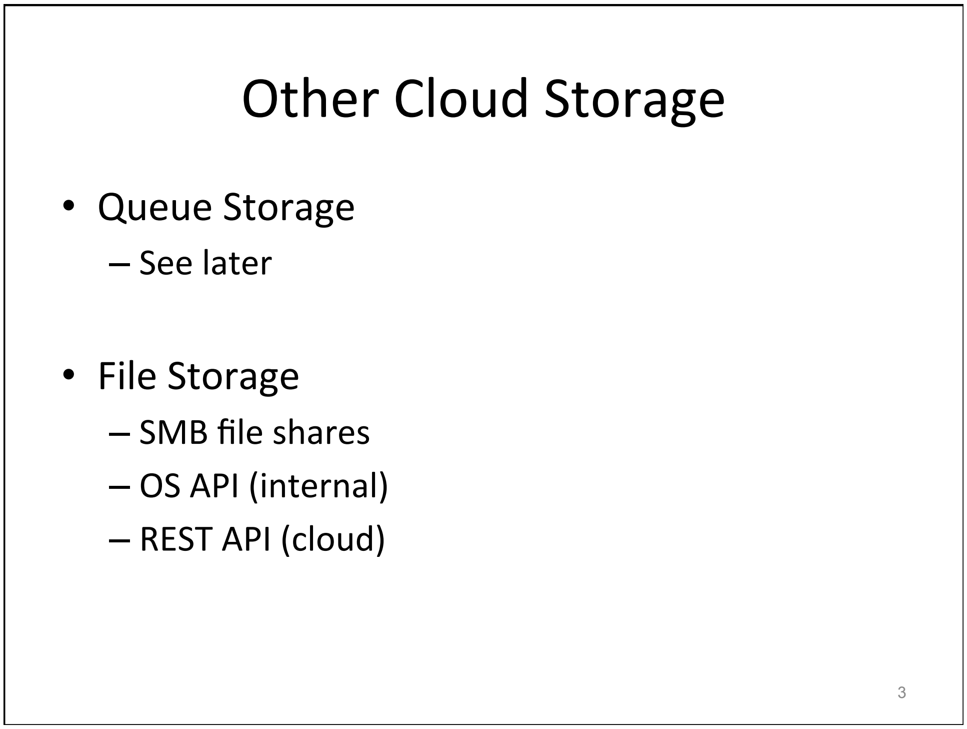
Cloud Storage

Dominic Duggan
Stevens Institute of Technology

Cloud Storage Models

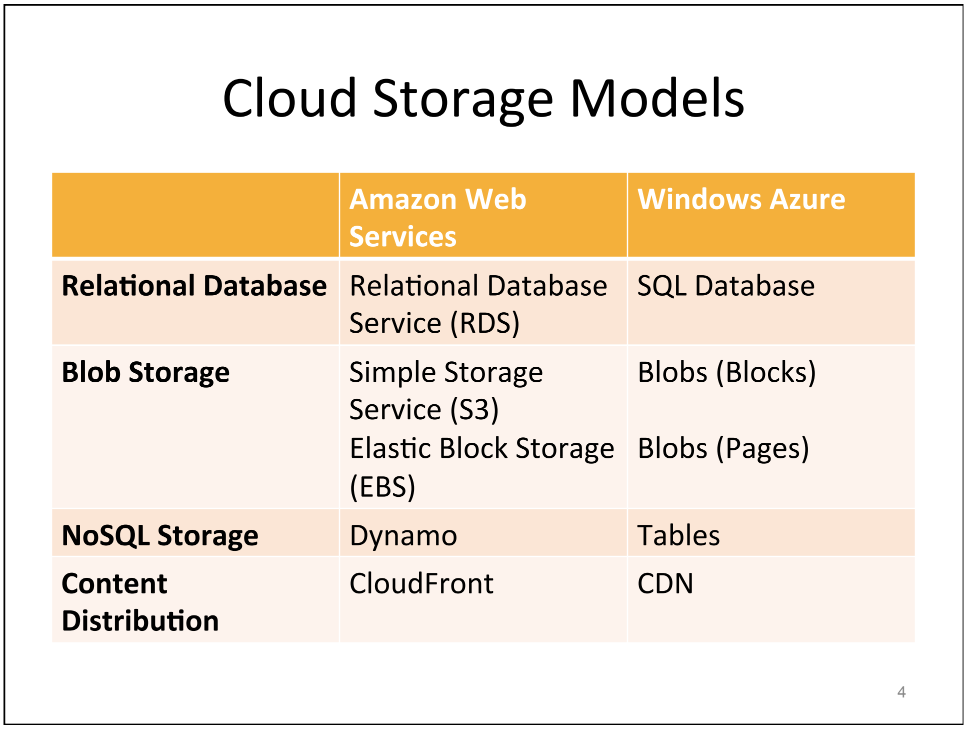
- Relational Database
- Blob Storage
- NoSQL Storage
- Content Distribution Networks



Other Cloud Storage

- Queue Storage
  - See later
- File Storage
  - SMB file shares
  - OS API (internal)
  - REST API (cloud)

3



Cloud Storage Models

	Amazon Web Services	Windows Azure
Relational Database	Relational Database Service (RDS)	SQL Database
Blob Storage	Simple Storage Service (S3) Elastic Block Storage (EBS)	Blobs (Blocks) Blobs (Pages)
NoSQL Storage	Dynamo	Tables
Content Distribution	CloudFront	CDN

4

5

BLOB STORAGE

6

Blob Storage

- Blob: big list of binary data
  - Images, audio, video, etc
- Container: group of blobs
  - Flat file system
- Storage account
  - Max 100Tb

- Address:

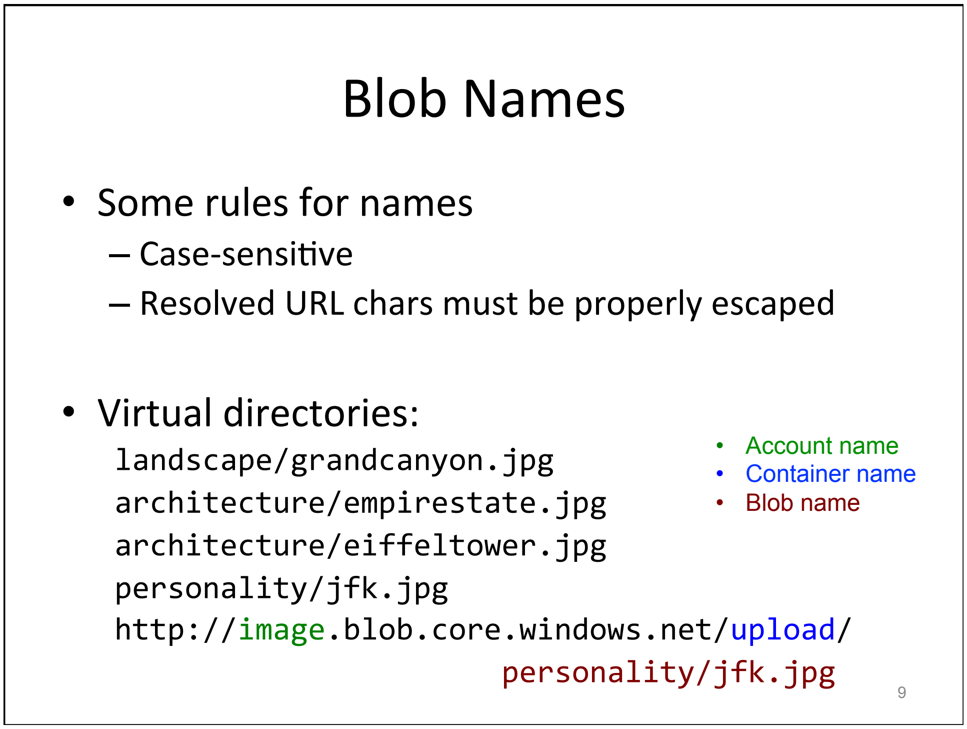
```
http://account.blob.core.windows.net/
                       container-name/blob-name
```

Container Names

- Some rules for names
  - Valid DNS name
  - Letters, numbers, dash
  - *All letters must be lowercase*

- Root container

```
http://image.blob.core.windows.net/upload/1.jpg
http://image.blob.core.windows.net/$root/1.jpg
http://image.blob.core.windows.net/1.jpg
```



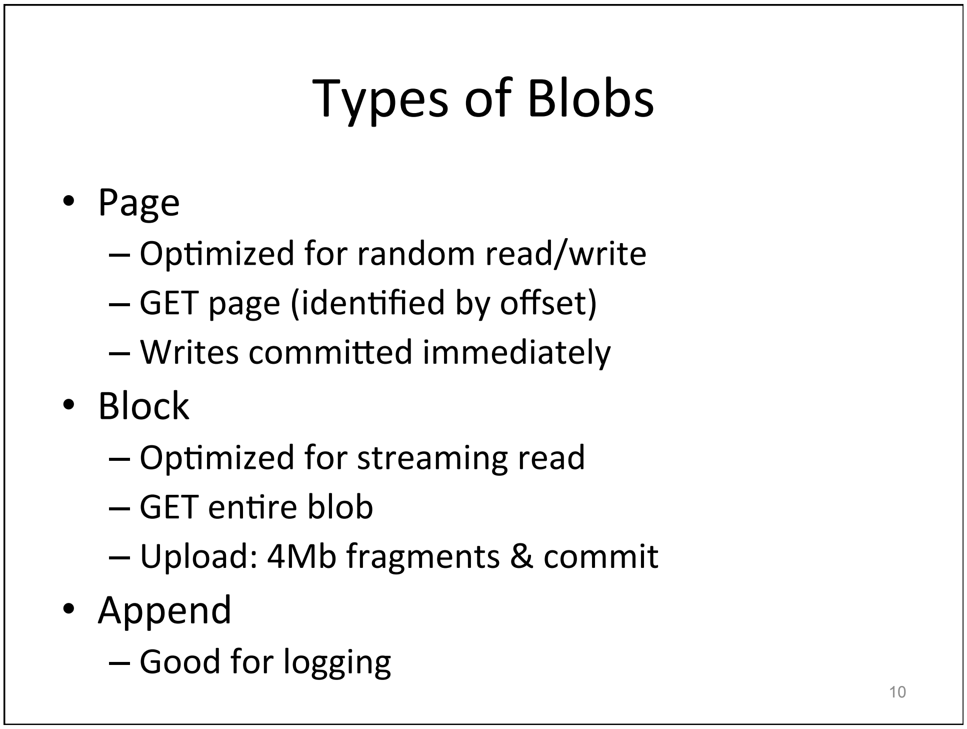
Blob Names

- Some rules for names
  - Case-sensitive
  - Resolved URL chars must be properly escaped

- Virtual directories:

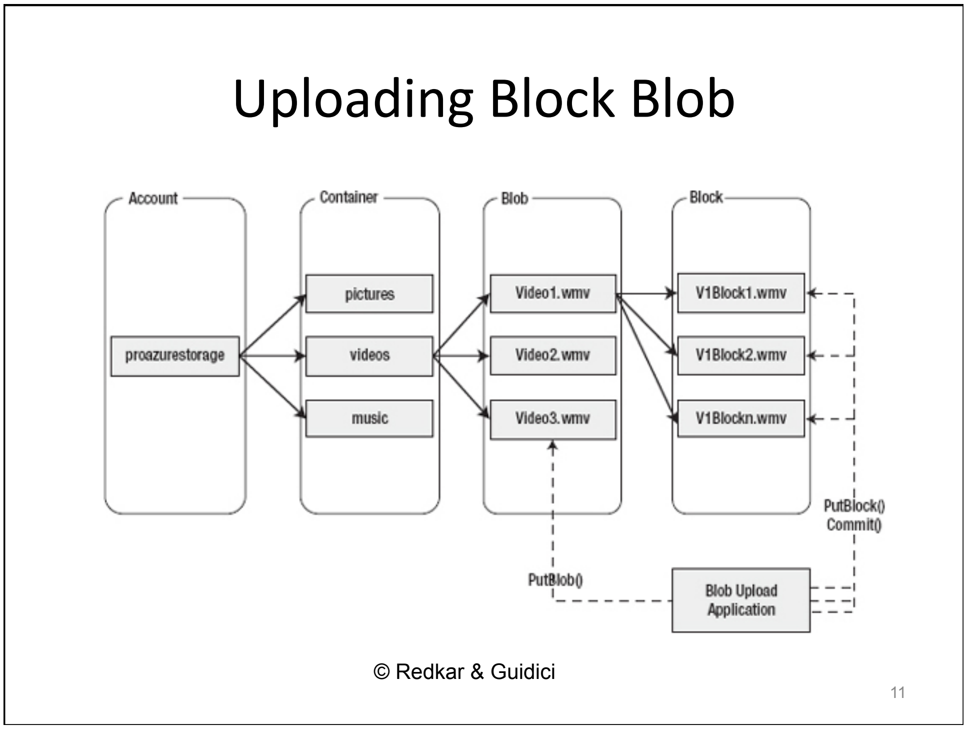
```
landscape/grandcanyon.jpg
architecture/empirestate.jpg
architecture/eiffeltower.jpg
personality/jfk.jpg
http://image.blob.core.windows.net/upload/
                      personality/jfk.jpg
```

- Account name
- Container name
- Blob name



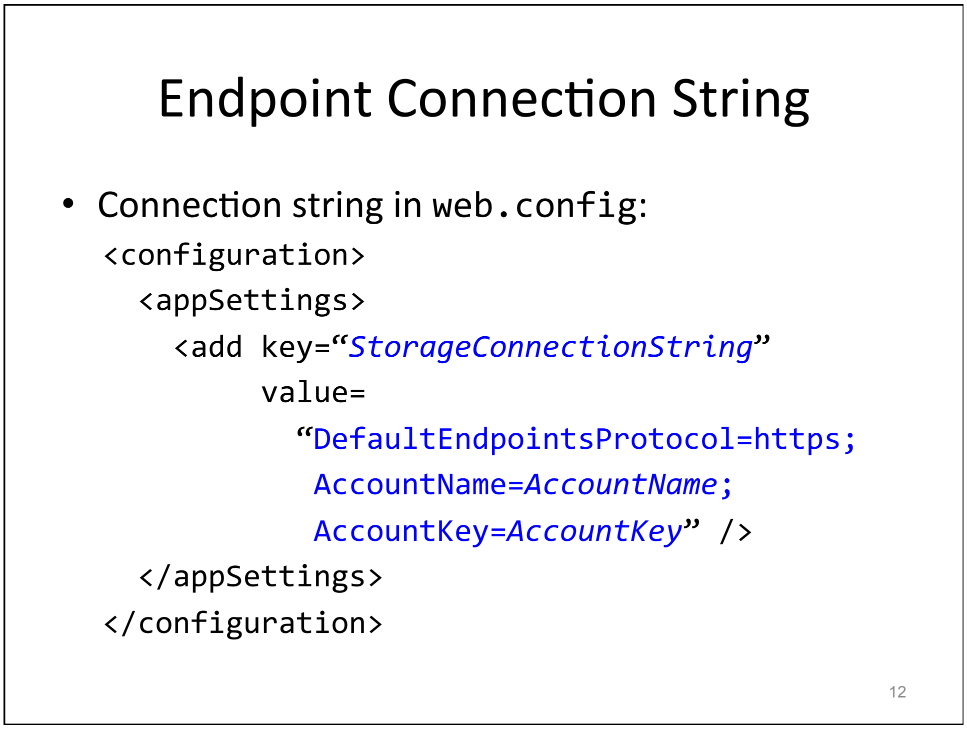
Types of Blobs

- Page
  - Optimized for random read/write
  - GET page (identified by offset)
  - Writes committed immediately
- Block
  - Optimized for streaming read
  - GET entire blob
  - Upload: 4Mb fragments & commit
- Append
  - Good for logging



Uploading Block Blob

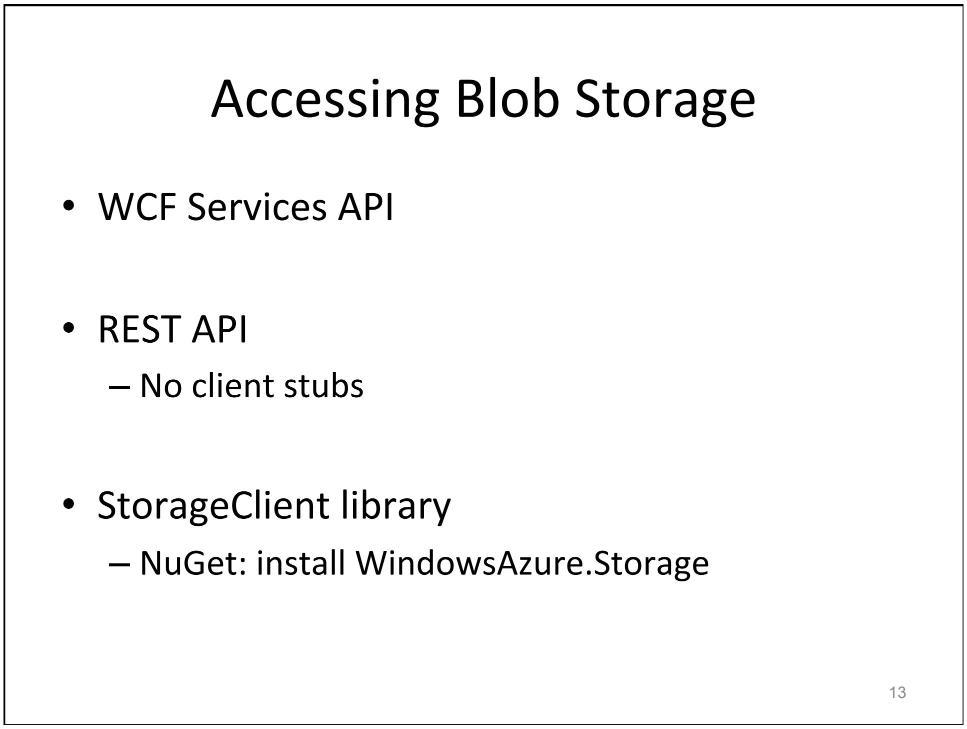
© Redkar & Guidici



Endpoint Connection String

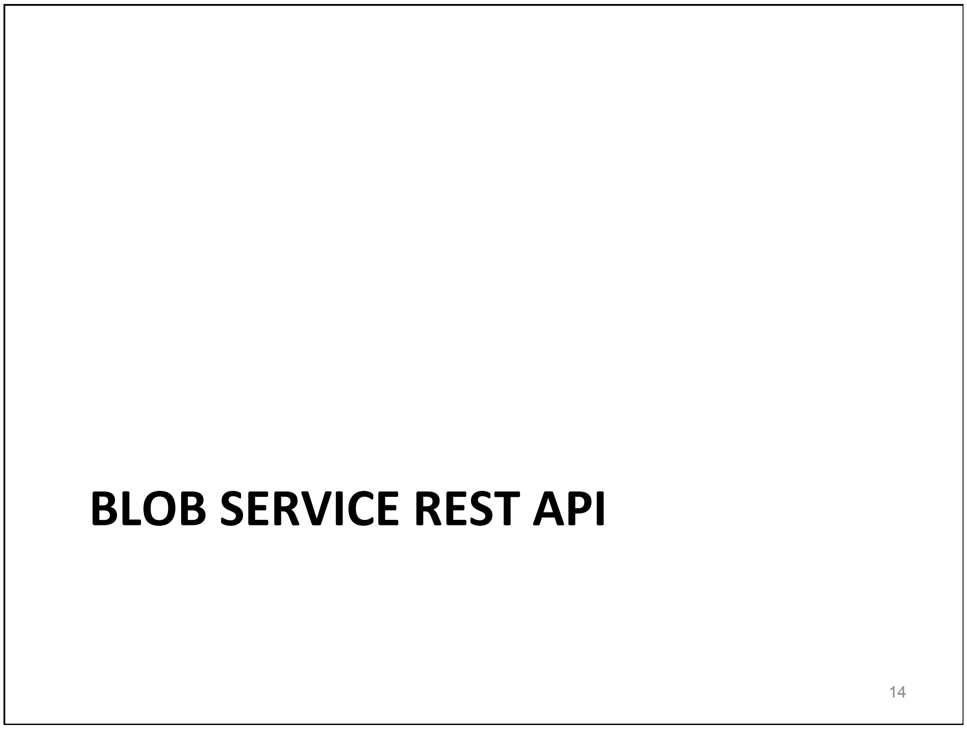
- Connection string in `web.config`:

```
<configuration>
  <appSettings>
    <add key=“StorageConnectionString”
         value=
           “DefaultEndpointsProtocol=https;
            AccountName=AccountName;
            AccountKey=AccountKey” />
  </appSettings>
</configuration>
```



Accessing Blob Storage

- WCF Services API
- REST API
  - No client stubs
- StorageClient library
  - NuGet: install WindowsAzure.Storage

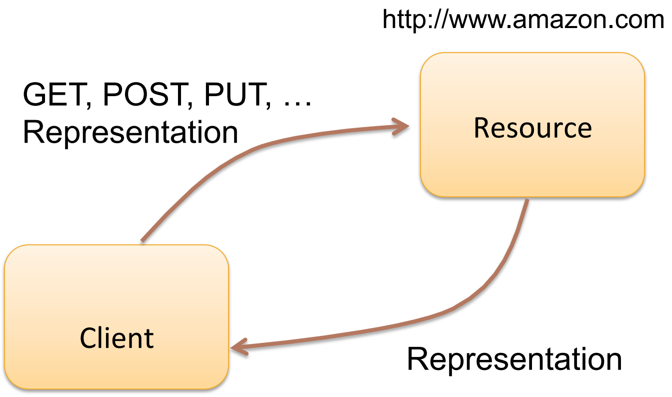


BLOB SERVICE REST API

REST is not HTTP!

- …but for this discussion we will assume it is
- Resources
  - Identified by URIs
- Representations
  - E.g. HTML files

Representational State Transfer



HTTP Request

```
GET /index.html HTTP/1.1
Host: www.example.org

...request headers...
```

HTTP Response

```
HTTP/1.1 200 OK
Date: Mon, 1 May 2011 21:38:14 GMT
Server: Apache/1.3.34 (Debian) mod ssl/2.8.25
OpenSSL/0.9.8c ...
Last-Modified: Wed, 25 Nov 2009 12:27:01 GMT
ETag: "7496a6-a0c-4b0d2295"
Accept-Ranges: bytes
Content-Length: 2572
Content-Type: text/html
Via: 1.1 www.example.org
Vary: Accept-Encoding
...
```

