Agenda: Azure Storage Service

- About Storage Service and Account
- Creating a Storage Account
- Working with Blob Storage
    - Types of Blobs (Block, Append, Page)
    - Container and Metadata
    - Programming Blob Storage
    - Async Blob Copy
    - Access blobs securely
    - Configure CDN
    - Configure Custom Domain
    - Scale Blob Storage
- Working with Table Storage
    - Understanding NoSQL Database
    - Creating Table and Entities using Storage Explorer
    - Programming Table Storage
    - Entities and Properties
- Azure Queues Storage
    - Understanding Async Communication using Queues
    - Comparing Azure Queues and Service Bus Queues
    - Programming Queues
- Azure SMB File Storage
    - Common usage of File Storage
    - Shares, Directory and File
    - Managing Using Azure Portal
    - Programming File Storage
- Manage access
    - Create and Manage Shared Access Signature
    - Account SAS vs Service SAS
    - Using Stored Access Policies
    - Regenerating Keys
    - Encrypt Keys using Azure Key Vault integration
    - Programming in C#
- Configure diagnostics, monitoring and analytics
    - Capturing Metrics Data
    - Analyzing Diagnostic Data

- Capturing Logs
- Retention Policies and Logging Levels
- Analyze Logs
- Enabling Monitoring
- Enabling Alerts

- Storage Service Encryption

About Azure Storage Service and Account

Cloud computing enables new scenarios for applications requiring **scalable, durable, and highly available** storage for their data – which is exactly why Microsoft developed **Azure Storage Service.**

- Azure Storage is a PaaS service that you can use to store both **unstructured** and **partially structured** data.
- **Azure Storage is massively scalable and elastic:** It can store and process **hundreds of terabytes of data** to support the big data scenarios required by scientific, financial analysis, and media applications. Or you can store the **small amounts of data** required for a small business website.
- By default, you can create up to **100 storage accounts** in a single Azure subscription. Each standard storage account can contain up to **500 TB** of combined blob, queue, table and file data.
- As the demands on your storage application grow, Azure Storage **automatically allocates** the appropriate resources to meet them. **We are charged only for what we use**.

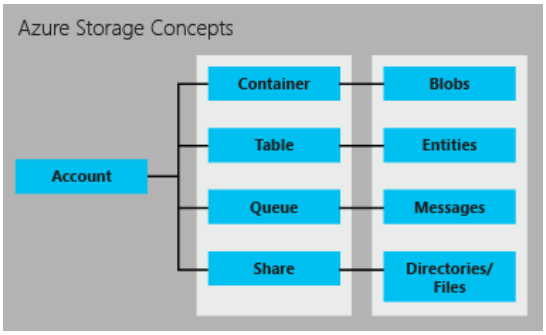
Azure Storage Account

An Azure storage account is a **secure account** that gives you access to services in Azure Storage. Your storage account provides the unique namespace for your storage resources. There are two types of storage accounts:

1. A **standard storage** account includes Blob, Table, Queue, and File storage.
2. A **premium storage** account currently supports Azure Virtual Machine disks only.

It offers **four types of storage services**, depending on the type of data that they are designed to store:

1. **Blob Storage** stores file data. A blob can be any type of **text or binary data**, such as a document, media file, or application installer. Blob Storage is sometimes referred to as **Object storage**.
2. **Table Storage** stores partially structured datasets. Table storage is a **NoSQL** key-attribute data store, which allows for rapid development and fast access to large quantities of data.
3. **Queue Storage** provides **reliable messaging** for workflow processing and for communication between components of cloud services.
4. **File Storage** Similar to blobs, these provide storage for unstructured files, but they offer support for file sharing in the same manner as traditional on-premises Windows file shares.



Creating Storage Account

1. Azure Portal → Browse Storage Accounts → **New** → **Data + Storage** → **Storage account**
2. Enter Name (must be all lowercase)
3. Deployment Model = Resource Manager
4. and select Subscription, Resource Group, Location
5. Account Kind: Storage (general purpose v1) / **StorageV2 (general purpose v2)** / Blob Storage

   Note: A Blob storage account is a specialized storage account for storing your unstructured data as blobs (objects) in Azure Storage. For applications requiring only **block or append blob** storage, we recommend using Blob storage accounts.
6. Performance: **Standard / Premium**

   Standard use HDD Drives and Premiun use SSD Drives

   Premium is used only for disks of VMs

   Note that it is not possible to convert a Standard storage account to Premium storage account or vice versa.
7. Access tier: Cool / **Hot**

   Account kind: **Blob storage,** Performance**: Standard**
   - Access tier: **Hot**, if objects will be **more** frequently accessed. This allows you to store data at a **lower access cost.**
   - Access tier: **Cool**, if objects will be **less** frequently accessed. This allows you to store data at **a lower data storage cost.**
8. Replication: **LRS.**

Locally redundant storage (LRS):

- **Replicates 3 times within a single data center in a single region where Storage Account is created.**
- A request returns successfully only once it has been written to all three replicas.
- LRS protects your data against server hardware failures but not against a failure of the facility itself.
- LRS is less expensive than GRS and also offers higher throughput.
- For Premium Storage accounts - This is the only option available.

Zone-redundant storage (ZRS)

- Replicates your data across three (3) storage clusters in a single region. Each storage cluster is physically separated from the others and resides in its own availability zone. Each availability zone, and the ZRS cluster within it, is autonomous, with separate utilities and networking capabilities.
- ZRS is not yet available in all regions.
- Once you have created your storage account and selected ZRS, you cannot convert it to use to any other type of replication, or vice versa.
- Consider ZRS for scenarios that require strong consistency, strong durability, and high availability even if an outage or natural disaster renders a zonal data center unavailable.

Geo-redundant storage (GRS)

- GRS maintains **six copies** of your data. 3 replicas in **primary region** and also replicates your data 3 additional times to a **secondary region** that is hundreds of miles away from the primary region.
- Data is durable even in the case of a complete regional outage or a disaster in which the primary region is not recoverable.
- If failure occurs in the primary region, Azure Storage automatically failover to the secondary region.
- An update is first committed to the primary region, where it is replicated three times. Then the update is replicated to the secondary region, where it is also replicated three times.
- Requests to write data are replicated **asynchronously** to the secondary region. It is important to note that opting for GRS does not impact latency of requests made against the primary region.
- The secondary region is **automatically determined** based on the primary region, and cannot be changed.

Read-access geo-redundant storage (RA-GRS)

- As with GRS, your data replicates asynchronously across two regions and synchronously within each region, yielding six copies of a storage account.
- This is default option when we create a storage account.
- In the event that data becomes unavailable in the primary region, your application can **read data** from the secondary region.
- If your primary endpoint for the Blob service is **myaccount.blob.core.windows.net**, then your secondary endpoint is **myaccount-secondary.blob.core.windows.net**. The **access keys** for your storage account are the **same** for both the primary and secondary endpoints.

