Store data in Azure

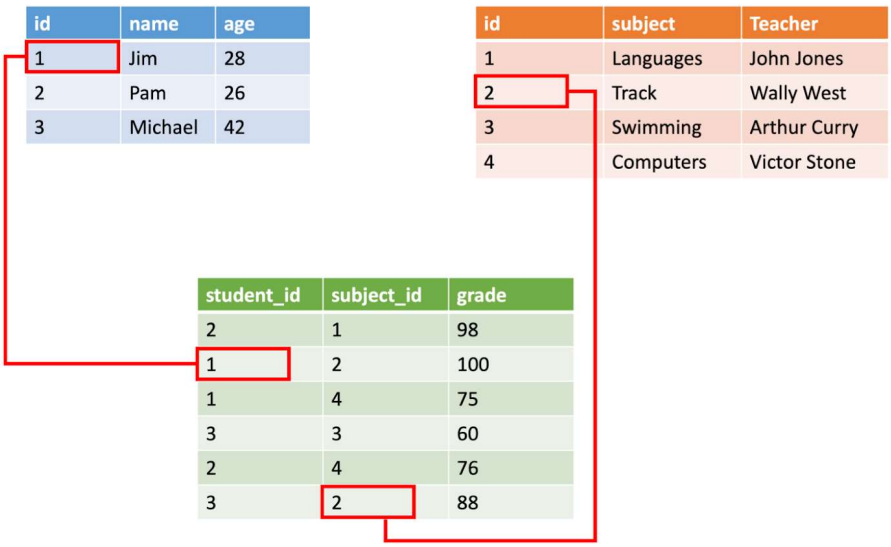
Azure provides a variety of ways to store data: unstructured, archival, relational, and more. Learn the basics of storage management in Azure, how to create a Storage Account, and how to choose the right model for the data you want to store in the cloud.

Choose a data storage approach in Azure

Structured data

Structured data, sometimes referred to as relational data, is data that adheres to a strict schema, so all of the data has the same fields or properties. The shared schema allows this type of data to be easily searched with query languages such as SQL (Structured Query Language). This capability makes this data style perfect for applications such as CRM systems, reservations, and inventory management.

Structured data is often stored in database tables with rows and columns with key columns to indicate how one row in a table relates to data in another row of another table. The below image shows data about students and classes with a relationship to grades that ties them together.



id	name	age
1	Jim	28
2	Pam	26
3	Michael	42

id	subject	Teacher
1	Languages	John Jones
2	Track	Wally West
3	Swimming	Arthur Curry
4	Computers	Victor Stone

student_id	subject_id	grade
2	1	98
1	2	100
1	4	75
3	3	60
2	4	76
3	2	88

Structured data is straightforward in that it's easy to enter, query, and analyze. All of the data follows the same format. However, forcing a consistent structure also means evolution of the data is more difficult as each record has to be updated to conform to the new structure.

Semi-structured data

Semi-structured data is less organized than structured data, and is not stored in a relational format, as the fields do not neatly fit into tables, rows, and columns. Semi-structured data contains tags that make the organization and hierarchy of the data apparent - for example,

key/value pairs. Semi-structured data is also referred to as non-relational or NoSQL data. The expression and structure of the data in this style is defined by a serialization language.

For software developers, data serialization languages are important because they can be used to write data stored in memory to a file, sent to another system, parsed and read. The sender and receiver don't need to know details about the other system, as long as the same serialization language is used, the data can be understood by both systems.

Today, there are three common serialization languages you're likely to encounter:

1. **XML**, or extensible markup language, was one of the first data languages to receive widespread support. It's text-based, which makes it easily human and machine-readable. In addition, parsers for it can be found for almost all popular development platforms. XML allows you to express relationships and has standards for schema, transformation, and even displaying on the web. XML expresses the shape of the data using tags. XML is flexible and can express complex data easily. However it tends to be more verbose making it larger to store, process, or pass over a network. As a result, other formats have become more popular.
2. **JSON –** or JavaScript Object Notation, has a lightweight specification and relies on curly braces to indicate data structure. Compared to XML, it is less verbose and easier to read by humans. JSON is frequently used by web services to return data. Notice that this format isn't as formal as XML. It's closer to a key/value pair model than a formal data expression. As you might guess from the name, JavaScript has built-in support for this format - making it very popular for web development. Like XML, other languages have parsers you can use to work with this data format. The downside to JSON is that it tends to be more programmer-oriented making it harder for non-technical people to read and modify.
3. **YAML –** or YAML Ain't Markup Language, is a relatively new data language that's growing quickly in popularity in part due to its human-friendliness. The data structure is defined by line separation and indentation, and reduces the dependency on structural characters like parentheses, commas and brackets. This format is more readable than JSON and is often used for configuration files that need to be written by people but parsed by programs. However, YAML is the newest of these data formats and doesn't have as much support in programming languages as JSON and XML.

Unstructured data

The organization of unstructured data is ambiguous. Unstructured data is often delivered in files, such as photos or videos. The video file itself may have an overall structure and come with semi-structured metadata, but the data that comprises the video itself is unstructured. Therefore, photos, videos, and other similar files are classified as unstructured data.

Examples of unstructured data include:

- Media files, such as photos, videos, and audio files
- Office files, such as Word documents
- Text files
- Log files

What is a transaction?

A transaction is a logical group of database operations that execute together.