Uniform Interface

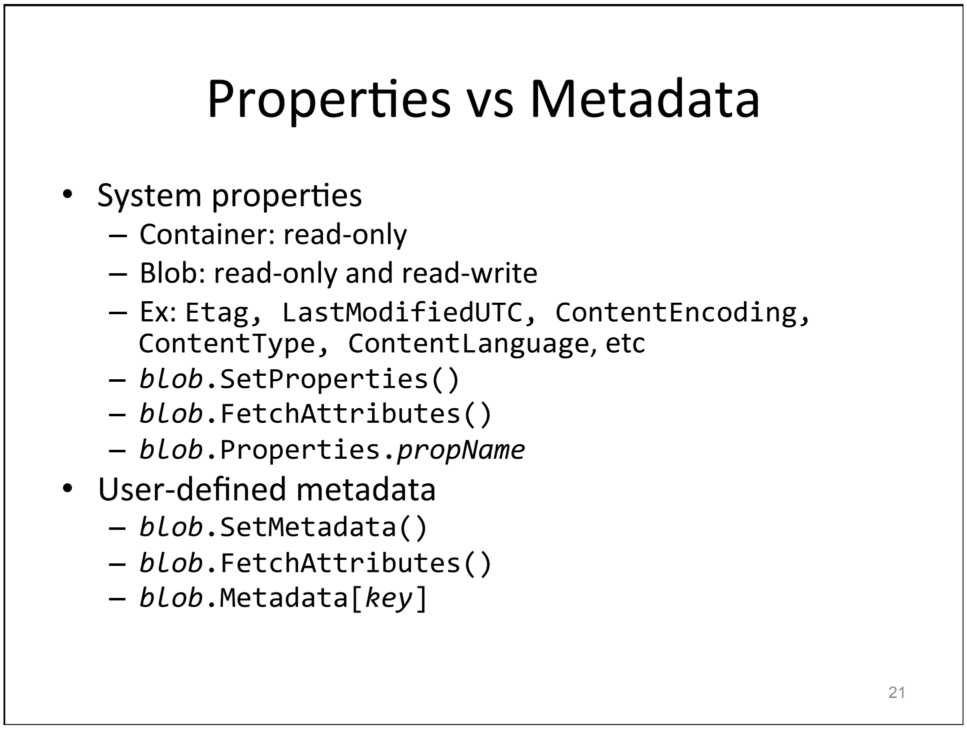
- Retrieve: HTTP GET
- Create:
  - HTTP PUT for new URI or
  - HTTP POST for existing URI (server decides result URI)
- Modify: HTTP PUT, PATCH to existing URI
- Delete: HTTP DELETE

- Retrieve metadata only: HTTP HEAD
- Check which methods are supported: HTTP OPTIONS

- No other operations besides these

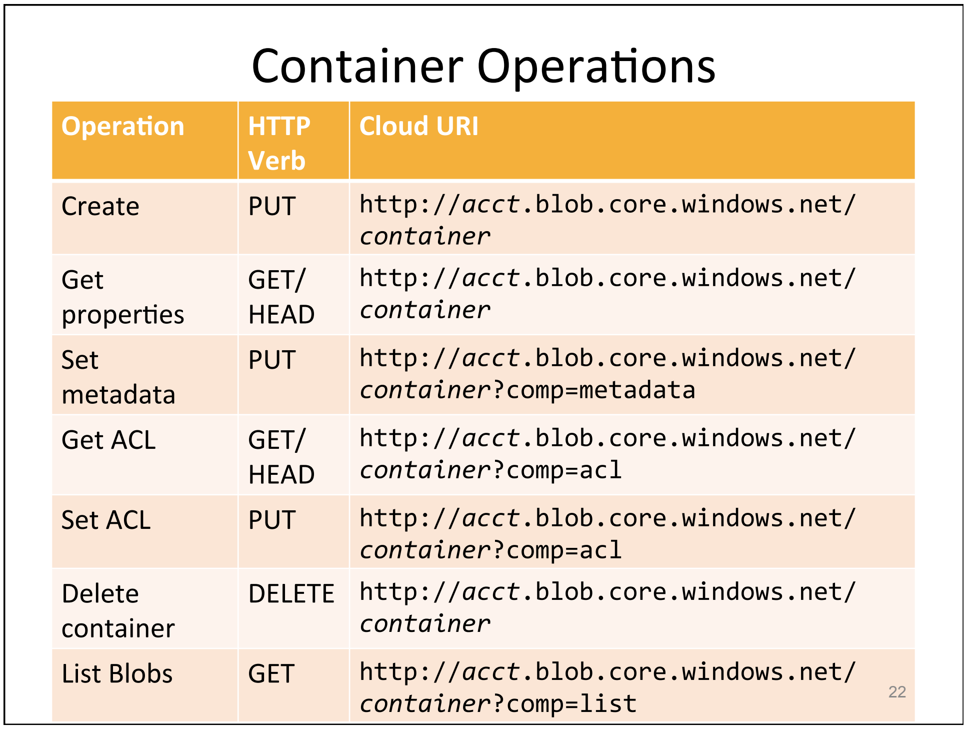
Account Operations

Operation	HTTP Verb	Cloud URI
List containers	GET	http://*acct*.blob.core.windows.net?comp=list



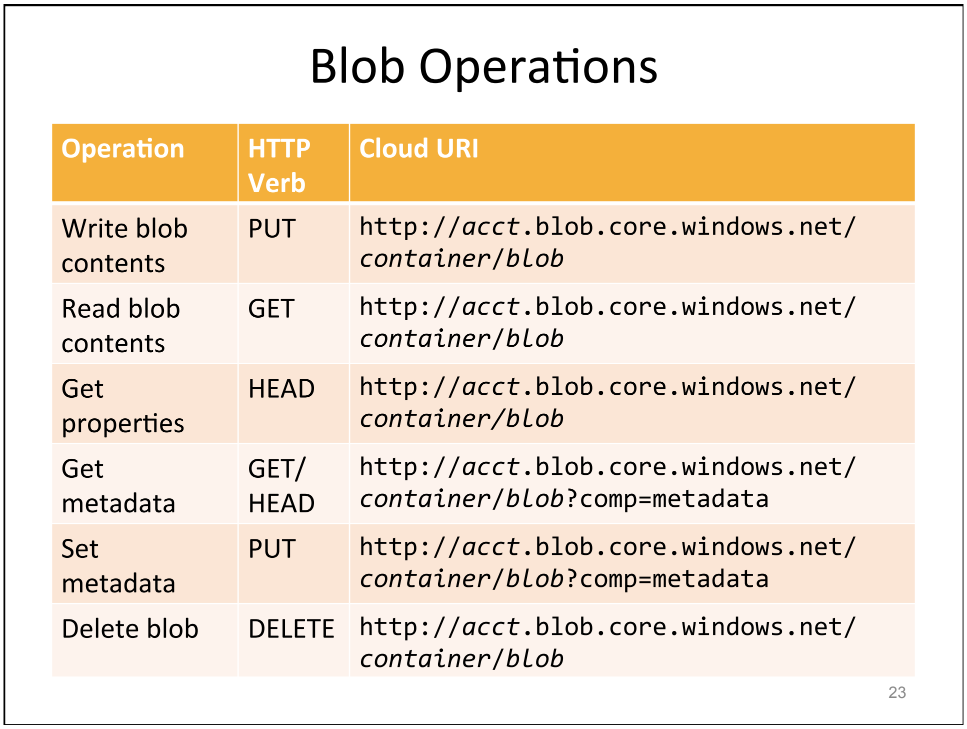
Properties vs Metadata

- System properties
  - Container: read-only
  - Blob: read-only and read-write
  - Ex: Etag, LastModifiedUTC, ContentEncoding, ContentType, ContentLanguage, etc
  - *blob*.SetProperties()
  - *blob*.FetchAttributes()
  - *blob*.Properties.*propName*
- User-defined metadata
  - *blob*.SetMetadata()
  - *blob*.FetchAttributes()
  - *blob*.Metadata[*key*]



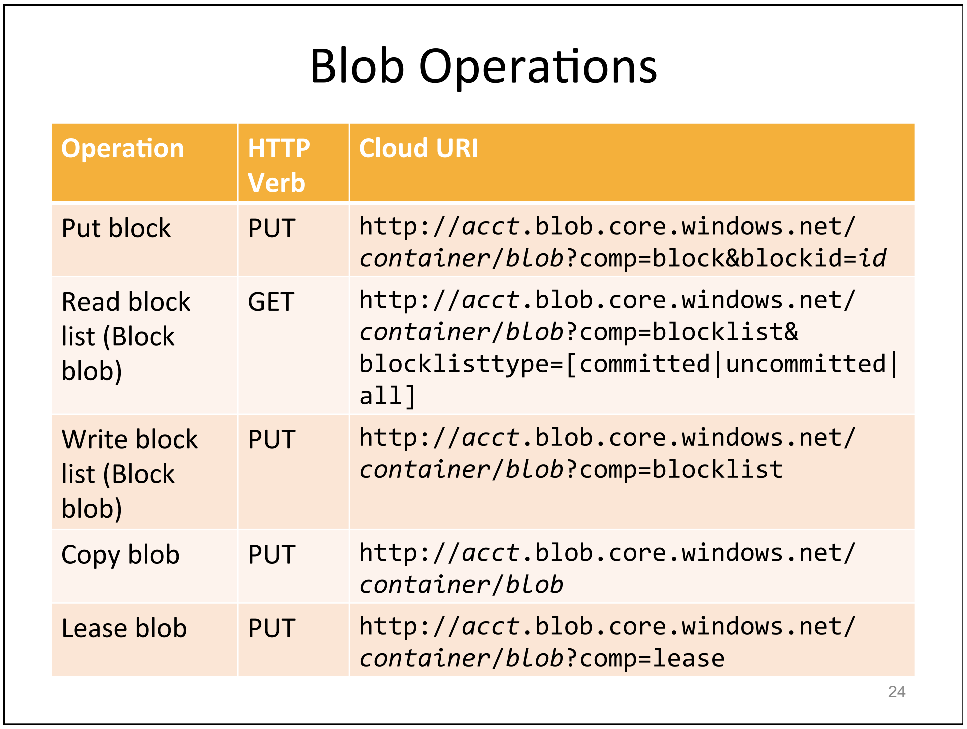
Container Operations

Operation	HTTP Verb	Cloud URI
Create	PUT	http://*acct*.blob.core.windows.net/*container*
Get properties	GET/HEAD	http://*acct*.blob.core.windows.net/*container*
Set metadata	PUT	http://*acct*.blob.core.windows.net/*container*?comp=metadata
Get ACL	GET/HEAD	http://*acct*.blob.core.windows.net/*container*?comp=acl
Set ACL	PUT	http://*acct*.blob.core.windows.net/*container*?comp=acl
Delete container	DELETE	http://*acct*.blob.core.windows.net/*container*
List Blobs	GET	http://*acct*.blob.core.windows.net/*container*?comp=list



Blob Operations

Operation	HTTP Verb	Cloud URI
Write blob contents	PUT	http://*acct*.blob.core.windows.net/*container*/*blob*
Read blob contents	GET	http://*acct*.blob.core.windows.net/*container*/*blob*
Get properties	HEAD	http://*acct*.blob.core.windows.net/*container*/*blob*
Get metadata	GET/ HEAD	http://*acct*.blob.core.windows.net/*container*/*blob*?comp=metadata
Set metadata	PUT	http://*acct*.blob.core.windows.net/*container*/*blob*?comp=metadata
Delete blob	DELETE	http://*acct*.blob.core.windows.net/*container*/*blob*



