Develop for Azure Storage

└─ **2.1 Develop solutions that use Azure Cosmos DB**

└─ **2.1.1 Perform operations on containers and items by using the SDK**

1. What SDKs are supported for Cosmos DB operations?
2. How do you create a container in Cosmos DB using the SDK?
3. How do you insert or update an item?
4. How do you query items using SQL syntax?
5. How do you delete an item by ID?
6. How do you use partition keys effectively?
7. What are common consistency levels and how do you set them?
8. How do you use the CosmosClient safely and efficiently?
9. How do you handle pagination (continuation tokens)?
10. How is exception handling and retry logic implemented?

---

1. What SDKs are supported for Cosmos DB operations?

.NET (Microsoft.Azure.Cosmos), Java, Python, Node.js

---

2. How do you create a container in Cosmos DB using the .NET SDK?

```
await database.CreateContainerIfNotExistsAsync("MyContainer", "/partitionKey");
```

- "/partitionKey" is required.
- Creates the container only if it doesn't exist.

---

3. How do you insert or update an item?

```
await container.UpsertItemAsync(item, new PartitionKey(item.partitionKey));
```

- UpsertItemAsync inserts or replaces item based on ID.
- Requires correct partition key.

---

4. How do you query items using SQL syntax?

```
var query = container.GetItemQueryIterator<MyItem>("SELECT * FROM c WHERE c.status = 'active'");
while (query.HasMoreResults)
{
   foreach (var item in await query.ReadNextAsync())
   {
      // Process item
   }
}
```

- Uses Cosmos SQL API.
- Handles pagination internally.

---

5. How do you delete an item by ID?

```
await container.DeleteItemAsync<MyItem>(id, new PartitionKey(partitionKey));
```

- Both ID and correct partition key are required.

---

6. How do you use partition keys effectively?

- Choose a key with high cardinality and even distribution (e.g., /userId).
- Required for most operations (read, update, delete).

7. What are common consistency levels and how do you set them?

- Levels: Strong, BoundedStaleness, Session (default), ConsistentPrefix, Eventual
- Set at CosmosClientOptions level:

```
new CosmosClient(endpoint, key, new CosmosClientOptions { ConsistencyLevel = ConsistencyLevel.Session });
```

---

8. How do you use the CosmosClient safely and efficiently?

- Reuse a single CosmosClient instance (thread-safe).
- Instantiate once at app startup (e.g., via dependency injection).

---

9. How do you handle pagination (continuation tokens)?

- Use FeedIterator<T> from GetItemQueryIterator<T>()
- Cosmos handles paging; iterate until HasMoreResults is false.

---

10. How is exception handling and retry logic implemented?

- Catch CosmosException for specific status codes:

```
catch (CosmosException ex) when (ex.StatusCode == HttpStatusCode.TooManyRequests)
{
   await Task.Delay(ex.RetryAfter);
}
```

- SDK includes automatic retry policies; customize via CosmosClientOptions.

Develop for Azure Storage

└─ **2.1 Develop solutions that use Azure Cosmos DB**

└─ **2.1.2 Set the appropriate consistency level for operations**

1. What are the consistency levels supported by Azure Cosmos DB?
2. What is the default consistency level and why is it recommended?
3. How do you configure consistency level at the account level?
4. How do you override the consistency level per request?
5. What is session consistency and when should it be used?
6. What are the trade-offs between strong and eventual consistency?
7. How does consistency affect performance and availability?
8. How do you check the current consistency level of an account?
9. Which operations are affected by the chosen consistency level?
10. What are best practices for setting consistency levels in real-world applications?

---

1. What are the consistency levels supported by Azure Cosmos DB?

- Strong
- BoundedStaleness
- Session *(default)*
- ConsistentPrefix
- Eventual

---

2. What is the default consistency level and why is it recommended?

- Session is default.
- Guarantees *read-your-own-writes* within a session.
- Balanced choice for consistency and performance.

