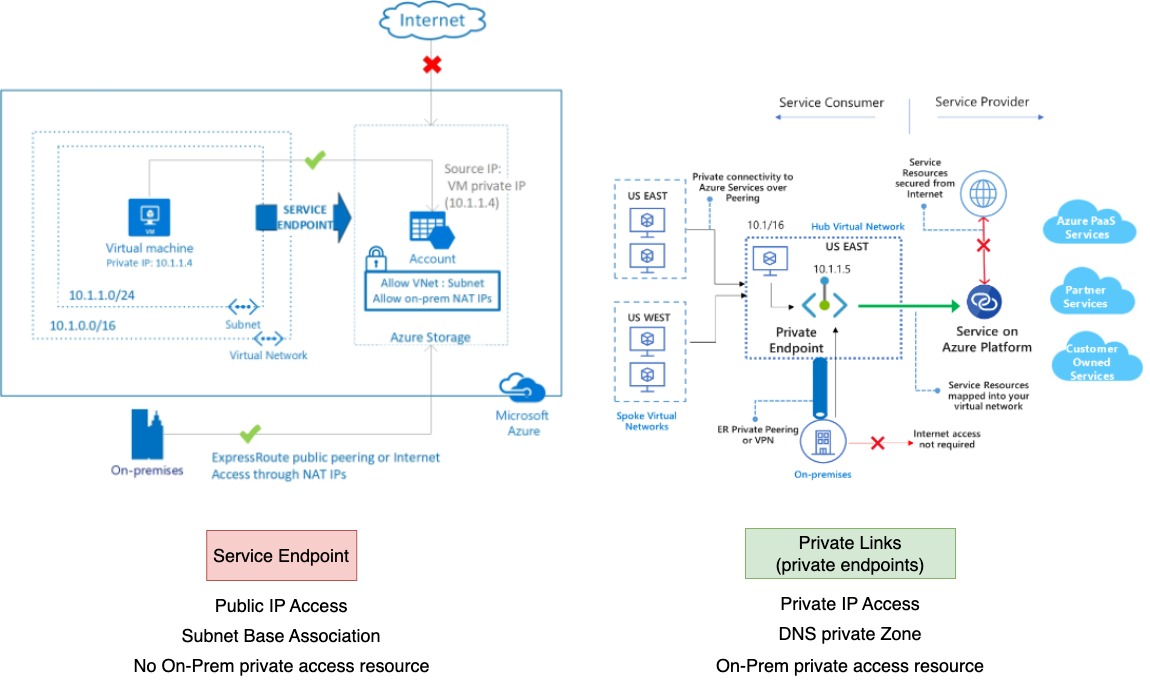
# AZ-104T00A – Administer Azure Storage

Good day everyone. Before we start today’s session, I would like to clarify the difference between Service Endpoint and Private Links, because last time, due to time constraint, it was not so clear.

The primary difference between these methods to restrict access is that while service endpoints keep PaaS resources outside your VNet, Private links injects the service directly into your VNet. Service Endpoints work by enabling a subnet, or subnets, on your virtual network to support Service Endpoints. Once this is done, you can configure your PaaS resource to only accept traffic from those subnets. There is no requirement to do any IP filtering or NAT translation; you tell the PaaS resource which vNet and Subnet to allow traffic from. When Service Endpoints are enabled, the PaaS resource sees traffic coming from your vNets private IP, not its public IP. Another advantage of using service endpoints is that traffic is routed to the Azure resources optimally. Even if you have UDRs on your vNet to route internet traffic back on-premises or through a firewall device, using a Service Endpoint means traffic is sent directly to the Azure Resource. The key difference between Private Link and Service Endpoints is that with Private Link you are injecting the multi-tenant PaaS resource into your virtual network. With Service Endpoints, traffic still left your vNet and hit the public endpoint of the PaaS resource, with Private Link the PaaS resource sits within your vNet and gets a private IP on your vNet. When you send traffic to the PaaS resource, it does not leave the virtual network. Another key difference with Private Link is that when enabled, you are granting access to a specific PaaS resource in your virtual network. That means you can control egress to PaaS resources. For example, if you wanted to, you could use NSG’s to block access to all Azure SQL databases and then use Private Link to grant access only to your specific Azure SQL Server. Unlike Service Endpoints, Private Link allows access from resources on your on-premises network through VPN or ExpressRoute, and from peered networks. You can also connect to resources across region. The implementation of Private link is more complex than service endpoints because it involves also Azure DNS private zone as we saw last time. Everything is clear?

Today we’re going to talk about the core storage services in Azure and about the different storage account types that are available. You also get to see a demonstration that shows you how to create a storage account in Microsoft Azure. We’ll then take a look at the different storage services in a little more detail. We're going to dive into storage, understanding what are the different types of storage requirement we may have and what are the services available in Azure to actually go ahead and meet those requirements. But then also what's special about storage, when I think about using it with a virtual machine and what I say virtual machine realize many other services build on top of that virtual machine foundational block. Virtual machine scale sets, AKS, node pools, app services, database services. Many of those build on this humble virtual machine. So, when we think about the storage available with a VM, that applies to many other types of service. They're all very different requirements. We have different workloads have needs for or sometimes it's ephemeral, it's just temporary. Maybe it's a cache of some kind. There's data stored in a durable, IE long-term survives, reboots, power off, etc. But maybe I have some local cache of it to improve performance. But I don't need to store it in a durable service as a power off type way because I can repopulate that data. When I start up so sometimes, we need ephemeral storage. It's not that long term durable. Sometimes hey I need it durable. I need to make sure it's always retained no matter what that type of event is.

Azure Storage Layers

Azure does not use traditional storage most of the time. There aren't big storage area networks that maybe the EMC o Netapp sans we're used to in data centers. Instead, there's this three-tiered architecture model based around storage stamps or cluster if you prefer. Azure divides up its hardware into racks with servers in them that are managed by certain controllers, and we can think about certain groups. Those racks make up a cluster. It could be a compute cluster that runs VMS. It could be a storage cluster that provides storage services. So, we're focused on the idea of a storage stamp or storage cluster. So, this uses this idea of layers, says a streaming layer, a partition layer and a front-end layer. Now you do not have to understand the details of this. I stress this only so you understand it's not using some SAN. It's built around an architecture that's designed for these huge hyperscale clouds, hyperscale performance type scenarios.

**Stream Layer**

We, have this stream layer. They're the bits on the disk that's responsible for distributing and replicating data across servers within the stamp. So, it's durable. It's never one copy of your data. A minimum there's three when we talk about Azure storage.

**Partition Layer**

Then we have the partition layer, and this understands higher level abstractions like a BLOB, a table, a queue. It gives you this scalable namespace into which you can interact

**Front-End Layer**

The front-end is responsible for actually taking those requests. They're stateless servers. They take the request, they might look up account names, they do some authentication, and then they route it through to the partition layer to actually go and get those bits. DNS is used for all of the namespaces. So, when you go and look at a storage account, what you'll see always is these endpoint names.

