Windows Azure

- Windows Azure is the foundation of Microsoft’s Cloud Platform
- It is an “Operating System for the Cloud” and provides Essential Services for the Cloud
  - Virtualized Computation
  - Scalable Storage
  - Automatic Management
  - Developer SDK
Cloud Storage is part of Windows Azure
Windows Azure Storage

- The goal is to allow users and applications to:
  - Access their data efficiently from anywhere at anytime
  - Store data for any length of time
  - Scale to store any amount of data
  - Be confident that the data will not be lost
  - Pay for only what they use/store
Windows Azure Storage

• Storage that is
  – Durable
  – Scalable (capacity and throughput)
  – Highly Available
  – Security
  – Performance Efficient

• Rich Data Abstractions
  – Service communication: queues, locks, …
  – Large user data items: blobs, blocks, …
  – Service state: tables, caches, …

• Simple and Familiar Programming Interfaces
  – REST Accessible and ADO.NET
Windows Azure Storage Account

• User creates a globally unique storage account name
  – Receive a 256 bit secret key when creating account

• Provides security for accessing the store
  – Use secret key to create a HMAC SHA256 signature for each request
  – Use signature to authenticate request at server
SETTING UP CLOUD STORAGE
Fundamental Data Abstractions

• **Blobs** – Provide a simple interface for storing named files along with metadata for the file

• **Tables** – Provide structured storage. A Table is a set of entities, which contain a set of properties

• **Queues** – Provide reliable storage and delivery of messages for an application
Blob Storage Concepts

Account
  - sally

Container
  - pictures
  - movies

Blob
  - IMG001.JPG
  - IMG002.JPG
  - MOV1.AVI
Storage Account And Blob Containers

• Storage Account
  – An account can have many Blob Containers

• Container
  – A container is a set of blobs
  – Sharing policies are set at the container level
    • Public READ or Private
  – Associate Metadata with Container
    • Metadata is <name, value> pairs
    • Up to 8KB per container
  – List the blobs in a container
Blob Namespace

• Blob URL
  
  http://<Account>.blob.core.windows.net/<Container>/<BlobName>  

Example:

• Account – sally
• Container – music
• BlobName – rock/rush/xanadu.mp3
• URL: http://sally.blob.core.windows.net/music/rock/rush/xanadu.mp3
Blob Features And Functions

• Store Large Objects (up to 50 GB each)
• Standard REST PUT/GET Interface
  – http://<Account>.blob.core.windows.net/<Container>/<BlobName>
    • Inserts a new blob or overwrites the existing blob
  – GetBlob
    • Get whole blob or by starting offset, length
  – DeleteBlob
  – Support for Continuation on Upload
• Associate Metadata with Blob
  – Metadata is <name, value> pairs
  – Set/Get with or separate from blob data bits
  – Up to 8KB per blob
Continuation On Upload Scenario

• Want to upload a large multi GB file into the cloud
• If upload fails in the middle, need an efficient way to resume the upload from where it failed
Uploading A Blob Via Blocks

• Uploading a Large Blob

Benefit:
• Efficient continuation and retry
• Parallel and out of order upload of blocks

```
blobName = “TheBlob.wmv”;
PutBlock(blobName, blockId1, block1Bits);
PutBlock(blobName, blockId2, block2Bits);
...........
PutBlock(blobName, blockIdN, blockNBits);
PutBlockList(blobName, blockId1,...,blockIdN);
```
PutBlockList Example

BlobName = ExampleBlob.wmv

Committed and readable version of blob

- Example Uploading
  - Blocks Out of Order
  - Same Block IDs
  - Unused Blocks

- PutBlock(BlockId1)
- PutBlock(BlockId3)
- PutBlock(BlockId4)
- PutBlock(BlockId2)
- PutBlock(BlockId4)