Here's the question to ask yourself regarding whether you need to use transactions in your application: Will a change to one piece of data in your dataset impact another? If the answer is yes, then you'll need support for transactions in your database service.

Transactions are often defined by a set of four requirements, referred to as ACID guarantees. ACID stands for Atomicity, Consistency, Isolation, and Durability:

- **Atomicity** means a transaction must execute exactly once and must be atomic; either all of the work is done, or none of it is. Operations within a transaction usually share a common intent and are interdependent.
- **Consistency** ensures that the data is consistent both before and after the transaction.
- **Isolation** ensures that one transaction is not impacted by another transaction.
- **Durability** means that the changes made due to the transaction are permanently saved in the system. Committed data is saved by the system so that even in the event of a failure and system restart, the data is available in its correct state.

When a database offers ACID guarantees, these principles are applied to any transactions in a consistent manner.

Transactional databases are often called OLTP (Online Transaction Processing) systems. OLTP systems commonly support lots of users, have quick response times, and handle large volumes of data. They are also highly available (meaning they have very minimal downtime), and typically handle small or relatively simple transactions.

On the contrary, OLAP (Online Analytical Processing) systems commonly support fewer users, have longer response times, can be less available, and typically handle large and complex transactions.

Product catalog data

Data classification: Semi-structured because of the need to extend or modify the schema for new products
Operations:

- Customers require a high number of read operations, with the ability to query on many fields within the database.
- The business requires a high number of write operations to track the constantly changing inventory.

Latency & throughput: High throughput and low latency
Transactional support: Required

Recommended service: Azure Cosmos DB

Azure Cosmos DB supports semi-structured data, or NoSQL data, by design.

Azure Cosmos DB supports SQL for queries and every property is indexed by default. You can create queries so that your customers can filter on any property in the catalog.

Azure Cosmos DB is also ACID-compliant, so you can be assured that your transactions are completed according to those strict requirements.

As an added plus, Azure Cosmos DB also enables you to replicate your data anywhere in the world with the click of a button. So, if your e-commerce site has users concentrated in the US, France, and England, you can replicate your data to those data centers to reduce latency, as you've physically moved the data closer to your users.

Even with data replicated around the world, you can choose from one of five consistency levels. By choosing the right consistency level, you can determine the tradeoffs to make between consistency, availability, latency, and throughput. You can scale up to handle higher customer demand during peak shopping times, or scale down during slower times to conserve cost.

Azure Cosmos DB is better choice for highly unstructured and variable data where you cannot predict what are the properties that should be indexed.

Photos and videos

Data classification: Unstructured
Operations:

- Only need to be retrieved by ID.
- Customers require a high number of read operations with low latency.
- Creates and updates will be somewhat infrequent and can have higher latency than read operations.

Latency & throughput: Retrievals by ID need to support low latency and high throughput. Creates and updates can have higher latency than read operations.
Transactional support: Not required

Recommended service: Azure Blob storage

Azure Blob storage supports storing files such as photos and videos. It also works with Azure Content Delivery Network (CDN) by caching the most frequently used content and storing it on edge servers. Azure CDN reduces latency in serving up those images to your users.

By using Azure Blob storage, you can also move images from the hot storage tier to the cool or archive storage tier, to reduce costs and focus throughput on the most frequently viewed images and videos.

Business data

Data classification: Structured
Operations: Read-only, complex analytical queries across multiple databases
Latency & throughput: Some latency in the results is expected based on the complex nature of the queries.
Transactional support: Required

Recommended service: Azure SQL Database

Business data will most likely be queried by business analysts, who are more likely to know SQL than any other query language. Azure SQL Database could be used as the solution by itself, but pairing it with Azure Analysis Services enables data analysts to create a semantic model over the data in SQL Database. The data analysts can then share it with business users, so that they only need to connect to the model from any business intelligence (BI) tool to immediately explore the data and gain insights.

Summary

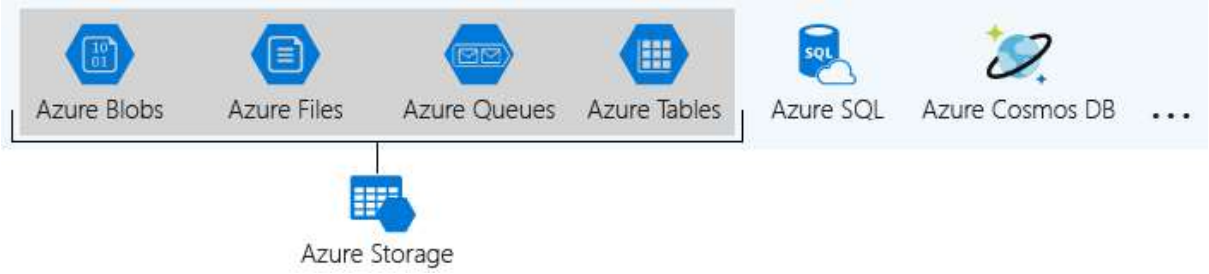
1. **A JSON file is an example of which type of data?**
Semi-structured
Semi-structured data contains tags that make the organization and hierarchy of the data apparent.
2. **A video is an example of which type of data?**
Unstructured
Unstructured data is often delivered in files. A video may have an overall structure but the data that comprises the video itself is unstructured.
3. **Which type of transactional database system would work best for product data?**
OLTP
OLTP systems support a large set of users, have quick response times, handle large volumes of data, are highly available, and are great for small or relatively simple transactions.
4. **Suppose the operations to update inventory and process payments are in the same transaction. A user is attempting to apply store credit for the full amount of an order, and submitted the exact same order (for the full amount) using their phone and laptop at the same time - so two identical orders are received. The database behind the scenes is an ACID-compliant database, what would happen?**
One order would be processed and use the in-store credit, and the other order would not be processed.
Once the second order determined that the in-store credit has already been used, it would roll back the transaction.