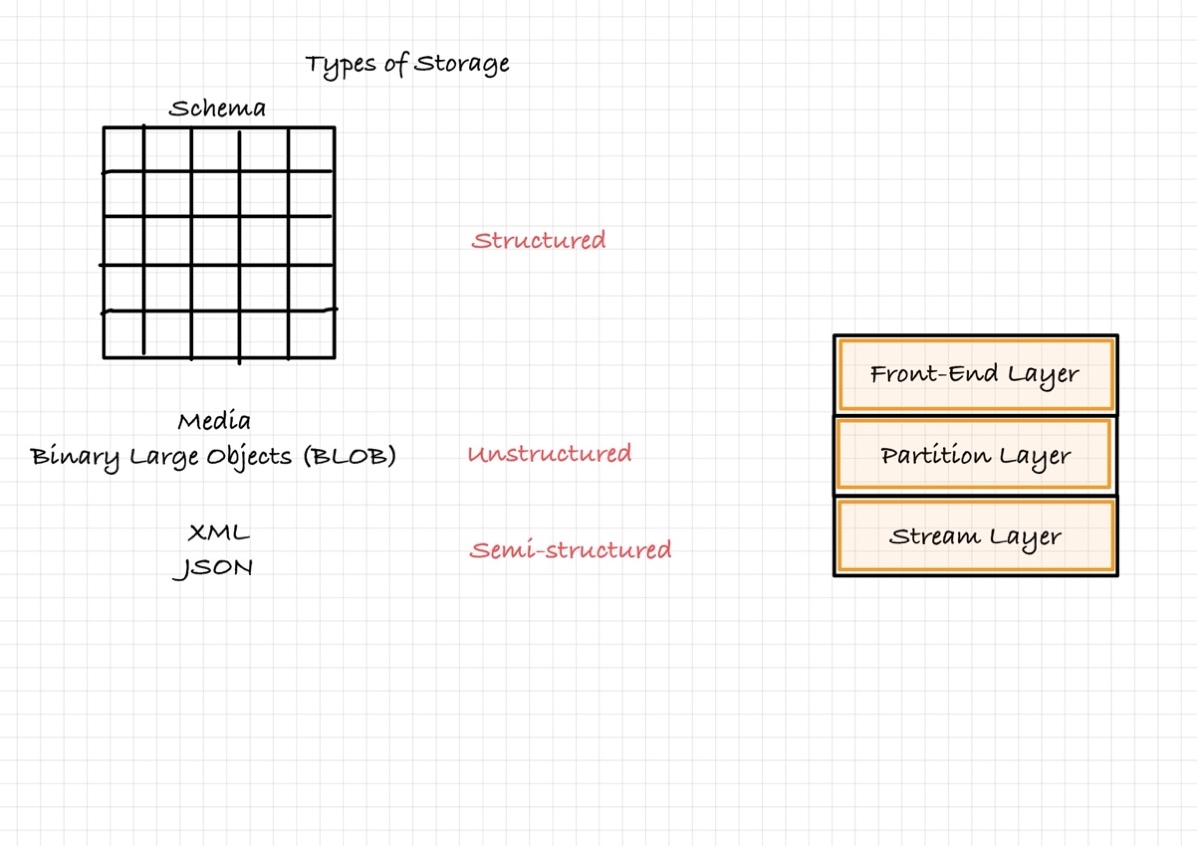
**Demo Endpoints**

Open Azure Portal and see endpoint names in a Storage account.

## Configure Storage Accounts

Because the Azure storage platform is designed for virtually all modern-day data storage requirements, it was built to offer several key benefits. First and foremost, all Azure storage services are durable and highly available. Built-in redundancy keeps data safe in the event of underlying hardware failures within the Azure infrastructure. In addition to this redundancy, you can also cross multiple data centers and even multiple geographical regions. This provides protection against natural disasters or local failures at the data center level. Because all data that gets written to Azure storage accounts is encrypted automatically, it is inherently secure as well. Azure also offers the ability to maintain fine-grained control over data access. Azure storage offerings are also scalable and widely accessible. As I mentioned earlier, the Azure storage platform is designed to support all modern-day data storage requirements. That being the case, it’s designed to be massively scalable. You can access Azure data from anywhere in the world over HTTP or HTTPS. There are also many client libraries available for Azure storage in many languages. This means you can access Azure storage using things like .NET, Java, Python, PHP, and many others. You can also access Azure storage through PowerShell scripting, through the Azure CLI, through the Azure portal, and through Azure Storage Explorer.

Implement Azure Storage

**Storage Types**

The types of data we might have vary greatly as well. There are three key types that most things fall into. We're used to the idea of structured.

**Structured Data is stored in a relational format that has a shared schema,** there is some well-defined blueprint of. These are the columns. This is the type of data can be stored in each column, integer, a string, whatever that might be. And then the table consists of number of those columns and it has entities records within it. And this is when we think of a database. So, we have this very structured type data that's very useful when I want to be able to interact and do queries against it, types of transactions. **Structured data is often contained in a database table with rows, columns, and keys. Tables are an autoscaling NoSQL store. Structured data can be stored by using Azure Table Storage, Azure Cosmos DB and Azure SQL Database. Cosmos DB is a globally distributed database service. Azure SQL Database is a fully managed database-as-a-service built on SQL.**

**Unstructured Data**

Then I also have things just like media files, a document, a PDF, Images, videos, audio tracks, blobs (binary large objects). These have no fixed structure, so we'd say they're unstructured. So, I need services to be able to handle just, hey, I want to store some stuff. You don't need to understand what it is, but I need to be able to store it. **Unstructured Data is the least organized. It can be a mix of information that’s stored together, but the data doesn’t have a clear relationship. The format of unstructured data is referred to as non-relational. Unstructured data can be stored by using Azure Blob Storage and Azure Data Lake Storage. Blob storage is highly scalable, REST-based cloud object store. Azure Data Lake is the Hadoop Distributed File System (HDFS) as a service.**

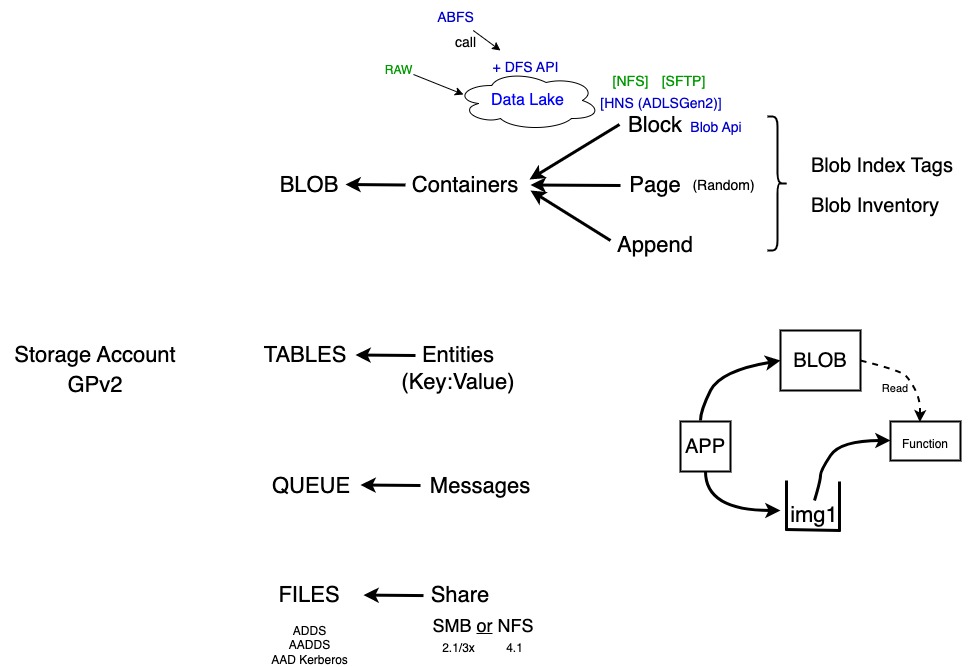
**Semi-structured Data**

And then we have almost an in between. We have things like XML, JSON and we think of these as semi structured. They're not adhering to some very strict schema in the traditional sense, but they do have some kind of structure. They might have some kind of entity, some kind of hierarchy. Typically, they're self-describing in terms of the actual data, the attributes they have.

So, we have these different types of data available to us and depending on what we're doing, we need different services to actually go ahead and support those. So, the different capabilities are needed across and to support the different types of data based on those applications. There really isn't one better type of storage. Hey if I want to store big media files, well I want an unstructured data store. If I have records about customers and sell orders, hey I want a structured data, I want to use a database. If I have some configuration type files that's probably a JSON, hey I want semi structured so it's not there. Is one better than the others?

Determine Storage Account Kinds

Before we get into the different types of storage accounts. Let’s talk a little bit about what a storage account is and what it’s used for. A storage account in Azure can be viewed as the container, so to speak, that houses all of our Azure storage data objects. A storage account can host blobs, Azure Files, queues, tables, and disks. When you provision a storage account, you are asked to provide a unique name for that storage account. This is necessary because data within a storage account is accessible from anywhere in the world. That being the case, the storage account namespace must be unique across the Azure landscape. There are several different types of storage accounts available. Each type offers different features, and each has a different pricing model. The first type of storage account is the general-purpose V2 account. A general-purpose V2 is a basic storage account that can be used, to host blobs, files, queues, and tables. Microsoft recommends using the general-purpose V2 storage account for most scenarios that require Azure storage. The next type of storage account we are going to take a look at here is the block blob storage account. Block blob storage accounts offer premium performance for block blobs and append blobs. You would typically use a block blob storage account for situations where high transaction rates are in play. Block blob storage accounts are also a good choice for scenarios that require low storage latency. File storage accounts are exactly what the name says they are. They are files only storage accounts. Because they feature high-performance characteristics, Microsoft recommends using these kinds of accounts for enterprise applications or for high-performing applications. The last storage account type touch on here is the page blob storage account. Page blobs are ideal for storing index-based and sparse data structures, such as operating systems, data disks for virtual machines, and databases. **You can’t convert a Standard storage account to Premium or viceversa.**

Explore Azure Storage Services

So, what are the storage account services I can actually use? We have different types for the different scenarios I might want. So, I create a storage account and I'm going to say again, for right now this is a general purpose V2 so I can do all of them because if it again was premium, it would be a particular, it would be files, it would be block-blob, it would be page BLOB, and that's it.

