**Classify your data**

1. Application data can be classified in one of three ways: **structured**, **semi**-**structured**, and **unstructured**
2. **Structured data**
   1. Structured data, sometimes referred to as relational data, is data that adheres to a strict schema
   2. Searched with SQL
   3. Structured data is often stored in database tables with rows and columns and keys
   4. Structured data is straightforward in that it's easy to enter, query, and analyse
3. **Semi-structured data**
   1. Less organized than structured
   2. Not stored in a relational format, as the fields do not neatly fit into tables, rows, and columns
   3. Semi-structured data contains **tags** that make the organization and hierarchy of the data
   4. Semi-structured data is also referred to as non-relational or NoSQL data
4. **Common formats**
   1. XML:
      1. Text-based, which makes it easily human and machine-readable
      2. XML allows you to express relationships and has standards for schema, transformation, and even displaying on the web
      3. XML expresses the shape of the data using tags.
      4. 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
   2. JSON:
      1. JavaScript Object Notation
      2. Lightweight specification and relies on curly braces to indicate data structure
      3. Compared to XML, it is less verbose and easier to read by humans
      4. JSON is frequently used by web services to return data
      5. It's closer to a key/value pair model than a formal data expression
   3. YAML
      1. Ain’t Markup Language
      2. The data structure is defined by line separation and indentation, and reduces the dependency on structural characters like parentheses, commas and brackets
      3. YAML is the newest of these data formats and doesn't have as much support in programming languages as JSON and XML
5. **What is NoSQL / semi-structured data?**
   1. DataModels in NoSQL
      1. Key-Value: Store in key-value pairs. Don’t use query language. Use commands such as get, put, delete
      2. Graph: Uses nodes to make relationships.
      3. Document: Store data use JSON or XML

**Determine operational needs**

1. Operations and latency
   1. Ask yourself these questions:
      1. Will you be doing simple lookups using an ID
      2. Do you need to query the database for one or more fields
      3. How many create, update, and delete operations do you expect
      4. Do you need to run complex analytical queries
      5. How quickly do these operations need to complete

**Group multiple operations in a transaction**

1. Transactions enable you to group updates so that if one event in a series of updates fails, the entire series can be rolled back, or undone
2. What is a transaction?
   1. A transaction is a logical group of database operations that execute together
   2. Question to ask: Will a change to one piece of data in your dataset impact another?
   3. If the answer is yes, then you'll need support for transactions in your database service.
3. Transactions defined by ACID
   1. **Atomicity**: means a transaction must execute exactly once and must be atomic; either all of the work is done, or none of it it
   2. **Consistency**: Ensures that the data is consistent before and after a transaction
   3. **Isolation**: Ensures that one transaction is not impacted by another transaction
   4. **Durability**: Changes are permanently saved in the system

**OLTP vs OLAP**

1. Transactional databases are often called OLTP (Online Transaction Processing) systems. Support a lot of users
2. On the contrary, OLAP (Online Analytical Processing) systems commonly support fewer users

**Product catalog data**

1. Product catalog data should be stored in a transactional database

**Photos and videos**

1. Photos and videos in a product catalog don't require transactional support
2. These files are changed only when an update is made or new files are added

**Business data**

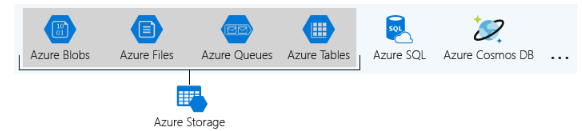
1. Because all of the data is historical and unchanging, transactional support is not required