Create an Azure Storage account

What is Azure Storage?

Azure provides many ways to store your data. There are multiple database options like Azure SQL Database, Azure Cosmos DB, and Azure Table Storage. Azure offers multiple ways to store and send messages, such as Azure Queues and Event Hubs. You can even store loose files using services like Azure Files and Azure Blobs.

Azure selected four of these data services and placed them together under the name Azure Storage. The four services are Azure Blobs, Azure Files, Azure Queues, and Azure Tables.

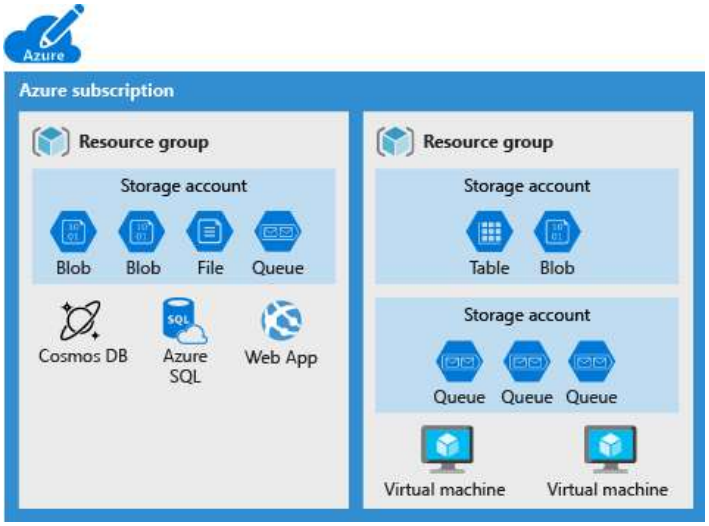


What is a storage account?

A storage account is a container that groups a set of Azure Storage services together. Only data services from Azure Storage can be included in a storage account (Azure Blobs, Azure Files, Azure Queues, and Azure Tables).

Combining data services into a storage account lets you manage them as a group. The settings you specify when you create the account, or any that you change after creation, are applied to everything in the account. Deleting the storage account deletes all of the data stored inside it.

A storage account is an Azure resource and is included in a resource group. Other Azure data services like Azure SQL and Azure Cosmos DB are managed as independent Azure resources and cannot be included in a storage account.



Storage account settings

A storage account defines a policy that applies to all the storage services in the account. For example, you could specify that all the contained services will be stored in the West US datacenter, accessible only over https, and billed to the sales department's subscription.

The settings that are controlled by a storage account are:

- **Subscription**: The Azure subscription that will be billed for the services in the account.
- **Location**: The datacenter that will store the services in the account.
- **Performance**: Determines the data services you can have in your storage account and the type of hardware disks used to store the data. Standard allows you to have any data service (Blob, File, Queue, Table) and uses magnetic disk drives. Premium introduces additional services for storing data. For example, storing unstructured object data as block blobs or append blobs, and specialized file storage used to store and create premium file shares. These storage accounts use solid-state drives (SSD) for storage.
- **Replication**: Determines the strategy used to make copies of your data to protect against hardware failure or natural disaster. At a minimum, Azure will automatically maintain three copies of your data within the data center associated with the storage account. This is called locally-redundant storage (LRS), and guards against hardware failure but does not protect you from an event that incapacitates the entire datacenter. You can upgrade to one of the other options such as geo-redundant storage (GRS) to get replication at different datacenters across the world.
- **Access tier**: Controls how quickly you will be able to access the blobs in this storage account. Hot gives quicker access than Cool, but at increased cost. This applies only to blobs, and serves as the default value for new blobs.
- **Secure transfer required**: A security feature that determines the supported protocols for access. Enabled requires HTTPs, while disabled allows HTTP.
- **Virtual networks**: A security feature that allows inbound access requests only from the virtual network(s) you specify.

The number of storage accounts you need is typically determined by your data diversity, cost sensitivity, and tolerance for management overhead.

These settings impact how you manage your account and the cost of the services within it.

Name

Each storage account has a name. The name must be globally unique within Azure, use only lowercase letters and digits and be between 3 and 24 characters.

Deployment model

A deployment model is the system Azure uses to organize your resources. The model defines the API that you use to create, configure, and manage those resources. Azure provides two deployment models:

- **Resource Manager**: the current model that uses the Azure Resource Manager API

- **Classic**: a legacy offering that uses the Azure Service Management API

Most Azure resources only work with Resource Manager, and makes it easy to decide which model to choose. However, storage accounts, virtual machines, and virtual networks support both, so you must choose one or the other when you create your storage account.

The key feature difference between the two models is their support for grouping. The Resource Manager model adds the concept of a resource group, which is not available in the classic model. A resource group lets you deploy and manage a collection of resources as a single unit.