**BLOB**

The first service is BLOB. Now we've blocked the way I organize my content is I create containers. Like have multiple containers. And into those I create my different types of blob so it could be block. It could be page and it can be append. So, all three of those can be mixed in. They go into a container. Now, one thing I do want to stress with. It's flat, there is no hierarchy, although it might look like there were directories. You can go and create it, and it looks like there's a file system. There is not. That file system you see is a virtual directory. It's made-up of the name. The name has the folder name structure in it. So, operations when I'm listing a folder actually has to go and recurse and check all the names of the files. Realize that's really important in terms of performance and types of operations. You might do that, although it might look like, for example my block BLOB has this nice hierarchy of stuff. It isn't. It's all flat. So, if I did a rename operation or moved it between a folder, that's a copy and delete operation. It's very expensive. Which is why one of the things you can actually turn on, on top of block is thing called a hierarchical. This is optional hierarchical namespace called Azure Data Lake Storage Gen 2. That creates it as a true file system, POSIX style, ACLs. I can actually move things. I'm truly recursing through actual directories. Very efficient. And then once I turn on things like hierarchical name space, then I could turn on access using things like the NFS protocol. I can turn on things like the SFTP protocol, but that has to have that hierarchical namespace to enable me to do that. So, these will option that when I create my storage account.

**DEMO enabling Data Lake Storage Gen 2**

So, when I turn on things like this Data Lake over here, well this turns on even more types of APIs. There are things like the DFS API, which can then talk to the abfs drivers that's used by things like Hadoop. So why block is extremely powerful, extremely flexible in what I can do with that.

Obviously page we talked about, the whole big deal of page is it's these random types of operation and then we have append, as the name really suggests, I'm just adding to the end of it. Now one of the nice things we can do across these, there were some features like BLOB index tags. This enables me to add metadata and we saw this when we talked about governance, those role-based access controls. So, I can just add metadata to blobs. That could then be used maybe for inventory purposes. Maybe I want to use it as part of that attribute-based access control. I have this ability to have BLOB index tags, which is just this key value data that I can store along with the BLOB. And then there's different things I can do as part of that. I think the key is 128 characters, the value can be 256 characters and their maximums, and I think I have 10 index tags per block. So, it's a nice way to maybe add just metadata. Maybe I'll use it as part of some lifecycle management quickly finding things. It's just a cool thing. Over time I might get a lot of blobs, so the other thing I can do is having this BLOB inventory. I create a rule and what I say is, hey, maybe it's what types of blobs I care about. As you can see over here, I could restrict it. I could just, hey, do I care about blobs or do I care about containers? What do I want to include as part of that? I could absolutely. Add different prefix matches or only want certain ones. How do I want to export it? How often do I want to export it? And what this will do is then just in the background it will go. And create an inventory file for us. If you use this, you want to make sure you've got something to actually go and clean it up. So, we have things like BLOB index tags, and I also have BLOB inventory.

**Tables**

Then we have Tables. Table is all about key value pairs. It might look like, when you look at a table, then it's a schema, it's a database, but it isn't. Every single entity I create in a table can have its own columns which whatever values it wants, it doesn't have to adhere to any settings.

**Queues**

We have queues, which is messages. It's first in, first out. But it's not guaranteed. It should be, but it's not a guaranteed first in first out. If you need a guarantee first in first out, then it's saying like service bus, service bus can guarantee that first in first out. On a queue we put messages. These are fairly small pieces of data. Normally it might be used to drive something else, an event driven type compute. Hey I put a message on a queue and then it goes and triggers something else to then actually go and happen. For example, what would be very common, imagine BLOB and imagine this queue I could think about. Well maybe I have some application. When it actually does is. It's a BLOB. He writes an image, so it writes image1. But then what it also does is to the queue it just says hey I wrote image one and then there's some other function that is event driven. So it sees, hey, a message is being created on the queue, it takes the message off. So, this is the event. Oh well, image one, I'll go and read image one off the BLOB and then go and do something about it. So, it's very common that I'll use the multiple things together. I want you to go and do something. I'm going to dump a message and then I might go and do something else with that. That's super common thing.

**Files**

And then we also have files. So, this is going to be an SMB or an NFS 4.1 share. So, there's different things we can do. These are file-based protocols. So, what I do is I create a share. Now when I create that share it can be SMB Or NFS (it cannot be both) SMB it's 2.1 / 3.X, so I can do encryption and NFS, it's version 4.1. There were different capabilities, ones like create the share, obviously into that share I then create folders, I create files. The things we're used to. There's a number of capabilities we have as part of that, that we're used to. For example, I can have snapshots, different points in time, version of a file share. Then on Windows for example, if I go to the previous versions tab, I would see those snapshots to enable you to go back to those different points in time.

**DEMO Storage Browser**

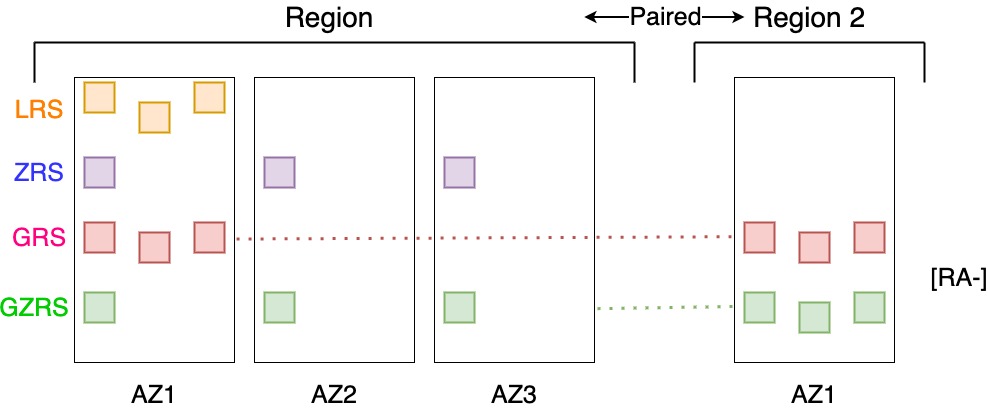
I want to stress about this Data Lake feature. Many, many times in any data flow today extract, transform, load, sources coming in. I've got a data pipeline that maybe transforms it, scrubs it, sends it somewhere else. Storage has got really cheap. And rather than getting data to come in, transform it and then store it, it's becoming more and more popular today to as it comes in that story in its raw format straight away. Because maybe in the future I want to transform it in a different way. Maybe in the future I want some different type of analysis done. So, the Azure Data Lake storage Gen 2 builds on top of BLOB, it provides a true data lake. We can use the POSIX ACLs or I can still use the Azure AD. I still have those capabilities available to me and I would think about if my workload is primarily analytics, I probably want to use data lake. If it's more, hey I have a bunch of media files. Want to store it? Well then maybe I'll use BLOB. With the block BLOB I still have the regular BLOB API for example that I can interact with, but now also what I can do with this hierarchical namespace once it's enabled. I'm adding this DFS API which I have these special abfs drivers that call the DFS API. So, my Hadoop for example, I don't have to do anything. And again, the whole point is my data pipeline. That initial raw data I just dump in the data lake whatever format it's in, I'll dump it in there and then I'll go and transform it and send it to a synapse or a SQL or cosmos, whatever it is. And some services today will even be able to interact with the data while it's still in the data lake. if I had Parquet files or Avro format files, I can actually connect to read it in place in the data. I can use it in maybe my analytics software. So, this is a hugely, hugely powerful thing.

**Hosting a WebSite**

I can host a website on BLOB. Now this is only static content, it has to be pre rendered. There's no server-side CPU processing available to this. But if I had HTML, images, cascading style sheet files, JavaScript embedded in the HTML, I've pre rendered it with something. Hey I just enable on the account level. Say I want to offer website, it creates a $web folder. I just put the stuff in. It's going to have a URL. I can hide that URL in my own DNS. I can create a cname. I can have an alias to this name and hey. It's just there now. We can use for sure Web apps, I think it’s just better since it uses the content delivery network. It's more geographically distributed and available. It ties into things like Azure functions really nicely. If I do want some server-side processing, there's a bunch of that's even just free. But it does exist. It's there. It's just an option in my storage account.

**DEMO Static Website**

Determine Replication Strategies

**Data Resiliency**

Now I can have different types of resiliency. This is the redundancy of my data. Now, this affects a number of different things. It might impact my ability to interact. I get read access to the data. There's a certain service level agreement about being able to actually interact. Then there's a very different SLA about the durability. I'm making sure no bits, no ones, and zeros get lost. The durability is going to be much, much higher than the SLA to be able to interact with the service, because you could think there could be problems with those front-end servers, there could be issues with DNS that's going to impact my ability to get and talk to the data, but the actual bits on disk are still safe, so there's very different SLAs. So, understand there's a big difference between those things, but we do have control of that resiliency. What is the redundancy of the data?