---

3. How do you configure consistency level at the account level?

- Set during Cosmos DB account creation or via Azure Portal:
  - *Settings → Default consistency*
- Or using SDK:

```
CosmosClientOptions.ConsistencyLevel = ConsistencyLevel.Session;
```

---

4. How do you override the consistency level per request?

```
var requestOptions = new QueryRequestOptions
{
   ConsistencyLevel = ConsistencyLevel.Eventual
};
```

- Applies only to the specific request.
- Must be equal or weaker than the account-level setting.

---

5. What is session consistency and when should it be used?

- Guarantees *read-your-own-writes* for a session token.
- Ideal for user-specific data scenarios (e.g., profile updates, shopping carts).

6. What are the trade-offs between strong and eventual consistency?

- Strong: Highest data accuracy, lowest availability across regions.
- Eventual: Best performance and availability, but stale reads are possible.

---

7. How does consistency affect performance and availability?

- Weaker levels (Eventual, ConsistentPrefix) offer lower latency and higher throughput.
- Stronger levels (Strong, BoundedStaleness) increase latency and reduce write availability in multi-region setups.

---

8. How do you check the current consistency level of an account?

- Use Azure Portal → *Settings* → *Default consistency*
- Or SDK:

```
var consistency = cosmosClient.ClientOptions.ConsistencyLevel;
```

---

9. Which operations are affected by the chosen consistency level?

- Read operations: The chosen level impacts how up-to-date the reads are.
- Write operations are always consistent.

---

10. What are best practices for setting consistency levels in real-world applications?

- Use Session for most app scenarios (low latency + strong enough).
- Use Strong only when global read consistency is critical.
- Use Eventual or ConsistentPrefix for high-throughput, read-heavy apps where data freshness is not critical.

Develop for Azure Storage
  └─ **2.1 Develop solutions that use Azure Cosmos DB**
    └─ **2.1.3 Implement change feed notifications**

1. What is the change feed in Azure Cosmos DB?
2. What types of changes does the change feed capture?
3. How do you read from the change feed using the SDK?
4. What is the difference between manual polling vs. Change Feed Processor?
5. How do you implement the Change Feed Processor in .NET?
6. How do you scale out a change feed listener?
7. What are common use cases for the change feed?
8. What is lease container and why is it required?
9. How do you resume reading from a specific point in the change feed?
10. What are best practices for change feed implementations?

---

1. What is the change feed in Azure Cosmos DB?

- A persistent, ordered log of item changes (inserts and updates) in a container.
- Enables event-driven processing without polling the whole dataset.

---

2. What types of changes does the change feed capture?

- Creates and updates only.
- Deletes are not included. You must implement soft delete patterns if needed.

---

3. How do you read from the change feed using the SDK?

```
var iterator = container.GetChangeFeedIterator<MyItem>(
    ChangeFeedStartFrom.Beginning(), ChangeFeedMode.Incremental);
while (iterator.HasMoreResults)
{
    var response = await iterator.ReadNextAsync();
    foreach (var item in response)
    {
        // Process item
    }
}
```

---

4. What is the difference between manual polling vs. Change Feed Processor?

- Manual polling: Directly queries the feed; full control but must manage state and scaling.
- Change Feed Processor: Auto-scales and handles lease/state tracking via a lease container.

---

5. How do you implement the Change Feed Processor in .NET?

```
var processor = container
    .GetChangeFeedProcessorBuilder<MyItem>("myProcessor", async (changes, token) =>
    {
        foreach (var item in changes) { /* process */ }
    })
    .WithInstanceName("worker1")
    .WithLeaseContainer(leaseContainer)
    .Build();
await processor.StartAsync();
```

- Requires a lease container for tracking progress.

6. How do you scale out a change feed listener?

- Use multiple instances of Change Feed Processor with the same lease container.
- The processor automatically partitions work across instances.

