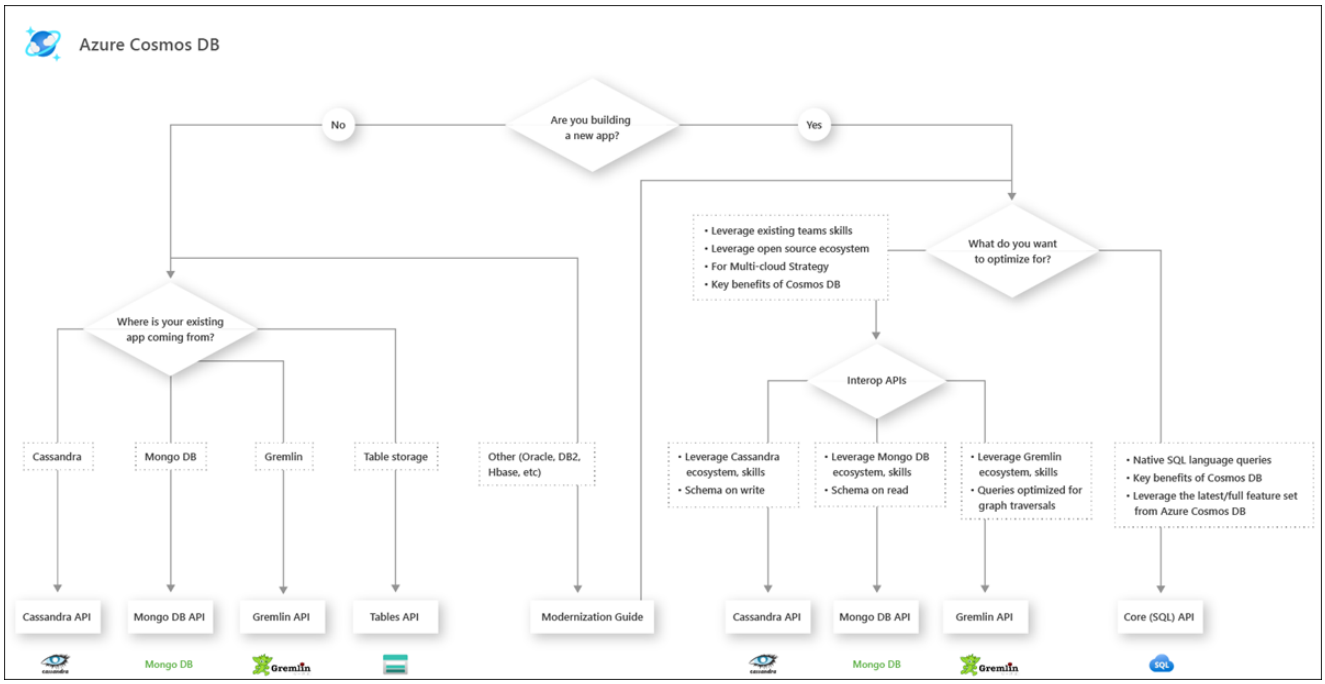
LEARING STORY 2

My recap of what I learned during my second learning story.

Brian Dekker
Brian.dekker@hva.nl

Index



Storage accounts.

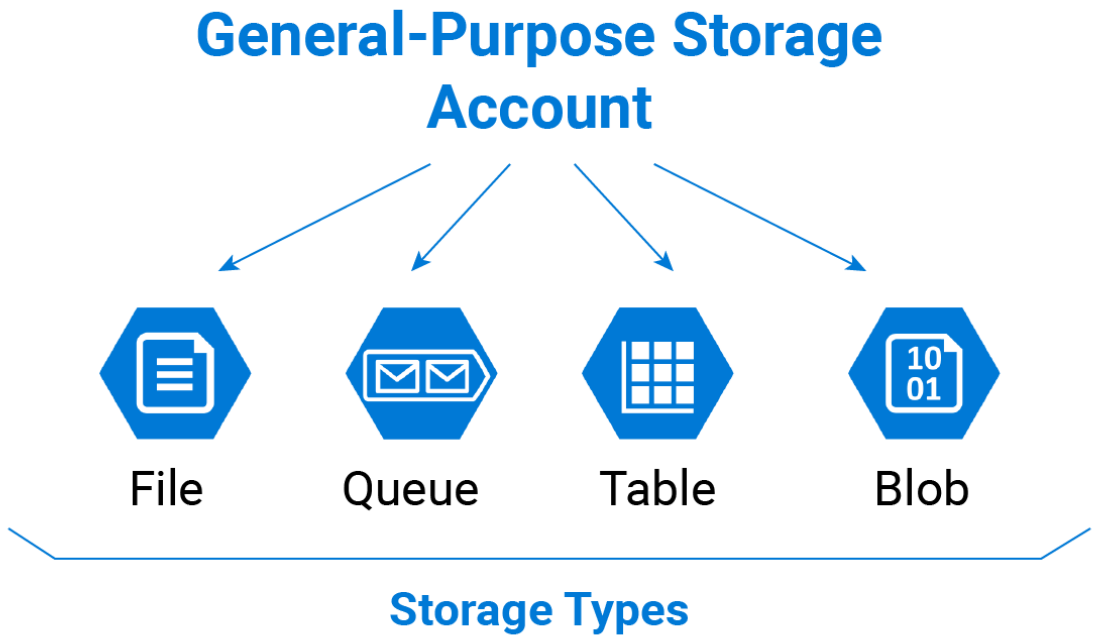
I did a lot of research on storage account and I will go into detail what they are and what they do in this recap chapter.

What is a storage account.

A storage account is basically a data repository on the cloud. In storage accounts you have 4 types of storage options.

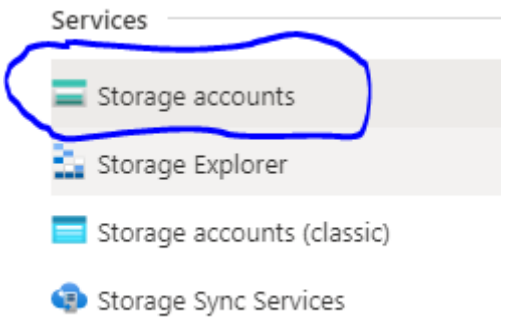
- Blob = Scalable data option for any type of file
- Table = A NoSQL data storage option
- Queue = A messaging storage for reliable messaging between apps.
- File share = Managed file shares for cloud or on premise deployment

I will go into detail of these four option in the upcoming chapters.