Now remember when we think about a region. A region is actually made-up of many physical data centers. And we often talked about availability zones, and I'm just gonna pretend for this instance, this region supports availability zones. So, there's a group of data centers that make up each AZ with independent power cabling networking. So, I can say, hey, this is AZ1, AZ2, and AZ3. Now, one of the types of data that we could have is:

**Local Redundant Storage**

We'll start with LRS Locally Redundant Storage. With LRS, there were three copies of the data, but they're all within the same storage stamp. Said my 3 copies. So potentially, if that whole data center went up in flames, I could lose my data. But more likely what could happen is there's some problem with power or cabling and I can't get to it for a period of time.

**Zone Redundant Storage**

Then I can think about there's ZRS. So, in zone redundant storage, there's still 3 copies. But now those 3 copies are distributed over the three Availability Zones in that region, so now there's an increased ability to get to it, even if AZ1 has a particular problem.

**Geo Redundant Storage**

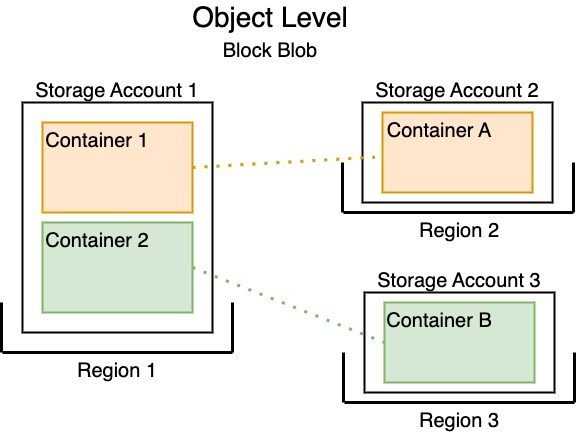
And then we have the idea of GRS. This is Geo globally redundant storage. Now remember. Regions are paired. So here this is region 2 and this is the paired region of region 1, so these are pairs, which means they're typically hundreds of miles apart. And region two it also has multiple data centers. I'm just going to join its AZ1 because for the purpose of what we're talking about, if I do GRS well, there's three copies of the data in the same storage stamp. And then it's asynchronously replicated to a storage stamp in the paired region where there were three copies in the same storage stamp there as well. Now when we say the same storage stamp, remember the storage stamp is made-up of fault domains, racks, so it is distributed over different racks in the same cluster. So, there's still a certain amount of blast radius protection there, but they're within that same building so to speak.

**Geo Zone Redundant Storage**

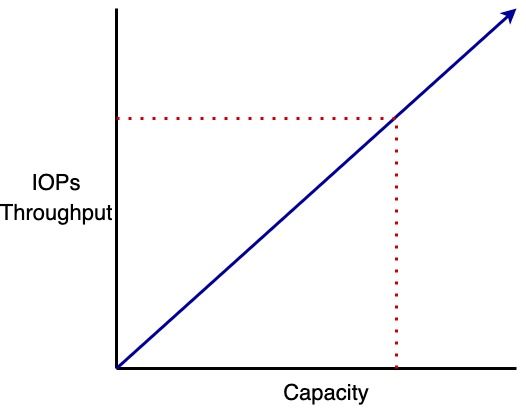
And the next thing we can do is combine. So, I can say GZRS. It's zone redundant so it's spread over the three AZ's and it's then asynchronously copied, but on the other side it's still in a single storage stamp. So those are the key combinations we have.

**Read-Access**

Now one of the optional things I can turn on is when I have the GRS or the GZRS there is the ability to say hey I want read access, it's optional. I don't have to have this, but if I turn this on, what this now lets me do is there's going to be a second set of endpoints for the BLOB, the queue, the table service is not files, files I can't do read access to the replica. There'll be a second set of endpoints, so my application could actually go and read. When there’s a failover, the endpoints the DNS changes to point to this copy of the data. But ordinarily, hey, it's pointing here. So, we do have a different job ability now. Those are the paired regions. I have no control. There's nothing else I can do about these.

**Object Level Replica (Block Blob only)**

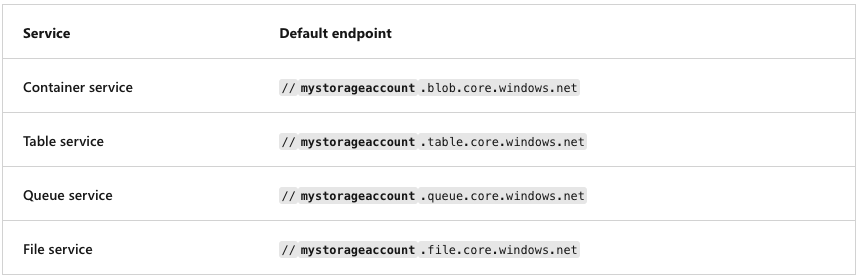
Well, there is something I can actually do for my block blobs, which is very, very commonly used, you're going to see this as probably one of the key services we ever leverage. We actually have a different replication option. So, one of the things I can do with block BLOB is thing called object level replication. I can do object level and if I think of the storage account, so I've created a storage account. Blobs, live in containers. So, I might have for example a container one in the same storage account and a container 2. Remember also that the storage account lives in a particular region, it's a region 1 and I want to point out this is block-blobs. Only block BLOB. Now what object level replication lets me do is hey look there are other regions, Region 2 and Region 3 and I created a storage account in this region Storage account 2 and Storage account 3. I created a container, container A and container B. I can configure a container level replica. Hey I want you to replicate to this, but I want this other container to replicate to there. It's not having to be the paired regions. I have complete control of this now, so this is a very powerful capability for block BLOB that I can now break away from those paired regions and I can essentially just have my own replication however I want. It's only to one. Each container can go to one other container. I can't do multiple chains of replication or multiple targets, it's a one to one. There's different API supported, so there's different ways to interact. There's restful APIs, it could be for something like files I can do SMB, I can do NFS for BLOB. There's different APIs exposed depending on which services I exposed. So, there's different ways to interact.

Money

So, money, everyone likes money. Standard performance is consumption based, so general purpose V2 I pay the amount of data I'm storing. I pay for the number of transactions and that's going to be common across all of them. But that's the key point. Hey, I pay for what I actually consume, and now that amount will vary. Is it hot? Is it cool? Is it archive? But it's always the amount I'm using. For premium, it's different. For premium, it may be based on the amount of space you provision and the reason for this, again, if I think about there's really multiple dimensions I care about, there's yes capacity, but in addition to capacity, there's also performance. So, if I think about the idea of capacity and then we add things like IOPS and throughput. How many operations can I do well depend on the size of the operations, what's the amount of throughput, megabytes per second supported. And what we see is, hey, as my capacity goes up, my amount of IOPS and throughput goes up as well. So, it's possible in the premium SKUs I might pick a certain size, not because I need to store that much data, but I need that much performance. So, when I look at these services like files premium and Page blob premium, when I create that share with files premium, I say the size, that's the provision size. I pay for that provision size. Doesn't matter how little I might write to it because I'm getting the benefit of the performance of that size. Sometime for different services, and check the pricing calculator to make sure you understand this, you might have a provisioned capacity, doesn't care how much you've actually written because maybe you created it at a certain size because I need this IOPS or this throughput. So that's why there can be these differences between them, and that's very common for files. In SMB, I need a certain performance for my application, it's using that file-based protocol or in my page blobs, this is storing a disk, I need to have huge IOPS or high throughput. It's not about the amount of data, it's the performance I care about. Managed disks, which we use for our virtual machines and some other workloads, they're always based on the provision size. If it's a standard hard disk, a premium SSD, a standard SSD and ultra-disk, I pay for the size of the disk, not the amount of data I write to it, and then sometimes I pay additionally or differently based on the performance. Premium SSD V2, ultra, I individually tuned the IOPS and throughput I want dynamically at any time. And also, don't forget about those operations. Don't forget them, if I turn on GRS, remember it has to copy the data to the other region. So, I pay for those operations as well.

So, if we go and look, this is the pricing, OK, don't forget it's not just the amount of data you're writing. What is the number of operations, write operations, read operations you're performing? That's going to be important thing to understand. Additionally, and there's small print, additional cross region data transfer network charges may apply for each storage access. Well, that's basically saying is, hey, if you've got GRS turned on, if I write a bunch of data that has to then replicate that over the network, I'm going to pay network egress charges. It's going outside of the region. I'm going to pay network egress, so make sure you understand all of those things apply. So, when I go into the pricing calculator and I start plugging in my numbers. Especially if you're a very high operational workload. Don't neglect the operations, don't neglect those network things, because that's going to come into play as well.