# **Choose a storage solution on Azure**

1. Product catalog data
   1. Semi-structured because of the need to extend or modify the schema for new products
2. Recommended service: Azure Cosmos DB (For semi-structured data)
   1. Azure Cosmos DB supports semi-structured data, or NoSQL data, by design
   2. Azure Cosmos DB supports SQL for queries
   3. Azure Cosmos DB is also ACID-compliant, so you can be assured that your transactions are completed according to those strict requirements
3. Why not other Azure services?
   1. Azure SQL Database enables you to combine structured data in the columns, and semi-structured data stored as JSON columns that can be easily extended
   2. Azure SQL Database can provide many of the same benefits of Azure Cosmos DB, but it provides little benefit if the structure of your data is changing in different entities
   3. Azure Cosmos DB is better choice for highly unstructured and variable data
   4. Other Azure services, such as **Azure** **Table** **storage**, **Azure** **HBase** as a part of **HDInsight**, and **Azure** **Cache** for **Redis**, can also store **NoSQL** data
4. Recommended service: Azure Blob storage (For unstructured data)
   1. Azure Blob storage supports storing files such as photos and videos
   2. Works with Azure Content Delivery Network (CDN) by caching the most frequently used content and storing it on edge servers
   3. Azure CDN reduces latency in serving up those images to your users
   4. You could upload your images to Azure App Service. This would work if you did not have many files
5. Recommended service: Azure SQL Database (For structured data)
   1. Business data will most likely be queried by business analysts, who are more likely to know SQL
   2. Combining Azure SQL database with Azure Analysis Services enables data analysts to create a semantic model over the data in SQL Database
6. Why not other Azure services?
   1. Azure Synapse supports OLAP solutions and SQL queries
   2. Azure Analysis Services could be used in addition to Azure SQL Database

**Create an Azure Storage account**

**Decide how many storage accounts you need**

1. What is Azure Storage?
   1. There are multiple **database** options like **Azure** **SQL** **Database**, **Azure** **Cosmos** **DB**, and **Azure** **Table** **Storage**
   2. Azure offers multiple ways to store and send **messages**, such as **Azure** **Queues** and **Event** **Hubs**
   3. Azure selected four of these data services and placed them together under the name Azure Storage
   4. The four services are **Azure** **Blobs**, **Azure** **Files**, **Azure** **Queues**, and **Azure** **Tables**
2. What is a storage account?
   1. A **storage** **account** is a **container** that groups a set of Azure Storage services together
   2. Combining data services into a storage account lets you manage them as a group
   3. Deleting the storage account deletes all of the data stored inside it
   4. A storage account is an Azure resource and is included in a resource group
   5. 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
3. Storage account settings
   1. The settings that are defined by a storage account are:
      1. **Subscription**: billed for the services
      2. **Location**: the datacenter
      3. **Performance**: *Standard* – magnetic disk drives, *Premium* - SSD.
      4. **Replication**: strategy used to make copies of your data to protect against hardware failure or natural disaster. Azure will automatically maintain three copies of your data, called locally-redundant storage (LRS). Geo-redundant storage (GRS) to get replication at different datacenters across the world
      5. **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
      6. **Secure** **Transfer** **Required**: Enabled requires HTTPs, while disabled allows HTTP
      7. **Virtual** **Network**: A security feature that allows inbound access requests only from the virtual network(s) you specify
4. How many storage accounts do you need?
   1. A storage account represents a collection of settings like location, replication strategy, and subscription owner
   2. The number of storage accounts you need is typically determined by your **data** **diversity**, **cost** **sensitivity**, and **tolerance** for **management** **overhead**
5. Data Diversity
   1. Two questions:
      1. Do you have data that is specific to a country or region?
      2. Do you have some data that is proprietary and some for public consumption?
   2. Increased diversity means an increased number of storage accounts
6. Cost sensitivity
   1. A storage account by itself has no financial cost
   2. However, the settings influence the service
   3. Geo-redundant storage costs more than locally-redundant storage
   4. Premium performance and the Hot access tier increase the cost of blobs
   5. You can use multiple storage accounts to reduce costs
7. Tolerance for management overhead
   1. Complexity is added for anyone who adds data to your cloud storage
   2. Storage accounts are a powerful tool to help you get the performance and security you need while minimizing costs

**Choose your account settings**

1. **Name**: The name must be globally unique within Azure, only lowercase letters and digits and be between 3 and 24 characters
2. **Deployment** **Model**: System Azure uses to organize your resources. The model defines the API that you use to create, configure, and manage those resources. Microsoft recommends that you use Resource Manager for all new resources. Two deployment models:
   1. **Resource Manager:** Current model
   2. **Classic:** Legacy offering
3. **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. Microsoft recommends that you use the General-purpose v2. Three types of storage accounts:
   1. **StorageV2**
   2. **Storage (general purpose V1)**
   3. **Blob Storage**
4. 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

**Choose an account creation tool**

1. Available tools
   1. Azure Portal
   2. Azure CLI (command line interface)
   3. Azure PowerShell
   4. Management Client Libraries
2. The portal provides a GUI with explanations for each setting
3. This makes the portal easy to use and helpful for learning about the options
4. How to choose a tool
   1. Storage-account creation is usually a one-time operation done at the start of a project
   2. Portal is the most common choice
   3. Scripts are typically faster to create and less work to maintain