More Details: https://azure.microsoft.com/en-in/documentation/articles/storage-redundancy/

9. Secure transfer required = **Disabled** / Enabled

   If Enabled only HTTPS requests will be accepted.

   This option doesn't work with Custom Domain Names for Storage account.
10. Select Subscription, Resource Group, Location
11. Virtual network: Disabled
12. Click on "Create".

PowerShell Command to Create Storage Account

Login-AzureRmAccount

Create a storage account.

- **New-AzureRmStorageAccount** -ResourceGroupName "DemoRG" -Name "dssdemostorage" -SkuName Standard_LRS -Location 'South India'

Modify storage account properties, such as type.

- **Set-AzureRmStorageAccount** -ResourceGroupName "DemoRG" -AccountName "dssdemostorage" -Type "Standard_RAGRS"

Retrieve a specific storage account or all the storage accounts in a resource group or subscription.

- **Get-AzureRmStorageAccount** -ResourceGroupName "DemoRG" -AccountName "dssdemostorage"

Working with Blob Storage

- **Blobs** are binary large objects. The Blob service stores text and binary data.
- Blob storage is also referred to as **object storage**.
- You can use Blob storage to store content such as:
  - Documents
  - Social data such as photos, videos, music, and blogs
  - Backups of files, computers, databases, and devices
  - Images and text for web applications
  - Configuration data for cloud applications
  - Big data, such as logs and other large datasets
- Every blob is organized into a **container**. Containers also provide a useful way to assign security policies to groups of objects. A storage account can contain any number of containers, and a container can contain any number of blobs, up to the **500 TB capacity** limit of the storage account.
- Every blob is **replicated** across **three computers** in a Windows Azure datacenter.
- **Creating BLOB Hierarchies:** The blob service in Azure Storage is based on a **flat storage** scheme. This means that creating a container one level below the **root is the only true level of container**. However, you can specify a delimiter as part of the blob name to create your own **virtual hierarchy**. For example, you could create a blob named **/January/Reports.txt** and **/February/Reports.txt**, and filter based on /January or /February in most tools that support Azure Storage. Most third-party storage tools allow you to create folders within a container, but they are actually being clever with the name of the blob itself.

Blobs are addressable using the following URL format:

http(s)://<storage account name>.**blob**.core.windows.net/<container>/<blob name>

Types of blobs:

1. **Block blobs** are optimized for streaming (sequential access) and for uploads and downloads, and are a good choice for storing documents, media files, backups etc. Azure divides data into smaller blocks of up to 4 megabytes (MB) in size, which subsequently upload or download in parallel. Individual block blobs can be up to 200 GB in size.
2. **Append blobs:** Append blobs are similar to block blobs, but are optimized for append operations. This works best with **logging and auditing** activities. Updating or deleting of existing blocks is not supported.
3. **Page blobs** are optimized for **random read/write** operations and provide the ability to write to a range of bytes in a blob. Blobs are accessed as pages, each of which is up to **512 bytes** in size. Each Page blob can be up to **8TB** each. Is best suited for **virtual machine disks**.

Azure Storage Explorer

Microsoft Azure Storage Explorer is a standalone app from Microsoft that allows you to easily work with Azure Storage data.

Some of the benefits of Azure Storage Explorer are:

a) Access multiple accounts and subscriptions across Azure

b) Create, delete, view, and edit storage resources

c) View and edit Blob, Queue, Table, File, Cosmos DB storage and Data Lake Storage.

d) Obtain shared access signature (SAS) keys

e) Available for Windows, Mac, and Linux

Azure Storage Explorer has many uses when it comes to managing your storage. See the following articles to learn more. Also, check out the videos that follow this topic.

- **Connect to an Azure subscription:** Manage storage resources that belong to your Azure subscription.
- **Work with local development storage:** Manage local storage by using the Azure Storage Emulator.
- **Attach to external storage:** Manage storage resources that belong to another Azure subscription or that are under national Azure clouds by using the storage account's name, key, and endpoints.
- **Attach a storage account by using an SAS:** Manage storage resources that belong to another Azure subscription by using a shared access signature (SAS).
- **Attach a service by using an SAS:** Manage a specific storage service (blob container, queue, or table) that belongs to another Azure subscription by using an SAS.
- **Connect to an Azure Cosmos DB account by using a connection string:** Manage Cosmos DB account by using a connection string.

Transfer data with the AzCopy on Windows

AzCopy is a command-line utility designed for copying data to/from Microsoft Azure Blob, File, and Table storage, using simple commands designed for optimal performance. You can copy data between a file system and a storage account, or between storage accounts.

There are two versions of AzCopy that you can download.

1. **AzCopy on Windows** is built with .NET Framework, and offers Windows style command-line options.
2. **AzCopy on Linux** is built with .NET Core Framework which targets Linux platforms offering POSIX style command-line options.

Path on Windows: C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy\AzCopy.exe

The basic syntax for AzCopy commands is:

```
AzCopy /Source:<source> /Dest:<destination> [Options]
```

Upload a blob based on its Pattern

AzCopy /Source:C:\myfolder /Dest:https://myaccount.blob.core.windows.net/mycontainer /DestKey:key /Pattern:"*.txt"

Upload all blobs in a folder and subfolder

AzCopy /Source:C:\myfolder /Dest:https://myaccount.blob.core.windows.net/mycontainer /DestKey:key /S

Download a single blob:

AzCopy /Source:https://myaccount.blob.core.windows.net/mycontainer /Dest:C:\myfolder /SourceKey:key /Pattern:"abc*.txt"

Download all blobs in a container

AzCopy /Source:https://myaccount.blob.core.windows.net/mycontainer /Dest:C:\myfolder /SourceKey:key /S

AzCopy on Windows

https://docs.microsoft.com/en-us/azure/storage/common/storage-use-azcopy

AzCopy on Linux

https://docs.microsoft.com/en-us/azure/storage/common/storage-use-azcopy-linux

Programming BLOB Storage using Powershell

Step 1: Copy Keys: Storage accounts → <Account Name> → Settings → Access Keys → Copy Keys for future use.