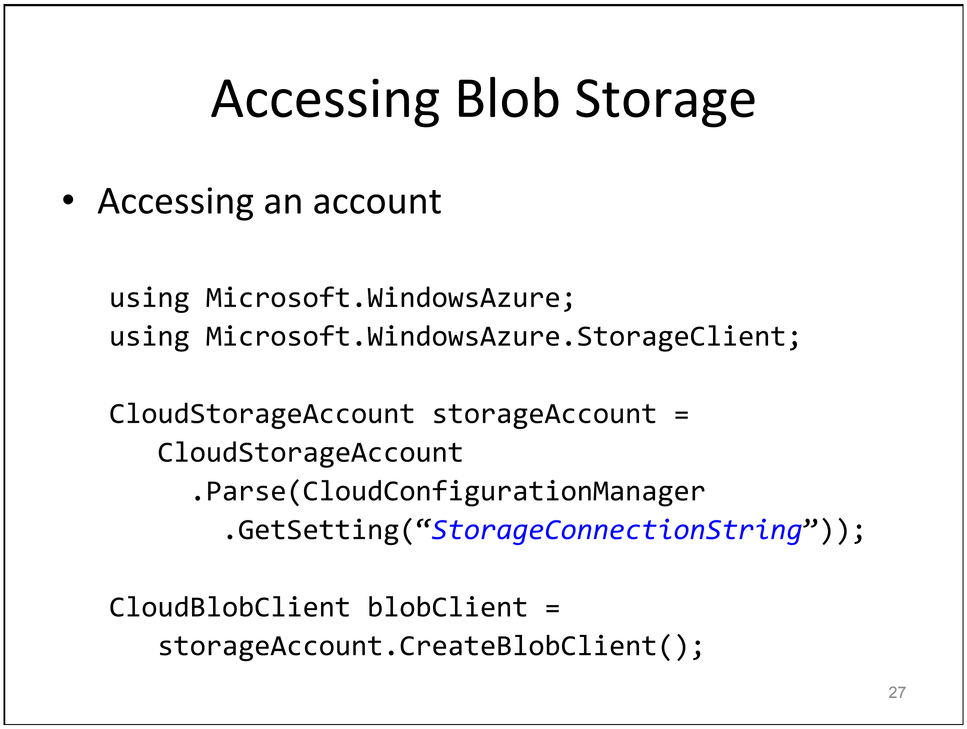
Blob Operations

Operation	HTTP Verb	Cloud URI
Put block	PUT	http://*acct*.blob.core.windows.net/*container*/*blob*?comp=block&blockid=*id*
Read block list (Block blob)	GET	http://*acct*.blob.core.windows.net/*container*/*blob*?comp=blocklist&blocklisttype=[committed|uncommitted|all]
Write block list (Block blob)	PUT	http://*acct*.blob.core.windows.net/*container*/*blob*?comp=blocklist
Copy blob	PUT	http://*acct*.blob.core.windows.net/*container*/*blob*
Lease blob	PUT	http://*acct*.blob.core.windows.net/*container*/*blob*?comp=lease

Blob Operations

Operation	HTTP Verb	Cloud URI
Snapshot	PUT	http://*acct*.blob.core.windows.net/ *container*/*blob*?comp=snapshot
Write page (Page blob)	PUT	http://*acct*.blob.core.windows.net/ *container*/*blob*?comp=page
Read page regions (Page blob)	GET	http://*acct*.blob.core.windows.net/ *container*/*blob*?comp=pagelist http://*acct*.blob.core.windows.net/ *container*/*blob*? comp=pagelist&snapshot=*datetime*

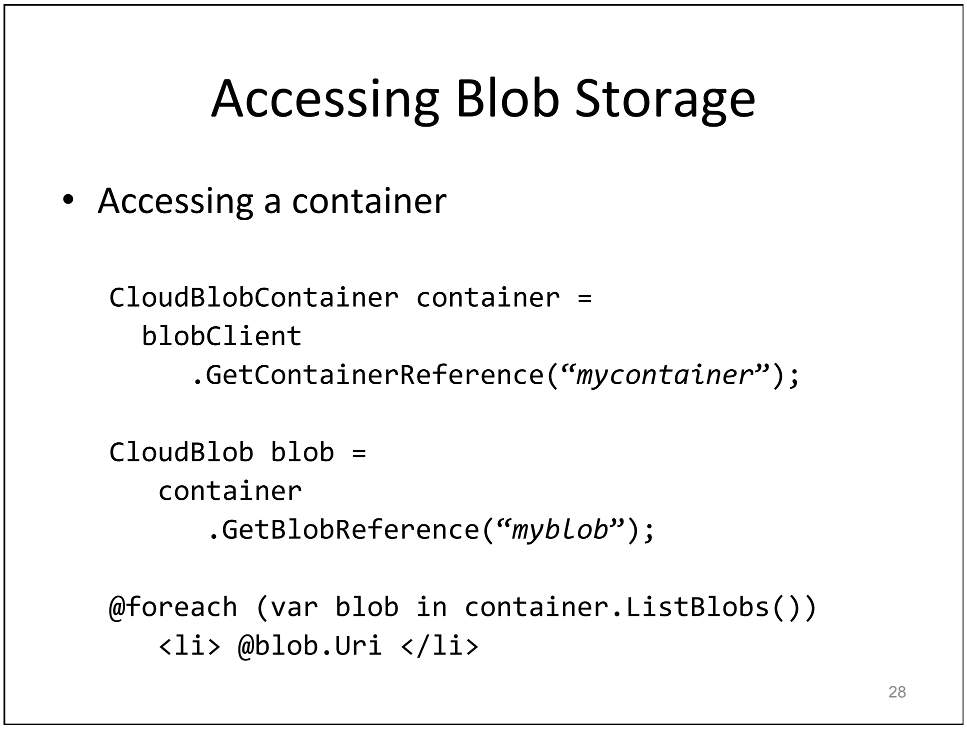
BLOB SERVICE: STORAGE CLIENT API



Accessing Blob Storage

- Accessing an account

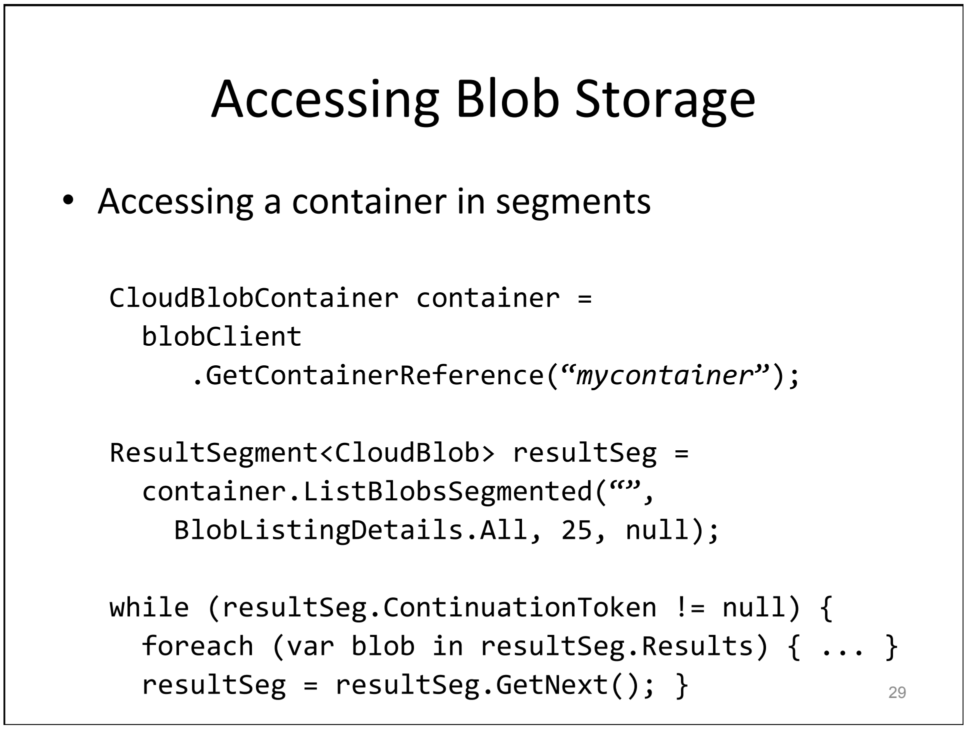
```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;

CloudStorageAccount storageAccount =
   CloudStorageAccount
     .Parse(CloudConfigurationManager
       .GetSetting(“StorageConnectionString”));

CloudBlobClient blobClient =
   storageAccount.CreateBlobClient();
```



Accessing Blob Storage

- Accessing a container

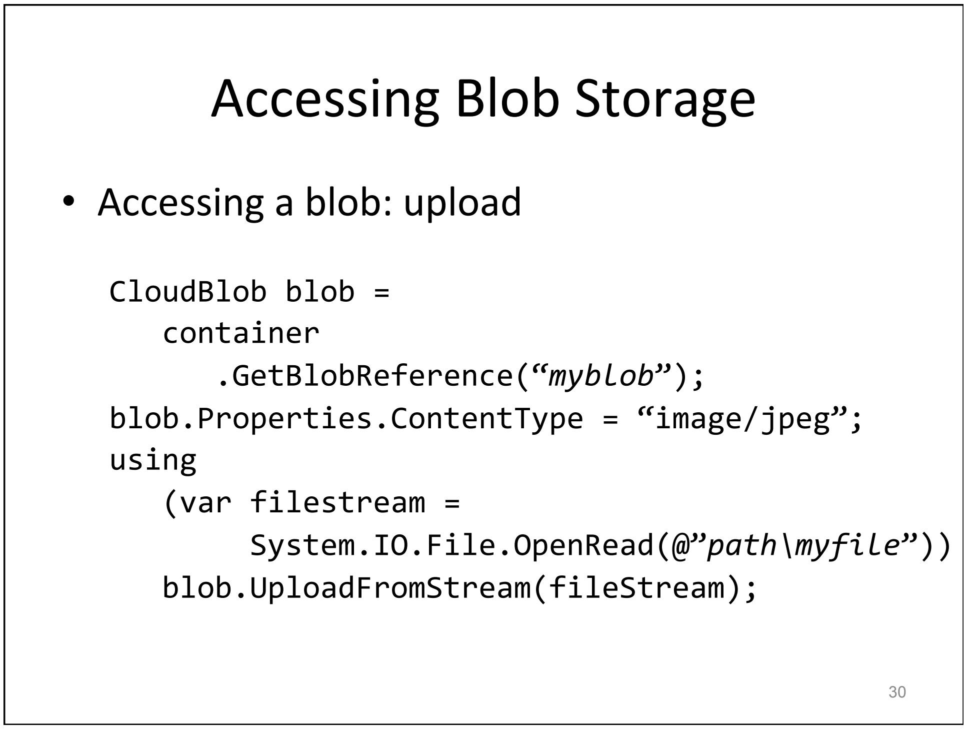
```
CloudBlobContainer container =
  blobClient
     .GetContainerReference(“mycontainer”);

CloudBlob blob =
   container
      .GetBlobReference(“myblob”);

@foreach (var blob in container.ListBlobs())
   <li> @blob.Uri </li>
```



Accessing Blob Storage

- Accessing a container in segments

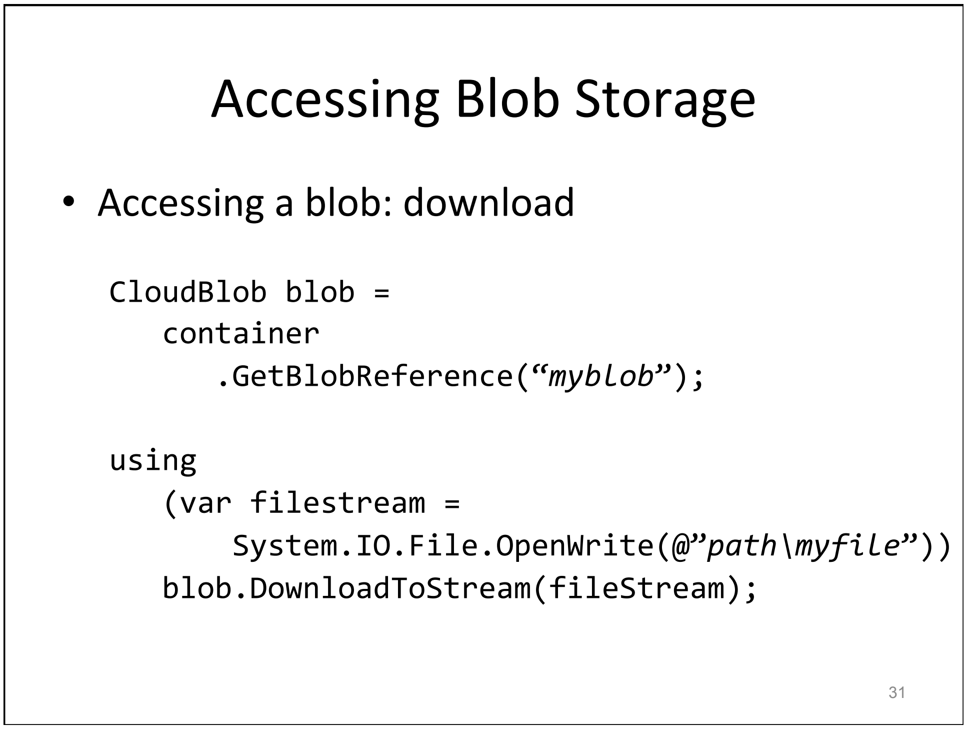
```
CloudBlobContainer container =
  blobClient
     .GetContainerReference(“mycontainer”);

ResultSegment<CloudBlob> resultSeg =
  container.ListBlobsSegmented(“”,
    BlobListingDetails.All, 25, null);

while (resultSeg.ContinuationToken != null) {
  foreach (var blob in resultSeg.Results) { ... }
  resultSeg = resultSeg.GetNext(); }
```