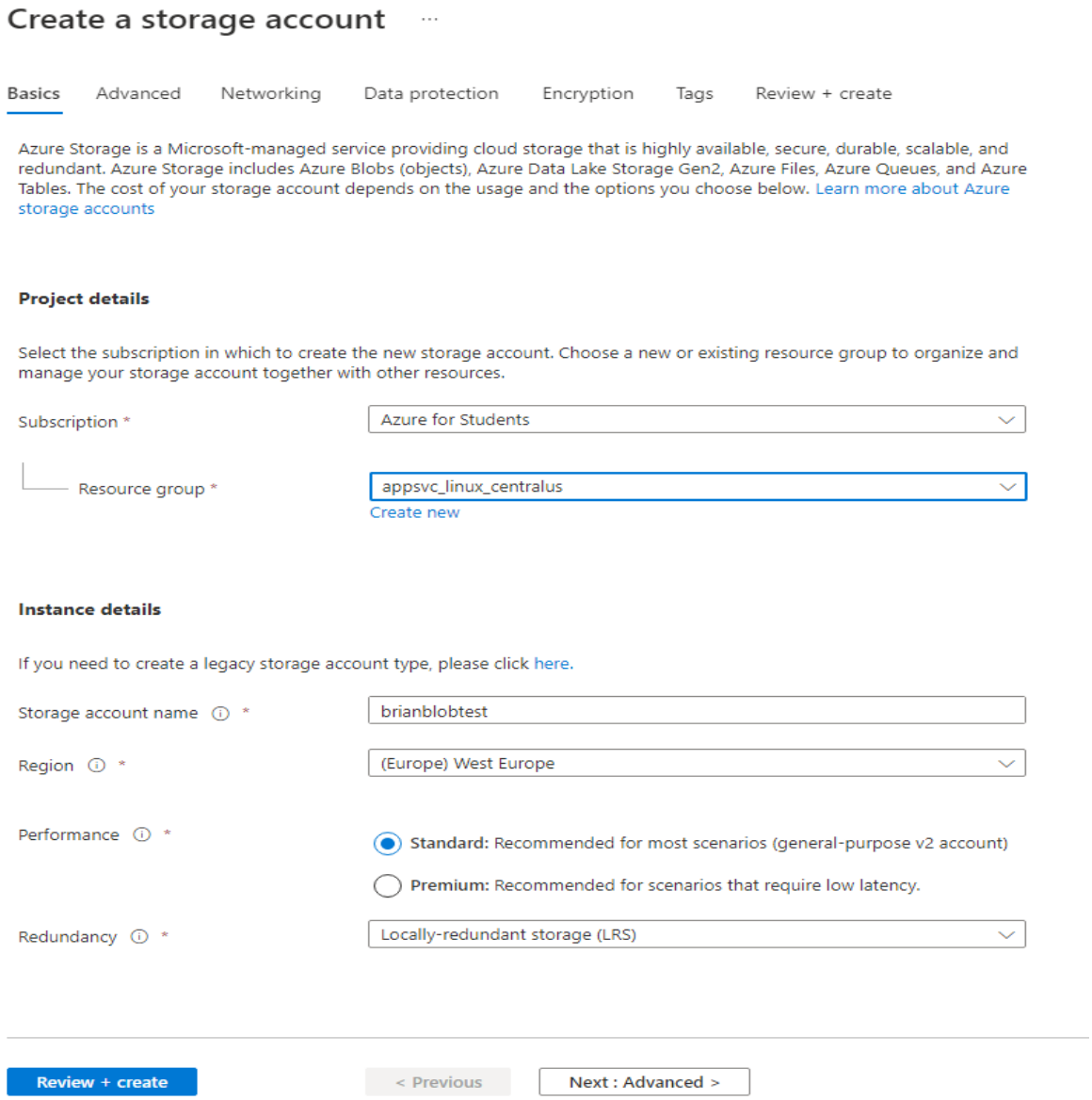


How to setup a storage account.

When you want to create a blob storage in azure u need to create a storage account. Make sure you choose storage account and not storage account (classic).



Now we need to create our account when u click create this screen will open.



Now all the options here we should already be familiar with from other services we created however the two at the bottom are unique to storage account so I will explain.

Performance: The performance option is pretty straight forward do you want high speed access to your files all the time or is it ok for a slower experience from time to time.

Redundancy: This option is a bit more complicated u get to choose between 4 different redundancy options. Here are the options in short
- **Locally-Redundant storage (LRS):** This is the most basic option that provides protection against server rack and drive failures. This is also the cheapest option.
- **Geo-redundant storage (GRS**) : This option makes sure that if something goes wrong a backup is ready in a secondary region.
- **Zone-redundant storage (ZRS)** : This option gives protection against datacentre level failures
- **Geo-zone-redundant storage** : This option gives you both the backup capability's or GRS and the protection of ZRS.

Below you can see these options represented in picture form with a more detailed explanation.

Single region redundancy

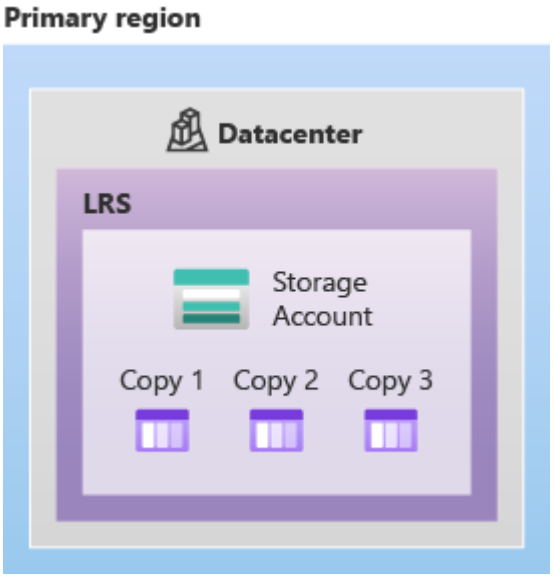
When using blob storage in a single region u have two redundancy option

Local redundancy storage(LRS) and Zero redundancy storage(ZRS).

LRS

Local redundancy storage replicates your data three times within a datacentre in the primary region.

LRS is the lowest cost option of the 4 options and offers the least durability compared to the other options. LRS protects data against server rack and drive failures. But lets say for example a massive fire breaks out in the datacentre all 3 copies will be lost forever since you wont be able to recover them in any way. To combat this risk you need to at least use one of the other 3 redundancy options.

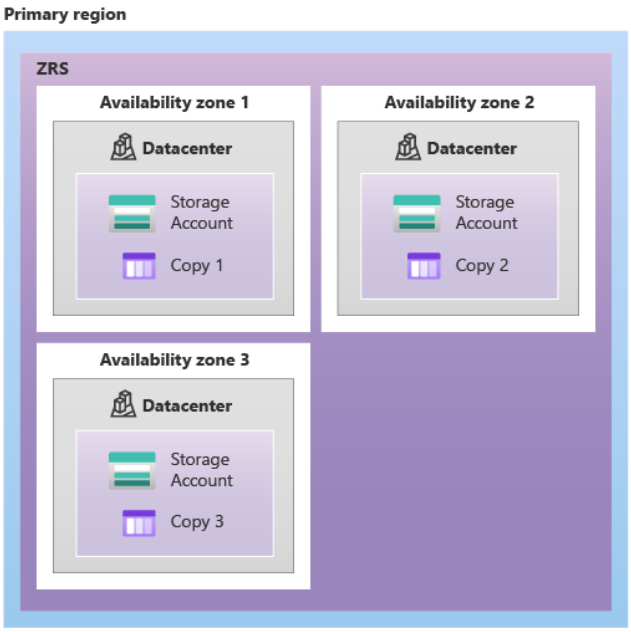


LRS is a good option when:

- Your data can easily be reconstructed if data loss occurs.
- If your data is restricted to a certain region or country because of government requirements. It can happen that if you pick the GRS or ZGRS option the data gets geo replicated in a region that goes against those rules.

ZRS

Zero-redundant storage (ZRS) replicates your Azure Storage data synchronously across three Azure availability zones in the primary region. Each zone has its own location with its own power source, cooling and network. So if one zone catches fire the other zones will still function and your data is safe.



ZRS is recommended when you require high availability of you data. ZRS is basically a upgraded version of LRS since u can still use ZRS under the same government rules as LRS because ZRS does not backup your data in different regions or countries.

Redundancy with a secondary region.

When using storage with a secondary region u have two redundancy option.

Geo redundancy storage(GRS) and Geo zone redundancy storage(GZRS).

For Data that requires high durability you can choose one of these two options. The secondary region is hundreds of miles away so even in the case of a complete region collapse your data is still safe in the secondary region.

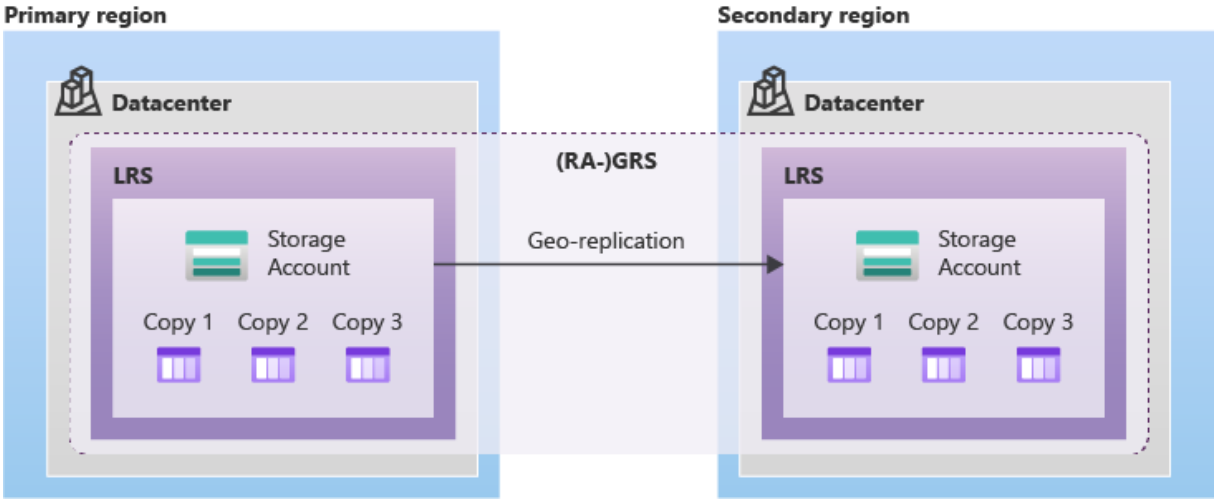
The secondary region is based on what primary region you picked when creating your storage account the secondary region can't be changed.

You cant write or read any of the data in the secondary region unless there is a failover to that region. For read access to the secondary region you need to configure your storage account to allow read access on the secondary region this is not on by default.