To Create a Container:

$ctx = **New-AzureStorageContext** -StorageAccountName "dssdemostorage" -StorageAccountKey "fTKZ. . . WA=="

New-AzureStorageContainer -Name "democontainer" -Context $ctx -Permission Off

You can optionally enable anonymous access at the container level for blob storage. The available options for this security policy are described as below:

Access Type	Resulting access
Off	No anonymous access (default)
Blob	Access blobs via anonymous requests
Container	List and access blobs via anonymous requests

To Upload a local file to Container:

Set-AzureStorageBlobContent -Container demoscontainer -File D:\happy.gif -Blob happy.gif **-Context $ctx**

OR

Set-AzureRmCurrentStorageAccount -StorageAccountName **dssdemostorage** -ResourceGroupName DemoRG

Set-AzureStorageBlobContent -Container imagescontainer -File D:\happy.gif -Blob happy.gif

To download BLOB

$blob = **Get-AzureStorageBlobContent** -Blob happy.gif -Container democontainer

Upload file and set Metadata:

$meta = @{"key" = "value"; "name" = "test"}

Set-AzureStorageBlobContent -File D:\demo.txt -Container democontainer -Metadata $meta **-Context $ctx**

To get Metadata of BLOB

$CloudBlockBlob = [Microsoft.WindowsAzure.Storage.Blob.**CloudBlockBlob**] $Blob.ICloudBlob

$CloudBlockBlob.**FetchAttributes**()

$meta = $CloudBlockBlob.**Metadata**

$meta["name"]

Using the Async blob copy service:

The async blob copy service is a server side based service that can copy files you specify from a source location to a destination in an Azure Storage account. The source blob can be located in another Azure Storage account, or it can even be outside of Azure as long as the storage service can access the blob directly for it to copy.

```
$vhdName = "[file name]"
$srcContainer = "[source container]"
```

```
$destContainer = "[destination container]"
$srcStorageAccount = "[source storage]"
$destStorageAccount = "[dest storage]"

Login-AzureRmAccount
$srcStorageKey = (Get-AzureStorageKey -StorageAccountName $srcStorageAccount).Primary
$destStorageKey = (Get-AzureStorageKey -StorageAccountName $destStorageAccount).Primary

$srcContext = New-AzureStorageContext –StorageAccountName $srcStorageAccount -StorageAccountKey
$srcStorageKey
$destContext = New-AzureStorageContext –StorageAccountName $destStorageAccount -StorageAccountKey
$destStorageKey

New-AzureStorageContainer -Name $destContainer -Context $destContext
$copiedBlob = Start-AzureStorageBlobCopy -SrcBlob $vhdName -SrcContainer $srcContainer -Context $srcContext
-DestContainer $destContainer -DestBlob $vhdName -DestContext $destContext
```

Copy SPECIFIC BLOB from one container to another:

Get-AzureStorageContainer -Name $srcContainer | **Start-AzureStorageBlobCopy** -SrcBlob $vhdName -DestContainer $destContext

Copy ALL BLOBS from one container to another

Get-AzureStorageBlob -Container $srcContainer | **Start-AzureStorageBlobCopy** -DestContainer $destContext

Copy a blob from a URI:

Start-AzureStorageBlobCopy -AbsoluteUri "http://www.contosointernal.com/planning" -DestContainer $destContext -DestBlob $vhdName -DestContext $Context

Import and Export Service

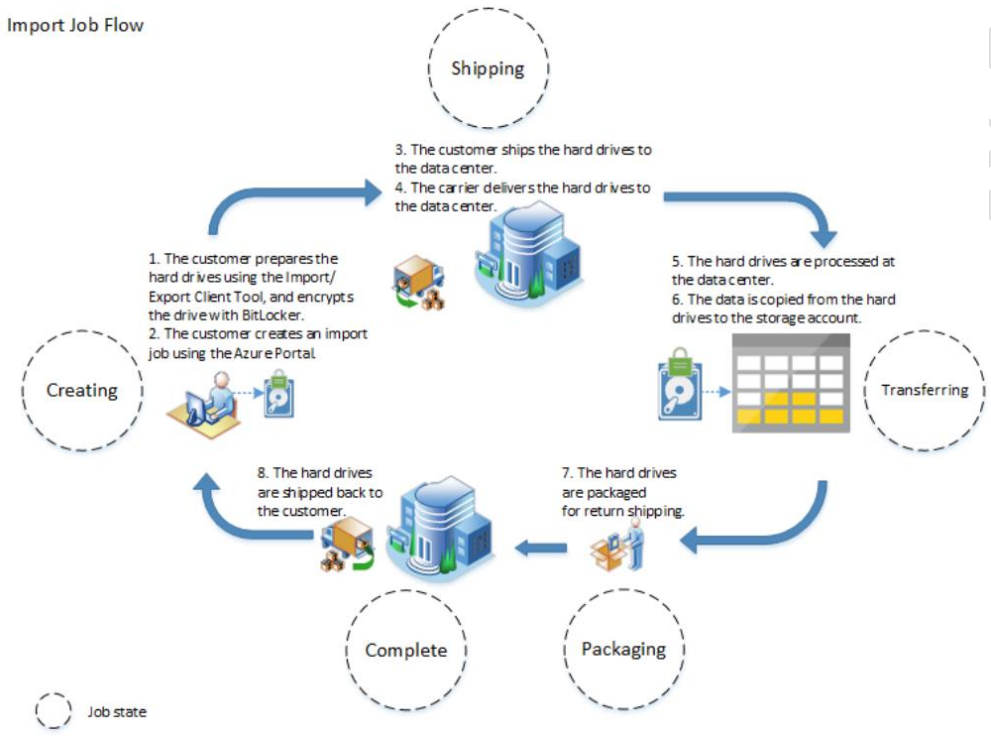
When it comes to transferring very large amounts of data to or from the cloud you will want to consider using the Azure Import/Export service. The Azure Import/Export Service allows you to:

- **Import to Azure Storage**. Securely transfer large amounts of data to Azure Blob storage (block and page blobs) and Azure Files by shipping disk drives to an Azure data center. In this case, you will be shipping hard drives containing your data.

- **Export from Azure Storage**. Transfer data from Azure storage to hard disk drives and ship to your on-premise sites. Currently, you can only export **Block** blobs, **Page** blobs or **Append** blobs from Azure storage using this service. Exporting Azure Files is not currently supported. In this case, you will be shipping empty hard drives.