7. What are common use cases for the change feed?

- Event-driven processing (e.g., send emails, process orders)
- Real-time analytics
- Data movement to other stores (e.g., SQL, Blob Storage)
- Cache invalidation or sync

8. What is lease container and why is it required?

- A separate Cosmos DB container used by Change Feed Processor to track progress.
- Stores checkpoints and ownership info for scaling and fault-tolerance.

9. How do you resume reading from a specific point in the change feed?

- Change Feed Processor resumes automatically via lease container.
- Manual method: use `ChangeFeedStartFrom.Time()` or `ChangeFeedStartFrom.ContinuationToken()`.

10. What are best practices for change feed implementations?

- Use dedicated lease container in same database.
- Ensure idempotent processing logic.
- Handle throttling and retries using SDK's retry policies.
- Monitor lag and exceptions for performance tuning.

Develop for Azure Storage
└─ **2.2 Develop solutions that use Azure Cosmos DB**
└─ **2.2.1 Set and retrieve properties and metadata**

1. What are blob properties and blob metadata?
2. How do you set blob properties (like content type, cache control)?
3. How do you retrieve blob properties?
4. How do you set custom metadata on a blob?
5. How do you retrieve metadata from a blob?
6. What are best practices when using metadata in Azure Blob Storage?
7. What happens if you overwrite a blob — are properties and metadata preserved?
8. How do you set or update metadata without overwriting the blob content?
9. How do you use Azure SDK (C#, Python) to manage properties and metadata?
10. How can you search or filter blobs by metadata?

---

1. What are blob properties and blob metadata?

- Blob properties are system-defined attributes like `Content-Type`, `Content-Encoding`, `Cache-Control`, and `Content-Length`.
- Blob metadata consists of user-defined key-value pairs that describe the blob but do not affect its behavior.

---

2. How do you set blob properties (like content type, cache control)?

- When uploading or updating a blob, set properties using `BlobClient.Upload()` with `BlobHttpHeaders`.
- Example (C# Azure SDK):

```
await blobClient.UploadAsync(fileStream, new BlobHttpHeaders { ContentType = "image/png", CacheControl = "no-cache" });
```

---

3. How do you retrieve blob properties?

- Use `BlobClient.GetPropertiesAsync()` method.
- Example (C# Azure SDK):

```
BlobProperties properties = await blobClient.GetPropertiesAsync();
Console.WriteLine(properties.ContentType);
Console.WriteLine(properties.CacheControl);
```

---

4. How do you set custom metadata on a blob?

- Use `BlobClient.SetMetadataAsync()` with a dictionary of key-value pairs.
- Example (C# Azure SDK):

```
var metadata = new Dictionary<string, string> { { "author", "john_doe" }, { "category", "images" } };
await blobClient.SetMetadataAsync(metadata);
```

---

5. How do you retrieve metadata from a blob?

- Call `BlobClient.GetPropertiesAsync()` and access the Metadata property.
- Example (C# Azure SDK):

```
BlobProperties properties = await blobClient.GetPropertiesAsync();
foreach (var item in properties.Metadata)
{
   Console.WriteLine($"{item.Key}: {item.Value}");
}
```

6. What are best practices when using metadata in Azure Blob Storage?

- Keep metadata size small (max 8 KB total per blob).
- Use lowercase keys; metadata keys are case-insensitive.
- Metadata is stored separately; retrieving it requires an extra API call (costs apply).

7. What happens if you overwrite a blob — are properties and metadata preserved?

- No, uploading a blob without explicitly setting metadata and properties will reset them to defaults.
- Always reapply desired metadata and properties during overwrite if needed.

8. How do you set or update metadata without overwriting the blob content?

- Use `BlobClient.SetMetadataAsync()` — it updates metadata without affecting the blob's content.
- No need to re-upload the blob when updating only metadata.

