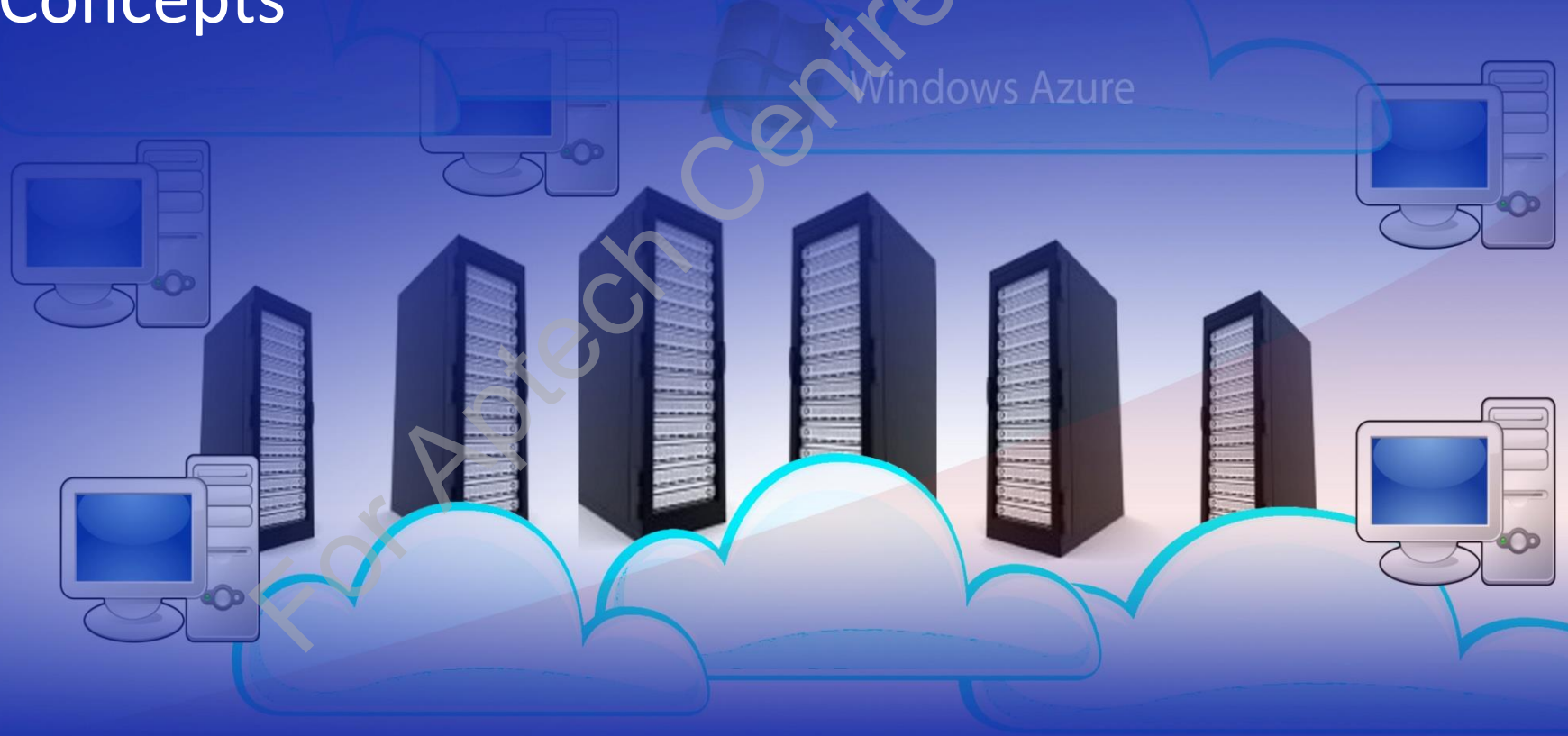
Enterprise Application Development Using Windows Azure and Web Services