**Exercise - Create a storage account using the Azure portal**

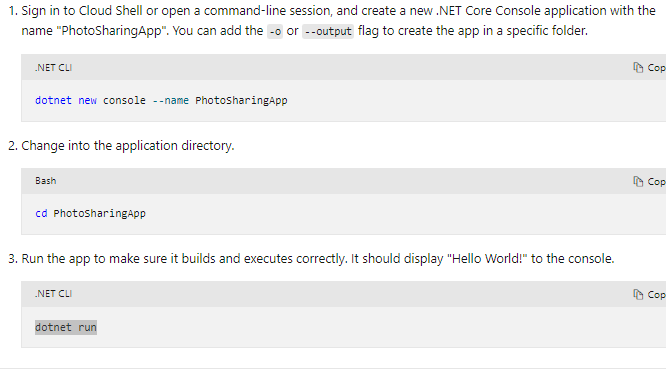
In general, increased diversity means an increased number of storage accounts. A storage account by itself has no financial cost. However, the settings you choose for the account do influence the cost of services in the account. Use multiple storage accounts to reduce costs.

**Connect an App to Azure Storage**

1. Explore Azure storage services
   1. **Durable**: Redundancy ensures that your data is safe in the event of hardware failure
   2. **Secure:** Encrypted by the service
   3. **Scalable:** Massively scalable to meet the data storage and performance needs
   4. **Managed:** Handles maintenance and any critical problems for you
   5. A single Azure subscription can host up to 200 storage accounts, each of which can hold 500 TB of data
2. Azure data services
   1. Azure storage includes four types of data:
      1. **Blobs**: Massively scalable object store for text and binary data. Support for Azure Data Lakes
      2. **Files**: For cloud or on-premises deployments
      3. **Queues**: A messaging store for reliable messaging
      4. **Table** **Storage**: A NoSQL store for schema-less storage of structured data
   2. Accessible anywhere in the world over HTTP or HTTPS
3. Blob storage
   1. Storing massive amounts of unstructured data, such as text or binary data
   2. Ideal for:
      1. **Serving** **images** or documents directly to a browser
      2. **Storing** **files** for distributed access
      3. **Streaming** **video** and **audio**
      4. **Storing** data for **backup** and **restoration**, **disaster** **recovery**, and **archiving**
      5. **Storing** **data** for **analysis** by an **on**-**premises** or **Azure**-**hosted** service
   3. Three kinds of blobs:
      1. **Block** **blobs**: Block blobs are used to hold text or binary files up to 5 TB
      2. **Page** **blobs**: Page blobs are used to hold random-access files up to 8 TB in size. Provide random read/write access to 512-byte pages
      3. **Append** **blobs**: Append blobs are made up of blocks like block blobs, but they are optimized for append operations. A single append blob can be up to 195 GB
4. Files
   1. Highly available network file shares that can be accessed using the standard Server Message Block (SMB) protocol.
   2. Scenarios:
      1. Storing shared configuration files for VMs
      2. Log files
      3. Shared data between on-premises and Azure VMs
5. Queues
   1. The Azure Queue service is used to store and retrieve messages
   2. Queue messages can be up to 64 KB in size, and a queue can contain millions of messages
   3. Processed asynchronously
6. Azure storage accounts
   1. To access any of these services from an application, you have to create a storage account
   2. The storage account provides a unique namespace in Azure to store and access your data objects
   3. Contains any blobs, files, queues, tables, and VM disks
7. Create a storage account
   1. You can create an Azure storage account using the Azure portal, Azure PowerShell, or Azure CLI
   2. Azure provides three distinct options:
      1. **General-purpose v2 (GPv2**): Support all of the latest features for blobs, files, queues, and tables.
      2. **General-purpose v1 (GPv1)**: May not have the latest features
      3. **Blob Storage**: A legacy account type, blob storage accounts support all the same block blob features as GPv2

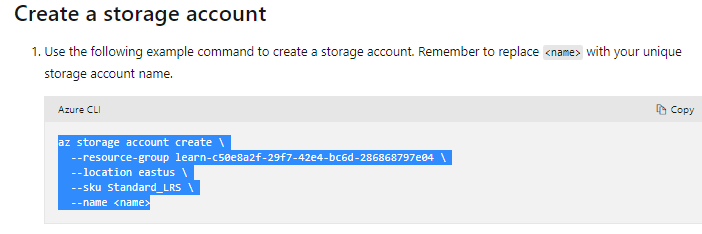
# **Exercise - Create a new app to work with Azure storage**