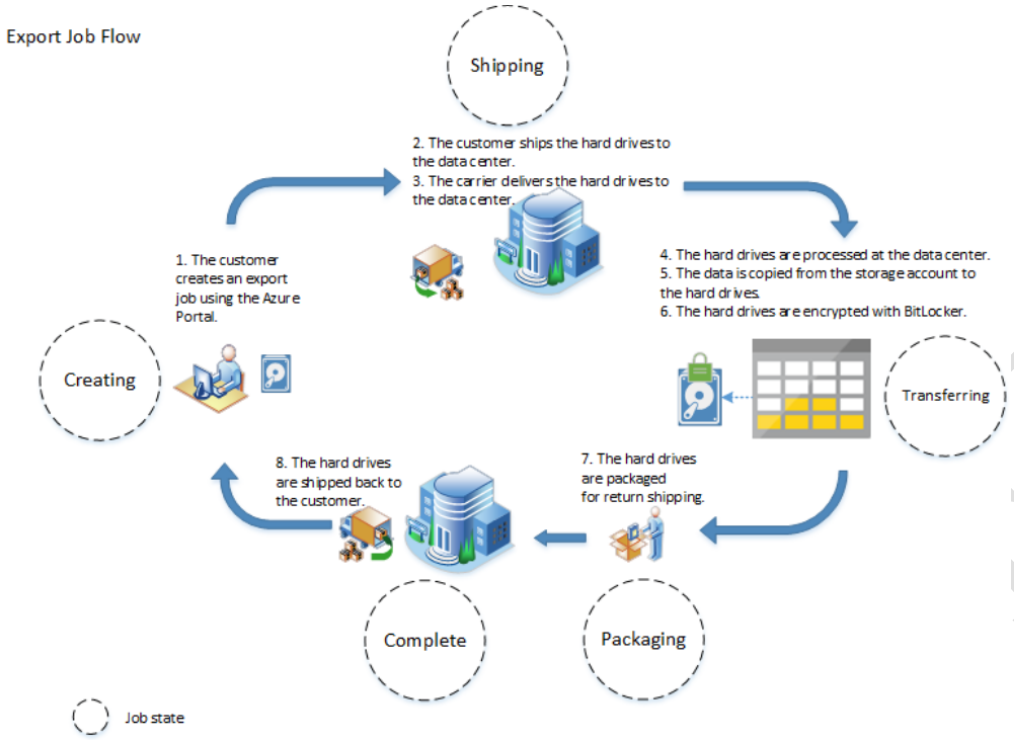
Consider using Azure Import/Export service when uploading or downloading data over the network is too slow or getting additional network bandwidth is cost-prohibitive. Scenarios where this would be useful include:

✓ Only 2.5" SSD or 2.5" or 3.5" SATA II or III internal HDD are supported for use with the Import/Export service.



At a high level, an import job involves the following steps:

1. Determine data to be imported, number of drives you need, destination blob location for your data in Azure storage.
2. Use the WAImportExport tool to copy data to disk drives. Encrypt the disk drives with BitLocker.
3. Create an import job in your target storage account in Azure portal. Upload the drive journal files.
4. Provide the return address and carrier account number for shipping the drives back to you.
5. Ship the disk drives to the shipping address provided during job creation.
6. Update the delivery tracking number in the import job details and submit the import job.
7. The drives are received and processed at the Azure data center.
8. The drives are shipped using your carrier account to the return address provided in the import job.



At a high level, an export job involves the following steps:

1. Determine the data to be exported, number of drives you need, source blobs or container paths of your data in Blob storage.
2. Create an export job in your source storage account in Azure portal.
3. Specify source blobs or container paths for the data to be exported.
4. Provide the return address and carrier account number for shipping the drives back to you.
5. Ship the disk drives to the shipping address provided during job creation.
6. Update the delivery tracking number in the export job details and submit the export job.
7. The drives are received and processed at the Azure data center.
8. The drives are encrypted with BitLocker and the keys are available via the Azure portal.
9. The drives are shipped using your carrier account to the return address provided in the import job.

Azure Import/Export Tool

The Microsoft Azure Import/Export Tool (**WAImportExport.exe**) is the drive preparation and repair tool that you can use with the Microsoft Azure Import/Export service. You can use the tool for the following functions:

- Before creating an import job, you can use this tool to copy data to the hard drives you are going to ship to an Azure data center.
- After an import job has completed, you can use this tool to repair any blobs that were corrupted, were missing, or conflicted with other blobs.
- After you receive the drives from a completed export job, you can use this tool to repair any files that were corrupted or missing on the drives.

Manage Access using Shared Access Signature

There are two techniques for controlling access to objects within an Azure Storage account.

1. Using the **authentication key and storage account name** is one technique.
2. Granting access using a **shared access signature** or **via a policy** to allow granular access with expiration is another technique.

A shared access signature (SAS) is a URI that grants **restricted access rights** to Azure Storage resources (a specific blob in this case). You can provide a shared access signature to clients who should not be trusted with your storage account key but whom you wish to delegate access to certain storage account resources. By distributing a shared access signature URI to these clients, you grant them access to a resource for a specified **period of time**.

- A shared access signature (SAS) is a **token** that can be appended to a URL that enables delegated access to a storage resource.
- Anyone who possesses the token can access the resource it points to with the permissions it specifies, for the period of time that it is valid.

Beginning with version 2015-04-05, Azure Storage supports two kinds of shared access signatures:

1. An **Account SAS** delegates access to resources in **one or more** of the storage services. 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.
2. The **Service SAS** delegates access to a resource in **just one** of the storage services: Blob, Queue, Table, or File service.
   Note that stored access policies are currently not supported for an account-level SAS.

Creating an **Account SAS** (for many operations)

1. Azure Portal → Storage Accounts → Select Account
2. Settings → Shared access signature
3. Provide the options as required → Generate SAS
4. Copy the SAS token and share it with the client.

https://dssdemostorage.blob.core.windows.net/container1/Azure%20Storage%20Service.pdf?sv=2017-11-09&ss=b&srt=co&sp=r&se=2018-08-13T10:05:57Z&st=2018-08-11T02:05:57Z&spr=https&sig=hFYQeZ2fvj52%2BQI0kg%2BbPmErr8J%2F5hKrGkAnF7Q7u%2F4%3D

Create Service SAS using Portal

1. Azure Portal → Storage Accounts → Select Account

2. BLOB → Container → <Select Blob Item> → Click on Generat SAS Tab
3. Provide the details →
4. Copy the Token and use with Blob URL.

Stored Access Policy and SAS Token if Deployment Model = Resource Manager

```
$storageAccountName = 'dssdemostorage'
$rgName = 'DemoRG'
$containerName = 'container1'

$accountKeys = Get-AzureRmStorageAccountKey -ResourceGroupName $rgName -Name $storageAccountName
$storageContext = New-AzureStorageContext -StorageAccountName $storageAccountName -StorageAccountKey
$accountKeys[0].Value
$expiryTime = (get-date).AddYears(1)
$startTime = (get-date).AddHours(-1)
$permission = "rwl"

//Creates a STORED ACCESS POLICY for an Azure storage container.
New-AzureStorageContainerStoredAccessPolicy -Context $storageContext -Container $containerName -Policy
"testPolicy" -ExpiryTime $expiryTime -Permission $permission

//Creates a SAS Token using a STORED ACCESS POLICY
$sasToken = New-AzureStorageContainerSASToken  -Name $containerName -Policy "testPolicy" -Context
$storageContext
Write-Host "SAS token (ref shared access policy): $sasToken"

//Creates a SAS Token with Adhoc Access Policy
$sasContainerToken2 = New-AzureStorageContainerSASToken -Name $containerName -Context $storageContext
-Permission rwl -StartTime $startTime -ExpiryTime $expiryTime
Write-Host 'SAS token: ' $($sasToken2)

//Create a SAS Token for a specific BLOB Item
$sasBlobToken3 = New-AzureStorageBlobSASToken -Container "container1" -Blob "SmileyNormal.png" -Context
$storageContext  -Permission "rwd" -StartTime $startTime -ExpiryTime $endTime
```

It's in query string format. This is a query string that can be appended to the full URI of the blob or container the SAS URI was created with, and passed to a client.