Session 8

Advanced Windows Azure Storage Concepts



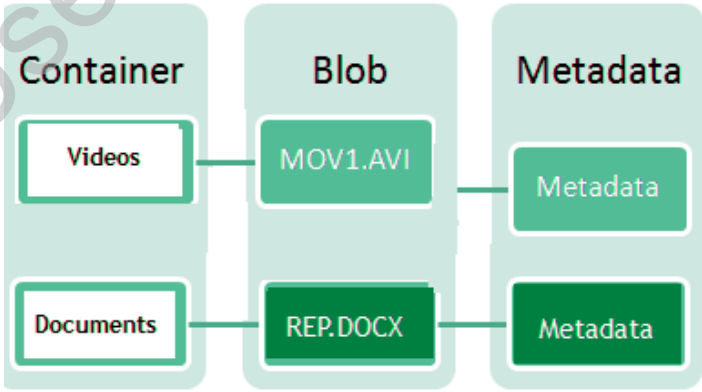
Learning Objectives

- Describe Blob storage
- Describe Table storage
- Describe Queue storage

Blob Storage 1-5

Blobs:

- Can store a large amount of data.
- Data can include images, videos, documents, and codes.
- Can store data in both structured and unstructured formats.
- Can store data in binary format.
- Are stored in containers.

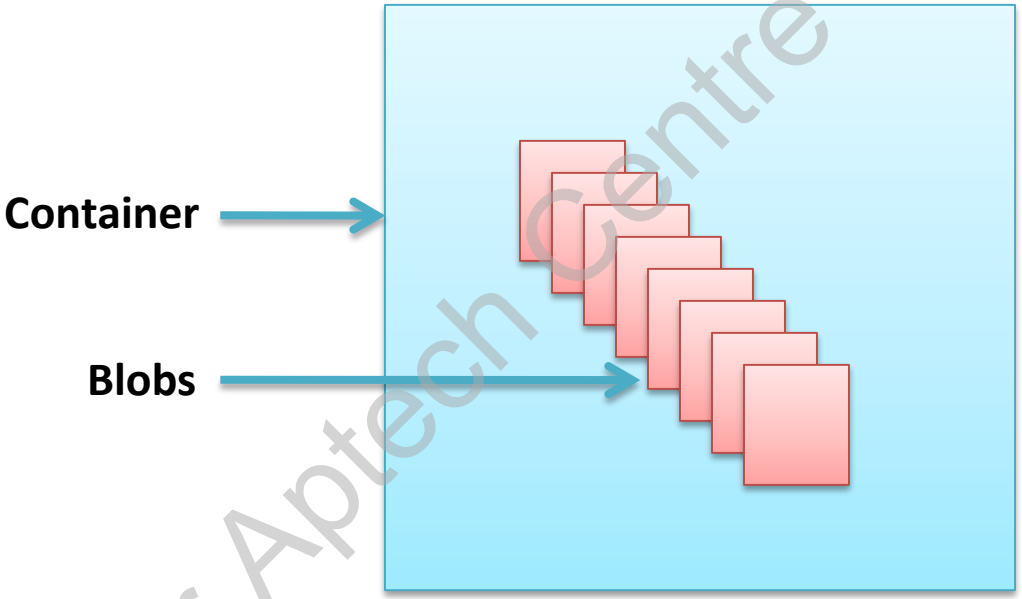


Containers:

- Can hold a number of blobs.
- Are similar to the directory structure on your hard drive.
- Can hold any number of directories with data as long as you do not run out of space.
- Can be created in your Windows Azure account.
- Used to contain blobs.

