

Corso di Sistemi Distribuiti

Corso di Laurea Magistrale in Ingegneria Informatica A.A. 2014/2015
DIMES - Università degli Studi della Calabria



DEVELOPMENT SERVICES ON WINDOWS AZURE

STORE DATA IN TABLES

STORE DATA IN BLOBS

Summary

- Store data in tables
- Store data in blobs

Summary

- Store data in tables
- Store data in blobs

The Table Service

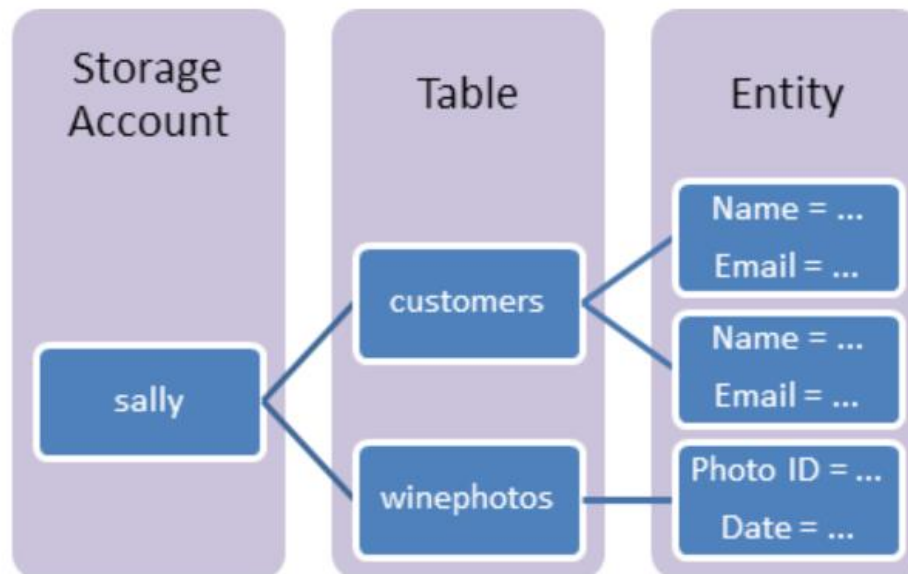
- The Azure Table storage service stores large amounts of structured data.
 - tables are ideal for storing structured, non-relational data
 - the service is a NoSQL datastore
 - accepts authenticated calls from inside and outside the Azure cloud

- Table service can be exploited to:
 - store TBs of structured data capable of serving web scale applications
 - store datasets that don't require complex joins, foreign keys, or stored procedures and can be denormalized for fast access
 - quickly query data using a clustered index
 - access data using the OData protocol and LINQ queries with WCF Data Service .NET Libraries

- Following the Cloud philosophy, tables will scale as demand increases

Concepts

- **Storage Account:** it needs to access to any Azure Storage service.
- **Table:** a collection of entities, no table schema enforced (i.e., a single table can contain entities that have different sets of properties).
- **Entity:** a set of properties, like a database row (size up to 1MB).
- **Property:** a name-value pair. Each entity can include up to 252 properties to store data. Each entity also has 3 system properties that specify a partition key, a row key, and a timestamp. Entities with the same partition key can be queried more quickly, and inserted/updated in atomic operations. An entity's row key is its unique identifier within a partition



Preliminary steps

- Obtain the assembly (by NuGet)
- Create and configure a `ConnectionString` (i.e, '`StorageConnectionString`')

Entity class

- Entities are custom classes derived from TableEntity
 - The entity class defines the properties of the entity
 - Each entity class (or type) must expose a parameter-less constructor.
 - About properties:
 - Row key
 - Partition key
 - Entities with the same partition key can be queried faster
 - Nevertheless, using diverse partition keys allows for greater parallel operation scalability.
 - Other properties..., but each one must be public and of a supported type
 - Each property must expose both get and set