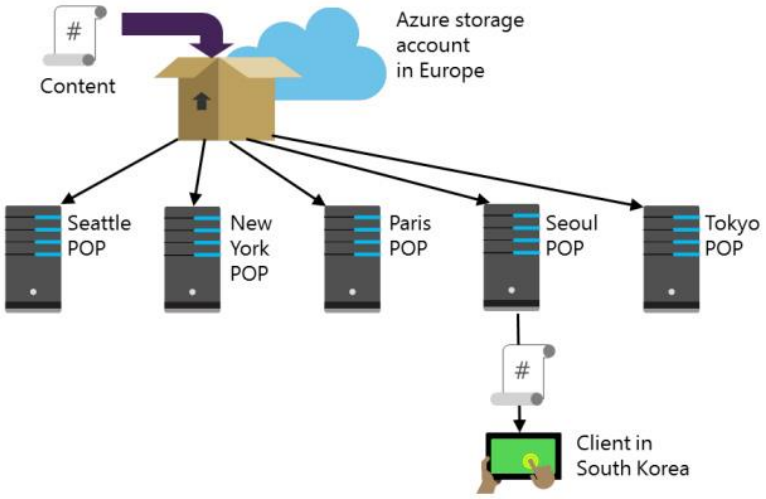
Example: http://dssdemostorage.blob.core.windows.net/secure/reports1.xlsx***?sv=2014-02-14&sr=b&sig=8qLv51D3ahgw9Zoyf9vhVfSvOuVdak%2Fdh1gIHmb6pll%3D&st=2014-11-29T12%3A17%3A50Z&se=2014-11-29T16%3A17%3A50Z&sp=rwd***

Similarly we have:

- New-AzureStorageQueueSASToken
- New-AzureStorageTableSASToken
- New-AzureStorageFileSASToken

Implementing Content Delivery Network

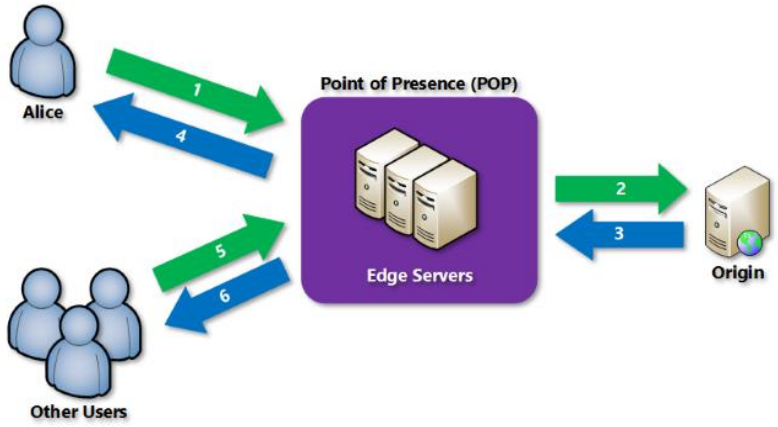
- Azure provides Content Delivery Network (CDN) functionality, which **decreases the time** it takes to download web content by first distributing it across multiple locations around the world and then delivering it from the location that is closest to the consumer of that content.
- CDNs are typically used to deliver **static content** such as images, style sheets, documents, client-side scripts, and HTML pages.



Azure content delivery networks cache content from

1. Azure Storage blobs
2. Web Apps
3. PaaS cloud services
4. Custom origin (any public web location that you can access by using HTTP or HTTPS)

How CDN Works



1. A user (Alice) requests a file (also called an asset) using a URL with a special domain name, such as <endpointname>.azureedge.net. DNS routes the request to the best performing Point-of-Presence (POP) location, which is usually the POP that is geographically closest to the user.
2. If the edge servers in the POP do not have the file in their cache, the edge server requests the file from the origin.
3. The origin returns the file to the edge server, including optional HTTP headers describing the file's Time-to-Live (TTL).
4. The edge server caches the file and returns the file to the original requestor (Alice). The file remains cached on the edge server until the TTL expires. Azure CDN automatically applies a default **TTL of seven days** unless you've set up caching rules in the Azure portal.
5. Additional users may then request the same file using that same URL and may also be directed to that same POP.
6. If the TTL for the file hasn't expired, the edge server returns the file from the cache.

Important points to be noted:

- If you modify an object that's currently cached in the CDN, the updated content **will not be** available via CDN until CDN refreshes its content after the **time-to-live period** for the cached content expires.
- CDNs are intended for static content. Dynamic content needs to be refreshed constantly from the content provider, minimizing and potentially eliminating any associated CDN benefits.
- The CDN service is **global** and not bound to a location, however you must specify a resource group location where the metadata associated with the CDN profile will reside. This location will not have any impact on the runtime availability of your profile.

CDNs offer a number of advantages:

1. **Improved user-experience**, especially if users reside in areas distant from the original content location.
2. **Improved scalability** by eliminating performance bottlenecks that are associated with hosting content in a single location.

3. **Increased resiliency** (capacity to recover) by eliminating a single point of failure. In particular, if one CDN node becomes unavailable, content transparently retrieves from the next nearest node.

Creating CDN profiles and endpoints

- A CDN profile is a collection of CDN endpoints with the same pricing tier and provider (origin).
- The profile constitutes an administrative and billing unit according to its pricing tier.
- The profile also provides additional features, such as country filtering, which includes blocking or allowing access to cached content from designated countries and analytics reporting.
- A CDN profile can contain up to **four endpoints**, and there is a limit of **eight CDN profiles** per Azure subscription.

Walkthrough:

1. All Services → CDN Profiles → + Add
2. Name = DemoProfile
3. In the **CDN profile** blade, click **+ Endpoint**.
4. In the **Add an endpoint** blade, specify the following:
   - Name. This is a unique name in the **azureedge.net** Domain Name System (DNS) namespace.
   - **Origin type.** This is **Storage**, Cloud service, Web app, or Custom origin.
   - **Origin hostname**. This is the name of the host that represents the origin type that you selected. This can be a name that displays automatically for Azure resources, an FQDN, or its corresponding IP address for custom origins.
   - **Origin path**. This allows you to specify a directory path to retrieve from the origin..
   - **Origin host header**. This is designates the host header value that should be sent to the origin with each request. This is useful if you host multiple virtual domains on a single target server.
   - **Protocol and origin port:** HTTP with the default port 80 and HTTPS with the default port 443.
5. Click **Add**.

For every endpoint, you can configure a number of settings, such as:

CDN Profiles → Select Profile → Click on Endpoints → Select Endpoint

- **Compression**: Compress x` CDN to reduce size and improve performance. All listed MIME types will be compressed when enabling the feature. Note that files are only compressed on the fly by the CDN if it is served from CDN cache. Compressed by the origin can still be delivered compressed to the client without being cached..
- **Cache rules: Query string caching behavior**. This setting controls caching behavior, depending on whether the request to the endpoint includes a query string or ignore query strings or ignore caching altogether.