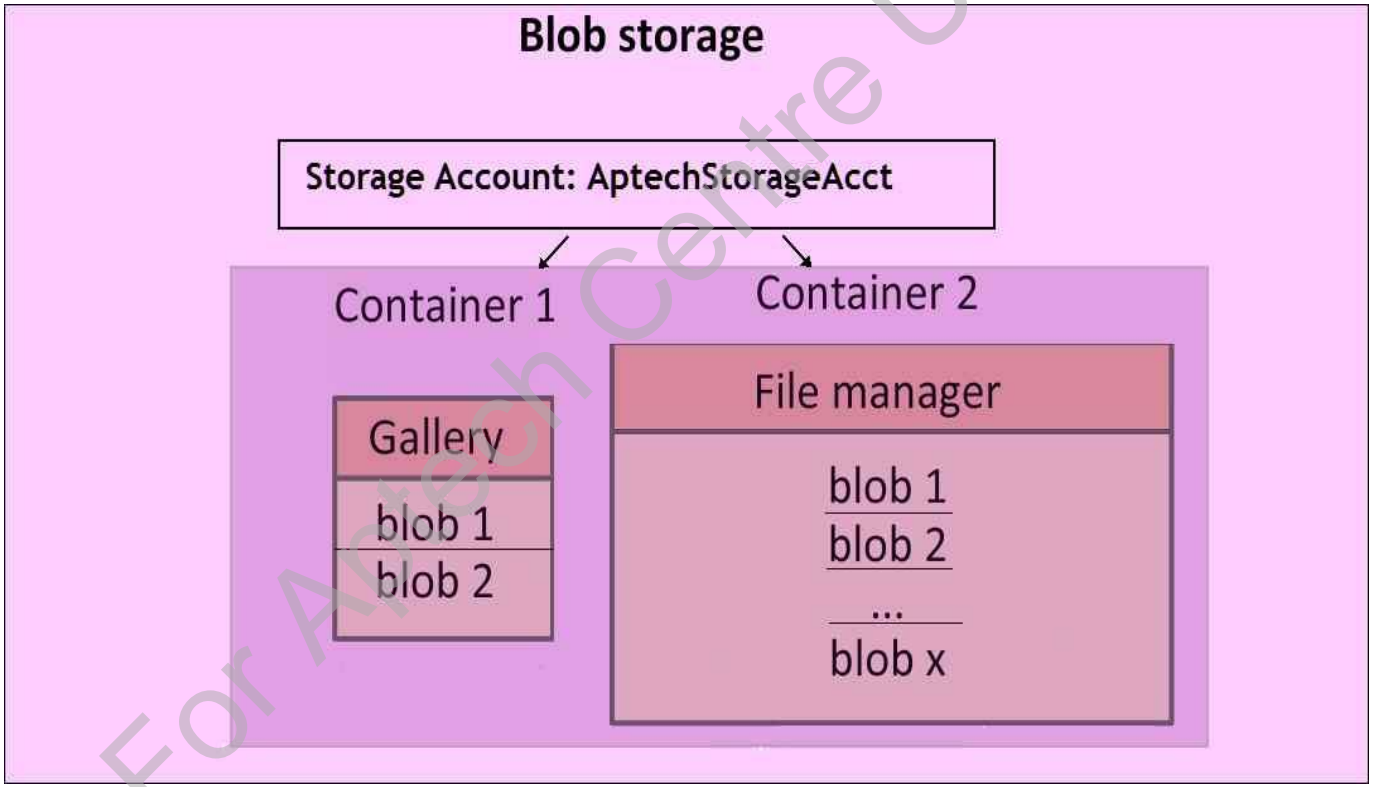
Blob Storage 2-5

- If you consider a container similar to a folder, then the blobs are similar to files within the folder.



Blob Storage 3-5

- Following figure displays the concept of containers and blobs:



Blob Storage 4-5

- You can essentially create two types of blobs:

Block blobs	Page blobs
• Mainly used for streaming data. • Each block blob can grow up to maximum of 200 GB.	• Mainly used for read/write access and can grow up to maximum of 1 TB in size.

- Each blob that you create:
  - Can have up to 8 KB of metadata.
  - Is replicated at least three times to ensure scalability and fault tolerance, after it is created.

