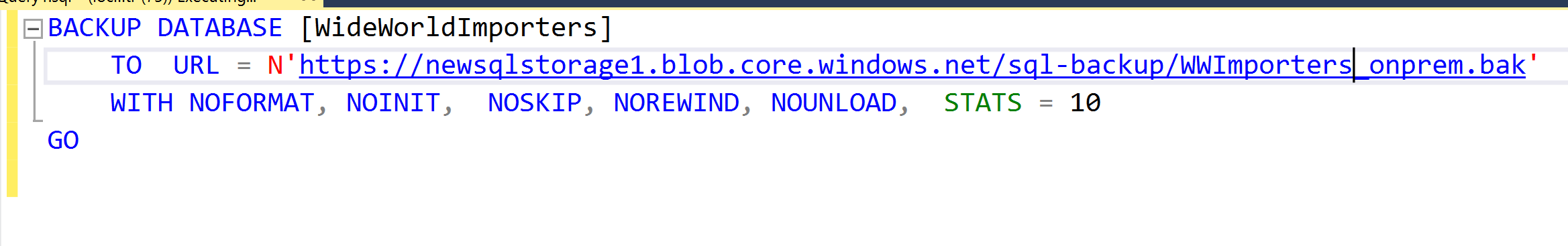
--ALTER DATABASE [WideWorldImporters] SET RECOVERY FULL;

USE master;

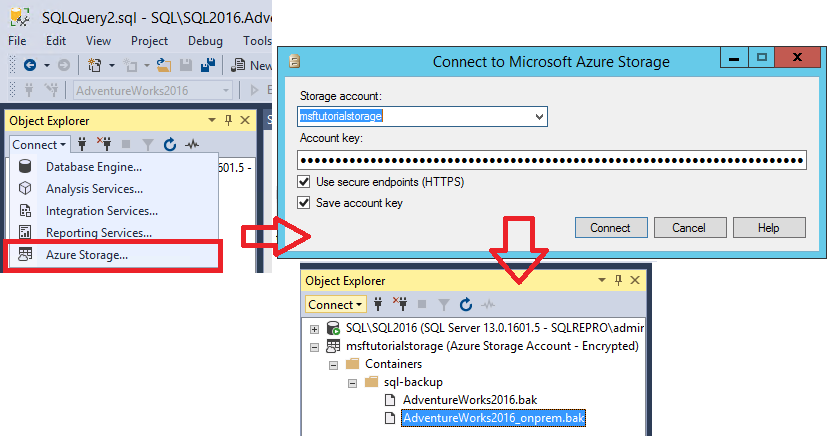
-- Back up the full AdventureWorks2016 database to the container that you created in section 1

BACKUP DATABASE [WideWorldImporters]

TO URL = 'https://<mystorageaccountname>.blob.core.windows.net/<mystorageaccountcontainername>/WideWorldImporters\_Full\_onprem.bak'



1. Open Object Explorer and connect to Azure storage using your storage account and account key.



Expand Containers, expand the container that you created in section 1 and verify that the backup from step 3 above appears in this container.

### Practice 2-2 : Backup database to Azure Blob using PowerShell

# Import the SqlServer module

#Import-Module SqlServer

Backup-SqlDatabase -ServerInstance "Computer\Instance" -Database "WideWorldImporters" -BackupFile "https://storageaccountname.blob.core.windows.net/containername/MainDB.bak" -SqlCredential "SQLCredentialName"

Practice 2-3 Backup multiple databases to Azure container with Encription

# Check if the SqlServer module is already installed, if not install it

$SQLModuleCheck = Get-Module -ListAvailable SqlServer

if ($SQLModuleCheck -eq $null)

{

write-host "SqlServer Module Not Found - Installing"

Set-PSRepository -Name PSGallery -InstallationPolicy Trusted

Install-Module -Name SqlServer –Scope AllUsers -Confirm:$false -AllowClobber

}

# Import the SqlServer module

$BackupType = 'Database' # Provide backup Type as 'Database' or 'Log'

Import-Module SqlServer

$SQLInstance = "localhost"

$LoggingFolder = "D:\SQL\_Azure\_Backup\_Logging\"

$DatabaseList = "master","model","WideWorldImporters" # Provide database names seperated by Comma "," e.b. "TestDB1, TestDB2"

$storageAccount = "blobtest666"

$blobContainer = "sql-backup"

$backupUrlContainer = "https://$storageAccount.blob.core.windows.net/$blobContainer/"

$credentialName = "<myCredential>"

#------------------

$BlobAccessKey = <myBlobAccessKey>" # Provide blob access key for accessing the above BLOB storage e.g. "dxJR6VOGrDyEJJEv5atJPSB7ElhZx+weKhIlWWE/WYuLo8mvM4Bg3mxp8e3FtNuCvvpoO1827gQmsKVemqb8Vyw=="

# check if logging folder and files exists

If(!(test-path $LoggingFolder)) { New-Item -ItemType Directory -Force -Path $LoggingFolder}

if(! (Test-Path $LoggingFolder\BackupLog.log )) { New-Item $LoggingFolder\BackupLog.log -type file }

if(! (Test-Path $LoggingFolder\BackupErrorLog.Log)) { New-Item $LoggingFolder\BackupErrorLog.Log -type file }