If a failover happens the secondary region becomes the new primary region. The data in the secondary region can miss some data compared to the primary region in case of a failover because both region sync asynchronously so it can happen that when the primary region fails the secondary region does not have the newest version of the data. At the moment the time it takes for the secondary to be updated after the primary region is less than 15 minutes.

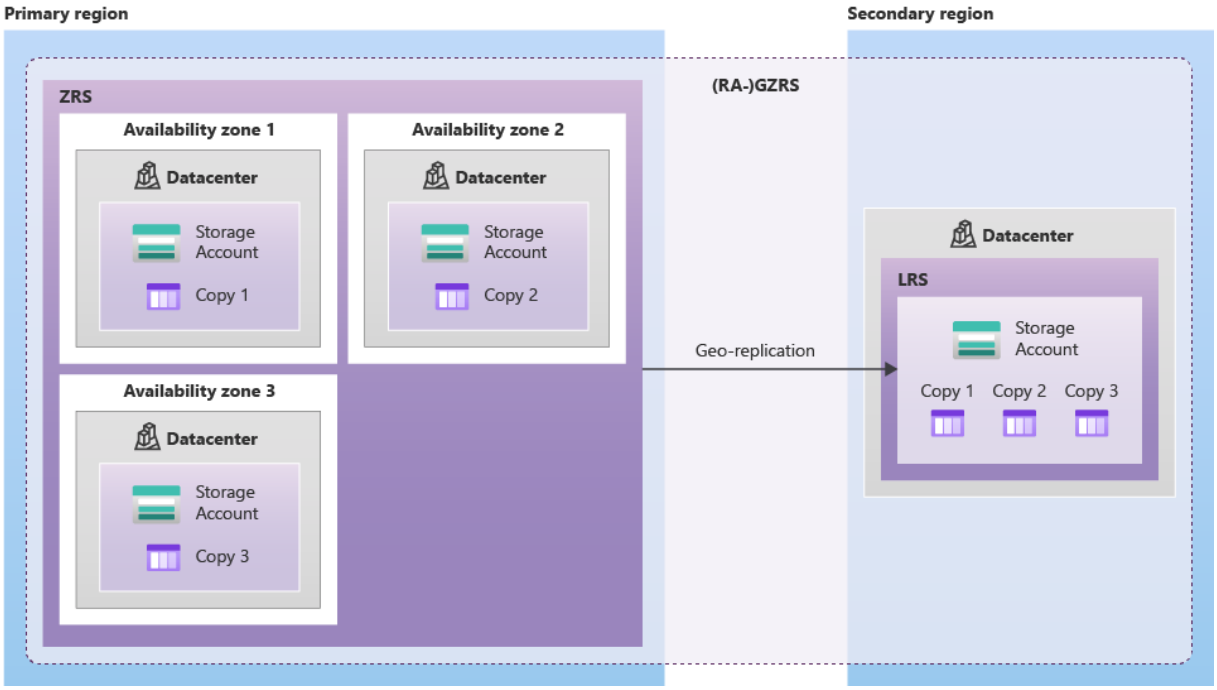
GRS

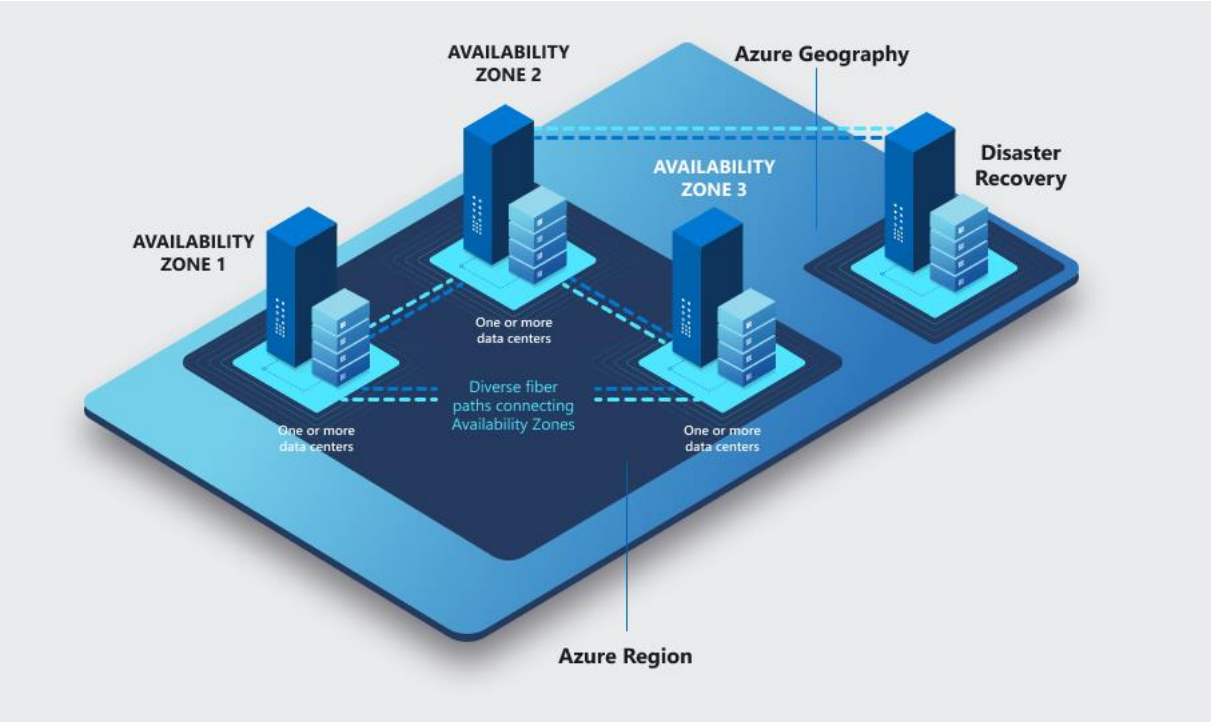
Geo-redundant storage (GRS) copies your data 3 times like LRS but also makes a backup of you data and copies in a secondary region. When you write data to the primary region first the data will be replicated in that region after that the data will be written to the secondary region and replicated after it has been written.



GZRS

Geo-zone-redundancy storage is ZRS and GRS combined. Data in a GZRS storage is copied across three zones in the primary region based on ZRS and then replicated in a secondary region based on LRS this option gives you everything maximum durability and excellent performance speed.





Ok now that we know what all the redundancy option do lets get back to creating our test storage account.

Advanced settings

Basics | Advanced | Networking | Data protection | Encryption | Tags | Review + create

Security

Configure security settings that impact your storage account.

Require secure transfer for REST API operations ⓘ ☑

Enable blob public access ⓘ ☑

Enable storage account key access ⓘ ☑

Default to Azure Active Directory authorization in the Azure portal ⓘ ☐

Minimum TLS version ⓘ Version 1.2

Data Lake Storage Gen2

The Data Lake Storage Gen2 hierarchical namespace accelerates big data analytics workloads and enables file-level access control lists (ACLs). Learn more

Enable hierarchical namespace ☐

Security

Require secure transfer for rest api operations: The secure transfer option enhances the security of your storage account by only allowing REST API operations on the storage account using HTTPs. Any requests using HTTP will be rejected when this setting is enabled. When you are using the Azure file service, connections without encryption will fail, including scenarios using SMB 2.1, SMB 3.0 without encryption, and some flavors of the Linux SMB client. Because Azure storage doesn't support HTTPs for custom domain names, this option is not applied when using a custom domain name. Connections via NFSv3 for blobs over TCP will succeed but will not be secured