Entity class – Customer & Employee

```
//class CustomerEntity
public class CustomerEntity : TableEntity {
    public CustomerEntity(string lastName, string firstName) {
        this.PartitionKey = lastName;
        this.RowKey = firstName;
    }
    public CustomerEntity() { }
    public string Email { get; set; }
    public string PhoneNumber { get; set; }
} //class CustomerEntity
```

```
//class EmployeeEntity
public class EmployeeEntity : TableEntity {
    public EmployeeEntity() { }
    public EmployeeEntity(int id, string name, double sal) {
        Id = id;
        Name = name;
        Salaray = sal;
        PartitionKey = id.ToString();
        RowKey = name;
    }
    public int Id { get; set; }
    public string Name { get; set; }
    public double Salary { get; set; }
} //class EmployeeEntity
```

← Auto-implemented property

Code snippets – insert an entity

```
protected void btn_Click(object sender, EventArgs e){

    // Retrieve the storage account from the connection string.
    CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
        CloudConfigurationManager.GetSetting("StorageConnectionString"));

    // Create the table client.
    CloudTableClient tableClient = storageAccount.CreateCloudTableClient();

    // Create the CloudTable object that represents the "people" table.
    CloudTable table = tableClient.GetTableReference("people");

    // Create the table if it doesn't exist.
    table.CreateIfNotExists();

    // Create a new customer entity.
    CustomerEntity customer = new CustomerEntity("Harp", "Walter");
    customer.Email = "Walter@contoso.com";
    customer.PhoneNumber = "425-555-0101";

    ...
}
```

Code snippets – insert an entity

```
protected void btn_Click(object sender, EventArgs e){  
  
...  
  
    // Create the TableOperation that inserts the customer entity.  
    TableOperation insertCustomerOperation = TableOperation.Insert(customer);  
  
    // Execute insert operations.  
    table.Execute(insertCustomerOperation);  
  
    EmployeeEntity employee = new EmployeeEntity(1, "BillG", 123456.33);  
    TableOperation insertEmployeeOperation = TableOperation.Insert(employee);  
  
    // Execute insert operations.  
    table.Execute(insertEmployeeOperation);  
  
} //btn_Click
```

View Content

■ How to view table entries?

- Azure Management Studio
- Azure Storage Explorer

The screenshot shows the Azure Management Studio interface. The left sidebar displays a tree view of the 'My Connection Group' with 'Storage Accounts' expanded. Under 'Storage Accounts', 'devstoreaccount1' is circled in red. Below it, 'Tables (2)' is expanded, showing 'people' and 'WADWindowsEventLogsTable'. The 'people' table is selected, and its data is displayed in the main pane. The data is presented in a table with columns: PartitionKey, RowKey, Timestamp, Id, Name, Salary, Email, and PhoneNumber. The first row shows a single entry with PartitionKey '1' and RowKey 'BillG'. The subsequent rows show a list of 'Harp' entries with RowKeys 'Harp1' through 'Harp5'. The 'Email' column contains the value 'Walter@contoso.com' for all rows, and the 'PhoneNumber' column contains the value '425-555-0101' for all rows. A red arrow points from the text 'Local Storage' to the 'people' table in the left sidebar.

PartitionKey	RowKey	Timestamp	Id	Name	Salary	Email	PhoneNumber
1	BillG	2014-05-22T10:13:05.7870000Z	1	BillG	123456,33		
Harp1	Walter1	2014-05-19T16:21:10.8470000Z				Walter@contoso.com	425-555-0101
Harp2	Walter2	2014-05-19T16:34:20.5170000Z				Walter@contoso.com	425-555-0101
Harp3	Walter3	2014-05-19T16:41:26.3830000Z				Walter@contoso.com	425-555-0101
Harp4	Walter4	2014-05-19T16:44:41.1470000Z				Walter@contoso.com	425-555-0101
Harp5	Walter5	2014-05-22T10:13:05.6800000Z				Walter@contoso.com	425-555-0101