Azure CDN caching rules are available only for **Azure CDN Standard from Verizon** and **Azure CDN Standard from Akamai**.

Custom Caching Rules

The **path** can **match** a single file (e.g. '/pictures/city.png') or a folder (e.g. '/pictures/' or '/pictures/cities/'). When a folder is specified (i.e. with a trailing slash), the filter applies to all files and sub-folders under it

- **Geo-filtering**: Create geo-filtering rules on specific paths on your endpoint to block or allow content in the selected countries.

Caching content from Azure blobs

- For a CDN to cache blobs, users must be able to access the **blobs anonymously**.
- After a CDN is implemented, all publicly available blobs in storage account will be cached.
- A blob stays in the CDN cache for a period known as the Time to Live (TTL), which by default **is seven days**. Therefore, if users access this content frequently in a seven-day period, the CDN will offer a significant performance gain. If users access this content every 10 days, CDN would provide no performance gains.

The **metadata** to control the **TTL**, and also the **content type**, is set on each blob as it is **uploaded**. The max-age attribute is measured in seconds. In the following example, the TTL is set to 86400 seconds, or 1 day.

$blobProperties = @{**ContentType**="img/png"; **CacheControl="**public, max-age=86400**"** }

Set-AzureStorageBlobContent -File $_.FullName -Container $container -Context $context **-Properties $blobProperties**

Caching content from cloud services and web apps

- The content to cache should be static and must be accessible via HTTP on port 80 or HTTPS an 443.
- The cloud service must be in the **production** deployment slot
- The content to cache must be in **/cdn** folder of cloud service.
- Similar to blob-based endpoints, cached content from cloud services has a **seven-day TTL** by default. You can modify this by specifying the **clientCache** setting in the **web.config** file in the **/cdn** folder.

Using custom domains to access CDNs

CDN Profiles → Select Profile → Click on Endpoints → Select Endpoint → **Custom domains**

In several scenarios, you might want to point to CDN-cached content by using names in your own custom DNS namespace. If that is the case, keep in mind that the target names must include a prefix, such as www.adatum.com," and they cannot take the form of a root domain, such as "adatum.com."

Register a custom domain for an Azure CDN endpoint using the intermediary cdnverify subdomain:

1. Navigate to your domain registrar's web site, Create a new CNAME record, and provide a subdomain alias that includes the **cdnverify** subdomain.

   For example, the subdomain that you specify will be in the format **cdnverify.www** or **cdnverify.cdn**. Then provide the host name, which is your CDN endpoint, in the format **cdnverify.<EndpointName>.azureedge.net**. Your DNS mapping should look like: **cdnverify**.www.consoto.com CNAME **cdnverify**.consoto.azureedge.net
2. Add custom domain name for CDN EndPoint. Azure will verify that the CNAME record exists for the cdnverify domain name you have entered.
3. At this point, your custom domain has been verified by Azure, but traffic to your domain is not yet being routed to your CDN endpoint. After waiting long enough to allow the custom domain settings to propagate to the CDN edge nodes (90 minutes for **Azure CDN from Verizon**, 1-2 minutes for **Azure CDN from Akamai**), return to your DNS registrar's web site and create another CNAME record that maps your subdomain to your CDN endpoint. For example, specify the subdomain as **www** or **cdn**, and the hostname as **<EndpointName>.azureedge.net**. With this step, the registration of your custom domain is complete.
4. Finally, you can delete the CNAME record you created using **cdnverify**, as it was necessary only as an intermediary step.

Current Status: cdn.deccansoft.com => someendpoint.azureedge.net

New Requirement is to Map cdn.deccansoft.com => dsscdnforstorage.azureedge.net

Existing State: cdn.deccansoft.com => OLDdsscdnforstorage.azureedge.net

Step1: DNS Server asverify.cdn.deccansoft.com => asverify.dsscdnforstorage.azureedge.net

//Custom is not able to connect to CDN...

Step2: CDN Profile - Add Custom Domain (requires verified domain)

dsscdnforstorage.azureedge.net = cdn.deccansoft.com

Step3: DNS Server cdn.deccansoft.com => dsscdnforstorage.azureedge.net

Note: Mapping a domain to an Azure Storage account in DNS with the asverify intermediary domain

CNAME RECORD	TARGET
asverify.blobs.contoso.com	asverify.contosoblobs.blob.core.windows.net
blobs.contoso.com	contosoblobs.blob.core.windows.net

Azure Table Storage

Category(PKCategoryId, CategoryName, ...) - SQL

1, Furniture

2, Pet

3, Electronics

4, Plants

Product(PKProduct, FKCategoryId, ProductName, FKProductAttributes) - SQL

1, 2, Dog01, 1

2, 2, Cat01, 4

3, 1, Chair01, 2

4, 1, Table01, 3

ProductAttributes (NoSQL)

ID: 1, **Age**:3, **Breed**:Pomerian, **Color**:White

ID: 2, **Weight**: 10KG, **Color**: Brown, **Type**: Chair

ID: 3, **Weight**: 30KG, **Color**: Black, **Type**: Table

ID: 4, **Age**:1, **Breed**:Indian, **Color**:Black

- The Azure Table storage service stores large amounts of **partially structured** data offering high availability and massively scalable storage.
- The service is a **NoSQL datastore** which accepts **authenticated calls** from inside and outside the Azure cloud.
- For today's Internet-based applications, NoSQL databases like Table storage offer a popular alternative to traditional relational databases.

Common uses of the Table service include:

- You can use Table storage to store flexible datasets, such as user data for web applications, address books, device information, and any other type of metadata that your service requires.
- Storing datasets that **don't** require complex joins, foreign keys, or stored procedures and can be de-normalized for fast access.
- Quickly querying data using a clustered index.
- Accessing data using the **OData protocol** and **LINQ queries** with WCF Data Service .NET Libraries.

You can use the Table service to store and query huge sets of structured, non-relational data, and your tables will scale as demand increases.

- **Table**: A table is a collection of entities. Tables don't enforce a schema on entities, which means a single table can contain entities that have different sets of properties. The number of tables that a storage account can contain is limited only by the storage account capacity limit.
- **Entity**: An entity is a set of properties, similar to a database row. An entity can be up to **1MB in size**.
- **Properties**: A property is a name-value pair. Each entity can include up to **252 properties** to store data. Each entity also has **3 system** properties that specify a **partition key, a row key, and a timestamp**. Entities with the same partition key can be queried more quickly, and inserted/updated in atomic operations. An entity's row key is its unique identifier within a partition.