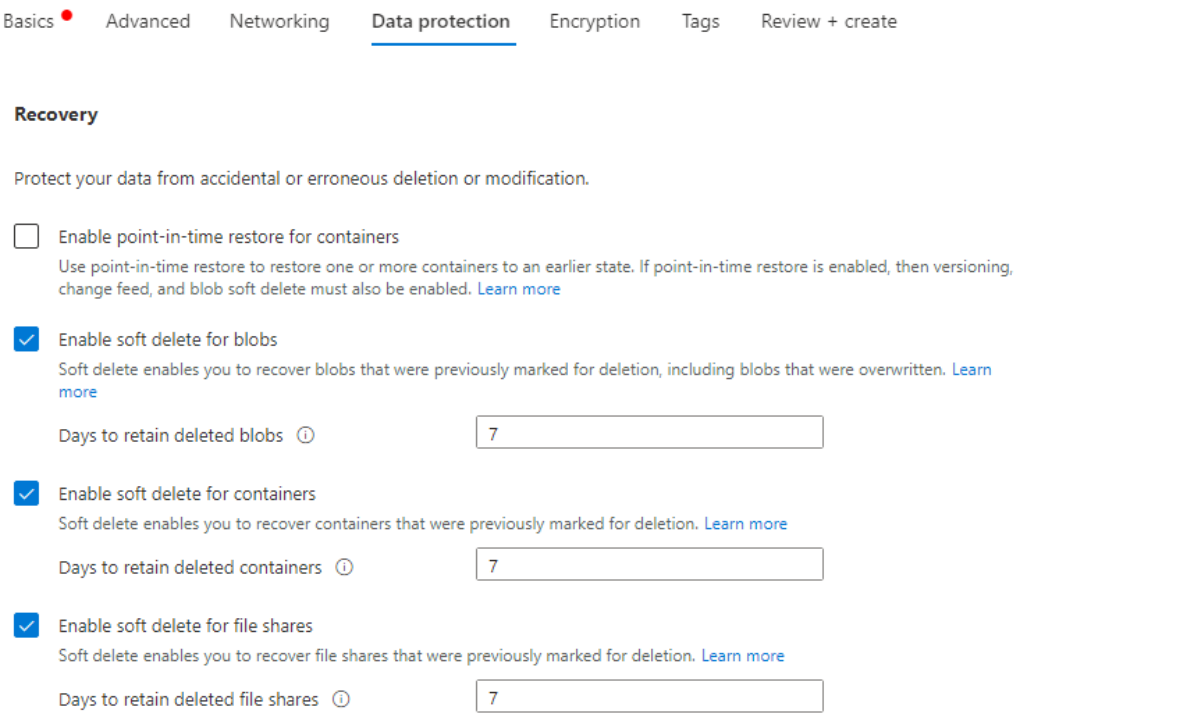
Enable blob public access: When blob public access is enabled, one is permitted to configure container ACLs (Access control list) to allow anonymous access to blobs within the storage account. When disabled, no anonymous access to blobs within the storage account is permitted, regardless of underlying ACL configurations

Enable storage account key access: When storage account key access is disabled, any requests to the account that are authorized with Shared Key, including shared access signatures (SAS), will be denied. Client applications that currently access the storage account using shared key will no longer work.

Default to azure directory: When this property is enabled, the Azure portal authorizes requests to blobs, queues, and tables with Azure Active Directory by default

Minimum TLS(Transport layer security version: Set the minimum TLS version needed by applications using your storage account's data

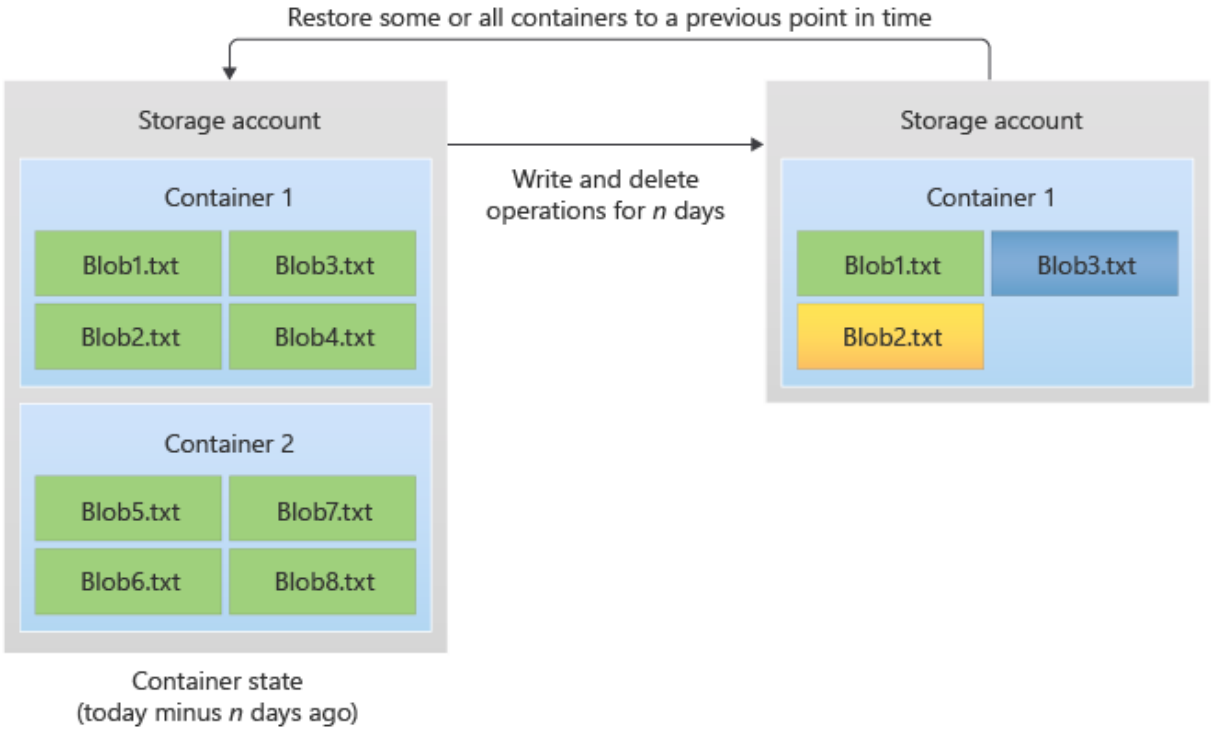
Data protection settings.



Recovery

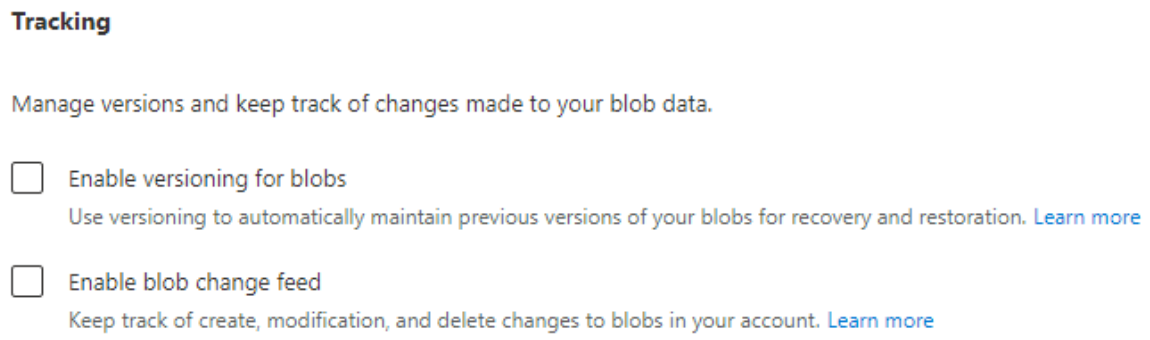
These settings determine what happens to your data after you delete or modify it.

Point in time restore for containers: This options means instead of only being able to restore single pieces of data you can restore entire containers with multiple pieces of data in one go. So in this example instead of restoring each blob individually you restore the entire container.



Soft delete: this option is very simple if u delete something it gets marked for deletion the time it takes for the object to actually be deleted is the time u specify in these options. Default is seven days.

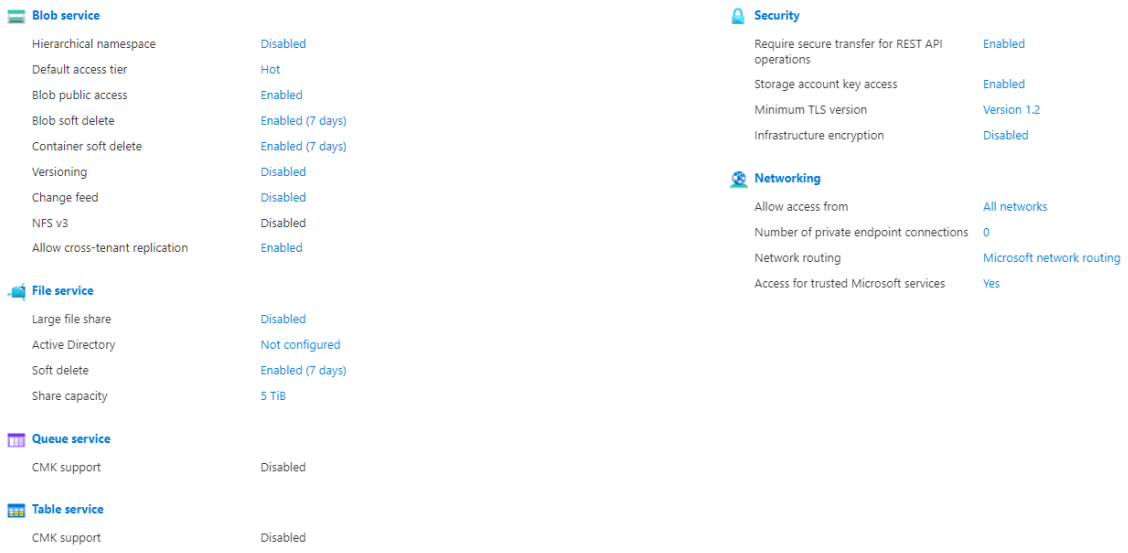
Tracking



The tracking options are pretty self-explanatory there are two options

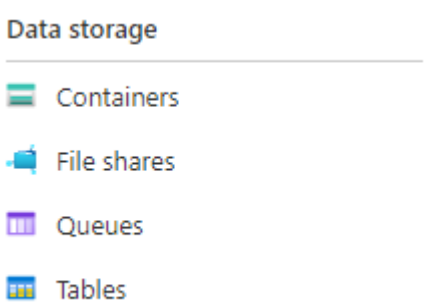
- **Versioning of blobs:** This option means the system will automatically maintain a previous version of you blobs for recovery and restoration purposes.
- **Blob change feed**: This option will automatically keep track of all create, modification and delete changes.