Access Storage

Every object you store in Azure Storage has a unique URL address. Your storage account name forms the subdomain portion of the URL address. The combination of the subdomain and the domain name, which is specific to each service, forms an endpoint for your storage account. Let's look at an example. If your storage account name is mystorageaccount, default endpoints for your storage account are formed for the Azure services as shown in this slide.

You can configure a custom domain to access blob data in your Azure storage account. **Direct mapping** lets you enable a custom domain for a subdomain to an Azure storage account. For this approach, you create a CNAME record that points from the subdomain to the Azure storage account as you can see in this example.

Secure Storage Endpoints

Storage accounts are managed through Azure Resource Manager. Management operations are authenticated and authorized using Azure Active Directory and RBAC. Each storage account service exposes its own endpoint used to manage the data in that storage service (blobs in Blob Storage, entities in tables, and so on). These service-specific endpoints are not exposed through Azure Resource Manager; instead, they are (by default) Internet-facing endpoints. Access to these Internet-facing storage endpoints must be secured, and Azure Storage provides several ways to do so. The **storage firewall** allows you to limit access to specific IP addresses or an IP address range. It applies to all storage account services (blobs, tables, queues, and files). For example, by limiting access to the IP address range of your company, access from other locations will be blocked. Service endpoints are used to restrict access to specific subnets within an Azure VNet.

**DEMO:** Open Azure Portal. Under the Storage Account Blade select Networking and in the tab Firewalls and Virtual Networks select “enabled from selected virtual networks and IP addresses” to show the options.

In some scenarios, a storage account is only accessed from within an Azure virtual network. In this case, it is desirable from a security standpoint to block all Internet access. Configuring **virtual network service endpoints** for your Azure Storage accounts allows you to remove access from the public Internet and only allow traffic from a virtual network for improved security. Another benefit of using service endpoints is optimized routing. Service endpoints create a direct network route from the virtual network to the storage service. If forced tunneling is being used to force Internet traffic to your on-premises network or to another network appliance, requests to Azure Storage will follow that same route. By using service endpoints, you can use direct route to the storage account instead of the on-premises route, so no additional latency is incurred.

**DEMO:** Configuring service endpoints requires two steps. First, from the virtual network subnet, choose **Microsoft.Storage** from the **Service Endpoints** drop-down menu. This creates the route from the subnet to the storage service but does not restrict which storage account the virtual network can use. To update the subnet settings, you should choose **demo-storage-vnet01** from the **Virtual Networks** blade. Then go to **Subnets** in the left pane under **Settings**. Click **Subnet-01** to access the subnet settings. The second step is to configure which virtual networks can access a particular storage account. From the storage account blade, click **Firewalls And Virtual Networks**.

Demonstration – Secure a Storage Endpoint

Create a storage account using PowerShell:

Get-AzLocation | Select Location

$location = “switzerlandnorth”

$resourceGroup = “Demo-Storage-AVQ”

New-AzResourceGroup -Name $resourceGroup -Location $location

New-AzStorageAccount -ResourceGroupName $resourceGroup -Name “stgavqdemo0101” -Location $location -SkuName Standard\_LRS -Kind StorageV2

## Configure Blob Storage

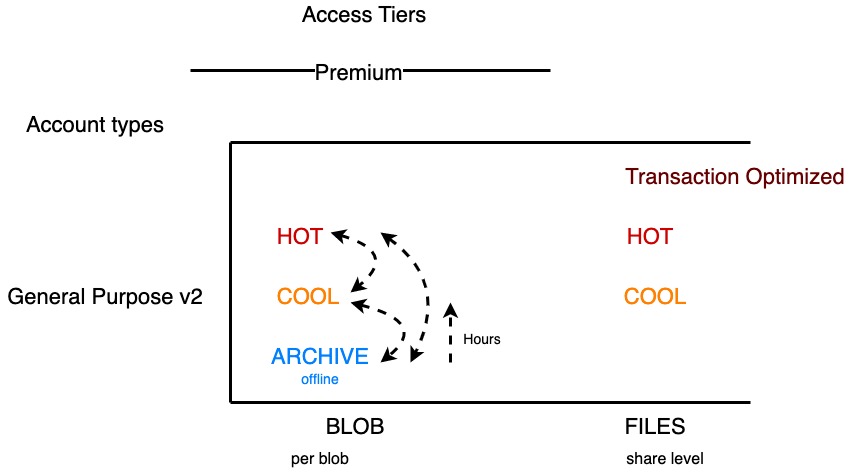
Implement Blob Storage

Azure Blob storage, as I mentioned earlier, is an object storage solution in Azure. It’s optimized to allow the storage of massive amounts of unstructured data, including text and binary data. You would typically use blob storage to host images and documents that you wish to serve up to a web browser. Think images on a website. Blob storage is also used when you wish to stream video or audio or to store log files. Organizations will also often use blob storage to store backup data, archive data, and data that needs to be analyzed by some on-prem process or Azure-hosted process. For accessing objects in blob storage, you can use HTTP or HTTPS. You can also access blob storage objects through the Azure storage rest API, through Azure PowerShell, through the Azure CLI, or through an Azure storage client library such as Java, PHP, .NET, or several others. There are three types of resources that you should be familiar with when discussing blob storage. They include the storage account that hosts the blob storage, containers within the storage account, and the blobs within those containers. The image on this slide depicts the relationship between these resources. The storage account creates the unique namespace in Azure that you use to access your data. When you access your data, you’ll use a combination of the storage account name and the Azure Storage blob endpoint that you are trying to access. Together, these two pieces form the base address for the objects that reside in the storage account. There are actually three types of blobs that Azure storage supports. They include block blobs, append blobs, and page blobs. Block blobs can contain up to about 190.7TiB of text and binary data. They are called block blobs because they consist of blocks of data that can be managed individually. Append blobs are similar to block blobs as they, too, consist of blocks of data. However, append blobs are optimized for append operations. This makes append blobs a good choice for logging data from virtual machines. Page blobs are used to store random access files up to 8 TiB in size. You would typically use page blobs to store VHD files, which would serve as disks for Azure virtual machines.

Create Blob Containers

Containers within a storage account are used to organize the blobs within the account. You can view containers like directories in a file system. You can create an unlimited number of containers within a storage account, and each container can store an unlimited number of blobs. Storage accounts support an additional access control mechanism that is limited only to Blob Storage. By default, no public read access is enabled for anonymous users, and only users with rights granted through RBAC or with the storage account name and key will have access to the stored blobs. To enable anonymous user access, you must change the container access level. The supported levels are as follows: **Private**, selecting which, only the storage account owner can access the container and its blobs. No one else would have access to them. **Blob**, with this option, only blobs within the container can be accessed anonymously. Or **Container**, selecting which, blobs and their containers can be accessed anonymously. You can change the access level through the Azure portal, Azure PowerShell, Azure CLI, programmatically using the REST API, or by using Azure Storage Explorer. The access level is configured separately on each blob container.

Create Blob Access Tier

**Access Tier**

There are different access tiers, so if we think, for example, our interactions with data are different. I might have the idea that there's certain data I'm constantly communicating with, so if I think tiers for a second, different access tiers. Now. Remember they were the premium account types and the general purpose. The tier availability varies greatly, so if I think for OK so I've got access tiers, then there's also the idea of the account type. And we're really focused on is a general purpose V2. So, there is the idea that, hey, I'm premium, but if I'm premium, I'm just premium. There's no other type of tearing available to me. If I'm a general purpose V2 I have different access tiers available to me.

**Hot Tier**

Now at the top we have hot and when I talk about, I guess at the top I should be specific. These are focused on BLOB because also Azure files has some different tiers as well available. To me that's a different type of service in the storage account. So what is data I'm constantly doing transactions against. I'm constantly reading it, writing to it. And the reason you have these, if it's hot, I pay more for the storage of the data, but I pay less for the transactions. So this is the most cost optimal way for me to use it. I'll pay more to store it, but I don't have to pay as much when I actually interact with it.

**Cool Tier**

Then there might be data that hey I need to have immediately available to me, but I don't intend to

interact with it very often. So, I pay less to store it, but I'll pay more for the transactions against it. So, that's the point of cool

**Archive Tier**

And then we actually have archive. And a key point of archive is this is offline. When I move it to archive, I no longer have a media access to it. If I want to be able to read the data, I have to hydrate it back into cool or hot to actually get access to it again. So, this is still drawable, it's resilient, but there is no instant access and it may take many hours. There's a priority mode. Well, I think it's an hour depending on the size, but it has to rehydrate this back and I can absolutely move data blobs between the tiers. This direction from Archive to Cool or Hot is hours. There is a priority model and depends on the size but it’s a fairly long amount of time.