The URI for a specific table access is structured as follows:

http://<account>.**table**.core.windows.net/<TableName>

Note:

- A **batch operation** is a collection of table operations which are executed by the Storage Service REST API as **a single atomic** operation.
- A batch operation may contain up to **100 individual table operations**, with the requirement that each operation entity must have **same partition key.**
- A batch with a retrieve operation cannot contain any other operations.
- The total payload of a batch operation is limited to 4MB.

Using Visual Studio Server Explorer

1. Server Explorer → … →Select Windows Azure Storage → Select and Expand Storage Account.
2. Expand to Tables → Create Table…, Enter Name of Table
3. Select Table → Right click → View Table
4. Use the Editor to manage Table.

Azure Queues Storage

Azure Queue storage is a service for storing large numbers of messages that can be accessed from anywhere in the world via authenticated calls using HTTP or HTTPS.

A single queue message can be up to 64 KB in size, and a queue can contain millions of messages, up to the total capacity limit of a storage account (i.e. 500TB)

Queues can have metadata associated with them. Metadata is in the form of <name, value> pairs.

Common uses of Queue storage include:

1. Creating a backlog of work to process asynchronously
2. Passing messages from an Azure App Service to an Azure Function.

- **Queue:** A queue contains a set of messages. All messages must be in a queue.
- **Message:** A message, **in any format**, of up to 64KB.

Difference between Storage Queues and Service Bus Queue

1. Azure storage queue does not provide publish/subscribe mechanism.
2. Azure storage queue can store data maximum for 7 days.
3. Azure storage queue **does not** provide a guaranteed FIFO delivery.

4. Azure storage queue does not provide transactional behavior.

Important Notes about Storage Queues:

- When a message is retrieved from the queue, the response includes the message and a pop receipt value, which is required to delete the message.
- The message is **not automatically deleted** from the queue, but after it has been retrieved, it is **not visible to other clients** for the time interval specified by the ***visibilitytimeout*** parameter.
- Every time it's received and not deleted, its **Dequeuecount is incremented**.

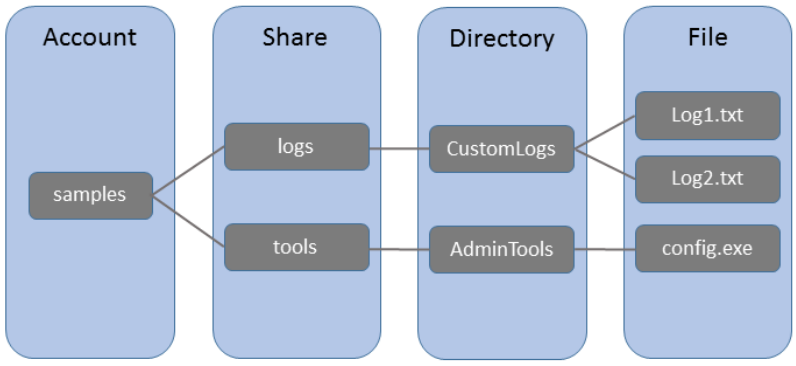
Azure SMB File Storage

- Azure File storage is a service that offers file shares in the cloud using the standard Server Message Block (SMB) Protocol.
- With Azure File storage, you can migrate **legacy applications** that rely on file shares to Azure quickly and without costly rewrites.
- Microsoft Azure virtual machines can share file data across application components via mounted shares, and on-premises applications can access file data in a share via the File storage API.

Common uses of File storage include:

- Migrating on-premises applications that rely on file shares to run on Azure virtual machines or cloud services, without expensive rewrites.
- Storing shared application settings, for example in configuration files.
- Storing diagnostic data such as logs, metrics, and crash dumps in a shared location.
- Storing tools and utilities needed for developing or administering Azure virtual machines or cloud services.

File storage concepts



- **Storage Account:** All access to Azure Storage is done through a storage account.

- **Share:** A File storage share is an SMB file share in Azure. All directories and files must be created in a parent share. An account can contain an unlimited number of shares, and a share can store an unlimited number of files, up to the 5 TB total capacity of the file share.
- **Directory:** An optional hierarchy of directories.
- **File:** A file in the share. A file may be up to 1 TB in size.
- **Max size of a File Share = 5TB**

URL format: https**://<storage account>**.file.core.windows.net/**<share>**/**<directory/directory>**/**<file>**

The following example URL could be used to address one of the files in the diagram above:

http://samples.file.core.windows.net/logs/CustomLogs/Log1.txt

Managing Using Azure Portal:

1. Azure Portal → **Storage accounts** → Create and Choose the Storage Account
   Note: File storage is replicated only via LRS or GRS right now…
2. Choose "Files" service → Click "+ File share" → New file share = Images, Quota=100GB
3. Optionally add directory and in the directory and upload the file[s].

Mapping the file share in Windows

4. Go to File Share Blade and click on **Connect** → Copy the net command edit the values
5. Open Command Prompt in Administrator Mode → Execute the net command
6. You can manage the File Share using the local drive.

Note: Ensure port 445 is open: Azure Files uses SMB protocol. SMB communicates over TCP port 445

Creating File Share using Powershell

Retrieve storage account and storage account key

$storageContext = **New-AzureStorageContext** <storage-account-name> <storage-account-key>

Create the file share, in this case "logs"

$share = **New-AzureStorageShare** logs -Context $storageContext

Azure File Sync

- Use Azure File Sync to centralize your organization's file shares in Azure Files, while keeping the flexibility, performance, and compatibility of an on-premises file server.
- Azure File Sync transforms Windows Server into a quick cache of your Azure file share.
- You can use any protocol that's available on Windows Server to access your data locally, including SMB, NFS, and FTPS.
- You can have as many caches as you need across the world.

Why Azure File is useful

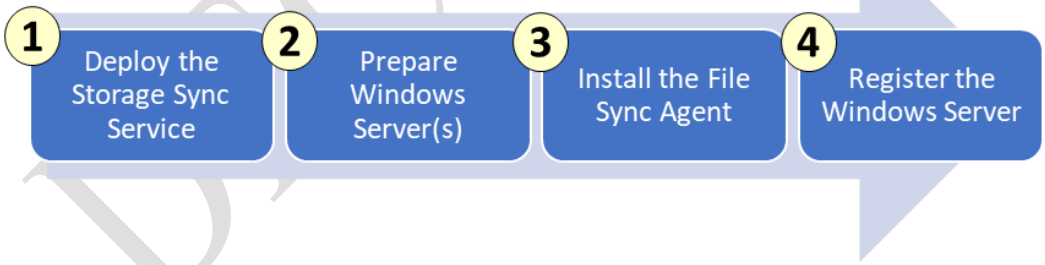
- Replace or supplement on-premises file servers.
- "Lift and shift" applications.
- Simplify cloud development
    - Shared application settings
    - Diagnostic share
    - Dev/Test/Debug

There are many uses and advantages to file sync.