Microsoft recommends that you use **Resource Manager** for all new resources.

Account kind

Storage account kind is a set of policies that determine which data services you can include in the account and the pricing of those services. There are three kinds of storage accounts:

- **StorageV2** (general purpose v2): the current offering that supports all storage types and all of the latest features
- **Storage** (general purpose v1): a legacy kind that supports all storage types but may not support all features
- **Blob storage**: a legacy kind that allows only block blobs and append blobs

Microsoft recommends that you use the General-purpose v2 option for new storage accounts.

There are a few special cases that can be exceptions to this rule. For example, pricing for transactions is lower in general purpose v1, which would allow you to slightly reduce costs if that matches your typical workload.

The core advice here is to choose the **Resource Manager** deployment model and the **StorageV2** (general purpose v2) account kind for all your storage accounts. The other options still exist primarily to allow existing resources to continue operation. For new resources, there are few reasons to consider the other choices.

Data diversity

Organizations often generate data that differs in where it is consumed, how sensitive it is, which group pays the bills, etc. Diversity along any of these vectors can lead to multiple storage accounts. Let's consider two examples:

1. Do you have data that is specific to a country or region? If so, you might want to locate it in a data center in that country for performance or compliance reasons. You will need one storage account for each location.
2. Do you have some data that is proprietary and some for public consumption? If so, you could enable virtual networks for the proprietary data and not for the public data. This will also require separate storage accounts.

In general, increased diversity means an increased number of storage accounts.

Cost sensitivity

A storage account by itself has no financial cost; however, the settings you choose for the account do influence the cost of services in the account. Geo-redundant storage costs more than locally-redundant storage. Premium performance and the Hot access tier increase the cost of blobs.

You can use multiple storage accounts to reduce costs. For example, you could partition your data into critical and non-critical categories. You could place your critical data into a storage account with geo-redundant storage and put your non-critical data in a different storage account with locally-redundant storage.

Tolerance for management overhead

Each storage account requires some time and attention from an administrator to create and maintain. It also increases complexity for anyone who adds data to your cloud storage; everyone in this role needs to understand the purpose of each storage account so they add new data to the correct account.

Storage accounts are a powerful tool to help you get the performance and security you need while minimizing costs. A typical strategy is to start with an analysis of your data and create partitions that share characteristics like location, billing, and replication strategy, and then create one storage account for each partition.

Available tools

The available tools are:

- Azure Portal
- Azure CLI (Command-line interface)
- Azure PowerShell
- Management client libraries

Summary

1. **Suppose you have two video files stored as blobs. One of the videos is business-critical and requires a replication policy that creates multiple copies across geographically diverse datacenters. The other video is non-critical, and a local replication policy is sufficient. Which of the following options would satisfy both data diversity and cost sensitivity consideration.**

Create a two storage accounts. The first account makes use of Geo-redundant storage (GRS) and hosts the business-critical video content. The second account makes use of Local-redundant storage (LRS) and hosts the non-critical video content.

In general, increased diversity means an increased number of storage accounts. A storage account by itself has no financial cost. However, the settings you choose for the account do influence the cost of services in the account. Use multiple storage accounts to reduce costs.

2**. The name of a storage account must be:**

Globally unique**.**

*The storage account name is used as part of the URI for API access, so it must be globally unique***.**

3. **In a typical project, when would you create your storage account(s)?**

At the beginning, during project setup**.**

Storage accounts are stable for the lifetime of a project. It's common to create them at the start of a project.

Connect an app to Azure Storage

Durable	Redundancy ensures that your data is safe in the event of transient hardware failures. You can also replicate data across datacenters or geographical regions for additional protection from local catastrophe or natural disaster. Data replicated in this way remains highly available in the event of an unexpected outage.
Secure	All data written to Azure Storage is encrypted by the service. Azure Storage provides you with fine-grained control over who has access to your data.
Scalable	Azure Storage is designed to be massively scalable to meet the data storage and performance needs of today's applications.
Managed	Microsoft Azure handles maintenance and any critical problems for you.

A single Azure subscription can host up to 200 storage accounts, each of which can hold 500 TB of data.

Azure data services

Azure storage includes four types of data:

- **Blobs**: A massively scalable object store for text and binary data. Can include **support** for Azure Data Lake Storage Gen2.
- Files: Managed file shares for cloud or on-premises deployments.
- **Queues**: A messaging store for reliable messaging between application components.
- **Table Storage**: A NoSQL store for schemaless storage of structured data. Table Storage is not covered in this module.

All of these data types in Azure Storage are accessible from anywhere in the world over HTTP or HTTPS. Microsoft provides SDKs for Azure Storage in a variety of languages, as well as a REST API.

Blob storage

Azure Blob storage is an object storage solution optimized for storing massive amounts of unstructured data, such as text or binary data. Blob storage is ideal for:

- Serving images or documents directly to a browser, including full static websites.
- Storing files for distributed access.

- Streaming video and audio.
- Storing data for backup and restoration, disaster recovery, and archiving.
- Storing data for analysis by an on-premises or Azure-hosted service.