- PutBlockList(BlockId2, BlockId3, BlockId4)
Blob As A List Of Blocks

- Blob
  - Consists of a List of Blocks

- Properties of Blocks
  - Each Block defined by a Block ID
    - Up to 64 Bytes, scoped by Blob Name
  - Blocks are immutable
  - A block is up to 64MB
    - Do not have to be same size
BlockList Operations

• PutBlockList for a Blob
  – Provide the list of blocks to comprise the readable version of the blob
  – If multiple blocks are uploaded with same Block ID
    – Last committed block wins
  – Blocks not used will be garbage collected

• GetBlockList for a Blob
  – Returns the list of blocks that represent the readable (committed) version of the blob
    • Block ID and Size of Block is returned for each block
demo

Blob Storage
Summary Of Windows Azure Blobs

• Easy to use REST Put/Get/Delete interface
• Can read from any Offset, Length of Blob
• Conditional Put and Get Blob
• Max Blob size
  – 50 GB using PutBlock and PutBlockList
  – 64 MB using PutBlob
• Blocks provide continuation for blob upload
• Put Blob/BlockList == Replace Blob for CTP
  – Can replace an existing blob with new blob/blocks
Future Windows Azure Blob Support

• Update Blob
  – Ability to replace, add, or remove blocks from a blob

• Append Blob
  – Ability to append a block to a blob

• Copy Blob
  – Ability to copy an existing blob to a new blob name
Fundamental Data Abstractions

- **Blobs** – Provide a simple interface for storing named files along with metadata for the file
- **Tables** – Provide structured storage. A Table is a set of entities, which contain a set of properties
- **Queues** – Provide reliable storage and delivery of messages for an application
Windows Azure Tables

• Provides Structured Storage
  – Massively Scalable Tables
    • Billions of entities (rows) and TBs of data
    • Automatically scales to thousands of servers as traffic grows
  – Highly Available
    • Can always access your data
  – Durable
    • Data is replicated at least 3 times

• Familiar and Easy to use Programming Interfaces
  – ADO.NET Data Services – .NET 3.5 SP1
    • .NET classes and LINQ
    • REST - with any platform or language
Table Data Model

• Table
  – A Storage Account can create many tables
  – Table name is scoped by Account

• Data is stored in Tables
  – A Table is a set of Entities (rows)
  – An Entity is a set of Properties (columns)

• Entity
  – Two “key” properties that together are the unique ID of the entity in the Table
    • PartitionKey – enables scalability
    • RowKey – uniquely identifies the entity within the partition
Partition Key And Partitions

• Every Table has a Partition Key
  – It is the first property (column) of your Table
  – Used to group entities in the Table into partitions

• A Table Partition
  – All entities in a Table with the same partition key value

• Partition Key is exposed in the programming model
  – Allows application to control the granularity of the partitions and enable scalability
### Partition Example

<table>
<thead>
<tr>
<th>Partition Key</th>
<th>Row Key Version</th>
<th>Property 3 Modification Time</th>
<th>......</th>
<th>Property N Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples Doc</td>
<td>V1.0</td>
<td>8/2/2007</td>
<td>......</td>
<td>Committed version</td>
</tr>
<tr>
<td>Examples Doc</td>
<td>V2.0.1</td>
<td>9/28/2007</td>
<td></td>
<td>Alice’s working version</td>
</tr>
<tr>
<td>FAQ Doc</td>
<td>V1.0</td>
<td>5/2/2007</td>
<td></td>
<td>Committed version</td>
</tr>
<tr>
<td>FAQ Doc</td>
<td>V1.0.1</td>
<td>7/6/2007</td>
<td></td>
<td>Alice’s working version</td>
</tr>
<tr>
<td>FAQ Doc</td>
<td>V1.0.2</td>
<td>8/1/2007</td>
<td></td>
<td>Sally’s working version</td>
</tr>
</tbody>
</table>

- **Table Partition** - all entities in table with same partition key value
- **Application controls granularity of partition**
Purpose Of The Partition