- **Lift and shift.** The ability to move applications that require access between Azure and on-premises systems. Provide write access to the same data across Windows Servers and Azure Files. This lets companies with multiple offices have a need to share files with all offices.
- **Branch Offices**. Branch offices need to backup files, or you need to setup a new server that will connect to Azure storage.
- **Backup and Disaster Recovery**. Once File Sync is implemented, Azure Backup will back up your on-premises data. Also, you can restore file metadata immediately and recall data as needed for rapid disaster recovery.
- **File Archiving**. Only recently accessed data is located on local servers. Non-used data moves to Azure in what is called Cloud Tiering. Cloud Tiering files will have greyed icons with an offline O file attribute to let the user know the file is only in Azure.

File Sync Service Deployment

There are a few things that need to be configured before you synchronize your files.

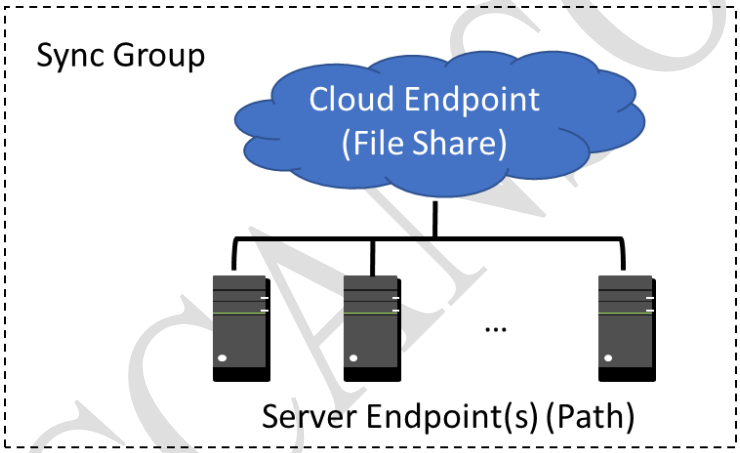


1. **Deploy the Storage Sync Service**. The Storage Sync Service is the top-level Azure resource for Azure File Sync. A distinct top-level resource from the storage account resource is required because the Storage Sync Service can create sync relationships with multiple storage accounts via multiple sync groups. A subscription can have multiple Storage Sync Service resources deployed.
2. **Prepare Windows Server to use with Azure File Sync**. For each server that you intend to use with Azure File Sync, including server nodes in a Failover Cluster, you will need to configure the server. Preparation steps

include temporarily disabling Internet Explorer Enhanced Security and ensuring you have latest PowerShell version.

3. **Install the Azure File Sync Agent**. The Azure File Sync agent is a downloadable package that enables Windows Server to be synced with an Azure file share. The Azure File Sync agent installation package should install relatively quickly. We recommend that you keep the default installation path and that you enable Microsoft Update to keep Azure File Sync up to date.
4. **Register Windows Server with Storage Sync Service**. When the Azure File Sync agent installation is finished, the Server Registration UI automatically opens. Registering Windows Server with a Storage Sync Service establishes a trust relationship between your server (or cluster) and the Storage Sync Service. Registration requires your Subscription ID, Resource Group, and Storage Sync Service (created in step one). A server (or cluster) can be registered with only one Storage Sync Service at a time.

File Sync Service Deployment (Synchronization)



A **Sync Group** defines the sync topology for a set of files. Endpoints within a sync group are kept in sync with each other. A sync group must contain at least one cloud endpoint, which represents an Azure file share created in your storage account, and at least one server endpoint, which represents a path on a Windows Server.

✓ As Azure File Sync is available in only few Regions: Eg: West US, remember that your storage account must be located in one of the regions in which Azure File Sync is supported.

Walkthrough

1. Create Storage Account
    a. Create File Share in Storage Account
2. Create **Azuer File Sync** Service
    a. Create Sync Group
    b. Select Storage Account

c. Register our On-Premise Server or local server
    i. RDP to VM
    ii. Ensure that Azure PowerShell Cmdlets are installed.
    iii. Download and Install File Sync Agent
    iv. Complete the Registration process, SignIn to Azure Account → Select the File Sync Service → Register

3. File Sync Group → Sync Group → Add server endpoint → Path = C:\AzureFiles.
    - **Registered server**. The name of the server or cluster where you want to create the server endpoint.
    - **Path**. The Windows Server path to be synced as part of the sync group. The path should not be the root volume.
    - **Cloud Tiering**. A switch to enable or disable cloud tiering.
    - **Volume Free Space**. The amount of free space to reserve on the volume on which the server endpoint is located. For example, if volume free space is set to 50% on a volume that has a single server endpoint, roughly half the amount of data is tiered to Azure Files.
4. RDP to VM and Note that the folder is created in that VM.
5. Dump some files in that folder and note in Portal that files are now present in Azure file share.

Monitoring Storage - Metrics and Alerts

Monitor Metrics

Azure Monitor provides unified user interfaces for monitoring across different Azure services. Azure Storage integrates Azure Monitor by sending metric data to the Azure Monitor platform. With metrics on Azure Storage, you can analyze usage trends, trace requests, and diagnose issues with your storage account.

Azure Monitor provides multiple ways to access metrics. You can access them from the Azure Portal, Monitor APIs (REST, and .Net) and analysis solutions such as the Operation Management Suite and Event Hubs. Metrics are enabled by default, and you can access the past 30 days of data. If you need to retain data for a longer period, you can archive metrics data to an Azure Storage account.

Metrics is a Shared Service where you can specify the resource, sub-service, metric, and aggregation criteria. Additionally, you specify more than one metric, filter by a metric, and Export to Excel.

Steps to View Metrics

a) Azure Portal → Storage Account → Settings → Monitoring → Metrics

b) Ad metric → Resource = <Storage Account>

When you select a Storage Account resource your sub-service choice is: Account, Blob, File, Queue, and Table and accordingly Metric field is filtered.

Note:

- **Capacity metrics** values are sent to Azure Monitor every hour. The values are refreshed daily. Your sub-service selection determines what Capacity metrics are available. For example, if you choose the Blob sub-service, then the Capacity metrics are: Blob Capacity, Blob Container Count, and Blob Count.
- **Transaction metrics** are sent from Azure Storage to Azure Monitor every minute. All transaction metrics are available at both account and service level (Blob storage, Table storage, Azure Files, and Queue storage).

Managing Alerts

Aerts can be authored in a consistent manner regardless of the monitoring service or signal type. All alerts fired and related details are available in single page.

Authoring an alert is a three-step task where the user first picks a target for the alert, followed by selecting the right signal and then specifying the logic to be applied on the signal as part of the alert rule.

Create Rule:

Monitoring → Alerts → New alert rule

1. **Select Resource:** For example, storage account.
2. **Add Condition**
   - Select signal. For example, Used Capacity.
   - Configure signal logic. For example, over a six-hour period whenever the Used Capacity is over 1000000 bytes.
3. **Define action group**. Create an action group to notify your team via email and text messages or automate actions using webhooks and runbooks.
4. **Define alert details:** Alert rule name, description.