View Content

The screenshot shows the Azure Management Studio interface. On the left, the 'My Connection Group' tree is expanded, showing 'Storage Accounts' > 'devstoreaccount1' > 'Tables (2)' > 'people'. The 'cesariostorage' connection group is highlighted with a red circle, and an arrow points to it from the text 'Remote Storage'. The main pane displays the 'people' table with the following data:

PartitionKey	RowKey	Timestamp	Id	Name	Salary	Email	PhoneNumber
1	BillG	2014-05-28T10:08:30.7681192Z	1	BillG	123456,33		
Harp1	Walter1	2014-05-19T16:24:30.5168839Z				Walter@contoso.com	425-555-0101
Harp5	Walter5	2014-05-28T10:08:30.6181192Z				Walter@contoso.com	425-555-0101

- Demo in classroom

- Local Storage
 - Fix connection string
 - Insert tuples
 - Check contents

- Remote Storage
 - Login the Azure Management Portal
 - Fix connection string
 - Insert tuples
 - Check contents

Code snippets - Insert a batch of entities

```
// Retrieve the storage account from the connection string.
// Create the table client.
...
// Create the CloudTable object that represents the "people" table.
CloudTable table = tableClient.GetTableReference("people");

// Create the batch operation.
TableBatchOperation batchOperation = new TableBatchOperation();

// Create a customer entity and add it to the table.
CustomerEntity customer1 = new CustomerEntity("Smith", "Jeff");
customer1.Email = "Jeff@contoso.com";
customer1.PhoneNumber = "425-555-0104";

// Create another customer entity and add it to the table.
CustomerEntity customer2 = new CustomerEntity("Smith", "Ben");
customer2.Email = "Ben@contoso.com";
customer2.PhoneNumber = "425-555-0102";

// Add both customer entities to the batch insert operation.
batchOperation.Insert(customer1);
batchOperation.Insert(customer2);

// Execute the batch operation.
table.ExecuteBatch(batchOperation);
```

Code snippets - Retrieve all entities in a partition

```
// Retrieve the storage account from the connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the table client.
CloudTableClient tableClient = storageAccount.CreateCloudTableClient();

// Create the CloudTable object that represents the "people" table.
CloudTable table = tableClient.GetTableReference("people");

// Construct the query operation for all customer entities where PartitionKey="Smith".
TableQuery<CustomerEntity> query =
    new TableQuery<CustomerEntity>().Where(TableQuery.GenerateFilterCondition(
        "PartitionKey", QueryComparisons.Equal, "Smith"));

// Print the fields for each customer.
foreach (CustomerEntity entity in table.ExecuteQuery(query))
{
    Console.WriteLine("{0}, {1}\t{2}\t{3}", entity.PartitionKey, entity.RowKey,
        entity.Email, entity.PhoneNumber);
}
```

TableQuery.GenerateFilterCondition

TableQuery.GenerateFilterCondition Method

0 out of 6 rated this helpful - [Rate this topic](#)

Updated: May 15, 2014

Generates a property filter condition string for the string value.

Namespace: Microsoft.WindowsAzure.Storage.Table

Assembly: Microsoft.WindowsAzure.Storage (in Microsoft.WindowsAzure.Storage.dll)

▲ Syntax

C# C++ VB

```
public static string GenerateFilterCondition (  
    string propertyName,  
    string operation,  
    string givenValue  
)
```

J#

JScript

Parameters

propertyName

A string containing the name of the property to compare.

operation

A string containing the comparison operator to use.

givenValue

A string containing the value to compare with the property.

Return Value

A string containing the formatted filter condition.