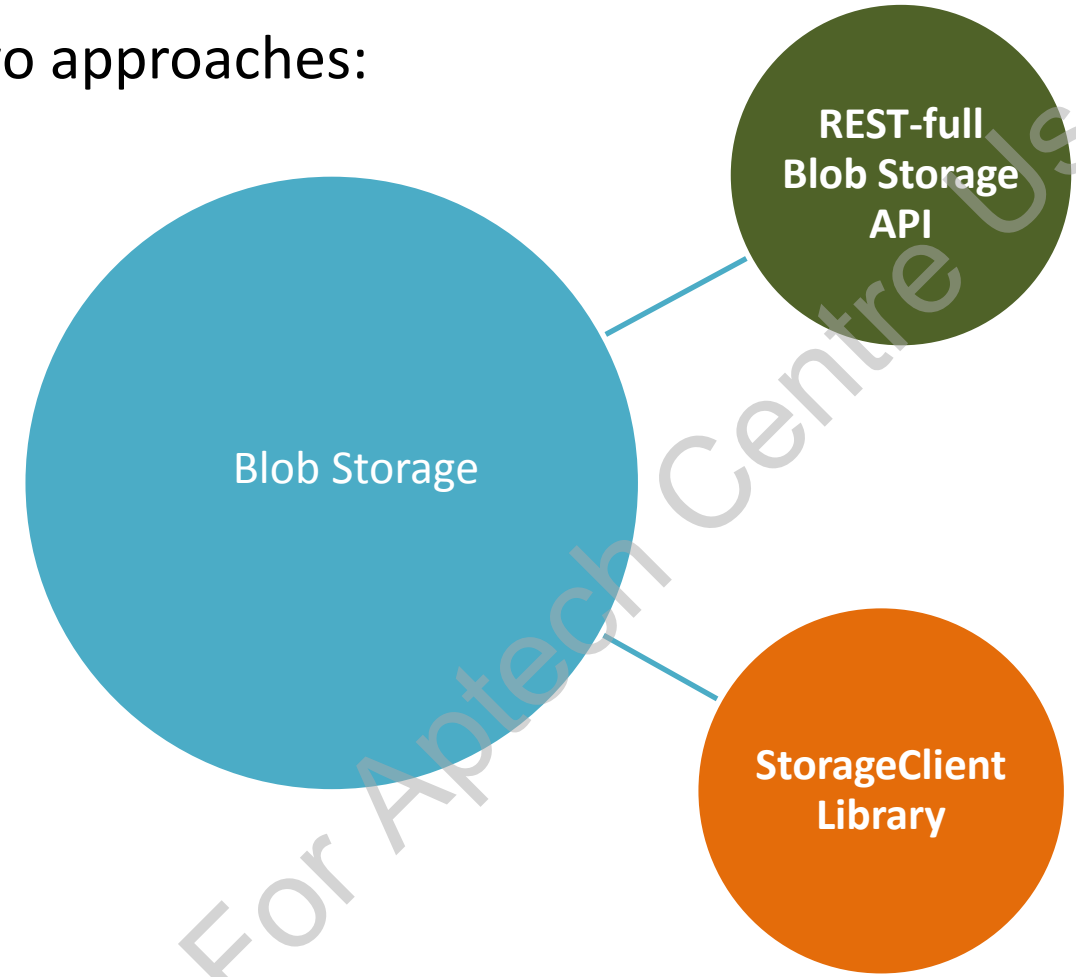
Blob Storage 5-5

- Four key elements in the Blob storage data model are:

Storage account	• Is required to log on to Windows Azure Storage, which can contain a number of containers.
Container	• Contains unlimited number of blobs.
Blobs	• Can be of 200 GB and can be upto 1 TB.
Blocks	• Are part of a blob. • Each block can be of 4 MB in size.

Accessing Blob Storage 1-2

Two approaches:



Accessing Blob Storage 2-2

REST-full Blob Storage API

Allows you to work with container and blob resources using the HTTP operations.

You can perform a number of operations using REST-full Blob Storage API.

Some of these operations are listing containers, create containers, and get container properties.

StorageClient Library

Provides a library for calling queries against the Windows Azure Tables.

It also provides a LINQ interface.

Table Storage 1-6

- Table storage in Windows Azure is used for storing structured data.
- Tables in table storage:
  - Does not follow relational mechanism, but they rather follow the object-based storage approach.
  - Can grow exceptionally large in size, such as in terabytes.
  - Can store billions of entities.
  - Will typically be spread over multiple servers that are part of the load balancing mechanism.
  - Also have names as the database tables.
  - Have entities and they are equal to the rows of database storage.
  - Can be split across multiple partitions or can even be moved from one partition to another partition.

Table Storage 2-6

- The data storage in Table storage is persistent.
- If servers shut down all of a sudden, the data will be retrieved from the exact point when the servers where shut down.
- The `PartitionKey` is used to keep track of all entities that are part of the table and glue them together.
- In the case of a table in Table storage, the `PartitionKey` and `RowKey` together form a key, which is known as primary key in relational databases.

Table Storage 3-6

- One of the biggest differences that tables in Table storage have as against database tables is that there is no fixed schema.
- Each entity can have a different set of properties.
- In database tables, each row is defined with fixed set of columns that define the same set of information.