# Log start

$MsgStart = $(get-date -format yyyy\_MM\_dd-HH:mm:ss) + [char]9 + "--- New Backup Task Started! --- , parameters Used: "

Write-Output $MsgStart | Out-File -filePath $LoggingFolder\BackupLog.log -append

if ($DatabaseList -EQ "ALL") { $WhereString = '$\_.Name -ne "tempdb" -and $\_.IsSystemObject -eq $False' }

elseif ($DatabaseList -EQ "ALLUSER") { $WhereString = '$\_.IsSystemObject -eq $False' }

else

{

$b = '"{0}"' -f ($DatabaseList -join '","')

$WhereString = '$\_.name -in ' + $b }

$WhereBlock = [scriptblock]::Create( $WhereString )

$currentDateTime = get-date -format yyyy-MM-dd-HHmmss

#Get-SqlDatabase -ServerInstance localhost | Where { $\_.Name -ne 'tempdb' } | Backup-SqlDatabase

#$DBList=Get-SqlDatabase -ServerInstance localhost | Where { $\_.Name -ne 'tempdb' }

$DBList=Get-SqlDatabase -ServerInstance localhost | Where $WhereBlock | Sort-Object -Property Name

$EncryptionOption = New-SqlBackupEncryptionOption -Algorithm Aes256 -EncryptorType ServerCertificate -EncryptorName "DBBackupEncryptCert"

foreach ($Database in $DBList )

$BckPath= $backupUrlContainer + $Database.Name +"\_" + $currentDateTime + "\_" + $BackupType +".bak"

TRY

{

{

$BckMsgStart= $(get-date -format yyyy\_MM\_dd-HH:mm:ss) + [char]9 + $Database.Name + " backup Started!"

Write-Output $BckMsgStart | Out-File -filePath $LoggingFolder\BackupLog.log -append

Backup-SqlDatabase -ServerInstance "localhost" -Database $Database.Name -BackupFile $BckPath -SqlCredential "SQLBackups" -CompressionOption On -BackupAction $BackupType -EncryptionOption $EncryptionOption

$BckMsgFinish= $(get-date -format yyyy\_MM\_dd-HH:mm:ss) + [char]9 + $Database.Name + " backup Completed!"

Write-Output $BckMsgFinish | Out-File -filePath $LoggingFolder\BackupLog.log -append

}

CATCH

{

$MsgErr = $(get-date -format yyyy\_MM\_dd-HH:mm:ss) + [char]9 + $Database.Name + " backup failed: " + $\_.Exception.Message

Write-Output $MsgErr | Out-File -filePath $LoggingFolder\BackupErrorLog.Log -append

}

}

## Blob Cleanup

$resourceGroup = "<ResGr>"

$storageAccountName = "<BlobName>"

$containerName = "<ContainerName>"

$DaySpan = 0

# get a reference to the storage account and the context

$storageAccount = Get-AzStorageAccount `

-ResourceGroupName $resourceGroup `

-Name $storageAccountName `

$ctx = $storageAccount.Context

$BlobAccessKey = "xyzww=="

$StorageAccountContext = New-AzureStorageContext -storageAccountName $StorageAccountName -StorageAccountKey $BlobAccessKey;

$StorageAccountContext;

# get a list of all of the blobs in the container

$listOfBLobs = Get-AzStorageBlob -Container $ContainerName -Context $ctx

# zero out our total

$Size = 0

$NumDel = 0

# this loops through the list of blobs and retrieves the length for each blob

# and adds it to the total

foreach($blob1 in $listOfBlobs)

{

if ([datetime]$blob1.LastModified.UtcDateTime -le $now.AddHours(-24\*$DaySpan))

{

Write-Host "Deleted:"$blob1.name

Remove-AzureStorageBlob -Container $containerName -Blob $blob1.Name -Context $StorageAccountContext

$NumDel ++

}

else

{

$Size = $Size + $blob1.Length

}

}

# output the blobs and their sizes and the total

Write-Host "Number of deleted items= " $NumDel

Write-Host "Total Length Remained= " $Size

## Blob Container usage

$resourceGroup = "<ResGr>"

$storageAccountName = "<BlobName>"

$containerName = "<ContainerName>"

# get a reference to the storage account and the context

$storageAccount = Get-AzStorageAccount `

-ResourceGroupName $resourceGroup `

-Name $storageAccountName

$ctx = $storageAccount.Context

# get a list of all of the blobs in the container

$listOfBLobs = Get-AzStorageBlob -Container $ContainerName -Context $ctx

# zero out our total

$length = 0

# this loops through the list of blobs and retrieves the length for each blob

# and adds it to the total

$listOfBlobs | ForEach-Object {$length = $length + $\_.Length}

# output the blobs and their sizes and the total

Write-Host "List of Blobs and their size (length)"

Write-Host " "

$listOfBlobs | select Name, Length

Write-Host " "

Write-Host "Total Length = " $length