Accessing Blob Storage

- Accessing a blob: upload

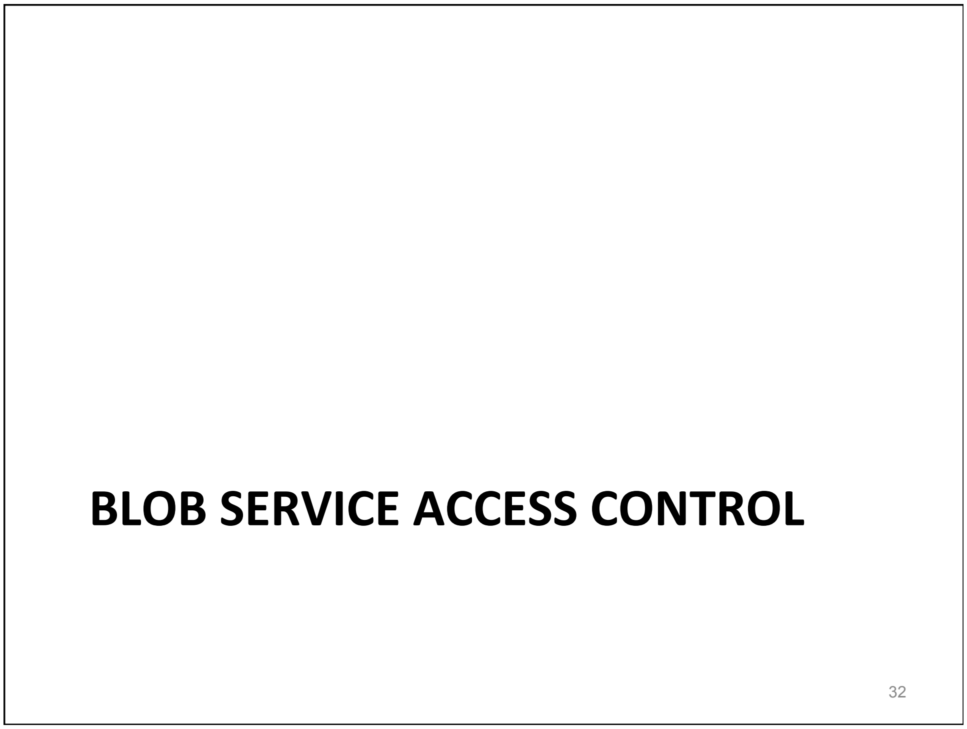
```
CloudBlob blob =
   container
      .GetBlobReference(“myblob”);
blob.Properties.ContentType = “image/jpeg”;
using
   (var filestream =
        System.IO.File.OpenRead(@”path\myfile”))
   blob.UploadFromStream(fileStream);
```



Accessing Blob Storage

- Accessing a blob: download

```
CloudBlob blob =
   container
      .GetBlobReference(“myblob”);

using
   (var filestream =
       System.IO.File.OpenWrite(@”path\myfile”))
   blob.DownloadToStream(fileStream);
```



BLOB SERVICE ACCESS CONTROL

Types of Access

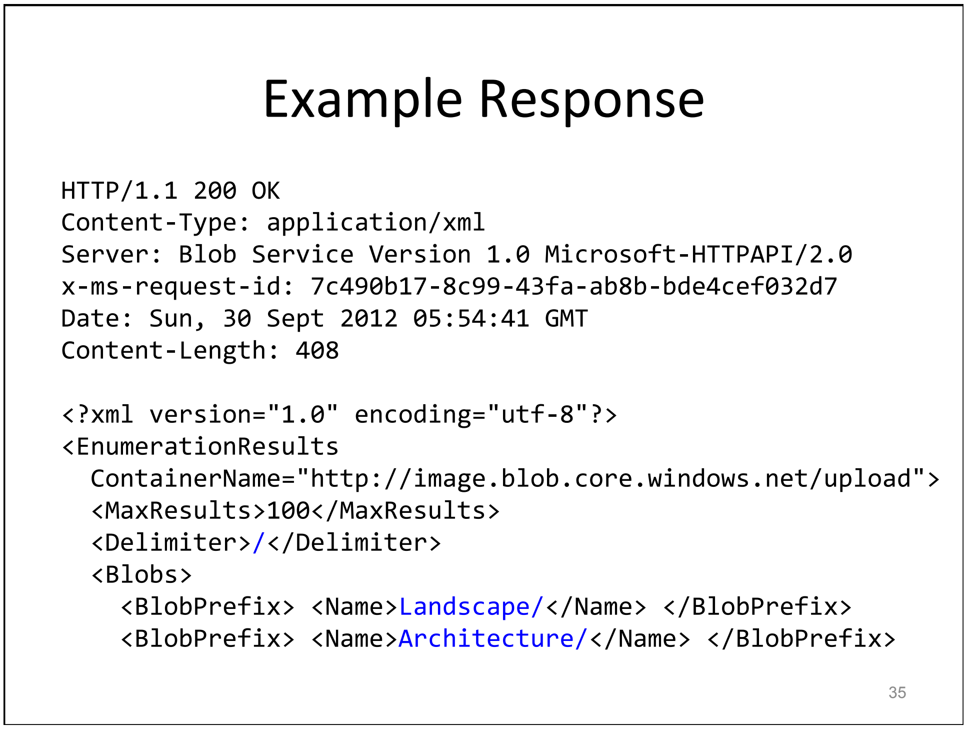
- Full public read access
  - Containers enumerable
- Public read access for blobs only
- Private
  - Only accessible to account owner
  - Authenticated with SHA256 HMAC

```
Authorization: [SharedKey|SharedKeyLite]
                account-name:signature
```

Example Request

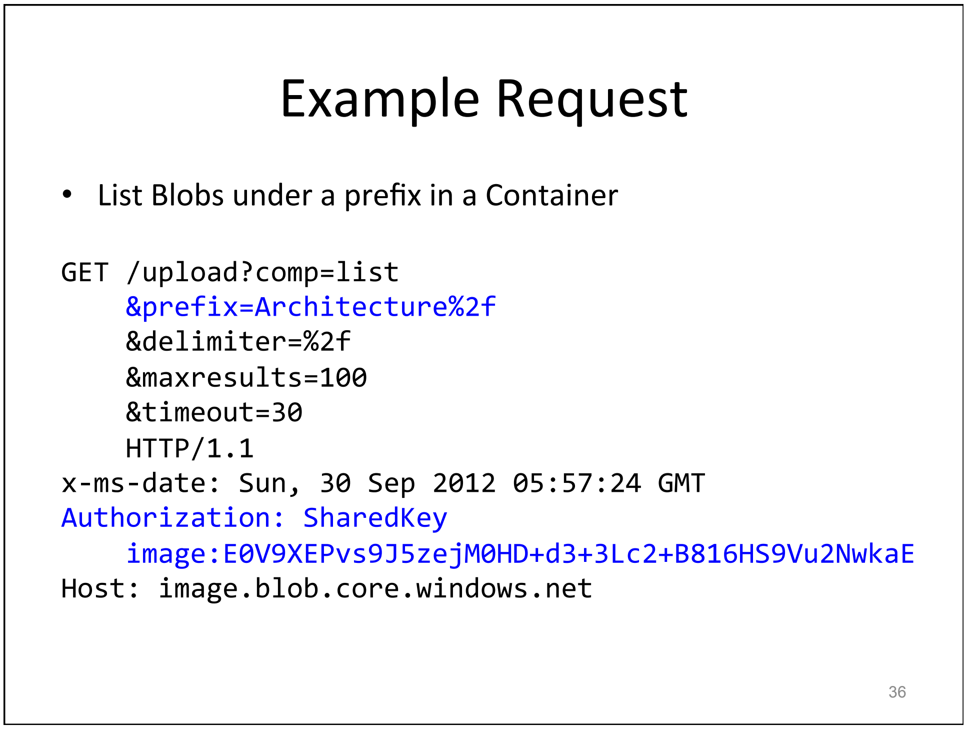
- List Prefixes in a Container

```
GET /upload?comp=list
    &delimiter=%2f
    &maxresults=100
    &timeout=30
    HTTP/1.1
x-ms-date: Sun, 30 Sep 2012 05:53:37 GMT
Authorization: SharedKey
    image:7euawYh5wNOGFJZGnvrn9vyR4y
Host: image.blob.core.windows.net
```



Example Response

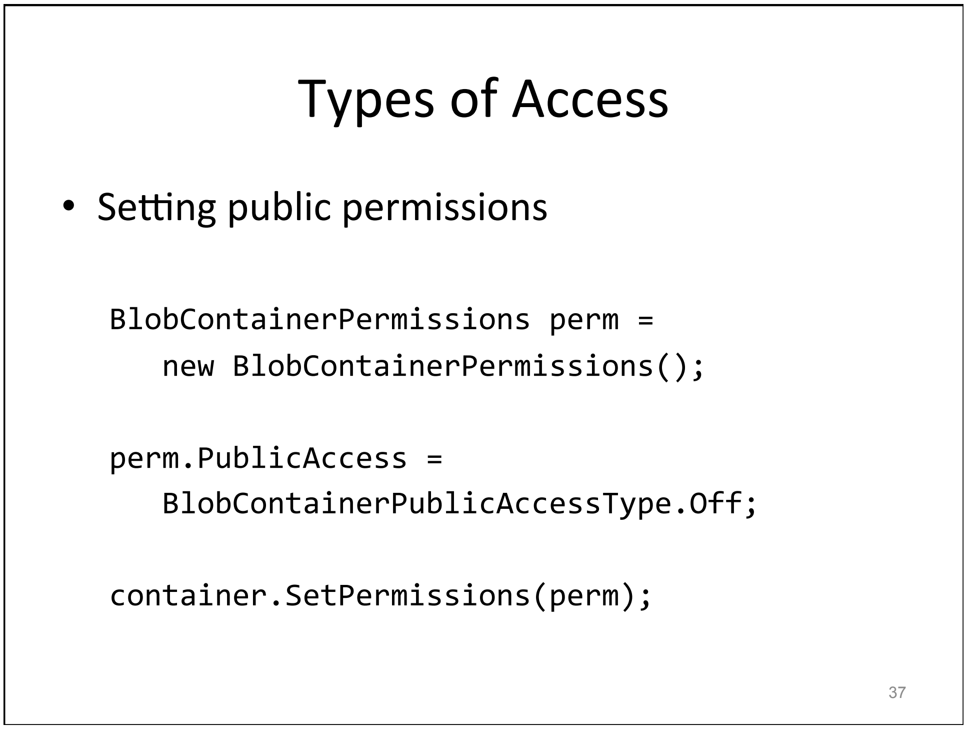
```
HTTP/1.1 200 OK
Content-Type: application/xml
Server: Blob Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: 7c490b17-8c99-43fa-ab8b-bde4cef032d7
Date: Sun, 30 Sept 2012 05:54:41 GMT
Content-Length: 408

<?xml version="1.0" encoding="utf-8"?>
<EnumerationResults
  ContainerName="http://image.blob.core.windows.net/upload">
  <MaxResults>100</MaxResults>
  <Delimiter>/</Delimiter>
  <Blobs>
    <BlobPrefix> <Name>Landscape/</Name> </BlobPrefix>
    <BlobPrefix> <Name>Architecture/</Name> </BlobPrefix>
```