• Performance and Entity Locality
  – Entities in the same partition will be stored together
  • Efficient querying and cache locality

• Table Scalability
  – We monitor the usage patterns of partitions
  – Automatically load balance partitions
  • Each partition can be served by a different storage node
  • Scale to meet the traffic needs of your table
# Performance And Scalability

## Table

<table>
<thead>
<tr>
<th>Partition Key Document Name</th>
<th>Row Key Version</th>
<th>Property 3 Modification Time</th>
<th>Property N Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples Doc</td>
<td>V1.0</td>
<td>8/2/2007</td>
<td>Committed version</td>
</tr>
<tr>
<td>Examples Doc</td>
<td>V2.0.1</td>
<td>9/28/2007</td>
<td>Alice’s working version</td>
</tr>
<tr>
<td>FAQ Doc</td>
<td>V1.0</td>
<td>5/2/2007</td>
<td>Committed version</td>
</tr>
<tr>
<td>FAQ Doc</td>
<td>V1.0.1</td>
<td>7/6/2007</td>
<td>Alice’s working version</td>
</tr>
<tr>
<td>FAQ Doc</td>
<td>V1.0.2</td>
<td>8/1/2007</td>
<td>Sally’s working version</td>
</tr>
</tbody>
</table>

- Efficient retrieval of all of the versions of FAQ Doc
  - Since we are accessing a single partition
- The two partitions can be served from different servers to scale out access
Choosing A Partition Key

• Use a PartitionKey that is common in your queries
  – If Partition Key is part of Query
    • Fast access to retrieve entities within a single partition
  – If Partition Key is not specified in a Query
    • Then every partition has to be scanned
## Query With And Without PartitionKey

<table>
<thead>
<tr>
<th>Partition Key Document Name</th>
<th>Row Key Version</th>
<th>Property 3 Modification Time</th>
<th>.....</th>
<th>Property N Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples Doc</td>
<td>V1.0</td>
<td>8/2/2007</td>
<td>.....</td>
<td>Committed version</td>
</tr>
<tr>
<td>Examples Doc</td>
<td>V2.0.1</td>
<td>9/28/2007</td>
<td>.....</td>
<td>Alice’s working version</td>
</tr>
<tr>
<td>FAQ Doc</td>
<td>V1.0</td>
<td>5/2/2007</td>
<td>.....</td>
<td>Committed version</td>
</tr>
<tr>
<td>FAQ Doc</td>
<td>V1.0.1</td>
<td>7/6/2007</td>
<td>.....</td>
<td>Alice’s working version</td>
</tr>
<tr>
<td>FAQ Doc</td>
<td>V1.0.2</td>
<td>8/1/2007</td>
<td>.....</td>
<td>Sally’s working version</td>
</tr>
</tbody>
</table>

- Getting all entities with (PartitionKey == “FAQ Doc”) is fast  
  - Access single partition  
- Get all docs with (ModifiedTime < 6/01/2007) is more expensive  
  - Have to traverse all partitions
Choosing A Partition Key

- Use a PartitionKey that is common in your queries
  - If Partition Key is part of Query
    - Fast access to retrieve entities within a single partition
  - If Partition Key is not specified in a Query
    - Then every partition has to be scanned
- Spread out load across partitions
  - Partition Key allows you to control what goes into your Table partitions
  - More partitions – makes it easier to automatically balance load
    - At the tradeoff of Entity Locality
# Primary Key For Entity