Azure Storage supports three kinds of blobs:

Blob type	Description
Block blobs	Block blobs are used to hold text or binary files up to ~5 TB (50,000 blocks of 100 MB) in size. The primary use case for block blobs is the storage of files that are read from beginning to end, such as media files or image files for websites. They are named block blobs because files larger than 100 MB must be uploaded as small blocks, which are then consolidated (or committed) into the final blob.
Page blobs	Page blobs are used to hold random-access files up to 8 TB in size. Page blobs are used primarily as the backing storage for the VHDs used to provide durable disks for Azure Virtual Machines (Azure VMs). They are named page blobs because they provide random read/write access to 512-byte pages.
Append blobs	Append blobs are made up of blocks like block blobs, but they are optimized for append operations. These are frequently used for logging information from one or more sources into the same blob. For example, you might write all of your trace logging to the same append blob for an application running on multiple VMs. A single append blob can be up to 195 GB.

Files

Azure Files enables you to set up highly available network file shares that can be accessed by using the standard Server Message Block (SMB) protocol. This means that multiple VMs can share the same files with both read and write access. You can also read the files using the REST interface or the storage client libraries. You can also associate a unique URL to any file to allow fine-grained access to a private file for a set period of time. File shares can be used for many common scenarios:

- Storing shared configuration files for VMs, tools, or utilities so that everyone is using the same version.
- Log files such as diagnostics, metrics, and crash dumps.
- Shared data between on-premises applications and Azure VMs to allow migration of apps to the cloud over a period of time.

Queues

The Azure Queue service is used to store and retrieve messages. Queue messages can be up to 64 KB in size, and a queue can contain millions of messages. Queues are generally used to store lists of messages to be processed asynchronously.

You can use queues to loosely connect different parts of your application together.

Account type	Description
General-purpose v2 (GPv2)	General-purpose v2 (GPv2) accounts are storage accounts that support all of the latest features for blobs, files, queues, and tables. Pricing for GPv2 accounts has been designed to deliver the lowest per gigabyte prices.
General-purpose v1 (GPv1)	General-purpose v1 (GPv1) accounts provide access to all Azure Storage services but may not have the latest features or the lowest per gigabyte pricing. For example, cool storage and archive storage are not supported in GPv1. Pricing is lower for GPv1 transactions, so workloads with high churn or high read rates may benefit from this account type.
Blob storage accounts	A legacy account type, blob storage accounts support all the same block blob features as GPv2, but they are limited to supporting only block and append blobs. Pricing is broadly similar to pricing for general-purpose v2 accounts.

`az storage account create`

Option	Description
--name	A Storage account name. The name will be used to generate the public URL used to access the data in the account. It must be unique across all existing storage account names in Azure. It must be 3 to 24 characters long and can contain only lowercase letters and numbers.
--resource-group	Use learn-22685b6d-b0fe-40ce-9a9f-a02794df91ea to place the storage account into the free sandbox.

--location	Select a location near you (see below).
--kind	This determines the storage account type. Options include BlobStorage, Storage, and StorageV2.
--sku	This decides the storage account performance and replication model. Options include Premium_LRS, Standard_GRS, Standard_LRS, Standard_RAGRS, and Standard_ZRS.
--access-tier	The Access tier is only used for Blob storage, available options are [Cool | Hot]. The Hot Access Tier is ideal for frequently accessed data, and the Cool Access Tier is better for infrequently accessed data. Note that this only sets the default value—when you create a Blob, you can set a different value for the data.

Security access keys

Each storage account has two unique access keys that are used to secure the storage account. If your app needs to connect to multiple storage accounts, then your app will require an access key for each storage account.

REST API endpoint

In addition to access keys for authentication to storage accounts, your app will need to know the storage service endpoints to issue the REST requests.
The REST endpoint is a combination of your storage account name, the data type, and a known domain. If you have a custom domain tied to Azure, then you can also create a custom domain URL for the endpoint.

Data type	**Example endpoint**
Blobs	`https://[name].blob.core.windows.net/`
Queues	`https://[name].queue.core.windows.net/`
Table	`https://[name].table.core.windows.net/`
Files	`https://[name].file.core.windows.net/`

Security

Access keys are critical to providing access to your storage account and as a result, should not be given to any system or person that you do not wish to have access to your storage account. Access keys are the equivalent of a username and password to access your computer.

Each storage account has two access keys. The reason for this is to allow keys to be rotated (regenerated) periodically as part of security best practice in keeping your storage account secure. This process can be done from the Azure portal or the Azure CLI / PowerShell command line tool.

Rotating a key will invalidate the original key value immediately and will revoke access to anyone who obtained the key inappropriately. With support for two keys, you can rotate keys without causing downtime in your applications that use them. Your app can switch to using the alternate access key while the other key is regenerated. If you have multiple apps using this storage account, they should all use the same key to support this technique. Here's the basic idea:

1. Update the connection strings in your application code to reference the secondary access key of the storage account.
2. Regenerate the primary access key for your storage account using the Azure portal or command line tool.
3. Update the connection strings in your code to reference the new primary access key.
4. Regenerate the secondary access key in the same manner.

Storage accounts offer a separate authentication mechanism called **shared access signatures** that support expiration and limited permissions for scenarios where you need to grant limited access. You should use this approach when you are allowing other users to read and write data to your storage account.

Summary

1. How many access keys are provided for accessing your Azure storage account?
2
Each storage account has two access keys. This lets you follow the best-practice guideline of periodically replacing the key used by your applications without incurring downtime.
2. You can use either the REST API or the Azure client library to programmatically access a storage account. What is the primary advantage of using the client library?
Convenience
Code that uses the client library is much shorter and simpler than code that uses the REST API. The client library handles assembling requests and parsing responses for you.
3. Which of the following is a good analogy for the access keys of a storage account?
Username and password
Possession of an access key identifies the account and grants you access. This is very similar to login credentials like a username and password.