Example Request

- List Blobs under a prefix in a Container

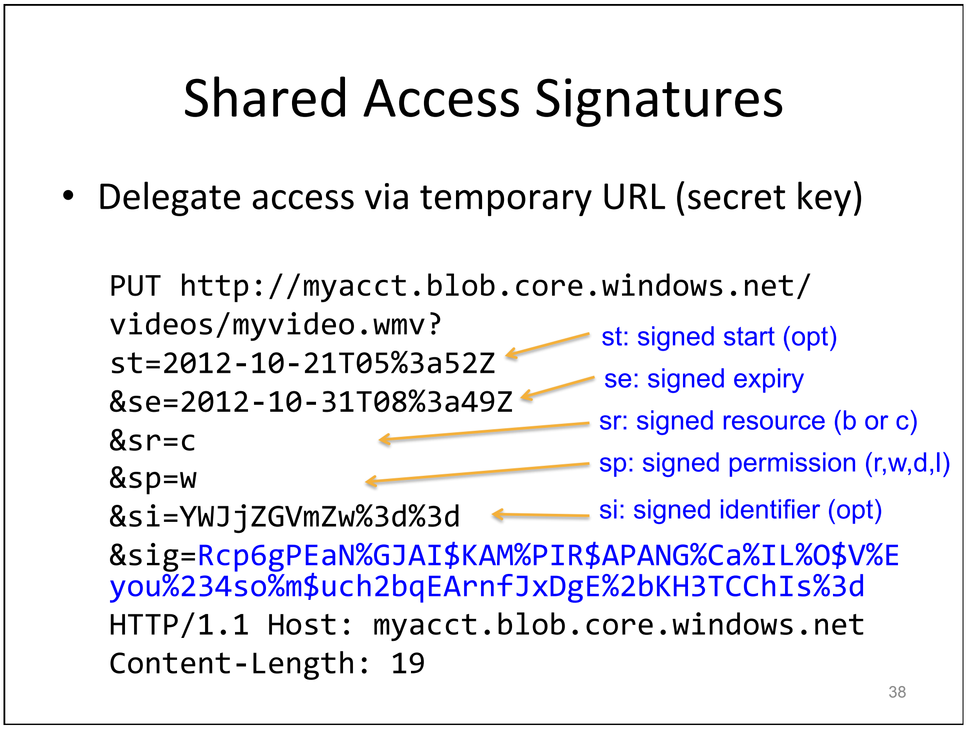
```
GET /upload?comp=list
    &prefix=Architecture%2f
    &delimiter=%2f
    &maxresults=100
    &timeout=30
    HTTP/1.1
x-ms-date: Sun, 30 Sep 2012 05:57:24 GMT
Authorization: SharedKey
    image:E0V9XEPvs9J5zejM0HD+d3+3Lc2+B816HS9Vu2NwkaE
Host: image.blob.core.windows.net
```



Types of Access

- Setting public permissions

```
BlobContainerPermissions perm =
   new BlobContainerPermissions();

perm.PublicAccess =
   BlobContainerPublicAccessType.Off;

container.SetPermissions(perm);
```



Shared Access Signatures

- Delegate access via temporary URL (secret key)

```
PUT http://myacct.blob.core.windows.net/
videos/myvideo.wmv?
st=2012-10-21T05%3a52Z
&se=2012-10-31T08%3a49Z
&sr=c
&sp=w
&si=YWJjZGVmZw%3d%3d
&sig=Rcp6gPEaN%GJAI$KAM%PIR$APANG%Ca%IL%O$V%E
you%234so%m$uch2bqEArnfJxDgE%2bKH3TCChIs%3d
HTTP/1.1 Host: myacct.blob.core.windows.net
Content-Length: 19
```

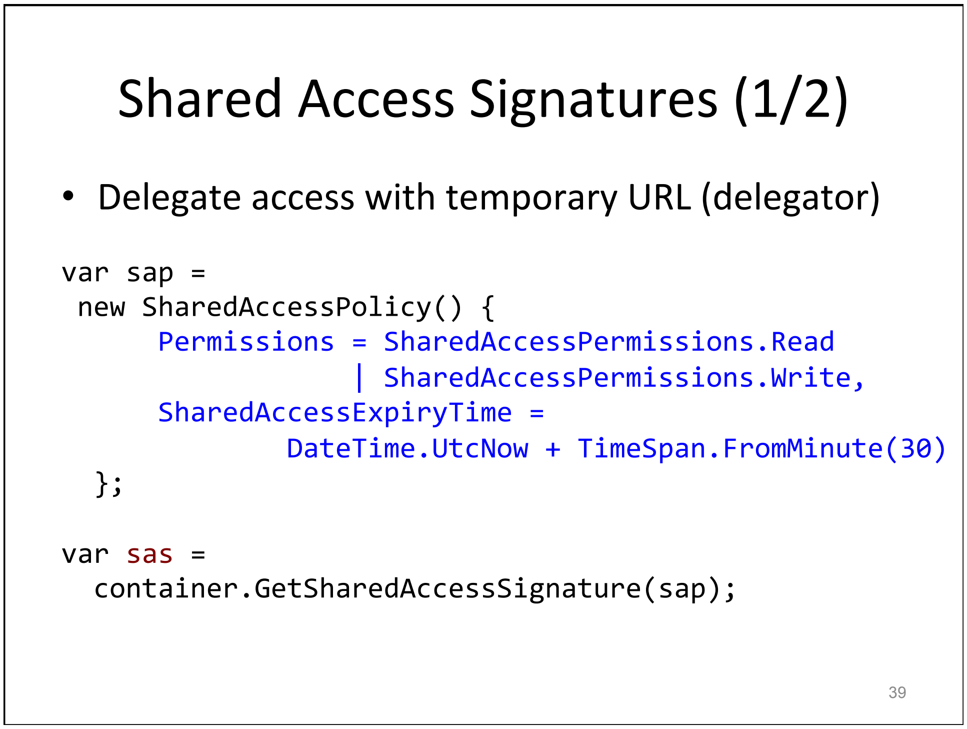
st: signed start (opt)

se: signed expiry

sr: signed resource (b or c)

sp: signed permission (r,w,d,l)

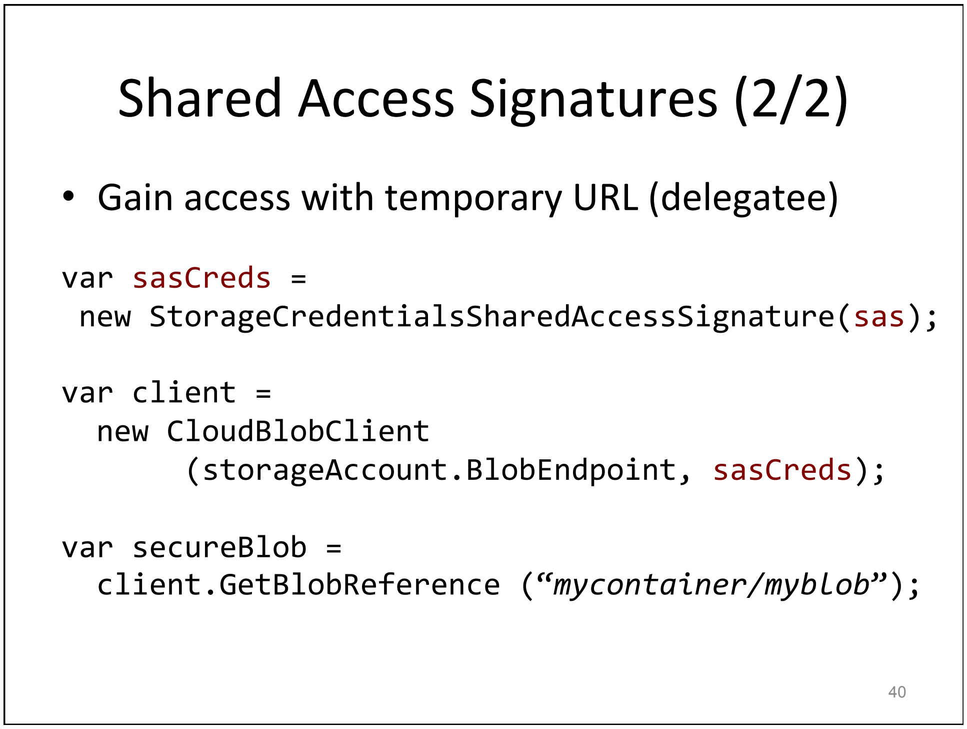
si: signed identifier (opt)



Shared Access Signatures (1/2)

- Delegate access with temporary URL (delegator)

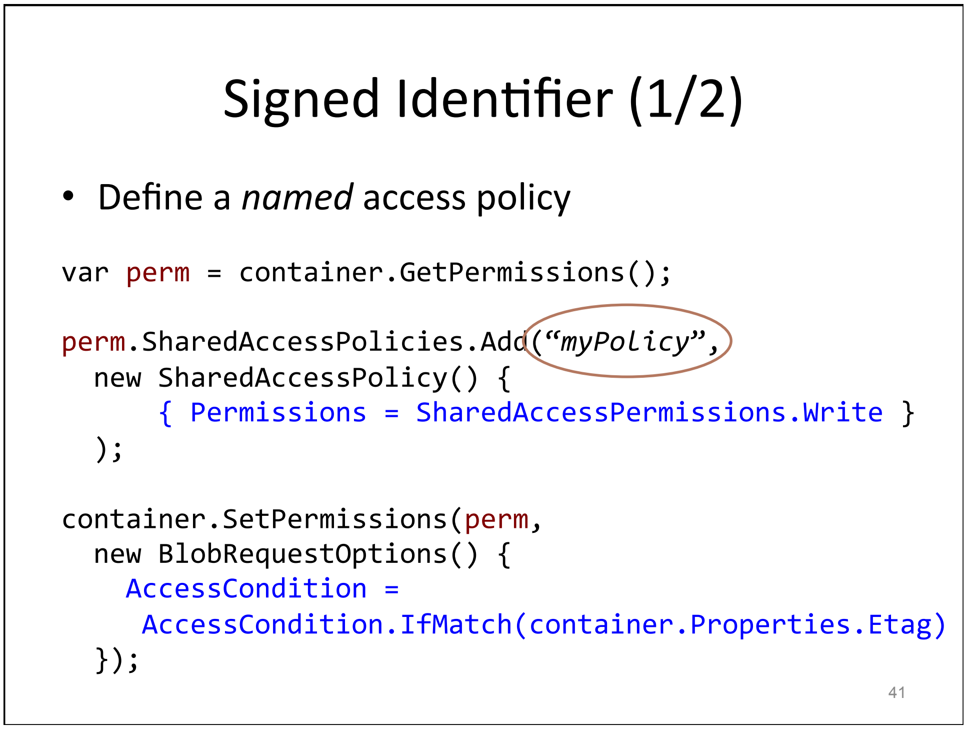
```
var sap =
 new SharedAccessPolicy() {
      Permissions = SharedAccessPermissions.Read
                  | SharedAccessPermissions.Write,
      SharedAccessExpiryTime =
              DateTime.UtcNow + TimeSpan.FromMinute(30)
  };

var sas =
  container.GetSharedAccessSignature(sap);
```



Shared Access Signatures (2/2)

- Gain access with temporary URL (delegatee)