Now that we have gone tough what I think are the most relevant options for us its time to look at the storage account we created.



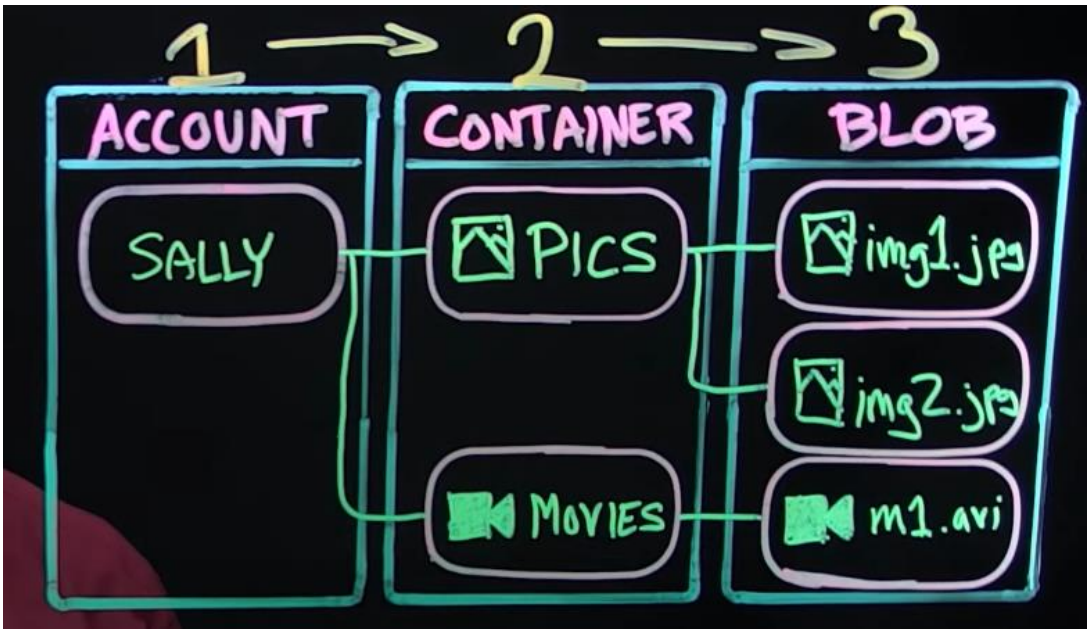
On the overview you can see all the setting you choose during the creation of the storage account here you can also choose to change a lot of the setting if you want to.

When we look at the sidebar we can see the four storage option that are available in a storage account I will now go into each of these options in detail.



Blob Storage

Blob storage can save all kinds of data. Blob storage is made up of 3 pillars that account, containers and blobs. Accounts can create containers and in those containers the user can save their blobs in this example sally has access or created the containers Pics and Movies and she can now add blob (files) to those containers these blobs are usually unstructured. In simple terms it's like your normal file system in windows.



Azure allows you to decide who has access to what container. Blob storage is useful when u need to share files or keep track of files with the people you work with. It is a way to centralize all your files in an organization in a safe and secure space that can be accessed from different locations.

Table Storage

Table storage is a storage accounts NoSQL database. It has key-value attributes but it does not follow a schema or has any relationships.

Table

A table is a collection of rows called entity's a table can store up to 500 TB of data and can process up to 2000 rows/s.

Entity

A entity is an object made out of a set of property's for example lets say we have a person entity that entity will be made out of the property's: "Person"(Partitionkey), ID(Rowkey), Timestamp , Firstname, Lastname and age for example. A entity can have a max size of 1MB and can have up to 255 property's this includes the mandatory (PartitionKey, RowKey and timestamp). A property is a name-value pair so you give a property a name and the cells that belong to that name all have a value of the same type.

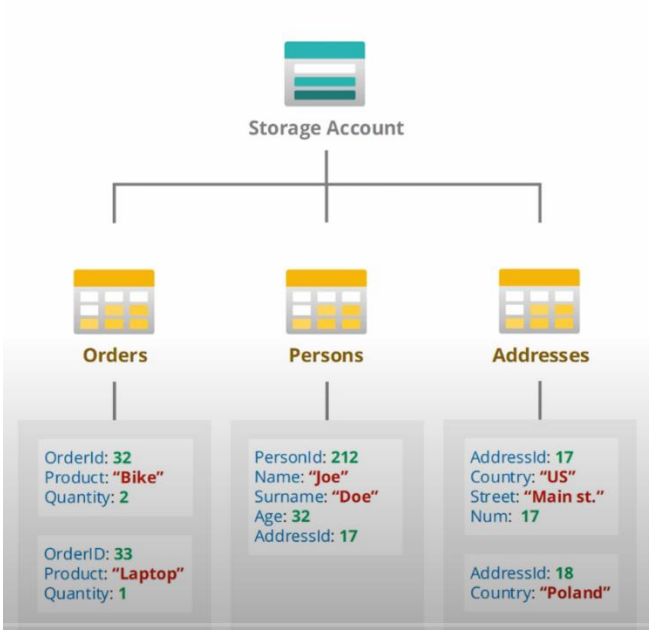


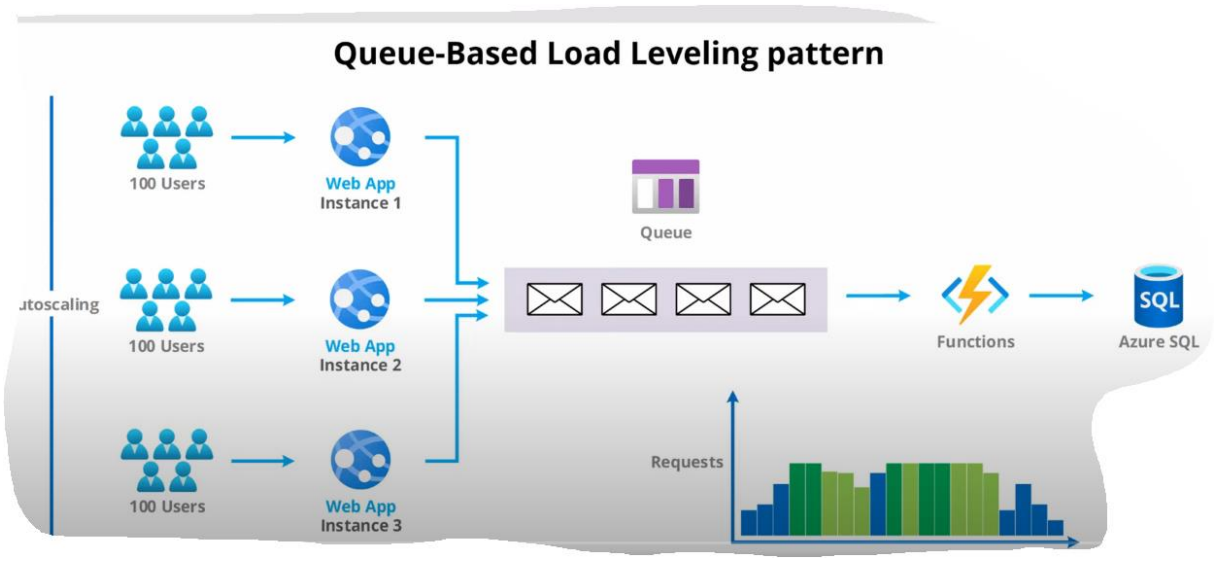
Table vs. Cosmos.