1. .NET Core is a cross-platform version of .NET that runs on macOS, Windows, and Linux



# **Exercise - Create an Azure storage account**

1. We'll use the az storage account create command to create a new storage account

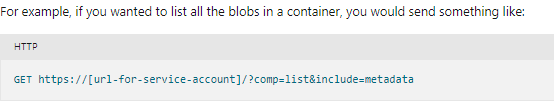


# **Interact with the Azure Storage APIs**

1. Azure Storage provides a REST API to work with the containers and data stored in each account
2. Recall four data types: Blob, Queues, Table and Files

# **Use the REST API**

1. The Storage REST APIs are accessible from anywhere on the Internet via HTTP or HTTPS



1. Azure provides pre-built client libraries that make working with the service easier

# **Use a client library**

1. Save a significant amount of work for app developers because the API is tested and it often provides nicer wrappers
2. Supports:
   1. .NET
   2. Java
   3. Python
   4. Node.js
   5. Go

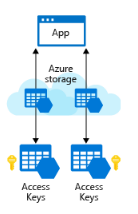


1. The client libraries are just thin wrappers over the REST API

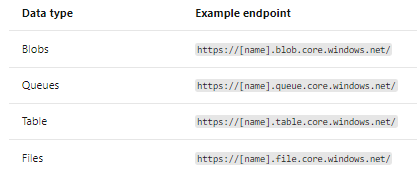
# **Exercise - Add the storage client library to your app**

# **Connect to your Azure storage account**

1. To work with data in a storage account, your app will need two pieces of data:
   1. Access Key
   2. REST API endpoint
2. Security access keys
   1. Each storage account has two unique access keys that are used to secure the storage account

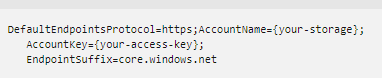


1. REST API endpoint
   1. In addition to access keys for authentication your pp will need to know the storage service endpoints to issue the REST requests
   2. The REST endpoint is a combination of your storage account name, the data type, and a known domain, for example:

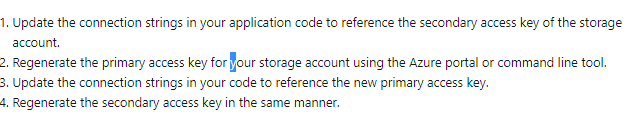


* 1. If you have a custom domain tied to Azure, then you can also create a custom domain URL for the endpoint

1. Connection strings
   1. The **simplest** way to handle access keys and endpoint URLs within applications is to use **storage** **account** **connection** **strings**
   2. Example:



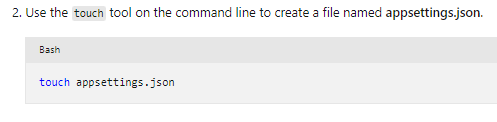
1. Security
   1. Access keys are critical to providing access to your storage account
   2. They should not be given to any system or person that you do not want to have access to your storage account
   3. Each storage account has two access keys
   4. Reason for this is to allow keys to be rotated (regenerated) periodically
   5. Rotating a key will invalidate the original key value immediately and will revoke access to anyone who obtained the key inappropriately
   6. With support for two keys, you can rotate keys without causing downtime in your applications
   7. Process:

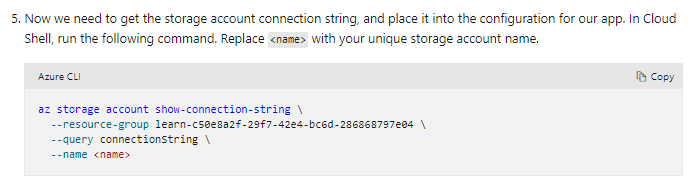


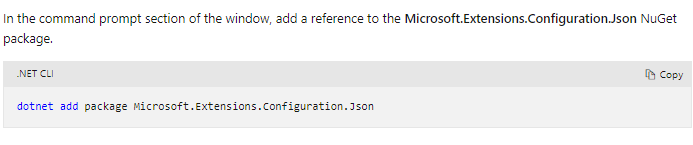
* 1. If you are using the key in a server application, you can use an Azure Key Vault to store the access key for you