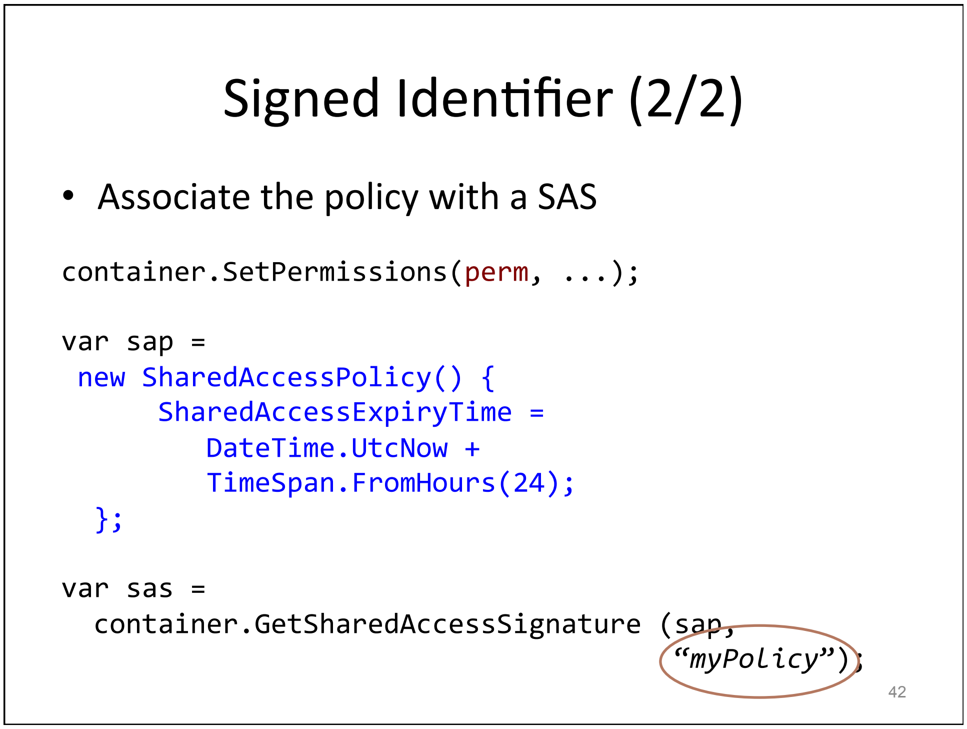
```
var sasCreds =
 new StorageCredentialsSharedAccessSignature(sas);

var client =
  new CloudBlobClient
       (storageAccount.BlobEndpoint, sasCreds);

var secureBlob =
  client.GetBlobReference (“mycontainer/myblob”);
```



Signed Identifier (1/2)

- Define a *named* access policy

```
var perm = container.GetPermissions();

perm.SharedAccessPolicies.Add(“myPolicy”,
  new SharedAccessPolicy() {
      { Permissions = SharedAccessPermissions.Write }
  );

container.SetPermissions(perm,
  new BlobRequestOptions() {
    AccessCondition =
     AccessCondition.IfMatch(container.Properties.Etag)
  });
```



Signed Identifier (2/2)

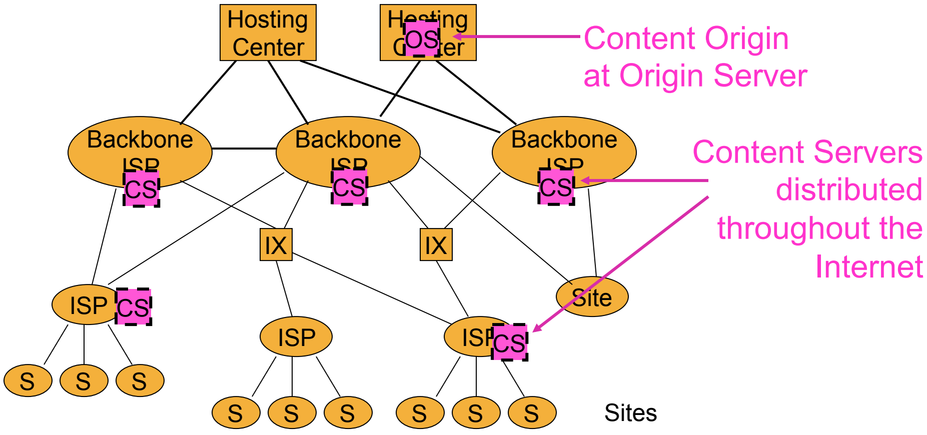
- Associate the policy with a SAS

```
container.SetPermissions(perm, ...);

var sap =
 new SharedAccessPolicy() {
      SharedAccessExpiryTime =
         DateTime.UtcNow +
         TimeSpan.FromHours(24);
  };

var sas =
  container.GetSharedAccessSignature (sap,
                                       “myPolicy”);
```

CONTENT DISTRIBUTION NETWORKS

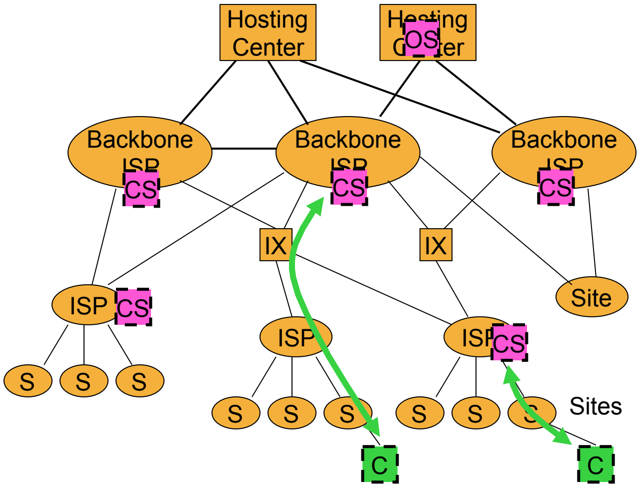
Content Routing Principle

(a.k.a. Content Distribution Network)



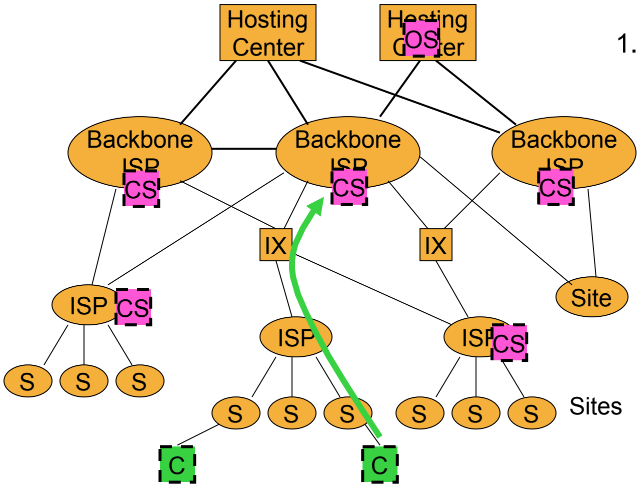
Content Routing Principle

(a.k.a. Content Distribution Network)



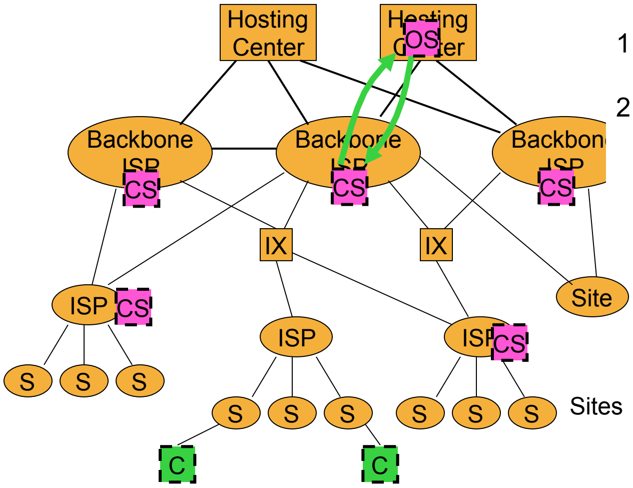
Content is served from content servers nearer to the client

Cached CDN



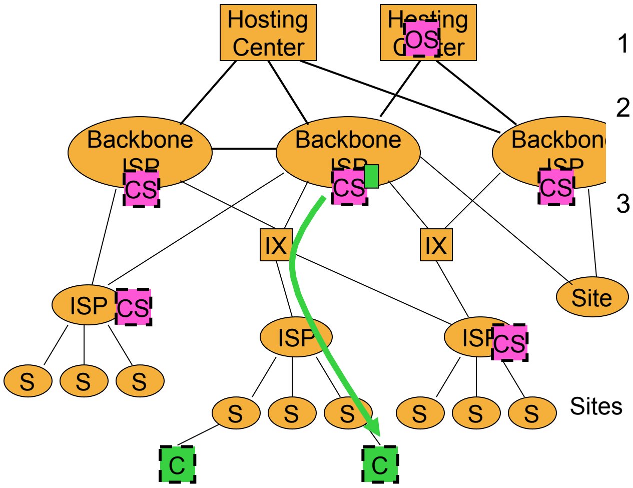
1. Client requests content.

Cached CDN



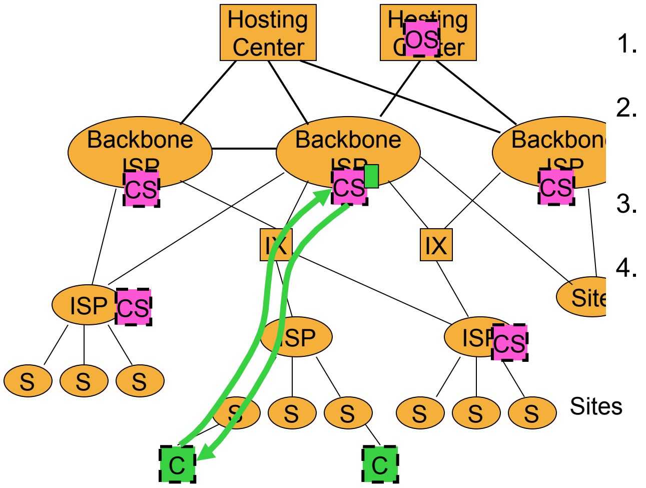
1. Client requests content.
2. CS checks cache, if miss gets content from origin server.

Cached CDN



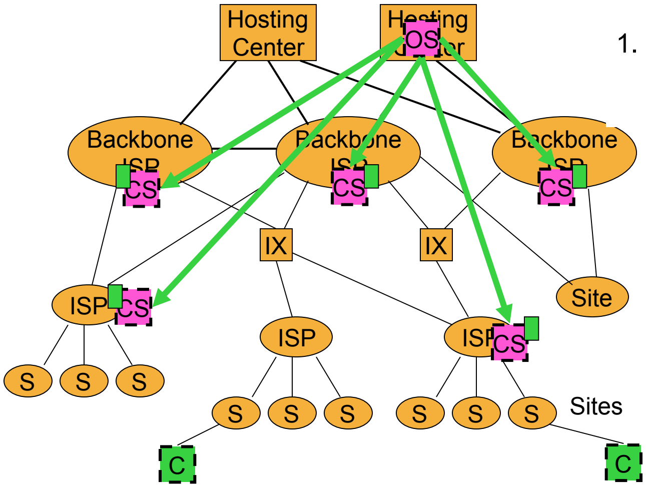
1. Client requests content.
2. CS checks cache, if miss gets content from origin server.
3. CS caches content, delivers to client.

Cached CDN



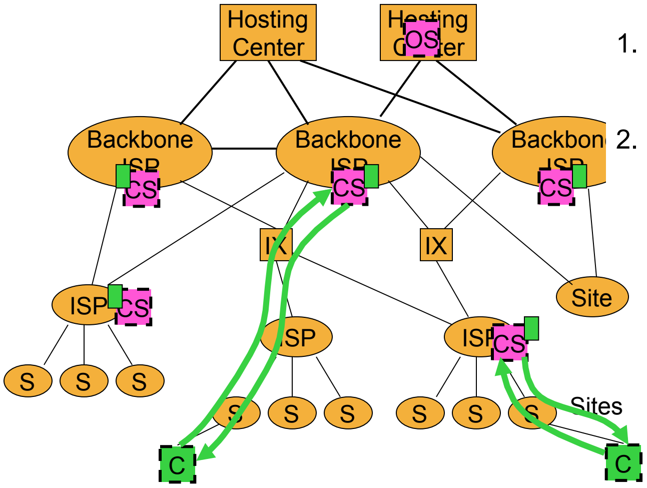
1. Client requests content.
2. CS checks cache, if miss gets content from origin server.
3. CS caches content, delivers to client.
4. Delivers content out of cache on subsequent requests.