<table>
<thead>
<tr>
<th>Partition Key</th>
<th>Row Key Version</th>
<th>Property 3 Modification Time</th>
<th>.....</th>
<th>Property N Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples Doc</td>
<td>V1.0</td>
<td>8/2/2007</td>
<td>.....</td>
<td>Committed version</td>
</tr>
<tr>
<td>Examples Doc</td>
<td>V2.0.1</td>
<td>9/28/2007</td>
<td></td>
<td>Alice’s working version</td>
</tr>
<tr>
<td>FAQ Doc</td>
<td>V1.0</td>
<td>5/2/2007</td>
<td></td>
<td>Committed version</td>
</tr>
<tr>
<td>FAQ Doc</td>
<td>V1.0.1</td>
<td>7/6/2007</td>
<td></td>
<td>Alice’s working version</td>
</tr>
<tr>
<td>FAQ Doc</td>
<td>V1.0.2</td>
<td>8/1/2007</td>
<td></td>
<td>Sally’s working version</td>
</tr>
</tbody>
</table>

- **Primary Key for the Entity** is the composite of
  - **PartitionKey**
    - Unique ID of the partition the entity belongs to within the Table
  - **RowKey**
    - Unique ID for the entity within the partition
Entities And Properties

• Each Entity can have up to 255 properties
• Mandatory Properties for every Entity in Table
  – Partition Key
  – Row Key
• All entities have a system maintained version
  – Each property is stored as a <name, typed value> pair
    • No schema stored for a table
  – 2 entities within the same table can have different properties
Property Types Supported

• Partition Key and Row Key
  – String (up to 64KB)

• Property Types
  – String (up to 64KB)
  – Binary (up to 64KB)
  – Bool
  – DateTime
  – GUID
  – Int
  – Int64
  – Double
Table Programming Model

• Provide familiar and easy to use interfaces
  – Leverage your .NET expertise
• Table Entities are accessed as objects via ADO.NET Data Services – .NET 3.5 SP1
  – LINQ – Language Integrated Query
  – RESTful access to table and entities
• Insert/Update/Delete Entities over the Table
• Query over Tables
  – Get back a list of structured Entities
Example Table Definition

- Example using ADO.NET Data Services
- Table Entities are represented as Class Objects

```
[DataServiceKey("PartitionKey", "RowKey")]
public class Customer
{
    // Partition key - Customer Last name
    public string PartitionKey { get; set; }
    // Row Key - Customer First name
    public string RowKey { get; set; }

    // User defined properties here
    public DateTime CustomerSince { get; set; }
    public double Rating { get; set; }
    public string Occupation { get; set; }
}
```
Create Customers Table

- Every Account has a master table called “Tables”
  - It is used to keep track of the tables in your account
  - To use a table it has to be inserted into “Tables”

```csharp
[DataServiceKey("TableName")] public class TableStorageTable
{
    public string TableName { get; set; }
}

// serviceUri is "http://<Account>.table.core.windows.net/"
context = new DataServiceContext(serviceUri);

TableStorageTable table = new TableStorageTable("Customers");
context.AddObject("Tables", table);
DataServiceResponse response = context.SaveChanges();
```
Table Storage
Query A Table

- LINQ

```csharp
DataServiceContext context = new DataServiceContext("http://myaccount.table.core.windows.net");

var customers = from o in context.CreateQuery<Customer>("Customers")
    where o.PartitionKey == "Lee"
    select o;

foreach (Customer customer in customers) { }
```

- REST

GET http://myaccount.table.core.windows.net/Customer?
$filter=PartitionKey eq 'Lee'
Summary Of Windows Azure Tables

• Built to provide Massively Scalable, Highly Available and Durable Structured Storage
  – Partition Key is exposed to the application

• Automatic Load Balancing and Scaling of Tables

• Familiar and Easy to use LINQ and REST programming interfaces
  – ADO.Net Data Services

• Not a “relational database”
  – No joins, no maintenance of foreign keys, etc
Future Windows Azure Table Support

• At CTP
  – Single Index
    • Query and retrieve results sorted by PartitionKey and RowKey
  – Single Entity Transactions
    • Atomically Insert, Update, or Delete a Single Entity

• Future Support for
  – Secondary Indexes
    • Query and retrieve results sorted by other properties
  – Entity Groups
    • Atomic transactions across multiple entities within same partition
Fundamental Data Abstractions