Links

Azure Storage Services REST API Reference
Using shared access signatures to provide limited access to a storage account
Using Azure Key Vault Storage Account Keys with server apps
Source code for the .NET Azure SDKs
Azure Storage Client Library for JavaScript

Secure your Azure Storage account

Encryption at rest

All data written to Azure Storage is automatically encrypted by Storage Service Encryption (SSE) with a 256-bit Advanced Encryption Standard (AES) cipher. SSE automatically encrypts data when writing it to Azure Storage. When you read data from Azure Storage, Azure Storage decrypts the data before returning it. This process incurs no additional charges and doesn't degrade performance. It can't be disabled.

For virtual machines (VMs), Azure lets you encrypt virtual hard disks (VHDs) by using Azure Disk Encryption. This encryption uses BitLocker for Windows images, and it uses dm-crypt for Linux.

Azure Key Vault stores the keys automatically to help you control and manage the disk-encryption keys and secrets. So even if someone gets access to the VHD image and downloads it, they can't access the data on the VHD.

Encryption in transit

Keep your data secure by enabling transport-level security between Azure and the client. Always use HTTPS to secure communication over the public internet. When you call the REST APIs to access objects in storage accounts, you can enforce the use of HTTPS by requiring secure transfer for the storage account. After you enable secure transfer, connections that use HTTP will be refused. This flag will also enforce secure transfer over SMB by requiring SMB 3.0 for all file share mounts.

CORS support

Azure Storage supports cross-domain access through cross-origin resource sharing (CORS). CORS uses HTTP headers so that a web application at one domain can access resources from a server at a different domain. By using CORS, web apps ensure that they load only authorized content from authorized sources.

CORS support is an optional flag you can enable on Storage accounts. The flag adds the appropriate headers when you use HTTP GET requests to retrieve resources from the Storage account.

Role-based access control

To access data in a storage account, the client makes a request over HTTP or HTTPS. Every request to a secure resource must be authorized. The service ensures that the client has the permissions required to access the data. You can choose from several access options. Arguably, the most flexible option is role-based access.

Azure Storage supports Azure Active Directory and role-based access control (RBAC) for both resource management and data operations. To security principals, you can assign RBAC roles that are scoped to the storage account. Use Active Directory to authorize

resource management operations, such as configuration. Active Directory is supported for data operations on Blob and Queue storage.

To a security principal or a managed identity for Azure resources, you can assign RBAC roles that are scoped to a subscription, a resource group, a storage account, or an individual container or queue.

Auditing access

Auditing is another part of controlling access. You can audit Azure Storage access by using the built-in Storage Analytics service.

Storage Analytics logs every operation in real time, and you can search the Storage Analytics logs for specific requests. Filter based on the authentication mechanism, the success of the operation, or the resource that was accessed.

Understand storage account keys

Azure Storage accounts can create authorized apps in Active Directory to control access to the data in blobs and queues. This authentication approach is the best solution for apps that use Blob storage or Queue storage.

For other storage models, clients can use a shared key, or shared secret. This authentication option is one of the easiest to use, and it supports blobs, files, queues, and tables. The client embeds the shared key in the HTTP Authorization header of every request, and the Storage account validates the key.
Authorization: SharedKey
myaccount:CY1OP3O3jGFpYFbTCBimLn0Xov0vt0khH/E5Gy0fXvg=

The storage account has only two keys, and they provide full access to the account. Because these keys are powerful, use them only with trusted in-house applications that you control completely.

If the keys are compromised, change the key values in the Azure portal. Here are several reasons to regenerate your storage account keys:

- For security reasons, you might regenerate keys periodically.
- If someone hacks into an application and gets the key that was hard-coded or saved in a configuration file, regenerate the key. The compromised key can give the hacker full access to your storage account.
- If your team is using a Storage Explorer application that keeps the storage account key, and one of the team members leaves, regenerate the key. Otherwise, the application will continue to work, giving the former team member access to your storage account.

To refresh keys:

1. Change each trusted app to use the secondary key.
2. Refresh the primary key in the Azure portal. Consider this as the new secondary key value.

Understand shared access signatures

As a best practice, you shouldn't share storage account keys with external third-party applications. If these apps need access to your data, you'll need to secure their connections without using storage account keys.

For untrusted clients, use a shared access signature (SAS). A shared access signature is a string that contains a security token that can be attached to a URI. Use a shared access signature to delegate access to storage objects and specify constraints, such as the permissions and the time range of access.

You can give a customer a shared access signature token, for example, so they can upload pictures to a file system in Blob storage. Separately, you can give a web application permission to read those pictures. In both cases, you allow only the access that the application needs to do the task.