Table Storage 4-6

- Following table displays different properties for each entity of a table in table storage:

Contacts table		
PartitionKey	**RowKey**	**Properties**
JohnBee	NameRow	FirstName,John:LastName,Bee
JohnBee	CareerRow	Company,Comp|Compdocs
JohnBee	PersonalRow	Hobby,Reading
ElizabethSmith	NameRow	FirstName, Elizabeth:LastName,Smith
ElizabethSmith	CareerRow	Company,SoftEducation
ElizabethSmith	PersonalRow	Hobby,PlayingFootball
DenWills	NameRow	FirstName,Den:LastName,Wills
DenWills	CareerRow	Company,TecPark|Systemplus
DenWills	PersonalRow	Hobby,branding

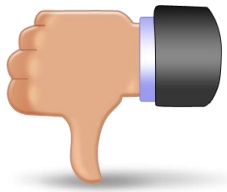
Table Storage 5-6

- Following are advantages of tables in Table Storage:

Size	• Each table can grow without limitations on the size.
Spread	• Each table can be spread across multiple partitions on multiple servers.
Persistent	• Each table is persistent, which means that it retains the last known status of the table when the servers are shut down or turned off.
Access	• Each table can be accessed via REST API or queried using client library for ADO.NET Data Services.

Table Storage 6-6

- Following are limitations of tables in Table Storage:



Size	• Each entity can be of maximum of 1 MB in size.
PartitionKey and RowKey	• Can be of only Data type.
Entity Properties	• Each entity can have maximum of 255 properties. • There can be 252 user-defined properties and remaining three properties are fixed, which are PartitionKey, RowKey, and Timestamp.
Partial Modification	• An entity's individual properties cannot be modified. • You must rewrite the entire entity.

Queue Storage 1-8

- **Queues in Windows Azure:**

  Are mainly used for notifications and task scheduling.

- **The basic intent of Queue storage is:**

  To allow frontend servers to communicate with the backend servers, which are typically the database servers.

- **Queue storage is:**

  Automatically created when you create a storage service in your Windows Azure Storage account.

Queue Storage 2-8

- You can view the default Queue storage with the following URL:

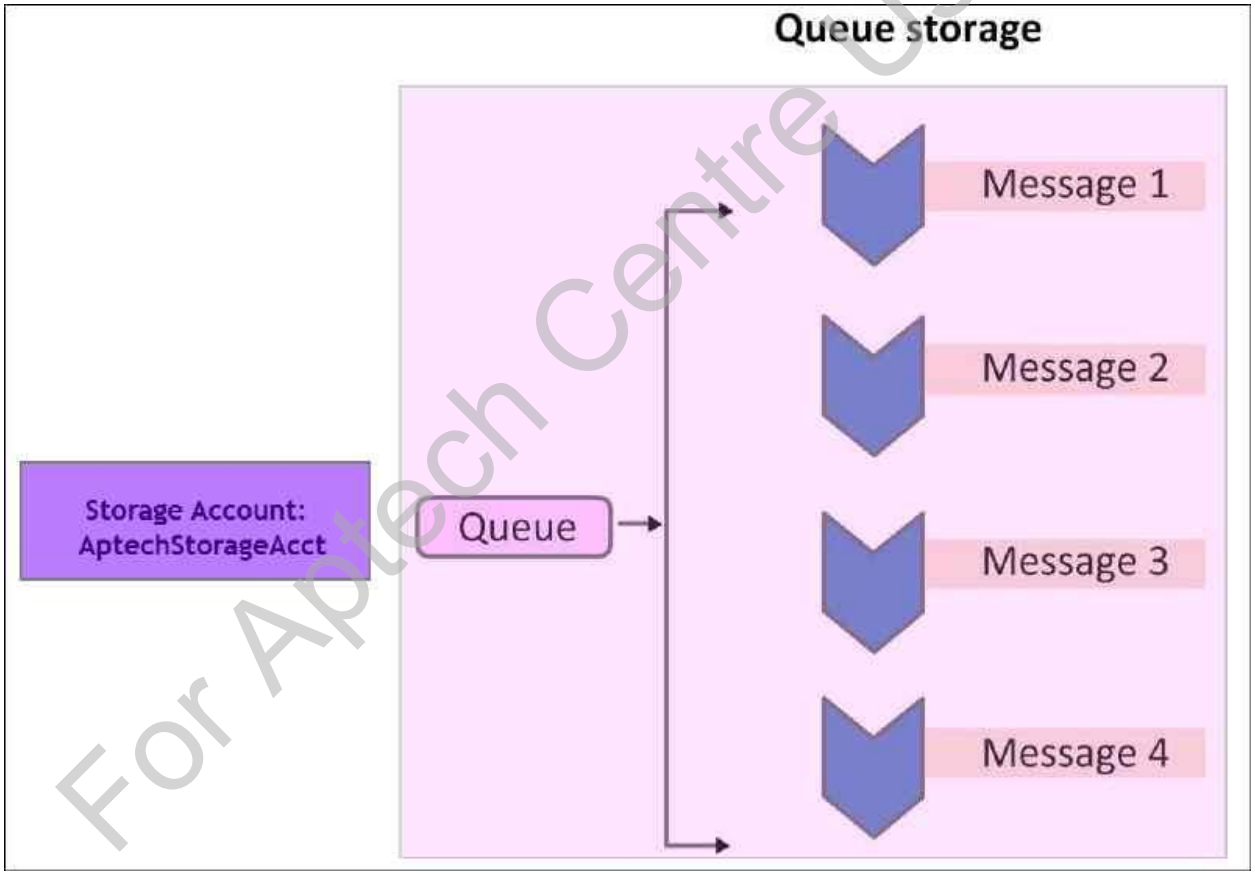
  http://<accountname>/queue.core.windows.net/