Code snippets - Retrieve a range of entities in a partition

```
// Retrieve the storage account from the connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the table client.
CloudTableClient tableClient = storageAccount.CreateCloudTableClient();

//Create the CloudTable object that represents the "people" table.
CloudTable table = tableClient.GetTableReference("people");

// Create the table query.
TableQuery<CustomerEntity> rangeQuery = new TableQuery<CustomerEntity>().Where(
    TableQuery.CombineFilters(
        TableQuery.GenerateFilterCondition("PartitionKey", QueryComparisons.Equal, "Smith"),
        TableOperators.And,
        TableQuery.GenerateFilterCondition("RowKey", QueryComparisons.LessThan, "E")));

// Loop through the results, displaying information about the entity.
foreach (CustomerEntity entity in table.ExecuteQuery(rangeQuery))
{
    Console.WriteLine("{0}, {1}\t{2}\t{3}", entity.PartitionKey, entity.RowKey,
        entity.Email, entity.PhoneNumber);
}
```

Code snippets - Retrieve a single entity

```
// Retrieve the storage account from the connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the table client.
CloudTableClient tableClient = storageAccount.CreateCloudTableClient();

// Create the CloudTable object that represents the "people" table.
CloudTable table = tableClient.GetTableReference("people");

// Create a retrieve operation that takes a customer entity.
TableOperation retrieveOperation = TableOperation.Retrieve<CustomerEntity>("Smith", "Ben");

// Execute the retrieve operation.
TableResult retrievedResult = table.Execute(retrieveOperation);

// Print the phone number of the result.
if (retrievedResult.Result != null)
    Console.WriteLine(((CustomerEntity)retrievedResult.Result).PhoneNumber);
else
    Console.WriteLine("The phone number could not be retrieved.");
```

Code snippets - Replace an entity

```
// Retrieve the storage account from the connection string.
// Create the table client
// Create the CloudTable object that represents the "people" table.
...
// Create a retrieve operation that takes a customer entity.
TableOperation retrieveOperation = TableOperation.Retrieve<CustomerEntity>("Smith", "Ben");

// Execute the operation.
TableResult retrievedResult = table.Execute(retrieveOperation);

// Assign the result to a CustomerEntity object.
CustomerEntity updateEntity = (CustomerEntity)retrievedResult.Result;

if (updateEntity != null) {
    // Change the phone number.
    updateEntity.PhoneNumber = "425-555-0105";
    // Create the InsertOrReplace TableOperation
    TableOperation updateOperation = TableOperation.Replace(updateEntity);
    // Execute the operation.
    table.Execute(updateOperation);
    Console.WriteLine("Entity updated.");
}
else
    Console.WriteLine("Entity could not be retrieved.");
```

Code snippets - Insert-or-replace an entity

```
// Retrieve the storage account from the connection string.
// Create the table client.
// Create the CloudTable object that represents the "people" table.
...

// Create a retrieve operation that takes a customer entity.
TableOperation retrieveOperation = TableOperation.Retrieve<CustomerEntity>("Smith", "Ben");

// Execute the operation.
TableResult retrievedResult = table.Execute(retrieveOperation);

// Assign the result to a CustomerEntity object.
CustomerEntity updateEntity = (CustomerEntity)retrievedResult.Result;

if (updateEntity != null) {
    // Change the phone number.
    updateEntity.PhoneNumber = "425-555-1234";
    // Create the InsertOrReplace TableOperation
    TableOperation insertOrReplaceOperation = TableOperation.InsertOrReplace(updateEntity);
    // Execute the operation.
    table.Execute(insertOrReplaceOperation);
    Console.WriteLine("Entity was updated.");
}
else
    Console.WriteLine("Entity could not be retrieved.");
```