Types of shared access signatures

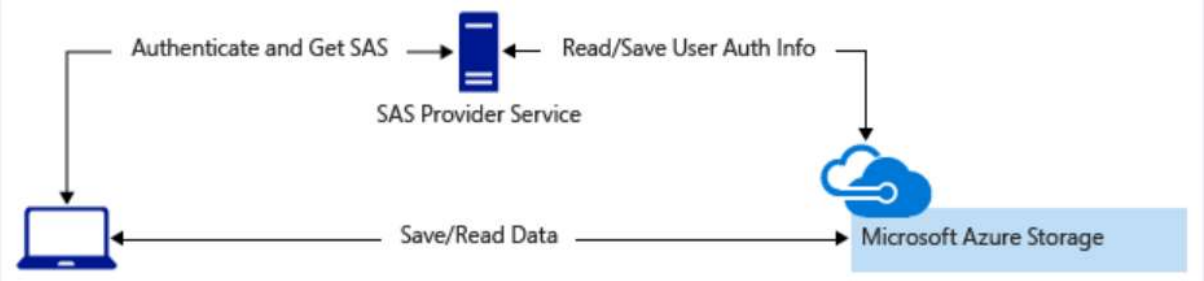
You can use a *service-level* shared access signature to allow access to specific resources in a storage account. You'd use this type of shared access signature, for example, to allow an app to retrieve a list of files in a file system or to download a file.

Use an *account-level* shared access signature to allow access to anything that a service-level shared access signature can allow, plus additional resources and abilities. For example, you can use an account-level shared access signature to allow the ability to create file systems.

You'd typically use a shared access signature for a service where users read and write their data to your storage account. Accounts that store user data have two typical designs:

- Clients upload and download data through a front-end proxy service, which performs authentication. This front-end proxy service has the advantage of allowing validation of business rules. But if the service must handle large amounts of data or high-volume transactions, you might find it complicated or expensive to scale this service to match demand.

- A lightweight service authenticates the client as needed. Then it generates a shared access signature. After receiving the shared access signature, the client can access storage account resources directly. The shared access signature defines the client's permissions and access interval. The shared access signature reduces the need to route all data through the front-end proxy service.



Control network access to your storage account

By default, storage accounts accept connections from clients on any network. To limit access to selected networks, you must first change the default action. You can restrict access to specific IP addresses, ranges, or virtual networks.

Understand Advanced Threat Protection for Azure Storage

Advanced Threat Protection detects anomalies in account activity. It then notifies you of potentially harmful attempts to access your account. You don't have to be a security expert or manage security monitoring systems to take advantage of this layer of threat protection.

Currently, Advanced Threat Protection for Azure Storage is available for the Blob service. Security alerts are integrated with Azure Security Center. The alerts are sent by email to subscription admins.

Explore Azure Data Lake Storage security features

Azure Data Lake Storage Gen2 provides a first-class data lake solution that allows enterprises to pull together their data. It's built on Azure Blob storage, so it inherits all of the security features we've reviewed in this module.

Along with role-based access control (RBAC), Azure Data Lake Storage Gen2 provides access control lists (ACLs) that are POSIX-compliant and that restrict access to only authorized users, groups, or service principals. It applies restrictions in a way that's flexible, fine-grained, and manageable. Azure Data Lake Storage Gen2 authenticates through Azure Active Directory OAuth 2.0 bearer tokens. This allows for flexible authentication schemes, including federation with Azure AD Connect and multifactor authentication that provides stronger protection than just passwords.

More significantly, these authentication schemes are integrated into the main analytics services that use the data. These services include Azure Databricks, HDInsight, and SQL Data Warehouse. Management tools such as Azure Storage Explorer are also included.

After authentication finishes, permissions are applied at the finest granularity to ensure the right level of authorization for an enterprise's big-data assets.

The Azure Storage end-to-end encryption of data and transport layer protections complete the security shield for an enterprise data lake. The same set of analytics engines and tools can take advantage of these additional layers of protection, resulting in complete protection of your analytics pipelines.

Summary

Azure Storage provides a layered security model. Use this model to secure your storage accounts to a specific set of supported networks. When you set up network rules, only applications that request data over the specified networks can access your storage account.

Authorization is supported by a public preview of Azure Active Directory credentials (for blobs and queues), a valid account access key, or a shared access signature (SAS) token. Data encryption is enabled by default, and you can proactively monitor systems by using Advanced Threat Protection.

1. You are working on a project with a 3rd party vendor to build a website for a customer. The image assets that will be used on the website are stored in an Azure Storage account that is held in your subscription. You want to give read access to this data for a limited period of time. What security option would be the best option to use?

Shared Access Signatures

Correct. A shared access signature is a string that contains a security token that can be attached to a URI. Use a shared access signature to delegate access to storage objects and specify constraints, such as the permissions and the time range of access.

2. When configuring network access to your Azure Storage Account, what is the default network rule?

To allow all connections from all networks

Correct. The default network rule is to allow all connections from all networks.

3. Which Azure service detects anomalies in account activities and notifies you of potential harmful attempts to access your account?

Advanced Threat Protection

Advanced Threat Protection, detects anomalies in account activity. It then notifies you of potentially harmful attempts to access your account.

Store application data with Azure Blob storage

Azure Blob storage is unstructured, meaning that there are no restrictions on the kinds of data it can hold. For example, a blob can hold a PDF document, a JPG image, a JSON file, video content, etc. Blobs aren't limited to common file formats — a blob could contain gigabytes of binary data streamed from a scientific instrument, an encrypted message for another application, or data in a custom format for an app you're developing.

Blobs are usually not appropriate for structured data that needs to be queried frequently. They have higher latency than memory and local disk and don't have the indexing features that make databases efficient at running queries. However, blobs are frequently used in combination with databases to store non-queryable data. For example, an app with a database of user profiles could store profile pictures in blobs. Each user record in the database would include the name or URL of the blob containing the user's picture.