Attribute	Table Storage	Cosmos DB Table API
Latency	**Fast**, but no upper bounds	**<15ms** for **read/writes globally**
Throughput	**20,000** operations/s per account **2,000** operations/s per table	No upper limit per account **>10,000,000** operations/s per table
Global distribution	**Single region** with one **optional readable secondary read region**	Turnkey **global distribution** from one to 30+ regions
Indexing	Only **primary index** on **PartitionKey** and **RowKey**	**Automatic indexing**
Query	**Index** on **primary key**, and **scans otherwise**	**Queries use automatic indexing**
Consistency	**Strong** within primary region and **eventual** within secondary region	**5 consistency levels**
Pricing	**Storage**-optimized, very cheap	**Throughput**-optimized, expensive

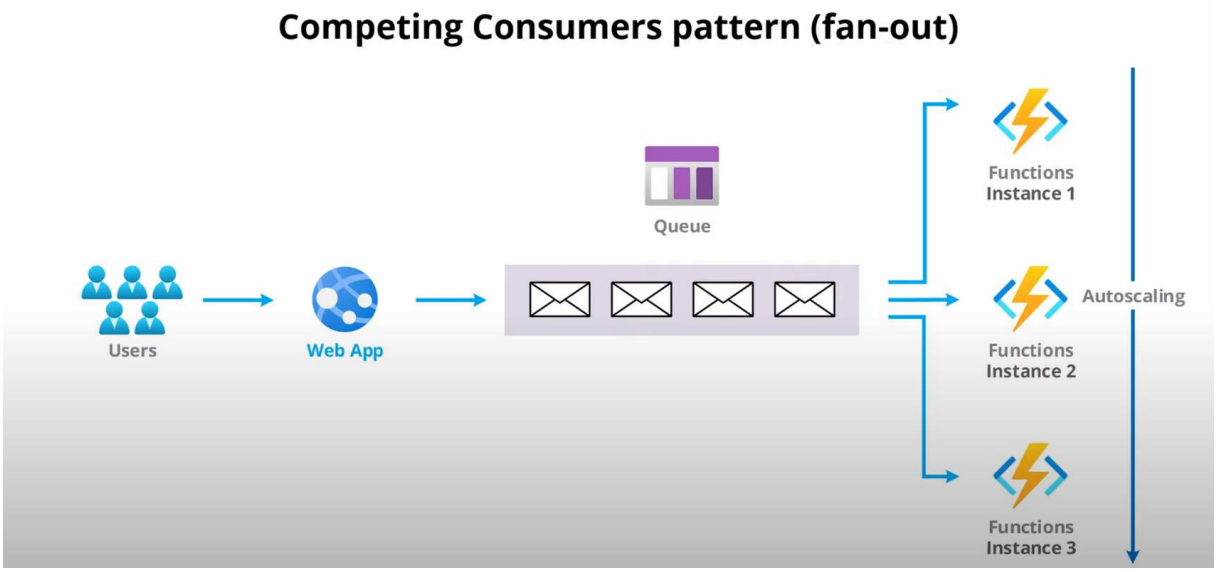
Queue storage

Queue storage is used to store a large number of massages and uses a FIFO rule.

Queue's are useful when for example you have an app that gets a lot of traffic and your database cannot handle it. Now the simple solution would be to just scale up your database. Another solution could be that you make a Queue between the app and the database that will send messages to the database at a steady rate that the database can handle.

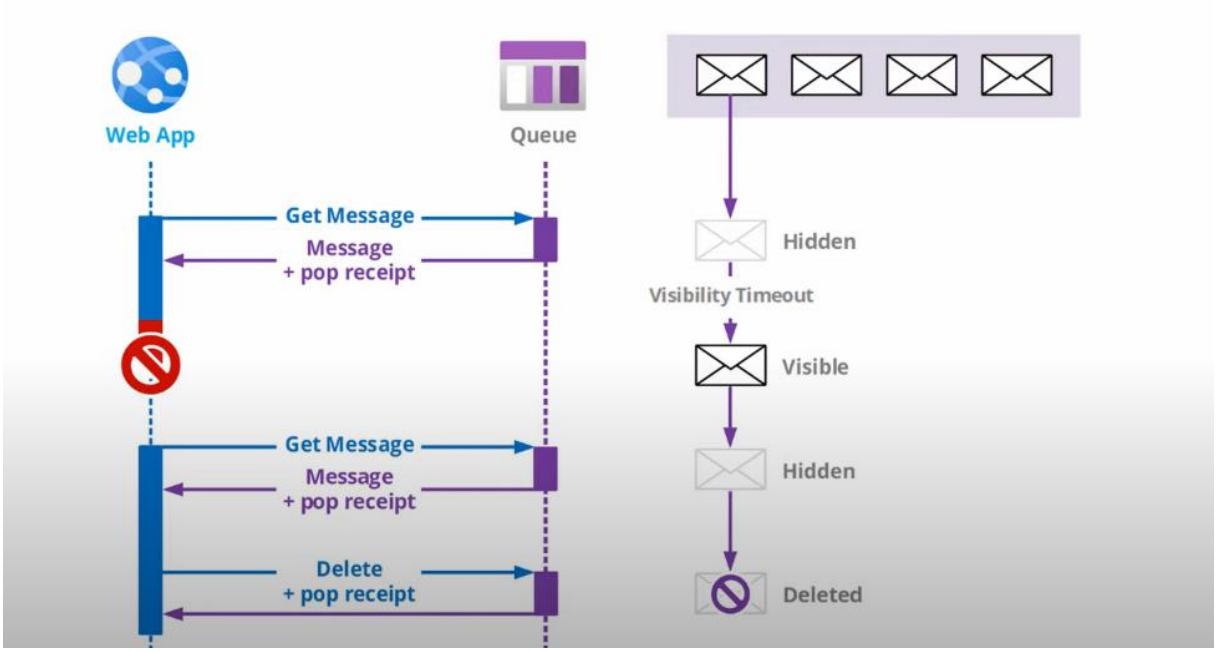


Another example use of a queue is a competing consumers pattern(fan-out). What this is your app sends messages to the queue and the messages get handled by functions in parallel after the queue the nice thing about this is the functions will scale based on the size of the queue.



Retry pattern

If you have an app and send a request to the queue to get a message that the queue does is makes this message hidden and returns the message to the app when a message is hidden no one else can see or use this message for the duration it is hidden. Now if the app uses this message but encounters a problem along the way so the message does not get fully processed the hidden message in the queue will become visible after a certain time to that our app can try and process the message again if it succeeds only then will the message be deleted from the queue.



File storage

One of the most useful uses for this service is for executing “Lift and Shift” operations on applications.

Cosmos DB

Azure Cosmos DB is all about scale of the amount of data you can store. Microsoft guarantees a 99.999% uptime and allows its users to quickly connect to any service it wants and use it all over the world. Cosmos DB is a NO SQL database so the user does not have to worry about any complexity that comes from using a traditional relations database.