**Azure Files**

When we think about of files, Azure files has the same idea of hot and cool. But it actually has something above even hot. It has transaction optimized. Once again, the whole logic of this is simply I set this at the share level. So, with the BLOB, this is per BLOB in one general purpose storage account in one container. One BLOB could be hot, one BLOB could be cool, one BLOB could be archive. In files it's at the share level, so I configure the share. So there's a big difference between those.

**Pricing**

So, we have these there and if we looked at the pricing, so we jump over for a second. We can see that in place. This is BLOB storage pricing. And what you'll see straight away is, well, this is the amount you pay to store it. So obviously premium is always going to be high. Because premium is this super low latency, very very high performance. But hey, hot is kind of two cents per GB, cool only one cent per GB, and archive is just dirt cheap. So, it's very, very cheap. But again, I don't have instant access to it, but that's the price to store it. But then the actual operations against it will for premium, they're very cheap. Some of them I don't pay anything. For Hot, well, it costs a bit more, but it's still pretty cheap. But for cool costs more because I'm paying less for the capacity storage of it. And then archive. There's still of the operations I do to move it around, but I have to actually hydrate it back to get it from archive. And then I have a cost associated as well. Just realized there is difference and we have those access tiers so that you can optimize how you store the data because you're have different requirements. So, for the data I'm working with right now, I'll probably put that in hot. For the data, I must have access to, but I don't interact with very often, I move that to cool, I'll spend less money. The data maybe I have to legally keep, but I don't intend to access, or if I do, I can wait a day, I can move that to archive. So, there's always to optimize how I'm spending my money.

**Monitoring**

There's monitoring, there's logging icon enable on those things. Just like everything else. I actually get some really nice data just for a regular storage account. I have insights just available to me. I can get nice information about transactions by the type of storage, I can see availability, used capacity, failures, performance. There are alerts, I can create these different metrics. I can diagnose they're diagnostic settings as always. I could send this to a log analytics workspace or a storage account. I have a lot of controls over these different types of service. I guess while I'm in here, if I was to go ahead and look at my containers. And I think maybe images. Notice I've got just images, image files in this container, so images of the container. But notice I've got a bunch of different files. But some of them are hot. Some of them are cool. One of them is archive. And I can go and view these files if it's in hot and cool. So, in hot if I did edit, I can actually see the content of it. I could do the same for the cool file, but I'm paying a bit more now for that transaction against it. But if I try and do it for the archive, I can't. It's not available. I would have to bring it back, which could take time, and there's certain limits on how long things have to stay in archive before I could actually go ahead and do that.

Add Blob Lifecycle Management Rules

Every data set has a unique lifecycle. Early in the lifecycle, users tend to access some of the data in the set, but not all of the data. As the data set ages, access to all of the data in the set tends to dramatically reduce. Some data set stays idle in the cloud and is rarely accessed after it's stored. Some data expires within a few days or months after it's created. Other data is actively read and modified throughout the data set lifetime. Azure Blob Storage supports lifecycle management for data sets. It offers a rich rule-based policy for GPv2 and Blob Storage accounts. You can use lifecycle policy rules to transition your data to the appropriate access tiers, and set expiration times for the end of a data set's lifecycle. In the Azure portal, you create lifecycle management policy rules for your Azure storage account by specifying several settings. For each rule, you create **If - Then** block conditions to transition or expire data based on your specifications. As you review these details, consider how you can set up lifecycle management policy rules for your data sets. **DEMO:** Create a “test rule” to move from Hot to Cool after 30 days then move from cool to Archive after 180 days and then delete the blobs after 365 days.

Determine Blob Object Replication

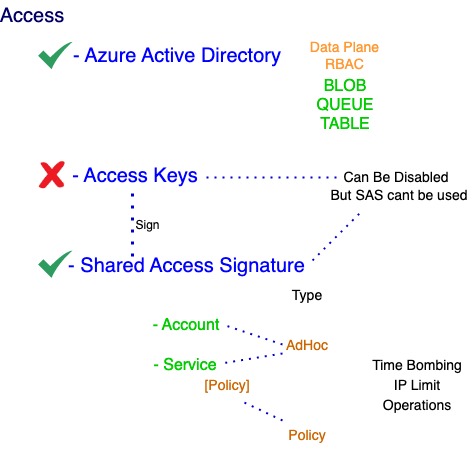
Object replication copies blobs in a container asynchronously according to policy rules that you configure. During the replication process, some elements are copied from the source container to the destination container, such as the blob contents, the metadata and properties and any versions of data associated with the blob. There are many benefits to using blob object replication. Consider the following scenarios and think about how replication can be a part of your Blob Storage strategy. **Consider latency reductions**. Minimize latency with blob object replication. You can reduce latency for read requests by enabling clients to consume data from a region that's in closer physical proximity. **Consider efficiency for compute workloads**. Improve efficiency for compute workloads by using blob object replication. With object replication, compute workloads can process the same sets of blobs in different regions. **Consider data distribution**. Optimize your configuration for data distribution. You can process or analyze data in a single location and then replicate only the results to other regions. **Consider costs benefits**. Manage your configuration and optimize your storage policies to achieve cost benefits. After your data is replicated, you can reduce costs by moving the data to the Archive tier by using lifecycle management policies. **DEMO:** Create a new Storage account in another region and create an object replication policy and apply to a container.

## Configure Storage Security

Azure Storage provides a comprehensive set of security capabilities that work together to enable developers to build secure applications.

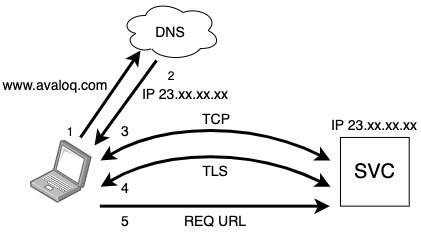
Review Storage Security Strategies

Administrators use different strategies to ensure their data is secure. Common approaches include encryption, authentication, authorization, and user access control with credentials, file permissions, and private signatures. Azure Storage offers a suite of security capabilities based on common strategies to help you secure your data as for example, **Encryption**. All data written to Azure Storage is automatically encrypted by using Azure Storage encryption. **Authentication**. Azure Active Directory (Azure AD) and role-based access control (RBAC) are supported for Azure Storage for both resource management operations and data operations. **Data in transit**. Data can be secured in transit between an application and Azure by using Client-Side Encryption, HTTPS, or SMB 3.0. **Disk encryption**. Operating system disks and data disks used by Azure Virtual Machines can be encrypted by using Azure Disk Encryption. **Shared access signatures**. Delegated access to the data objects in Azure Storage can be granted by using a shared access signature (SAS). **Authorization**. Every request made against a secured resource in Blob Storage, Azure Files, Queue Storage, or Azure Cosmos DB (Azure Table Storage) must be authorized. Authorization ensures that resources in your storage account are accessible only when you want them to be, and to only those users or applications whom you grant access.

Create shared access signatures

So, how do I control who has permissions to things? We have Azure AD for Data plane with Role based access control, I can create custom roles and I can do this for BLOB, Queue and table. For files, I can give a permission to access the share, but I can't do anything on the actual files and folders within the share by the SMB.

We have the access key that I don’t want to use and then we have shared access signatures. This is not as bad as just using the access key. These are signed with the account key. So, if I rotate the account key, it would invalidate the SAS that was signed with that account key. Otherwise, I can't revoke it, but there is a concept of a policy based shared access signature for service. There are two types, there's service access signature and account SAS. Account SAS, as the name suggests applies at account level. So, it could be given access to any of the services, BLOB, queue, table, whatever service. SAS it's specific and I create it at a particular service level, so it could be for files or for queue or for table. The benefit of service though. So, to recap, I could use Azure AD, is one of the options. We have the access key. We have two of them. Remember I do not want to use these. If I have these stored, I'm using this as part of my app as a bad thing to do. I don't want to use them, but maybe what I could use is a shared access signature and remember the access key is used to sign the shared access signature. There's the account level SAS. Which I can give permission for any of the different services within the account or there is the service which is going to be one of BLOB or queue or table. If I use service I can also at the container level create a policy that could then be used to create a policy types of access signature or both of them can create ad hoc types. Ad hoc is just. I select the settings and it's signed by the key. This means I can't revoke it. There's no way to revoke or change this. Unless I rotate the access key, which will then validate the signature, it won't work anymore. We've policy because it's built off of a certain policy. By changed the policy it changes what this can do. And then with these I can for example, I can time bomb it, there can be a time limit, I might do, IP limits, I can obviously restrict the operations. I can disable the access key but If I disable the access key, I can't use shared access signatures. So, although yes, I may want to turn off access keys from a security perspective, realize that also means I can't use these more granular shared access signatures.

**DEMO SAS**

It's got the signature all the way at the end of it now. One of the things that happens when you show this is people panic. They're like, wait, you just sent the checks, the signature in plain text over the Internet. Anyone could get that and get access to it. That is not how the Internet works.