- You can create a large number of queues in your Windows Azure account.

Queue Storage is mainly used for delivering messages from the frontend servers to the backend servers, which upon receiving the message, will process them.

Queue Storage 3-8

- Following figure displays Queue Storage message processing:



Queue Storage 4-8

- A queue can have any number of messages.
- You can control the number of messages that you want to have in the queue at any given point in time.
- The messages may be processed within the defined timeframe or may be delayed.
- There can be a number of reasons that can cause the delay.

Queue Storage 5-8

- Two key reasons of delaying the process of the messages:

Backend Processing

- Servers in the backend are not able to process the messages being received from the frontend servers.
- Queue is not able to release messages or releases with high latency.

Large Data

- Each message is limited to 8 KB in size.
- If the message is larger than this, then it is not going to go through the queue.

Solution

- Depending on the issue, either add more backend servers to speed up the processing or customize and optimize the existing processes in the backend serves.
- Your solution will largely depend on your existing infrastructure and requirements.

- Data should be stored in a blob or table.

Queue Storage 6-8

- Some of the reasons why queues are used, are as follows:

Persistent data

- Queue storage stores the data in a persistent manner. If an application crashes, you can recover the in-process data.
- It also offers recovery mechanism to recover the processes that fail to process in the queue.

Priority

- You can have multiple queues working at the same time. You can set high priority for a queue by redirecting more processes them.
- You can create and attach services for queues with higher importance.

Language

- You can use processes in different language in a single application.

Geography

- Processes can run from different geographical locations.

Queue Storage 7-8

Scalability

- Queue storage removes the dependency from the backend servers to process information.
- Consider the following scenario of an application:
  - In the application, you execute an action on the frontend server, which sends a message to the backend server.
  - The message is processed in the backend server.
  - The second message is processed when the processing of the first message is complete.
  - In the scenario where Queue storage is used, messages are processed differently.
  - Processes in Queue storage can work independently of each other.
  - You can have five frontend servers and a single backend server to handle the queue load.

Queue Storage 8-8

- Following table summarizes the comparison of blob, table, and message queue:

Storage Type	Purpose of Use	Maximum Capacity
Blob	Large binary objects	200 GB/1 TB
Table	Structured data	100 TB
Queue	Structured data	100 TB

Using Blobs, Tables, and Message Queues 1-2

- To use blob, tables, or message queues:

Create a storage account.

Set up a connection string to your storage account through Visual Studio or a .NET configuration file.

Put your storage connection string in a configuration file, than hard-coding it in your program.