9. How do you use Azure SDK (C#, Python) to manage properties and metadata?

- C# Example:

```
await blobClient.SetMetadataAsync(new Dictionary<string, string> { { "env", "prod" } });
BlobProperties props = await blobClient.GetPropertiesAsync();
Console.WriteLine(props.ContentType);
```

10. How can you search or filter blobs by metadata?

- Use Azure Blob Index Tags, not regular metadata.
- With tags, you can query blobs via FindBlobsByTags.
- Example (Azure CLI):

```
az storage blob query-tags --container-name mycontainer --where "tagName = 'value'"
```

Develop for Azure Storage
└─ **2.2 Develop solutions that use Azure Cosmos DB**
└─ **2.2.2 Perform operations by using the appropriate SDK**

1. What SDKs are supported for Azure Blob Storage operations?
2. How do you create and upload a blob using the Azure SDK (C#, Python)?
3. How do you download a blob using the Azure SDK?
4. How do you list blobs inside a container?
5. How do you delete a blob using the Azure SDK?
6. How do you perform conditional operations (e.g., upload if not exists)?
7. How do you use a stream to upload/download blobs?
8. How do you handle large blobs efficiently (e.g., upload in blocks)?
9. How do you set retries and timeouts in SDK operations?
10. What are best practices for SDK usage in production?

---

1. What SDKs are supported for Azure Blob Storage operations?

- Official Azure SDKs:
  - .NET (Azure.Storage.Blobs)
  - Python (azure-storage-blob)
  - Java (azure-storage-blob)
  - JavaScript/TypeScript (e.g., @azure/storage-blob)

---

2. How do you create and upload a blob using the Azure SDK (C#, Python)?

- C# Example:

```
BlobClient blobClient = containerClient.GetBlobClient("myblob.txt");
await blobClient.UploadAsync("localfile.txt", overwrite: true);
```

---

3. How do you download a blob using the Azure SDK?

```
BlobDownloadInfo download = await blobClient.DownloadAsync();
using (FileStream fs = File.OpenWrite("downloaded.txt"))
{
   await download.Content.CopyToAsync(fs);
}
```

---

4. How do you list blobs inside a container?

```
await foreach (BlobItem blobItem in containerClient.GetBlobsAsync())
{
   Console.WriteLine(blobItem.Name);
}
```

---

5. How do you delete a blob using the Azure SDK?

```
await blobClient.DeleteIfExistsAsync();
```

---

6. How do you perform conditional operations (e.g., upload if not exists)?

- Set the conditions parameter with an If-None-Match: * condition.
- C# Example:

```
var conditions = new BlobRequestConditions { IfNoneMatch = new ETag("*") };
await blobClient.UploadAsync("localfile.txt", conditions: conditions);
```

7. How do you use a stream to upload/download blobs?

- Upload Stream (C#):

```
using var stream = File.OpenRead("localfile.txt");
await blobClient.UploadAsync(stream, overwrite: true);
```

- Download Stream (C#):

```
BlobDownloadInfo download = await blobClient.DownloadAsync();
using var file = File.OpenWrite("output.txt");
await download.Content.CopyToAsync(file);
```

---

8. How do you handle large blobs efficiently (e.g., upload in blocks)?

- Use `UploadAsync` for files up to 256 MB (default).
- For larger files, use `UploadAsync` with automatic chunking or manually use `BlockBlobClient.StageBlockAsync` and `CommitBlockListAsync`.

---

9. How do you set retries and timeouts in SDK operations?

- C# Example:

```
BlobClientOptions options = new BlobClientOptions
{
   Retry =
   {
      MaxRetries = 5,
      Delay = TimeSpan.FromSeconds(2),
      MaxDelay = TimeSpan.FromSeconds(10),
      Mode = RetryMode.Exponential
   }
};
var blobServiceClient = new BlobServiceClient(connectionString, options);
```

---

10. What are best practices for SDK usage in production?

- Always set appropriate retry policies and timeouts.
- Prefer streams for large files.
- Handle exceptions explicitly (e.g., RequestFailedException in C#).
- Reuse BlobServiceClient, BlobContainerClient, and BlobClient instances (they are thread-safe).
- Secure secrets and connection strings (use Azure Managed Identity if possible).