• **Blobs** – Provide a simple interface for storing named files along with metadata for the file

• **Tables** – Provide structured storage. A Table is a set of entities, which contain a set of properties

• **Queues** – Provide reliable storage and delivery of messages for an application
Web + Worker Queue Example

Cloud Storage (blob, table, queue)
Windows Azure Queues

• Provide reliable message delivery
  – Simple, asynchronous work dispatch
  – Programming semantics ensure that a message can be processed at least once

• Queues are Highly Available, Durable and Performance Efficient

• Access is provided via REST
Account, Queues And Messages

• An Account can create many Queues
  – Queue Name is scoped by the Account

• A Queue contains Messages
  – No limit on number of messages stored in a Queue
    • But a Message is stored for at most a week

http://<Account>.queue.core.windows.net/<QueueName>

• Messages
  – Message Size <= 8 KB
  – To store larger data, store data in blob/entity storage, and the blob/entity name in the message
Queue Programming API

• Queues
  – Create/Clear/Delete Queues
  – Inspect Queue Length

• Messages
  – Enqueue (QueueName, Message)
  – Dequeue (QueueName, Invisibility Time T)
    • Returns the Message with a MessageID
    • Makes the Message Invisible for Time T
  – Delete(QueueName, MessageID)
Dequeue And Delete Messages

Producers

P₂

P₁

Consumers

C₁

C₂

1. Dequeue(Q, 30 sec) → msg 1

2. Dequeue(Q, 30 sec) → msg 2
Dequeue And Delete Messages

Producers

P₂

P₁

2. Dequeue(Q, 30 sec) → msg 2
3. C₂ consumed msg 2
4. Delete(Q, msg 2)
7. Dequeue(Q, 30 sec) → msg 1

Consumers

1. Dequeue(Q, 30 sec) → msg 1
5. C₁ crashed

6. msg1 visible 30 seconds after Dequeue

Benefit:
- Insures that every message can be processed at least once
Queue Storage
Summary Of Windows Azure Queues

• Provide reliable message delivery
  – Allows Messages to be retrieved and processed at least once

• No limit on number of messages stored in a Queue

• Message size is <=8KB
DURABILITY, AVAILABILITY AND SCALABILITY OF WINDOWS AZURE STORAGE
Storage Durability

• All data is replicated at least 3 times
  – Replicas are spread out over different fault and upgrade domains in same data center or geo-distribution and geo-replication
  – All of Storage (Blobs, Tables and Queues) is built on this replication layer
• Dynamic replication to maintain a healthy number of replicas
  – Recover from a lost/unresponsive Drive or Node
  – Recover from data bit rot
    • Data continuously scanned against bit rot
Availability And Scalability

• Efficient Failover
  – Data served immediately elsewhere within data center from available replicas

• Automatic Load Balancing of Hot Data
  – Monitor the usage patterns and load balance access to
    • Blob Containers, Table Partitions and Queues
  – Distribute access to the hot data over the data center according to traffic

• Caching of Hot Blobs, Entities and Queues
  – Hot Blobs are cached to scale out access to them
  – Hot Entity and Queue data pages are cached and served from memory
Windows Azure Storage Summary

• Essential Storage for the Cloud
  – Durable, Scalable, Highly Available, Security, Performance Efficient

• Familiar and Easy to Use Programming Interfaces
  – REST Accessible, LINQ and ADO.NET

• Rich Data Abstractions
  – Service communication: queues, locks, ...
  – Large user data items: blobs, blocks, ...
  – Service state: tables, caches, ...
Resources

• Azure portal: http://www.Azure.com
• Samples: Azure SDK, Cloud Storage sample
• User group: http://www.AzureUserGroup.com
• Forums: Azure Forums on MSDN
• Azure Storage Explorer: http://www.codeplex.com/azurestorageexplorer