Code snippets - Delete an entity

```
// Retrieve storage account from connection string
// Create the table client
//Create the CloudTable that represents the "people" table.
...
// Create a retrieve operation that expects a customer entity.
TableOperation retrieveOperation = TableOperation.Retrieve<CustomerEntity>("Smith", "Ben");

// Execute the operation.
TableResult retrievedResult = table.Execute(retrieveOperation);

// Assign the result to a CustomerEntity.
CustomerEntity deleteEntity = (CustomerEntity)retrievedResult.Result;

// Create the Delete TableOperation.
if (deleteEntity != null) {
    TableOperation deleteOperation = TableOperation.Delete(deleteEntity);

    // Execute the operation.
    table.Execute(deleteOperation);

    Console.WriteLine("Entity deleted.");
}
else
    Console.WriteLine("Could not retrieve the entity.");
```

Code snippets - Delete a table

```
// Retrieve the storage account from the connection string.

CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the table client.

CloudTableClient tableClient = storageAccount.CreateCloudTableClient();

//Create the CloudTable that represents the "people" table.

CloudTable table = tableClient.GetTableReference("people");

// Delete the table if it exists.

table.DeleteIfExists();
```

Summary

- Store data in tables
- Store data in blobs

The Blob Service

- The Azure Blob storage service stores large amounts of unstructured data that can be accessed via HTTP or HTTPS.
- A single blob can be hundreds of gigabytes in size.
- Blob storage can be used for:
 - Storing files for distributed access
 - Streaming video and audio
 - Performing secure backup and disaster recovery
 - Serving images or documents directly to a browser

Concepts

The Blob service contains the following components:

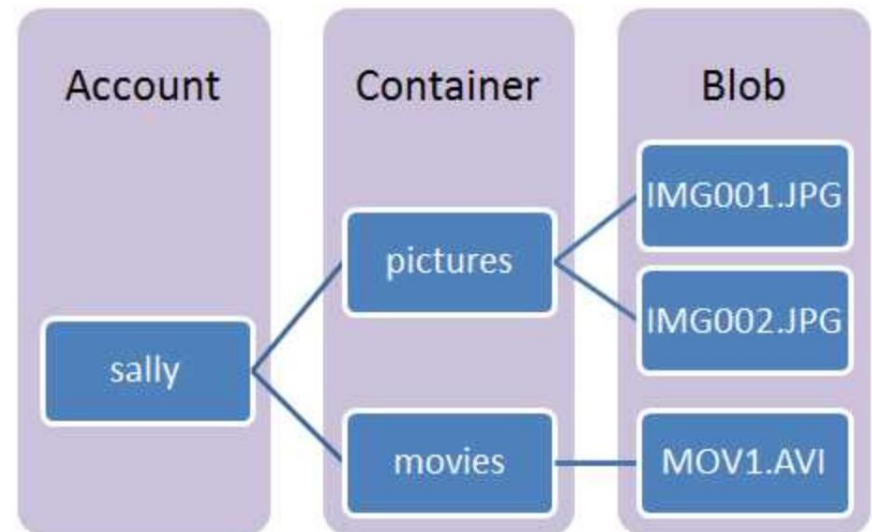
Storage Account: it is needed to access to any Azure Storage service

Container: it provides a grouping of a set of blobs

- All blobs must be in a container
- An account can contain an unlimited number of containers
- A container can store an unlimited number of blobs.

Blob: it is a file of any type and size

- two types of blobs: block and page blobs.
- A Block blob can be up to 200 GB in size.
- A Page blob, can be up to 1 TB in size
 - (more efficient when ranges of bytes in a file are modified frequently)
- A Directory blob is a directory



Preliminary steps

- Obtain the assembly (by NuGet)
- Create and configure a `ConnectionString` (i.e, 'StorageConnectionString')