1. Shared access signatures (SAS)
   1. Access keys are the easiest approach to authenticating access
   2. However they provide full access to anything in the storage account. Similar to root password of a computer
   3. Storage accounts offer a separate authentication mechanism called **shared** **access** **signatures**
   4. Support expiration and limited permissions

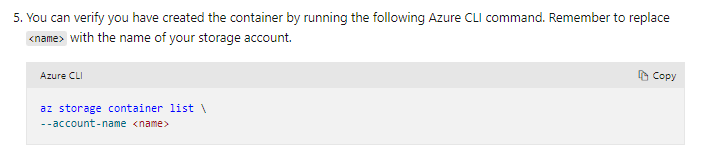
# **Exercise - Add Azure Storage configuration to your app**



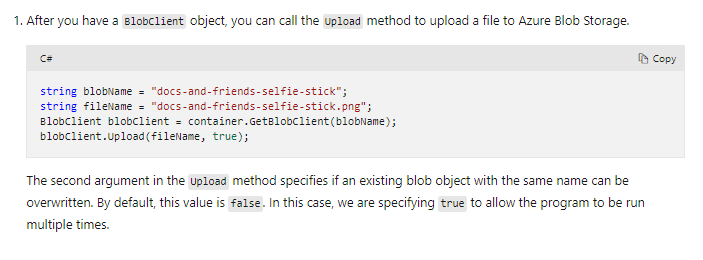




# **Exercise - Connect your application to your Azure Storage account**



# **Exercise - Upload an image to your Azure Storage account**



# **Explore Azure Storage security features**

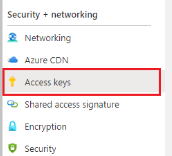
1. Azure Storage accounts provide several high-level security benefits for the data in the cloud:
   1. Protect the data at rest
   2. Protect the data in transit
   3. Support cross-domain access
   4. Control who can access the data
   5. Audit storage access
2. Encryption at rest
   1. All data written to Azure Storage is automatically encrypted by Storage Service Encryption (SSE) with a 256-bit Advanced Encryption Standard (AES) cipher
   2. When you read data from Azure Storage, Azure Storage decrypts the data before returning it
   3. This process incurs no additional charges
   4. For virtual machines (VMs), Azure lets you encrypt virtual hard disks (VHDs) by using Azure Disk Encryption
   5. Azure Key Vault stores the keys automatically to help you control and manage the disk-encryption keys and secrets
3. Encryption in transit
   1. Keep your data secure by enabling transport-level security between Azure and the client
   2. Always use HTTPS to secure communication over the public internet
   3. After you enable secure transfer, connections that use HTTP will be refused
4. CORS Support
   1. Azure Storage supports cross-domain access through cross-origin resource sharing (CORS)
   2. CORS uses HTTP headers so that a web application at one domain can access resources from a server at a different domain
   3. By using CORS, web apps ensure that they load only authorized content from authorized sources CORS support an optional flag, with appropriate headers for HTTP GET requests from the storage account
5. Role-based access control
   1. To access data in a storage account, the client makes a request over HTTP or HTTPS
   2. Azure Storage supports Azure Active Directory and role-based access control (RBAC)
6. Auditing access
   1. Auditing is another part of controlling access

# **Understand storage account keys**

1. Using a shared key or shared secret, this authentication option is one of the easiest to use, and it supports blobs, files, queues, and tables

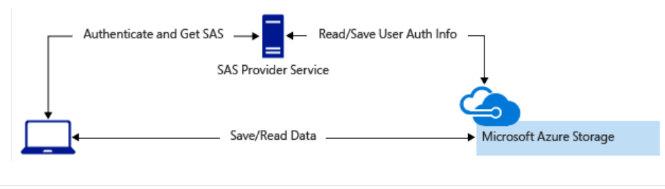


1. Storage account keys
   1. In Azure Storage accounts, shared keys are called storage account keys
   2. Azure creates two of these keys (primary and secondary) for each storage account you create. The keys give access to everything in the account
   3. You'll find the storage account keys in the Azure portal view of the storage account. In the left menu pane of your storage account, select **Security** + **networking** > **Access** **keys**



1. Protect shared keys
   1. The storage account has only two keys, and they provide full access to the account
   2. Use them only with trusted in-house applications
   3. If the keys are compromised, change the key values in the Azure portal
   4. Here are several reasons to regenerate your storage account keys:
      1. For security reasons
      2. Being hacked
      3. Team member leaves