You store your connection string using the Azure service configuration system (`*.csdef` and `*.cscfg` files), if Azure Cloud Services is being used.

Using Blobs, Tables, and Message Queues 2-2

- When using .NET, you can use the `web.config` or `app.config` files.
- The `<appSettings>` element needs to be configured to create the connection string.
- You can use the following syntax:

Syntax

```
<configuration>
<appSettings>
<add key="StorageConnectionString"
value="DefaultEndpointsProtocol=https;AccountName=<accou
ntname>;AccountKey=<your-account-key>" />
</appSettings>
</configuration>
```

Working with Blobs 1-6

- To use blob storage programmatically, you need to:

1. Import `Microsoft.WindowsAzure.Storage.dll`, which is provided in Azure SDK for .NET.

2. Declare the namespaces using the following code:

```
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Auth;
using Microsoft.WindowsAzure.Storage.Blob;
```

3. After namespace declaration, you need to retrieve your connection string from the Azure service configuration using the following code:

```
CloudStorageAccount
storageAccount=CloudStorageAccount.Parse(
CloudConfigurationManager.GetSetting("StorageConnection
String"));
```

Working with Blobs 2-6

4

If a container does not exist, you need to create the container by first creating the blob client and then creating the container by using the following code:

```
// Create the blob client
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Retrieve a reference to a container if it exists
CloudBlobContainer container = blobClient.GetContainerReference
                               ("photos");

// Create the container if it does not exist.
container.CreateIfNotExists();
```

- In the code:
  - An instance of `CloudBlobClient` is created.
  - The `CloudBlobClient` class defined in the `Microsoft.WindowsAzure.Storage.Blob` namespace which enables you to retrieve objects representing containers and blobs stored within the Blob Storage Service.

Working with Blobs 3-6

Reference a Blob

- You can reference a blob that exists or create a new blob using the following code:

```
CloudBlockBlob blockBlob=container.GetBlockBlobReference("photoblob");
```

Create or Overwrite a Blob

- You can create or overwrite a blob using contents present in a local file with the following code:

```
using(var fileStream=System.IO.File.OpenRead(@"path\photofiles"))
{
      blockBlob.UploadFromStream(fileStream);
}
```

Working with Blobs 4-6

List the existing blobs in a container

- You can also list the existing blobs in a container shown in the following code.
- The `CloudBlobContainer` class represents a single container and is defined in the `Microsoft.WindowsAzure.StorageClient` namespace.

```
CloudBlobContainer container =.GetContainerReference("photos");
foreach(IListBlobItem item in container.ListBlobs(null,false))
{
   if(item.GetType()==typeof(CloudBlockBlob))
   {
    CloudBlockBlob blob =(CloudBlockBlob)item;

    Console.WriteLine("Block blob of length {0}: 1}",
    blob.Properties.Length,blob.Uri);
   }
   elseif(item.GetType()==typeof(CloudPageBlob))
   {
```

Working with Blobs 5-6

```
        CloudPageBlob pageBlob = (CloudPageBlob)item;
        Console.WriteLine("Page blob of length {0}: {1}",
        pageBlob.Properties.Length,pageBlob.Uri);
    }
    elseif(item.GetType()==typeof(CloudBlobDirectory))
    {
        CloudBlobDirectory directory =(CloudBlobDirectory)item;
        Console.WriteLine("Directory: {0}",directory.Uri);
    }
}
```

Working with Blobs 6-6

Save the contents of a blob into a file

- Reference the blob and then save the contents into a file as shown in the following code:

```
CloudBlockBlob blockBlob=container.GetBlockBlobReference("pic.jpg");
using(var fileStream = System.IO.File.OpenWrite(@"path\photofiles"))
{
      blockBlob.DownloadToStream(fileStream);
      . . .
}
```

Delete a blob

- Delete a blob by referencing it and then sending the delete command by using the following code:

```
CloudBlockBlob blockBlob=container.GetBlockBlobReference("blockbob.txt");
blockBlob.Delete();
```

Working with Tables 1-5

- **Create a table:** Similar to blobs, refer following steps to create a table:

Step 1	• Create a storage account
Step 2	• Set up the connection string by using the same code
Step 3	• Create a table client
Step 4	• Create the table by using the following code:

```
CloudTableClient tableClient =
storageAccount.CreateCloudTableClient();
CloudTable table =tableClient.GetTableReference("employees");
table.CreateIfNotExists();
```