Code snippets - Upload a blob into a container

```
protected void btn_Click(object sender, EventArgs e) {

    // Retrieve storage account from connection string.
    CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
        CloudConfigurationManager.GetSetting("StorageConnectionString"));

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

    // Retrieve a reference to a container.
    CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

    // Create the container if it doesn't already exist.
    container.CreateIfNotExists();

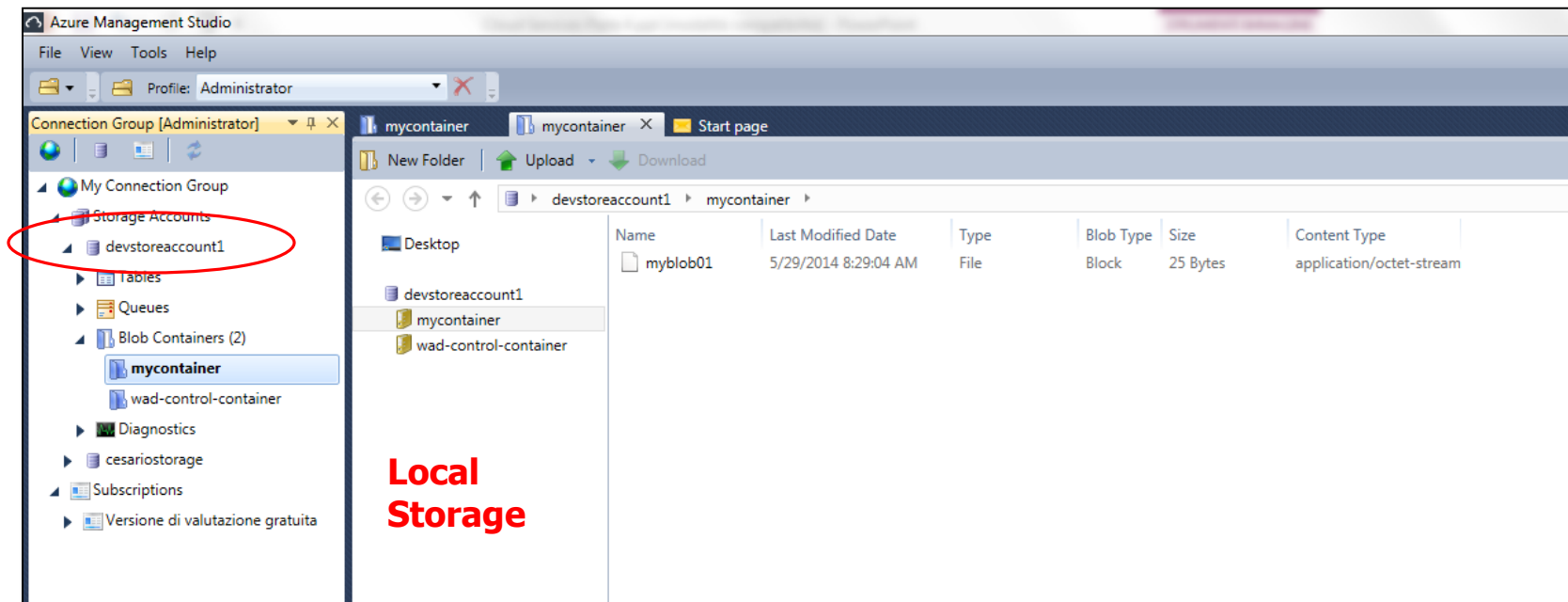
    // Retrieve reference to a blob named "myblob".
    CloudBlockBlob blockBlob = container.GetBlockBlobReference("myblob01");

    // Create or overwrite the "myblob" blob with contents from a local file.
    using (var fileStream = System.IO.File.OpenRead(@"C:\...\BlobFiles\myblob01.txt")) {
        blockBlob.UploadFromStream(fileStream);
    }

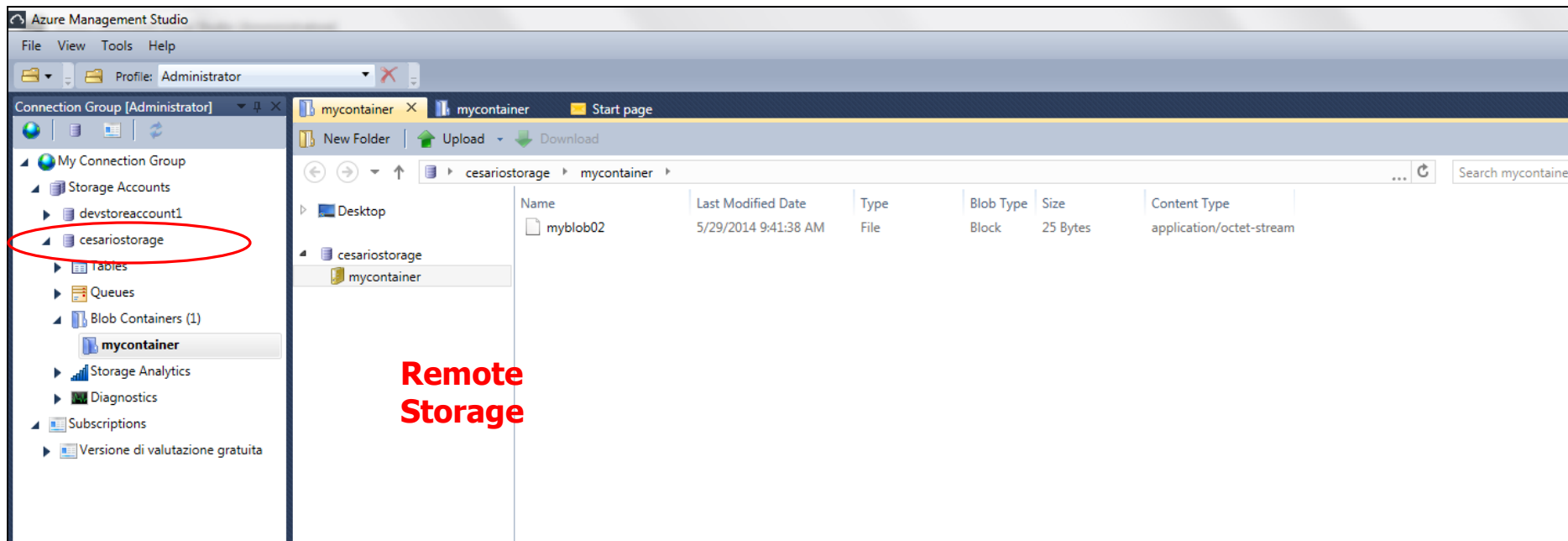
}
```