So, bit of a digression, but yes, it's a URL. It's added to the end of the request, but when you talk to a secure site you don't send the URL over the Internet say hey, I want this page. Well, not first if this is me. And then this is some service. This could be a storage account, it could be avaloq.com. It's all the same. The first thing you do is when I type in www.avaloq.com, but I actually have to convert www.avaloq.com to an IP address. So, my machine goes and talks to DNS and says hey what's the IP address of this service and it sends me back the IP. Then I go and talk to the IP of the service and there's kind of this 3-way handshake and establish a TCP session. Then I have to establish an TLS. And there's a, I think it's four sets of packets, the kind of a client Hello, server hello, you agree? The whole authentication and what level of TLS you support, what encryption algorithms you're going to use, all of that stuff happens, but the end result is I establish a TLS session and then over this TLS do I send an actual request which would be the actual URL. So, it's never sent plaintext. I established all of that stuff first before I ever actually send anything, and we'll see that like in this browser right here. If I super quickly just typed in that [www.avaloq.com](http://www.avaloq.com) if I look at this very very plain looking site right here, well actually what's happening is let's go. If I was to right click somewhere and do inspect, you get a whole bunch of troubleshooting stuff. But if I add an element and do security or maybe it's there already so. They got security. It will show you’re using TLS 1.3 with RSA. Advanced Encryption standard 256 with GCM like you've established this secure connection before you did anything else. And so that's when it actually would send that request over the URL. So, you're never sending that shared access just over plain text to web. That's not a concern you actually need to think about. Now, what I would say though is you need to be super, super careful that you don't put these access keys into code. So never in code, you're seeing GitHub, it will go and warn you. It will tell you if you have to use these. What maybe I have is some process generates the SAS, it then goes and writes it the site. Maybe like an Azure key vault as a secret and then my application can read it and then it might go and use it to actually go and talk to BLOB. Never ever end up with those in just a config file. It's a terrible place to be.

Azure files actually has different types of integration. Now it's got three options available to us, so it's really grown. The probably most common one you will see today is it can hook into Azure directory domain services. So, you're existing Active Directory you have the storage account gets an account created in your regular AD, and now my users just connect to the SMB share and the apples would be applied and maintained. So that's probably the easiest one for most companies. I can use Azure AD domain services, but then I have to go and deploy that and if I already have AD-DS I probably don't care or don't want to do that. The other thing I can actually do is Azure AD now does support Kerberos for limited scenarios, but one of them is access to Azure files I have to set up on my VM. So, then I'm actually using Azure AD to give me a Kerberos session ticket to authenticate to the Azure file share. So, there's different things depending on the scenario.

Identify URI and SAS Parameters

**DEMO:** create a SAS for a storage account. You can check the services, resource types, and permissions based on specific requirements, along with the duration for the SAS token validity and the IP addresses that are providing access. Lastly, you have an option to choose which key you want to use as the signing key for this token. Once the token is generated, it will be listed along with connection string and SAS URLs. Each SAS token is a query string parameter that can be appended to the full URI of the blob or other storage resource for which the SAS token was created. Create the SAS URI by appending the SAS token to the full URI of the blob or other storage resource.

Determine Storage Service Encryption

In Azure data is encrypted automatically before it's persisted to Azure Managed Disks, Azure Blob Storage, Azure Queue Storage, Azure Cosmos DB (Azure Table Storage), or Azure Files. Data is automatically decrypted before it's retrieved. Azure Storage encryption, encryption at rest, decryption, and key management are transparent to users. All data written to Azure Storage is encrypted through 256-bit advanced encryption standard (AES) encryption. Azure Storage encryption is enabled for all new and existing storage accounts and can't be disabled. In the Azure portal, you configure Azure Storage encryption by specifying the encryption type. You can manage the keys yourself, or you can have the keys managed by Microsoft. Consider how you might implement Azure Storage encryption for your storage security.

Create Customer Managed Keys

By creating your own keys (referred to as customer-managed keys), you have more flexibility and greater control. You can create, disable, audit, rotate, and define access controls for your encryption keys. Customer-managed keys can be used with Azure Storage encryption. You can use a new key or an existing key vault and key. The Azure storage account and the key vault must be in the same region, but they can be in different subscriptions. Azure Key Vault helps safeguard cryptographic keys and secrets used by cloud applications and services, such as authentication keys, storage account keys, data encryption keys, and certificate private keys.

Apply Storage Security Best Practices

1 – Encrypt data at rest and in-flight. Transparent Data Encryption (TDE) feature of Azure SQL database can help to protect a database for data at rest. To secure in-flight data Azure can help using Transport Layer Security TLS version 1.2. For most of the services it is enabled by default.

2 – Restrict access to your databases. In all Azure DB you can configure the firewall, which is enabled by default and blocks IP addresses that are not allowed. You can use also private link to segregate the access only from certain virtual networks or services.

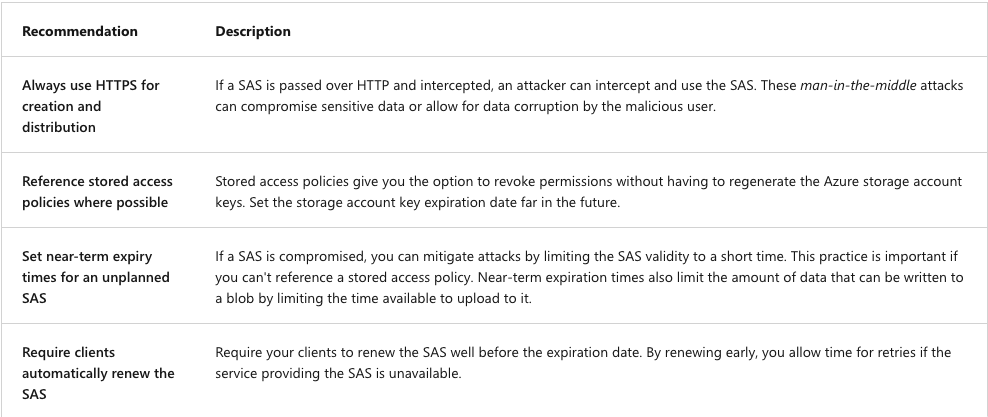
3 – Restrict access to your VMs. Opening RDP or SSH to reach your VMs from internet is not a best practice. Instead, you can use a Bastion to access them.

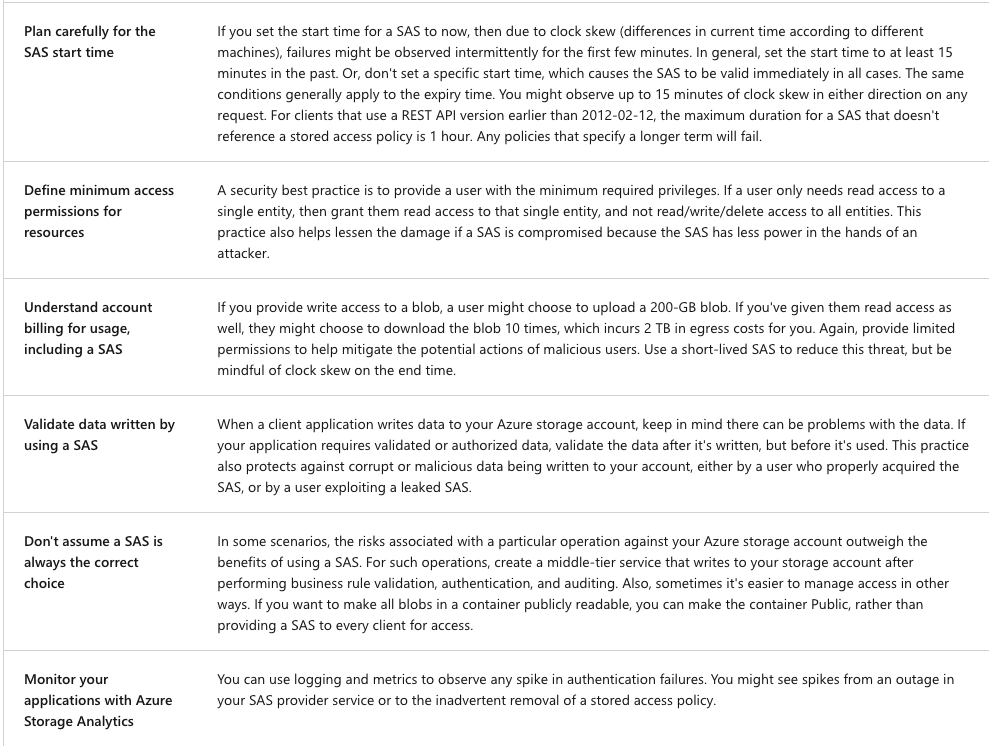
4 – Protect your application secrets. You shouldn’t store secrets like API secrets or storage encryption secrets in application code where every developer can access them. Azure Key Vault stores the secrets in a secure way using managed service identity to get the access.