Working with Tables 2-5

- **Create an entity in the table:** After creating a table, create an entity in the table with the help of a custom class named `TableEntity` and using the following code:

```
public class CustomerEntity:TableEntity
{
    public CustomerEntity(string lastName,stringfirstName)
    {
      this.PartitionKey=lastName;
      this.RowKey=firstName;
    }

    public CustomerEntity(){}

    public string Email{get;set;}

    public string PhoneNumber{get;set;}
}
```

Working with Tables 3-5

- **Insert the customer entity into the entity:** You can insert the customer entity into the entity with the help of following code:

```
CustomerEntity customer1 =newCustomerEntity("McClane","John");
customer1.Email="John@McClane.com";
customer1.PhoneNumber="123-111-1234";

// Create the TableOperation that inserts the customer entity.
TableOperation insertOperation=TableOperation.Insert(customer1);

// Execute the insert operation.
table.Execute(insertOperation);
```

Working with Tables 4-5

- **Delete an entity:** After creating an entity, you can later delete it when not required by using the following code:

```
CloudTable table =tableClient.GetTableReference("employees");
TableOperationretrieveOperation=TableOperation.Retrieve<Customer
Entity>
("McClane","John");
TableResultretrievedResult=table.Execute(retrieveOperation);
CustomerEntitydeleteEntity=(CustomerEntity)retrievedResult.Result;
if(deleteEntity!=null)
{
   TableOperationdeleteOperation=TableOperation.Delete(deleteEntity);
   table.Execute(deleteOperation);
   Console.WriteLine("Entity deleted.");
}
else
Console.WriteLine("Could not retrieve the entity.");
```

Working with Tables 5-5

- **Delete a table:** You can delete a table by using the following code:

```
CloudTable table =tableClient.GetTableReference("employees");
table.DeleteIfExists();
```

Working with Queue Storage 1-4

- **Create a message queue:** Similar to blobs and tables, refer the following steps to create a message queue:

Step 1
- Create a storage account.

Step 2
- Set up the connection string.

Step 3
- Create a message queue by creating an instance of `CloudQueueClient` as shown in the following code:

```
CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
```

Working with Queue Storage 2-4

- **Create the queue:** After you create the `queueClient` object, you need to:
  - Reference it.
  - Create the queue if it does not exist, by using the following code:

```
CloudQueue queue
=queueClient.GetQueueReference("ourmessagequeue");
// Create the queue if it doesnot already exist
queue.CreateIfNotExists();
```

- **Insert a message into the message queue:** After the message queue is created, you need to insert a message into it, as shown in the following code:

```
CloudQueueMessage message =new CloudQueueMessage("Hello and
Welcome");
queue.AddMessage(message);
```

Working with Queue Storage 3-4

- **Display the message:** After you insert the message, you can peek and display the message using the following code:

```
CloudQueueMessage peekedMessage=queue.PeekMessage();
Console.WriteLine(peekedMessage.AsString);
```

- **Get the message queue length:** If you have queued up a number of messages, you can get the message queue length with the help of code shown as follows:

```
CloudQueue queue
=queueClient.GetQueueReference("ourmessagequeue");
queue.FetchAttributes();
int? cachedMessageCount=queue.ApproximateMessageCount;
Console.WriteLine("Number of messages in queue:
"+cachedMessageCount);
```

Working with Queue Storage 4-4

- **Delete a message queue:** You can delete a message queue by using the following code:

```
CloudQueue queue =queueClient.GetQueueReference("myqueue");
queue.Delete();
```

Summary 1-2

- Blobs can store a large amount of data that can be in variety of forms, such as images, videos, documents, and codes.
- A container can hold a number of blobs.
- You can create two types of blobs: Block blobs and Page blobs.
- Each blob can have 8 KB of metadata.
- The Table storage in Windows Azure is used for storing structured data.

Summary 2-2

- The data storage in Table storage is persistent.
- A table in Table storage can be split across multiple partitions or can even be moved from one partition to another partition.
- Queues in Windows Azure are mainly used for notifications and task scheduling.
- You can create a large number of queues in your Windows Azure account.