View Blobs

- How to view blobs?
 - Azure Management Studio
 - Azure Storage Explorer



View Blobs



- Demo in classroom

- Local Storage
 - Fix connection string
 - Upload a blob
 - Check contents

- Remote Storage
 - Login the Azure Management Portal
 - Fix connection string
 - Upload a blob
 - Check contents

Code snippets - List the blobs in a container (not flat listing)

```
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

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

// Retrieve reference to a previously created container.
CloudBlobContainer container = blobClient.GetContainerReference("photos");

// Loop over items within the container and output the length and URI.
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);
    }
    else if (item.GetType() == typeof(CloudPageBlob)) {
        CloudPageBlob pageBlob = (CloudPageBlob)item;
        Console.WriteLine("Page blob of length {0}: {1}", pageBlob.Properties.Length, pageBlob.Uri);
    }
    else if (item.GetType() == typeof(CloudBlobDirectory)) {
        CloudBlobDirectory directory = (CloudBlobDirectory)item;
        Console.WriteLine("Directory: {0}", directory.Uri);
    }
}
```

Output: not flat listing

- Let us suppose the following set of block blobs in the container:
 - photo1.jpg
 - 2010/architecture/description.txt
 - 2010/architecture/photo3.jpg
 - 2010/architecture/photo4.jpg
 - 2011/architecture/photo5.jpg
 - 2011/architecture/photo6.jpg
 - 2011/architecture/description.txt
 - 2011/photo7.jpg