Blobs are used for data storage in many ways across all kinds of applications and architectures:

- Apps that need to transmit large amounts of data using messaging system that supports only small messages. These apps can store data in blobs and send the blob URLs in messages.
- Blob storage can be used like a file system for storing and sharing documents and other personal data.
- Static web assets like images can be stored in blobs and made available for public download as if they were files on a web server.
- Many Azure components use blobs behind the scenes. For example, Azure Cloud Shell stores your files and configuration in blobs, and Azure Virtual Machines uses blobs for hard-disk storage.

Storage accounts, containers, and metadata

In Blob storage, every blob lives inside a blob container. You can store an unlimited number of blobs in a container and an unlimited number of containers in a storage account. Containers are "flat" — they can only store blobs, not other containers.

Blobs and containers support metadata in the form of name-value string pairs. Your apps can use metadata for anything you like: a human-readable description of a blob's contents to be displayed by the application, a string that your app uses to determine how to process the blob's data, etc.

A single storage account is flexible enough to organize your blobs however you like, but you should use additional storage accounts as necessary to logically separate costs and control access to data.

Apps using blobs as part of a storage scheme that includes a database often don't need to rely heavily on organization, naming, or metadata to indicate anything about their data. Such apps commonly use identifiers like GUIDs as blob names and reference these identifiers in

database records. The app will use the database to determine where blobs are stored and the kind of data they contain.

Other apps may use Azure Blob storage more like a personal file system, where container and blob names are used to indicate meaning and structure. Blob names in these kinds of apps will often look like traditional file names and include file name extensions like .jpg to indicate what kind of data they contain. They'll use virtual directories (see below) to organize blobs and will frequently use metadata tags to store information about blobs and containers.

Public access and containers as security boundaries

By default, all blobs require authentication. However, individual containers can be configured to allow public downloading of their blobs without authentication. This feature supports many use cases, such as hosting static website assets and sharing files. This is because downloading blob contents works the same way as reading any other kind of data over the web: you just point a browser or anything that can make a GET request at the blob URL.

Enabling public access is important for scalability because data downloaded directly from Blob storage doesn't generate any traffic in your server-side app. Even if you don't immediately take advantage of public access or if you will use a database to control data access via your application, plan on using separate containers for data you want to be publicly available.

In addition to public access, Azure has a shared access signature feature that allows fine-grained permissions control on containers. Precision access control enables scenarios that further improve scalability, so thinking about containers as security boundaries in general is a helpful guideline.

Blob name prefixes (virtual directories)

Technically, containers are "flat" and do not support any kind of nesting or hierarchy. But if you give your blobs hierarchical names that look like file paths (such as finance/budgets/2017/q1.xls), the API's listing operation can filter results to specific prefixes. This allows you to navigate the list as if it was a hierarchical system of files and folders.

This feature is often called virtual directories because some tools and client libraries use it to visualize and navigate Blob storage as if it was a file system. Each folder navigation triggers a separate call to list the blobs in that folder.

Using names that are like file names for blobs is a common technique for organizing and navigating complex blob data.

Blob types

There are three different kinds of blobs you can store data in:

- **Block blobs** are composed of blocks of different sizes that can be uploaded independently and in parallel. Writing to a block blob involves uploading data to blocks and committing them to the blob.
- **Append blobs** are specialized block blobs that support only appending new data (not updating or deleting existing data), but they're very efficient at it. Append blobs are great for scenarios like storing logs or writing streamed data.
- **Page blobs** are designed for scenarios that involve random-access reads and writes. Page blobs are used to store the virtual hard disk (VHD) files used by Azure Virtual Machines, but they're great for any scenario that involves random access.

Block blobs are the best choice for most scenarios that don't specifically call for append or page blobs. Their block-based structure supports very fast uploads and downloads and efficient access to individual pieces of a blob. The process of managing and committing blocks is automatically handled by most client libraries, and some will also handle parallel uploads and downloads to maximize performance.

Summary

1. Suppose you need to store profile and order information about your customers. You need to query the data to answer questions like "who are my top 100 customers?" and "how many customers live in a given geographic region?". True or false: blob storage is a good choice for this data?
False
Blobs are not appropriate for structured data that needs to be queried frequently. They have higher latency than memory and local disk and don't have the indexing features that make databases efficient at running queries.
2. Blobs provide unstructured data storage. What does unstructured mean?
There are no restrictions on the type of data you can store in blobs.
Blobs do not impose any structure on your data, meaning your application can store any type of data in a blob.
3. Which of the following describes a good strategy for creating storage accounts and blob containers for your application?
Create Azure Storage accounts before deploying your app. Create containers in your application as needed.
Creating an Azure Storage account is an administrative activity and can be done prior to deploying an application. Container creation is lightweight and is often driven by run-time data which makes it a good activity to do in your application.
4. Which of the following can be used to initialize the Blob Storage client library within an application?
The Azure Storage account connection string.
A storage account connection string contains all the information needed to connect to Blob storage, most importantly the account name and the account key.
5. What happens when you call GetBlockBlobReference with the name of a blob?
A CloudBlockBlob object is created locally. No network calls are made.
Getting a blob reference does not make any calls to Azure Storage, it simply creates an object locally that can work with a stored blob.