# **Understand shared access signature**

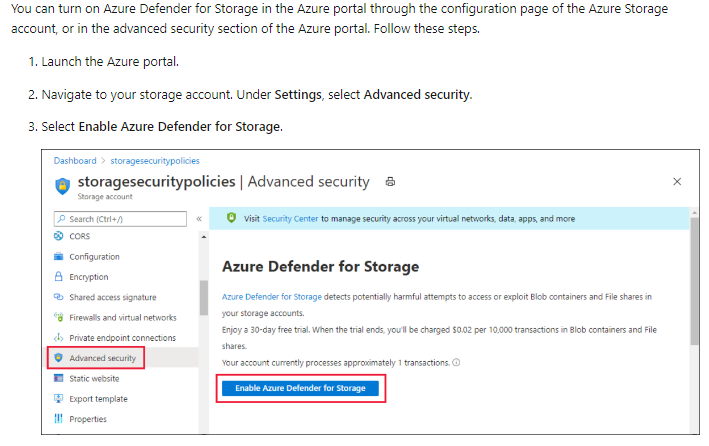
1. You shouldn't share storage account keys with external third-party applications
2. If these apps need access to your data, you'll need to secure their connections without using storage account keys
3. For untrusted clients, use a shared access signature (SAS).
4. SAS is a string that contains a security token that can be attached to a URI
5. You can give a customer a SAS token
6. Types of shared access signatures
   1. You can use a **service**-**level** **SAS** to allow access to specific resources in a storage account
   2. Use an **account**-**level** **SAS** to allow access to anything that a service-level SAS can allow, plus additional resources and abilities
   3. Accounts that store user data have two typical designs:
      1. Clients upload and download data through a front-end proxy service, which performs authentication. If the service must handle large amounts of data or high-volume transactions, you might find it complicated or expensive
      2. A lightweight service authenticates the client, as needed. Next, it generates a SAS. After receiving the SAS, the client can access storage account resources directly
      3. It reduces the need to route all data through the front-end proxy service

# **Control network access to your storage account**

1. By default, storage accounts accept connections from clients on any network.

# **Understand advanced threat protection for Azure Storage**

1. **Azure** **Defender** for **Storage** provides an extra layer of security
2. Security alerts are triggered when anomalies in activity occur
3. **Azure** **Defender** for **Storage** is currently available for Blob storage, Azure Files, and Azure Data Lake Storage Gen2



# **Explore Azure Data Lake Storage security features**

1. Azure Data Lake Storage Gen2 provides a first-class data lake solution
2. It's built on Azure Blob storage
3. Provides role based access control (RBAC) and provides access control lists (ACLs) that are POSIX-compliant
4. Services that use this are Azure Databricks, HDInsight, and Azure Synapse Analytics

# **Store application data with Azure Blob storage**

1. What are Blobs
   1. Blobs are files for the cloud
   2. Blobs can be reached from anywhere with an internet connection
   3. Azure Blob storage is unstructured - No restrictions on the kinds of data it can hold
   4. Can hold a PDF document, a JPG image, a JSON file, video content, etc
   5. Blobs are usually not appropriate for structured data
   6. They have higher latency than memory
   7. For example, an **app** with a **database** of **user** **profiles** could **store** profile **pictures** in **blobs**
   8. Blobs are used for data storage in many ways:
      1. Apps that need to transmit large amounts of data
      2. Blob storage can be used like a file system
      3. Static web assets like images can be stored in blobs
      4. Many Azure components use blobs behind the scenes

# **Storage accounts, containers, and metadata**

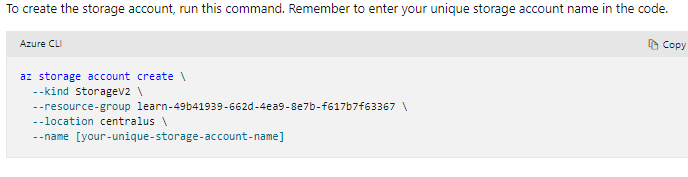
1. Every blob lives inside a blob container
2. You can store an **unlimited** **number** of **blobs** in a **container** and an **unlimited** **number** of **containers** in a **storage** **account**
3. Blobs and containers support metadata in the form of name-value string pairs