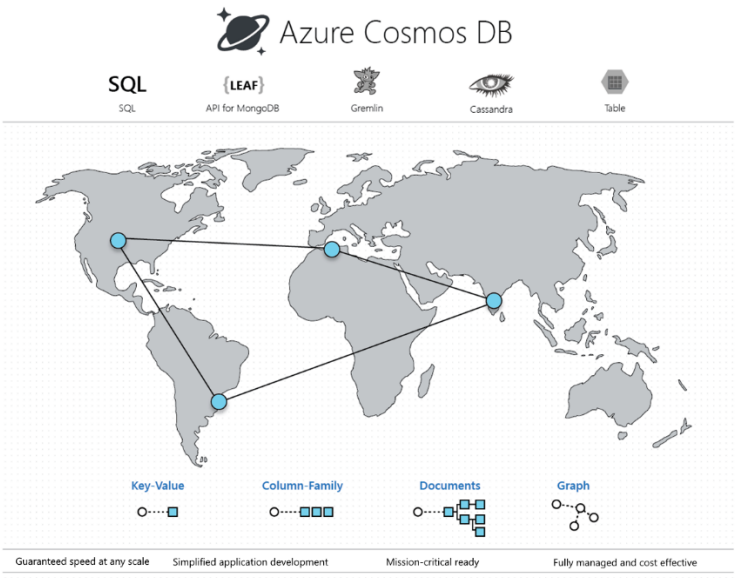
Key features

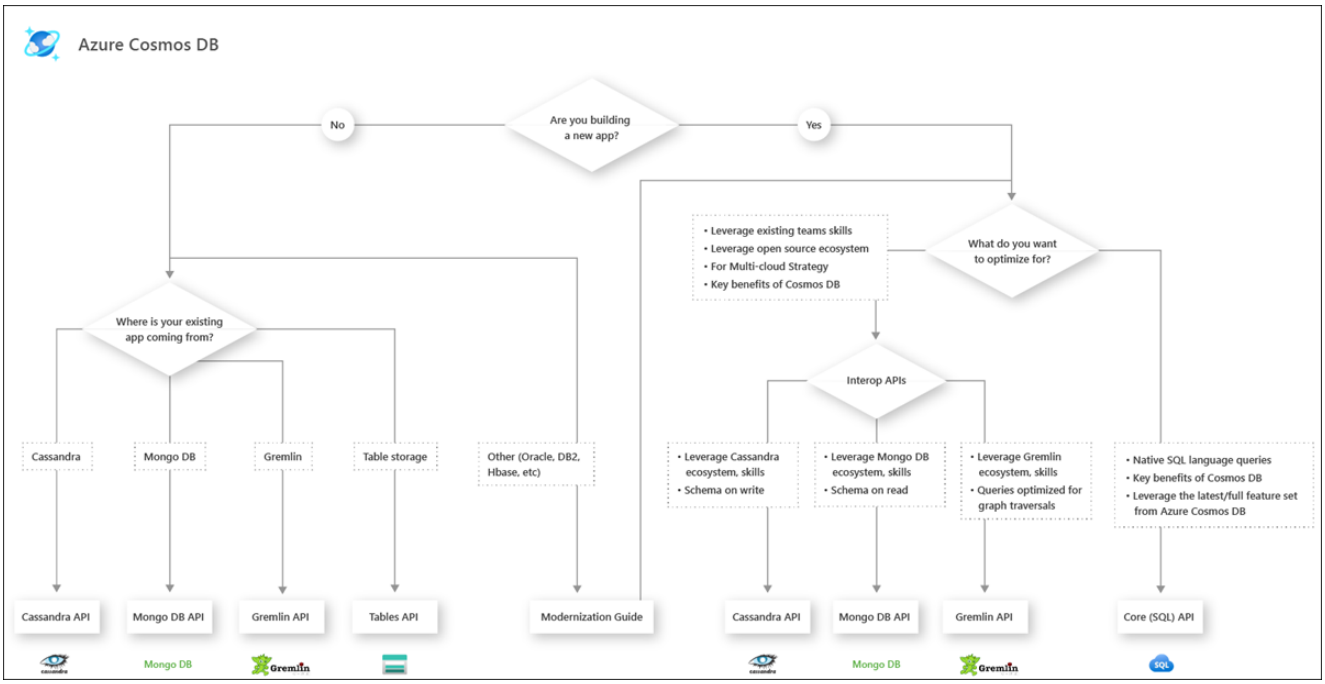
1. **Global Distribution**: Multi region distribution
2. **Regional presence**: Available in pretty much every region
3. **Always on**: 99.999% availability
4. **Elastic scaling**: Request per second can scale up into the hundreds of millions.
5. **Low latency**: under 10ms read and write request for 99th percentile
6. **Consistency options**: Choose the right balance between performance and consistent replication.
7. **No schema or index management**: schema-less service automatically indexes all your data, regardless of the data model

Api’s

When creating a cosmosDB you get to choose what API u want to use when its comes to working with your data these API’s are:

1. **SQL (a core API):** This is the default its basicly normal SQL so you can query your data just like you would with a normal database.
2. **Cassandra**
3. **MongoDB**
4. **Gremlin:** This one is specialized in graph data
5. **Azure Table Storage**

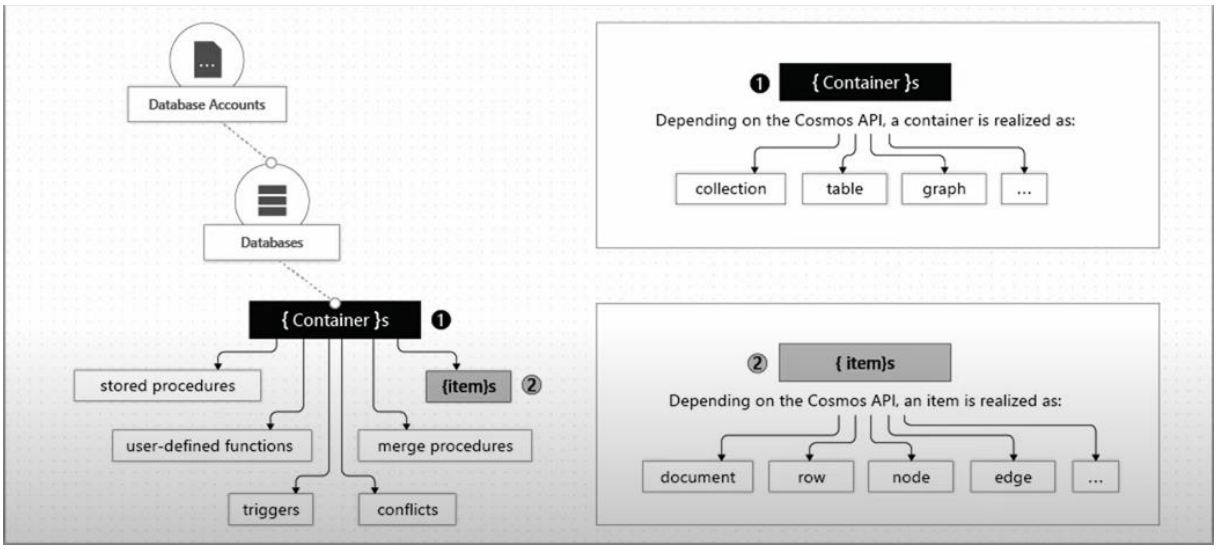




CosmosDB structure.

CosmosDB structure is as follow you start with a Database Account that account can have one or more databases and those databases have containers they are like tables in a normal database and in the container you can find you items these are you rows of data that you store in your database.

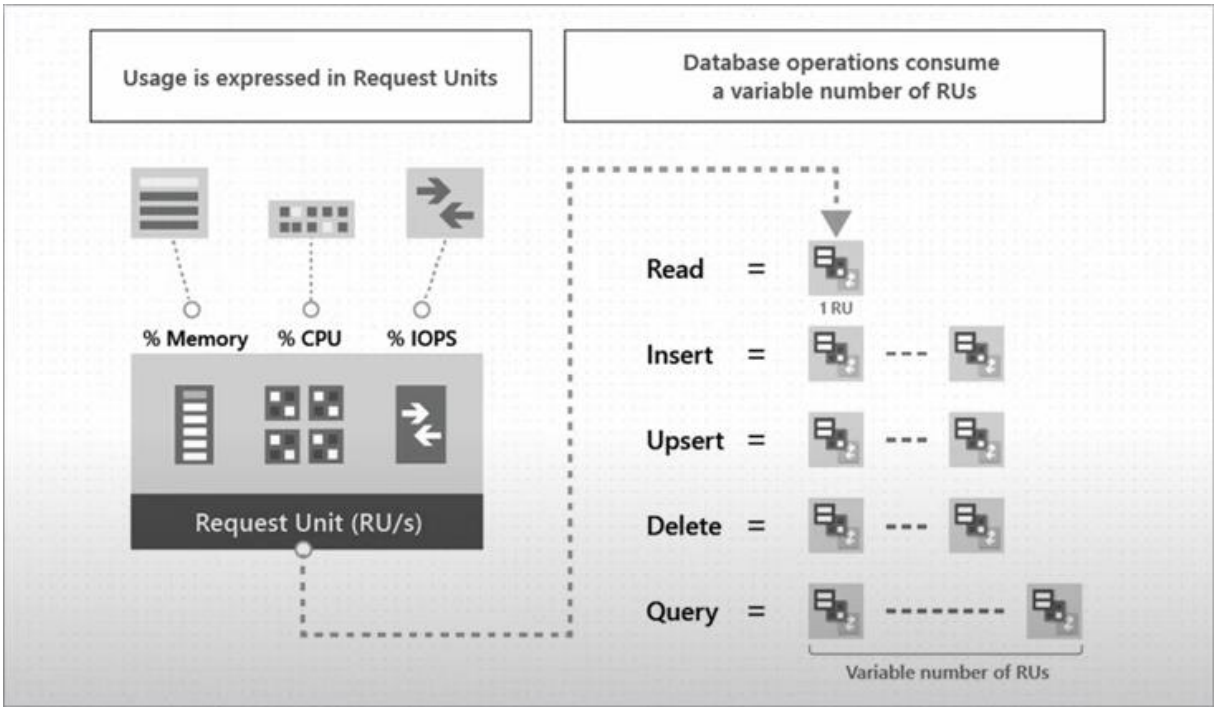
How your data is saved depends on the API you choose you can see this information in the graphic below:



Request Units (RU's)