5 – Use separate Subscription for Production Environment and implement Role-based Access Control to segregate Dev/Test environments from Production.

6 – Implement a Web Application Firewall. You should implement a gateway service for your application like Azure Front Door or Application Gateway that can route and filter the traffic. You can implement also on these services the Web Application Firewall feature that can detect the attacks and report them.

7 – Use Microsoft Defender for cloud to check the security of the resources we implemented in Azure.

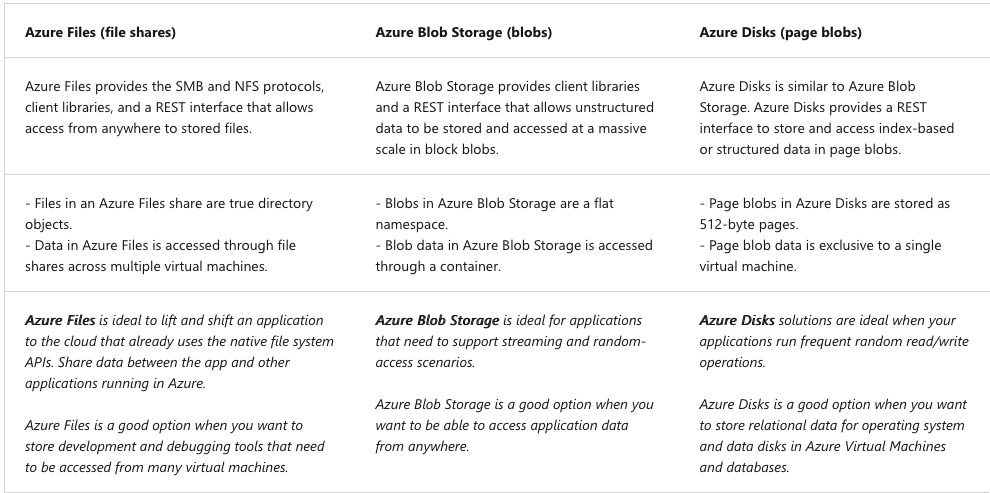
For the storage best practices, we can see some best practices on the slide.



## Configure Azure Files and File Sync

Azure Files is an offering that makes file shares available in the cloud. It’s a fully managed solution that supports access to these cloud-based file shares via the industry-standard server message block protocol, or SMB. You can mount Azure file shares from cloud deployments and on-prem deployments of not only Windows machines, but also Linux, and Mac OS machines. You can also use the Azure file sync service with Azure Files to cache your Azure file shares on Windows servers that are located close to your users. By leveraging Azure file shares with Azure file sync, you can speed data access for your end users. Organizations will often use Azure Files to replace on-prem file servers or to supplement them. While earlier iterations of Azure Files were not a good replacement for on-prem file servers, this is no longer the case. Because popular operating systems like Windows, Linux, and Mac OS can mount Azure file shares. Azure Files can now completely replace traditional on-prem file servers and even NAS devices. As a matter of fact, the release of Azure Files AD Authentication means Azure file share permissions can even be controlled through on-prem active directories. Azure Files is also helpful when lifting and shifting applications to the cloud. This is especially true for applications that require file shares to store application data and user data. Because Azure Files are fully managed, you can create as many file shares as you need without worrying about hardware management and OS installation. This means you also have no need for OS patching or security upgrades. You can also use familiar PowerShell commands and Azure CLI commands to create, mount, and manage Azure file shares. They can also be created and managed through the Azure portal and through Azure Storage Explorer. Because Azure Files are built to be resilient, you no longer need to worry about file server upgrades or local power outages and network issues that typically affect access to on-prem file shares.

Compare Files to Blobs

You can refer to this table for this comparison:

Manage Azure Files shares

There are two important settings for Azure Files that you need to be aware of when creating and configuring file shares. **Open port 445**. Azure Files uses the SMB protocol. SMB communicates over TCP port 445. Be sure port 445 is open. Also, make sure your firewall isn't blocking TCP port 445 from the client machine. **Enable secure transfer**. The Secure transfer required setting enhances the security of your storage account by limiting requests to your storage account from secure connections only. Consider the scenario where you use REST APIs to access your storage account. If you attempt to connect, and secure transfer required is enabled, you must connect by using HTTPS. If you try to connect to your account by using HTTP, and secure transfer required is enabled, the connection is rejected. Azure Files provides flexibility to use the following two types of identity-based authentication to access the shares.

**DEMO:** Create a simple share on the storageaccount and map on a Windows or MacOS system.

Create File Share Snapshots

Azure Files provides the capability to take share snapshots of file shares. File share snapshots capture a point-in-time, read-only copy of your data. The Azure Files share snapshot capability is provided at the file share level and the restore at file level. Share snapshots are incremental in nature. Only data changed since the most recent share snapshot is saved, this minimizes the time required to create share snapshots and saves on storage costs. If you want to delete a share that has share snapshots, you must first delete all of its' snapshots.

Implement Azure File Sync

Azure File Sync extends Azure Files to allow on-premises file services to be extended to Azure while maintaining performance and compatibility. Some of the key functionality Azure File Sync provides are: **multi-site access, the** ability to write files across Windows and Azure Files. **Cloud tiering**. Storage only recently accessed data on local servers. The rest of the data gets tiered to Azure in a storage account. **Azure Backup integration** Backup in the cloud. **Fast disaster recovery** Restore file metadata immediately and recall as needed.

Identify File Sync Components

Azure File Sync is composed of four main components that work together to provide caching for Azure Files shares on an on-premises Windows Server or cloud virtual machine. The **Storage Sync Service** is the top-level Azure resource for Azure File Sync. This resource is a peer of the storage account resource and can be deployed in a similar manner. The **registered server** object represents a trust relationship between your server (or cluster) and the Storage Sync Service resource. You can register as many servers to a Storage Sync Service resource as you want. The **Azure File Sync agent** is a downloadable package that enables Windows Server to be synced with an Azure Files share. The Azure File Sync agent has three main components: ***FileSyncSvc.exe*** is the background Windows service that's responsible for monitoring changes on server endpoints, and for initiating sync sessions to Azure. ***StorageSync.sys*** is the Azure File Sync file system filter that supports cloud tiering. The filter is responsible for tiering files to Azure Files when cloud tiering is enabled. **PowerShell cmdlets.** These PowerShell management cmdlets allow you to interact with the Microsoft.StorageSync Azure resource provider. A **server endpoint** represents a specific location on a registered server, such as a folder on a server volume. Multiple server endpoints can exist on the same volume if their namespaces are unique. A **cloud endpoint** is an Azure Files share that's part of a sync group. As part of a sync group, the entire cloud endpoint (Azure Files share) syncs. Then we have the last component. The **sync group** defines the sync topology for a set of files. Endpoints within a sync group are kept in sync with each other. Consider the scenario where you have two distinct sets of files that you want to manage with Azure File Sync. In this case, you create two sync groups and add different endpoints to each sync group. An instance of the Storage Sync Service can host as many sync groups as you need.

Setup File Sync

The setup of file sync is a 4 steps setup:

**Deploy the Storage Sync Service**. You can deploy the Storage Sync Service from the Azure portal.

**Prepare each Windows Server to use Azure File Sync**. After you deploy the Storage Sync Service, you configure each Windows Server or cloud virtual machine that you intend to use with Azure File Sync, including server nodes in a Failover Cluster.

**Install the Azure File Sync agent.** When the Windows Server configuration is complete, you're ready to install the Azure File Sync agent. The agent is a downloadable package that enables Windows Server to be synced with an Azure Files share. The Azure File Sync agent installation package should install relatively quickly.

**Register each Windows Server with the Storage Sync Service**. By registering the Windows Server with a Storage Sync Service, you establish a trust relationship between your server (or cluster) and the Storage Sync Service.

Configure Storage with Tools

**Azure Storage Explorer** is a standalone application that makes it easy to work with Azure Storage data on Windows, macOS, and Linux. With Azure Storage Explorer, you can access multiple accounts and subscriptions, and manage all your Storage content. **DEMO** Access Storage Explorer.

**Azure Import/Export service** is used to securely import large amounts of data to Azure Blob Storage and Azure Files by shipping disk drives to an Azure datacenter. This service can also be used to transfer data from Azure Blob Storage to disk drives and ship to your on-premises sites.

An alternate method for transferring data is the **AzCopy** tool. AzCopy v10 is the next-generation command-line utility for copying data to and from Azure Blob Storage and Azure Files. AzCopy v10 offers a redesigned command-line interface (CLI) and new architecture for high-performance reliable data transfers. You can use AzCopy to copy data between a file system and a storage account, or between storage accounts.