Develop for Azure Storage
└─ **2.2 Develop solutions that use Azure Cosmos DB**
└─ **2.2.3 Implement storage policies and data lifecycle management**

1. What is Azure Blob Lifecycle Management?
2. How do you define a lifecycle management rule?
3. How do you move blobs between access tiers (Hot, Cool, Archive)?
4. How do you configure auto-delete for old blobs?
5. How do you apply rules based on blob metadata or naming patterns?
6. How do you create lifecycle management rules using the Azure Portal?
7. How do you create lifecycle policies programmatically (Azure CLI, SDK)?
8. What are best practices for setting retention and tiering policies?
9. How does Archive rehydration work and what are the implications?
10. How do you monitor and troubleshoot lifecycle policy actions?

---

1. What is Azure Blob Lifecycle Management?

- A feature that automatically moves, deletes, or archives blobs based on rules and conditions like age, last modified date, or access tier.
- Reduces storage costs and enforces data retention policies.

---

2. How do you define a lifecycle management rule?

- A rule consists of:
  - Filters (prefix match, blob type, metadata conditions).
  - Actions (move to Cool/Archive tier, delete blob).
- Rules are evaluated daily by Azure.

---

3. How do you move blobs between access tiers (Hot, Cool, Archive)?

- Define a lifecycle rule that moves blobs based on conditions:
  - Example: Move to Cool if not modified for 30 days.
  - Example: Move to Archive if not modified for 180 days.
- No manual intervention needed once the policy is active.

---

4. How do you configure auto-delete for old blobs?

- Set a `Delete` action in a lifecycle management rule.
- Example: Delete blobs 90 days after the last modified date.
- Can combine delete action with filters (e.g., only for blobs with a specific prefix).

---

5. How do you apply rules based on blob metadata or naming patterns?

- Use filters when defining the rule:
  - Prefix match: Target blobs under a virtual folder path.
  - Blob index tags: Target blobs with specific metadata conditions (e.g., env=prod).
- Example filter:

```
"prefixMatch": ["logs/"],
"blobTypes": ["blockBlob"]
```

6. How do you create lifecycle management rules using the Azure Portal?

- Navigate to Storage Account → Data Management → Lifecycle Management.
- Add a rule:
  - Define conditions (e.g., last modified > 30 days).
  - Specify actions (move, delete).
- Save and enable the rule — applies automatically to matching blobs.

---

7. How do you create lifecycle policies programmatically (Azure CLI, SDK)?

- Azure CLI Example:

```
az storage account management-policy create \
   --account-name <storageaccount> \
   --resource-group <resourcegroup> \
   --policy @"policy.json"
```

- C# SDK Example:

```
var managementPolicy = new ManagementPolicy
{
   Policy = JsonConvert.DeserializeObject<ManagementPolicySchema>(policyJson)
};
await storageAccount.UpdateAsync(managementPolicy: managementPolicy);
```

- Policy is defined in a JSON file describing rules and actions.

---

8. What are best practices for setting retention and tiering policies?

- Use Cool tier for infrequently accessed data (accessed > 30 days).
- Use Archive tier for rarely accessed data (accessed > 180 days).
- Avoid aggressive delete rules unless compliance requirements mandate it.
- Test lifecycle rules in non-production first to verify behavior.

---

9. How does Archive rehydration work and what are the implications?

- Rehydration moves blobs from Archive to Hot or Cool tier to make them accessible.
- It can take hours (up to 15 hours standard, faster rehydration possible).
- Rehydration is asynchronous; status must be polled using blob properties.

---

10. How do you monitor and troubleshoot lifecycle policy actions?

- Review Activity Log in Azure Portal for lifecycle policy operations.
- Check Blob properties to verify last tier change date.
- Use Azure Monitor metrics to track storage capacity changes by tier.