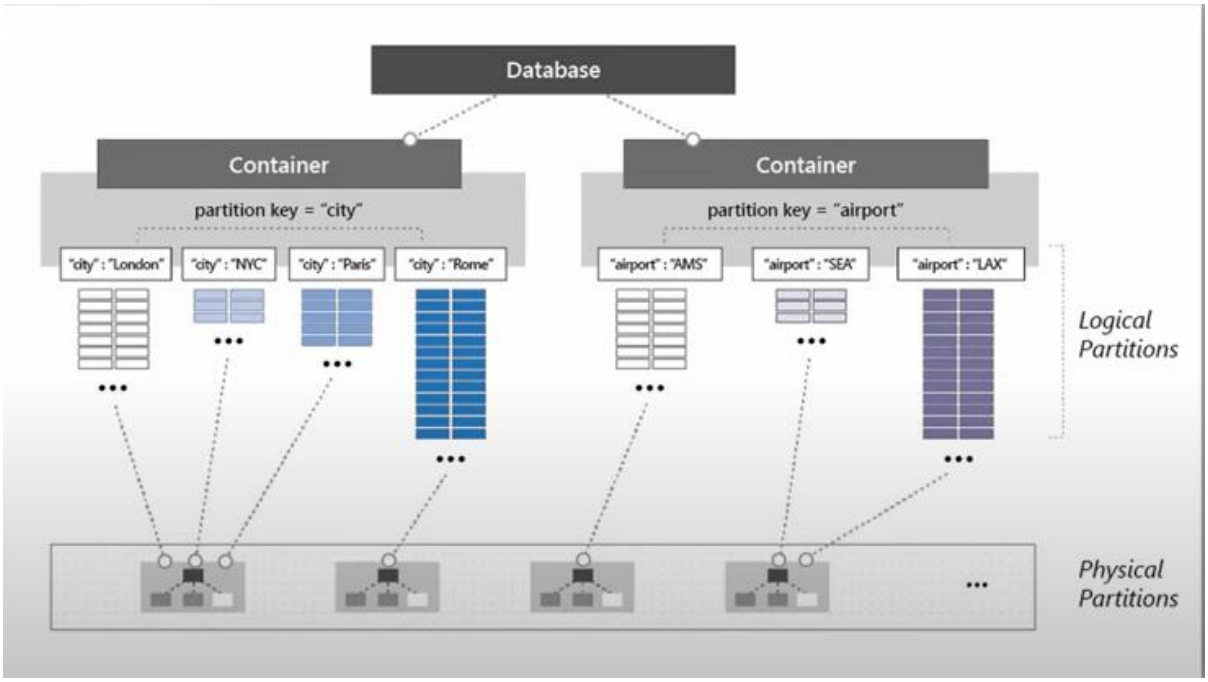
The cost of all database operation for CosmosDB is normalized and is expressed as Request Units (RU's).

So the amount of Request Units you use depends on that you are doing to your data. For example reading a single piece of data is 1 RU but inserting, upserting and deleting data cost multiple RU's depending on how big the changes are and querying data can cost a lot of RU's depending on what your querying.



Partitions

Partitions are important because u can change them after creating you container if u want to change it you would have to recreate your container and reupload your data.



API problems

I ran into some API problems with the CLI that I have yet to solve. When you create a logic app and add triggers and actions it will ask you to log in or connect to certain resources when you do this it will automatically create a API connection for you inside of your resource group. The problem is that if I try to run my logic app cli script for an empty resource group it will give an error that it cant find the API connections in that resource group and I have yet to find a way to create API connections in the portal or through cli.

Managed identity's

Here i wil recap my progress of researching managed identity's

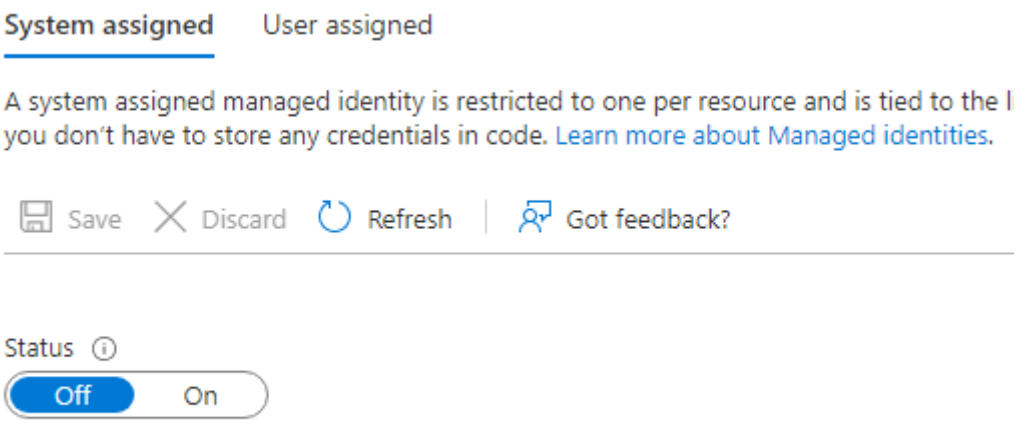
Fisrt watched the 48min video.

https://www.youtube.com/watch?v=rC1TV0_sIrM&ab_channel=JohnSavill%27sTechnicalTraining

Recap of the video.

Managed identity's fix the issue of having to use secrets or certificates to access service principles. A service principle is a representation of a app instance. So let's say I want to use a Netflix AAD in my AAD(Azure Active Directory) it will create a service principle in my AAD that points to the Netflix AAD. Now for the app in my resource to use my AAD we need to connect the two one way of doing this is by using secret keys or certificates but we don't want to do that so we will use managed identity's instead.

In a resource you can turn on a identity that azure is going to manage.

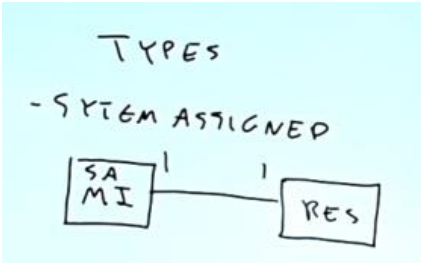


When you turn this on it will create service principle in you AAD with the same name as the azure resource. Now this service principle and Resource are linked with a shared life cycle so if I delete the resource the SP is also deleted. Now if we create a second resource we can give permissions to Resource 1 service principle to do things in this resource with RBAC(Resource based access control).

Now there are two different types of managed identity's

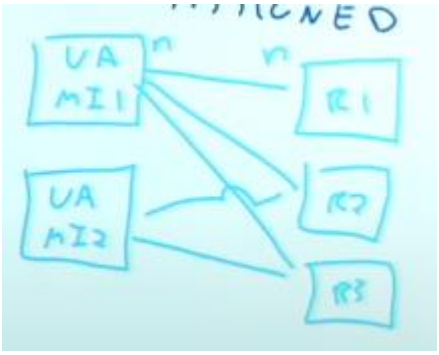
System assigned

System assigned managed identity's are managed identity's that can only have one resource connected to it and a resource can only have one managed identity. The life cycle between the two is shared if you delete the resource the managed identity is also deleted. System identity is a One to One relationship with resources.



User assigned

With this managed identity the user creates a managed identity as a separate resource. A user can create a managed identity in a AAD and then assign it to multiple resources. So the life cycle between the two is different if you delete a resource the managed identity stays. So user identity is a many to many relationship with resources.



So lets say for example I have a storage account with some blob storage I can now create a managed identity and give it permission to read the blob data in that storage account now any resource that I give that manged identity to has permission to read the blob data.

Used sources.

https://powerusers.microsoft.com/t5/General-Power-Automate/How-to-extract-body-of-calendar-events-without-html-coding-in/m-p/924654#M70568

https://docs.microsoft.com/en-us/azure/storage/blobs/point-in-time-restore-overview

https://www.youtube.com/watch?v=ZNuzmUKt6IE&ab_channel=JohnSavill%27sTechnicalTraining

https://www.youtube.com/watch?v=UzTtastcBsk&list=PLLGbQ0QttreqrItNw0y4sK4XSS1QsFxbZ&index=2&t=567s&ab_channel=AdamMarczak-AzureforEveryone

https://www.youtube.com/watch?v=R_Fi59j6BMo&list=PLLGbQ0QttreqrItNw0y4sK4XSS1QsFxbZ&index=5&ab_channel=AdamMarczak-AzureforEveryone

https://www.youtube.com/watch?v=FkekFVRTASM&list=PLLGbQ0QttreqrItNw0y4sK4XSS1QsFxbZ&index=4&ab_channel=ITProTV

https://www.youtube.com/watch?v=HSL1poL1VR0&list=PLLGbQ0QttreqrItNw0y4sK4XSS1QsFxbZ&index=6&ab_channel=AdamMarczak-AzureforEveryone

https://www.youtube.com/watch?v=JQ6KhjU5Zsg&list=PLLGbQ0QttreqrItNw0y4sK4XSS1QsFxbZ&index=7&ab_channel=AdamMarczak-AzureforEveryone

https://www.youtube.com/watch?v=Zm7vPBlq8Wg&list=PLLGbQ0QttreqrItNw0y4sK4XSS1QsFxbZ&index=2&ab_channel=ITProTV