- The resulting output could be:
 - Directory: <https://<accountname>.blob.core.windows.net/photos/2010/>
 - Directory: <https://<accountname>.blob.core.windows.net/photos/2011/>
 - Block blob of length 505623: <https://<accountname>.blob.core.windows.net/photos/photo1.jpg>

Code snippets - List the blobs in a container (flat listing)

```
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

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

// Retrieve reference to a previously created container.
CloudBlobContainer container = blobClient.GetContainerReference("photos");

// Loop over items within the container and output the length and URI.
foreach (IListBlobItem item in container.ListBlobs(null, true)) {
    if (item.GetType() == typeof(CloudBlockBlob)) {
        CloudBlockBlob blob = (CloudBlockBlob)item;
        Console.WriteLine("Block blob of length {0}: {1}", blob.Properties.Length, blob.Uri);
    }
    else if (item.GetType() == typeof(CloudPageBlob)) {
        CloudPageBlob pageBlob = (CloudPageBlob)item;
        Console.WriteLine("Page blob of length {0}: {1}", pageBlob.Properties.Length, pageBlob.Uri);
    }
    else if (item.GetType() == typeof(CloudBlobDirectory)) {
        CloudBlobDirectory directory = (CloudBlobDirectory)item;
        Console.WriteLine("Directory: {0}", directory.Uri);
    }
}
```

Output: flat listing

■ Let us suppose the following set of block blobs in the container:

- photo1.jpg
- 2010/architecture/description.txt
- 2010/architecture/photo3.jpg
- 2010/architecture/photo4.jpg
- 2011/architecture/photo5.jpg
- 2011/architecture/photo6.jpg
- 2011/architecture/description.txt
- 2011/photo7.jpg

■ The resulting output could be:

- Block blob of length 4: <https://<accountname>.blob.core.windows.net/photos/2010/architecture/description.txt>
- Block blob of length 314618: <https://<accountname>.blob.core.windows.net/photos/2010/architecture/photo3.jpg>
- Block blob of length 522713: <https://<accountname>.blob.core.windows.net/photos/2010/architecture/photo4.jpg>
- Block blob of length 4: <https://<accountname>.blob.core.windows.net/photos/2011/architecture/description.txt>
- Block blob of length 419048: <https://<accountname>.blob.core.windows.net/photos/2011/architecture/photo5.jpg>
- Block blob of length 506388: <https://<accountname>.blob.core.windows.net/photos/2011/architecture/photo6.jpg>
- Block blob of length 399751: <https://<accountname>.blob.core.windows.net/photos/2011/photo7.jpg>
- Block blob of length 505623: <https://<accountname>.blob.core.windows.net/photos/photo1.jpg>

Code snippets - Download blobs

```
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

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

// Retrieve reference to a previously created container.
CloudBlobContainer container =
    blobClient.GetContainerReference("mycontainer");

// Retrieve reference to a blob named "photo1.jpg".
CloudBlockBlob blockBlob = container.GetBlockBlobReference("photo1.jpg");

// Save blob contents to a file.
using (var fileStream = System.IO.File.OpenWrite(@"path\myfile"))
{
    blockBlob.DownloadToStream(fileStream);
}
```

Code snippets - Download blobs (as a text string)

```
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

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

// Retrieve reference to a previously created container.
CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

// Retrieve reference to a blob named "myblob.txt"
CloudBlockBlob blockBlob2 = container.GetBlockBlobReference("myblob.txt");

string text;
using (var memoryStream = new MemoryStream())
{
    blockBlob2.DownloadToStream(memoryStream);
    text = System.Text.Encoding.UTF8.GetString(memoryStream.ToArray());
}
```

Code snippets - Delete blobs

```
// Retrieve storage account from connection string.

CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the blob client.

CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Retrieve reference to a previously created container.

CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

// Retrieve reference to a blob named "myblob.txt".

CloudBlockBlob blockBlob = container.GetBlockBlobReference("myblob.txt");

// Delete the blob.

blockBlob.Delete();
```