# **Design a storage organization strategy**

1. Storage accounts
   1. You should use additional storage accounts as necessary to logically separate costs and control access to data
2. Containers and blobs
   1. The **nature** of your **app** and the data that it stores should **drive** your strategy for **naming** and **organizing** containers and blobs
   2. Some apps may use Azure Blob storage more like a personal file system, where container and blob names are used to indicate meaning and structure.
   3. **Blob** **names** in these kinds of apps will often look like traditional file names and include **file** name **extensions** like .**jpg**
3. Public access and containers as security boundaries
   1. By default, all blobs require authentication to access
   2. However, individual containers can be configured to allow public downloading
   3. This is because downloading blob contents works the same way as reading any other kind of data over the web
   4. Never put blob data in a public container that you don't intend to share publicly
   5. Azure has a **shared** **access** **signature** feature that allows **fine**-**grained** **permissions** control on containers.
4. Blob name prefixes (virtual directories)
   1. Technically, containers are "flat" and do not support any kind of nesting or hierarchy
   2. But if you give your blobs hierarchical names that look like file paths (such as finance/budgets/2017/q1.xls), this enables a hierarchical system of files and folders
   3. This is called virtual directories
5. Blob types
   1. **Block** **Blobs:** Composed of blocks of different sizes that can be uploaded independently and in parallel. Involves uploading data to blocks and committing them to the blob
   2. **Append** **Blobs:** Are Block Blobs that support only appending new data. Good for storing logs
   3. **Page** **Blobs:** For random-access reads and writes
6. Blobs don't have the indexing features that make databases efficient at running queries

# **Exercise - Create Azure storage resources**

1. Storage Accounts
   1. Accounts are usually created by a deployment or environment setup script, an Azure Resource Manager template, or manually by an admin
2. It's not uncommon for apps to create and delete containers as part of their work
3. Typical practice is to let the app create the containers it needs on startup

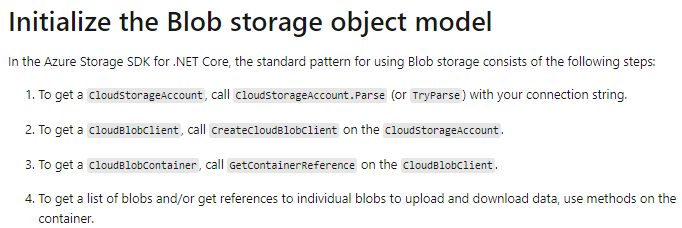


To view the documentation run the following in the Cloud Shell Terminal:

**az storage container create –h**

# **Exercise - Configure and initialize the client library**

1. Before running your app, you'll need the connection string for the storage account
2. Use Azure management interface to get it, including the Azure portal, the Azure CLI, or Azure PowerShell
3. Storage account connection strings include the account key
4. The account key is considered a secret and should be stored securely



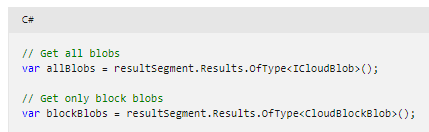
# **Exercise - Get blob references**

1. You can get an **ICloudBlob** by requesting it with the blob's name
2. Getting blobs by name
   1. To get an **ICloudBlob** by name, call one of the **GetXXXReference** methods on a **CloudBlobContainer**
   2. Specific methods (**GetBlockBlobReference**, **GetAppendBlobReference**, or **GetPageBlobReference**)
   3. These only create a blob reference object **locally**
   4. A separate method, **GetBlobReferenceFromServerAsync**, does call the Blob storage API
3. Listing blobs in a container
   1. You can get a list of the blobs in a container using **CloudBlobContainer's** **ListBlobsSegmentedAsync** method
   2. Segmented refers to the separate pages of results returned



* 1. Never assume that ListBlobsSegmentedAsync results will arrive in a single page

1. Processing list results



# **Exercise - Blob uploads and downloads**

1. To create a new blob, call one of the Upload methods on a reference to a blob that doesn't exist in storage
2. This does two things:
   1. Creates the blob in storage
   2. Uploads the data

## Move data to and from blobs

1. Moving data to and from a blob is a network operation that takes time
2. All methods that require network activity return **Tasks**

## Concurrent access

1. Other processes may be adding, changing, or deleting blobs as your app is using them
2. Always code defensively and think about problems caused by concurrency
3. Such as blobs that are deleted right as you try to download from them, or blobs whose contents change when you don't expect them to