Pushed CDN



1. Origin Server pushes content out to all CSs.

Pushed CDN



1. Origin Server pushes content out to all CSs.
2. Request served from CSs.

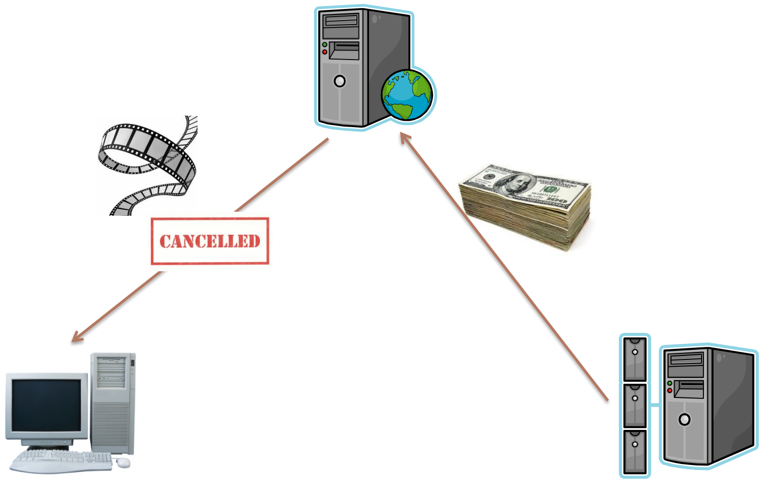
CDN benefits

- Content served closer to client
  - Less latency, better performance
- Load spread over multiple distributed CSs
  - More robust (to ISP failure)
  - Flash loads spread over ISPs

CDN challenges

- Managing content distribution
  - Content lifetimes vs cache performance
  - Content synchronized and current
- Streaming content
  - Streaming vs file download?
  - Avoid proprietary protocols
- Live content
  - Television over IP

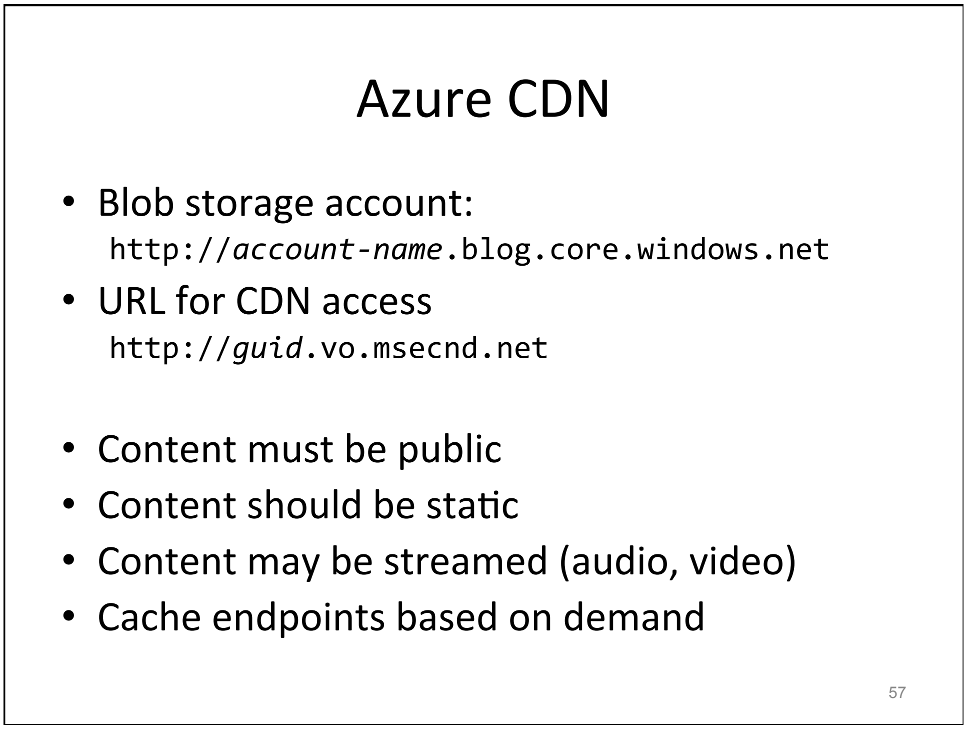
Streaming Content



Streaming Content

- Issue: Billing for content viewed
- Required:
  - Progressive download
    - e.g. Youtube
  - Throttling
- Solutions
  - Streaming HTTP
  - Adaptive Bit Rate (ABR) streaming

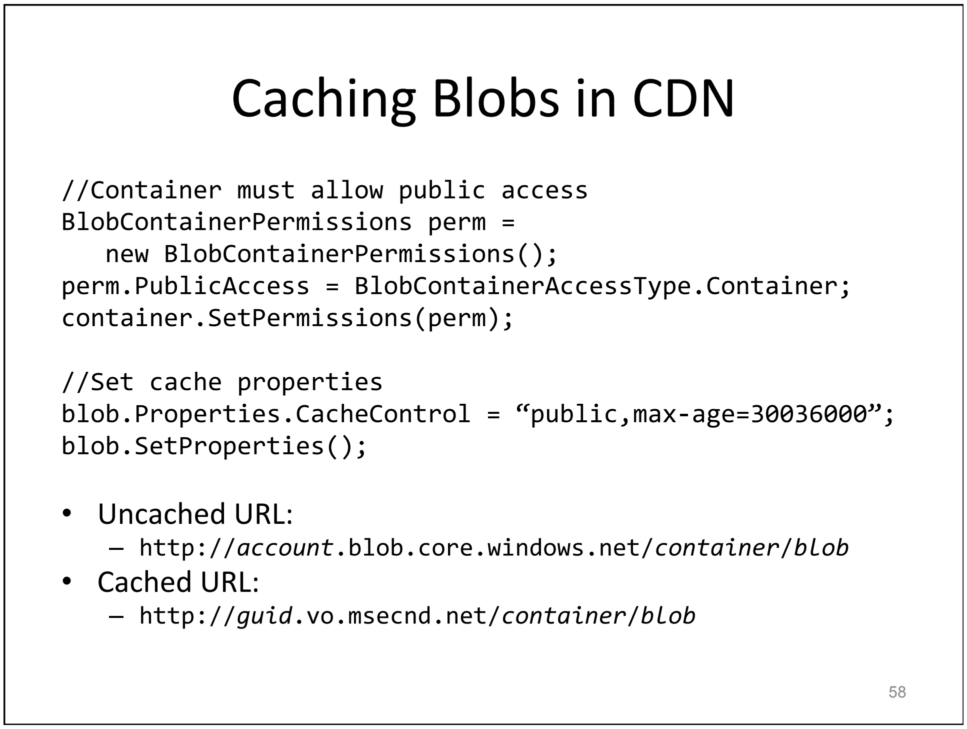
AZURE CDN



Azure CDN

- Blob storage account:
  `http://`*`account-name`*`.blog.core.windows.net`
- URL for CDN access
  `http://`*`guid`*`.vo.msecnd.net`

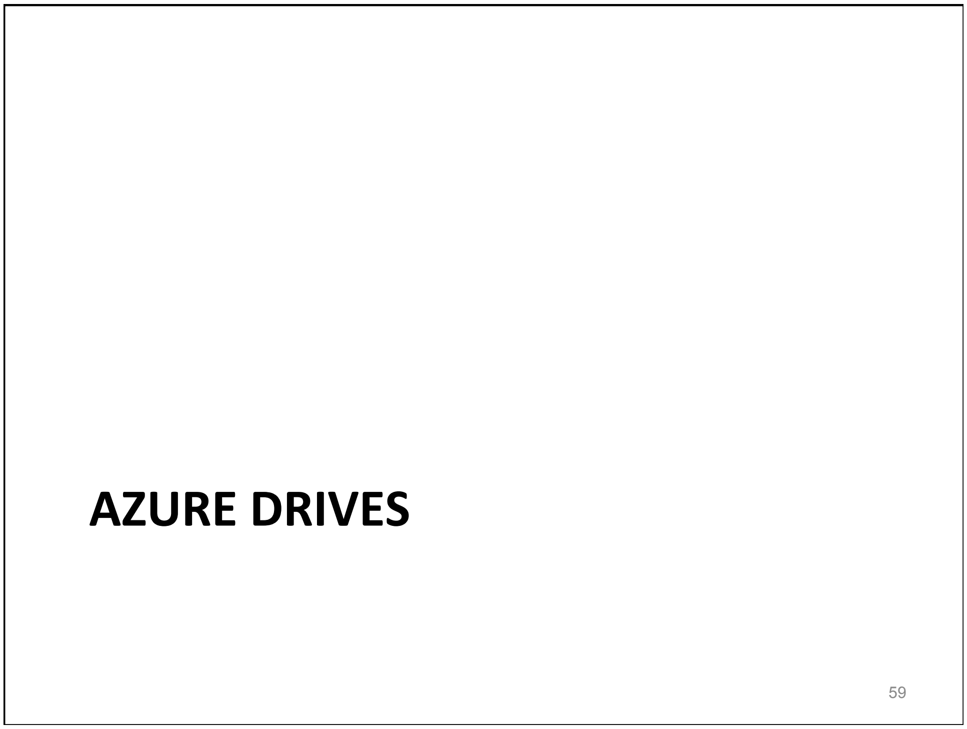
- Content must be public
- Content should be static
- Content may be streamed (audio, video)
- Cache endpoints based on demand



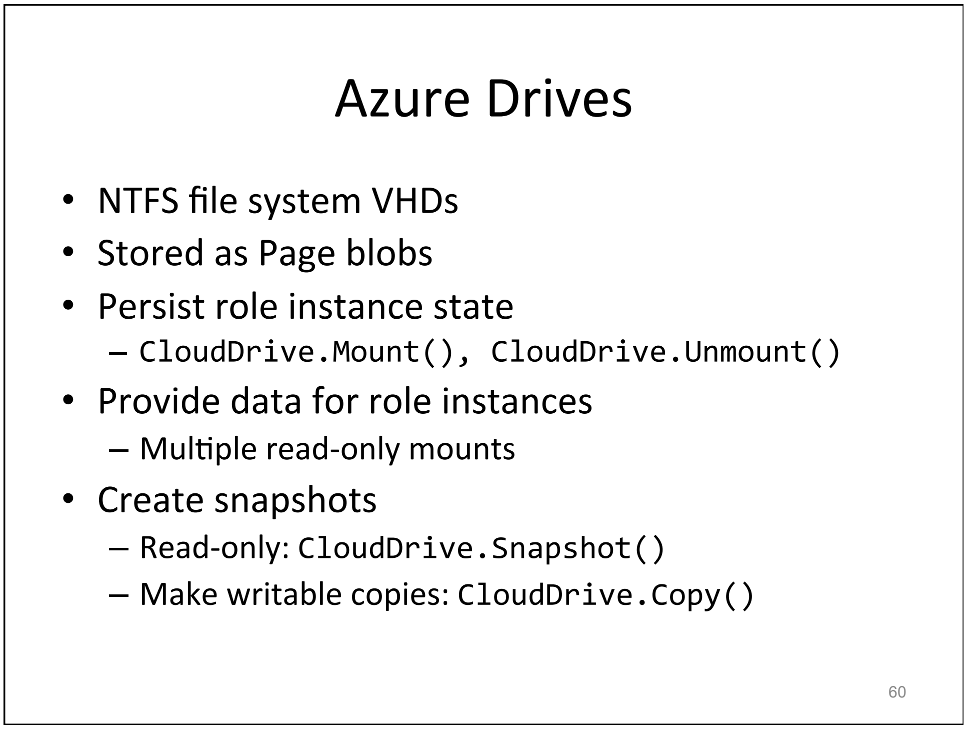
Caching Blobs in CDN

```
//Container must allow public access
BlobContainerPermissions perm =
   new BlobContainerPermissions();
perm.PublicAccess = BlobContainerAccessType.Container;
container.SetPermissions(perm);

//Set cache properties
blob.Properties.CacheControl = “public,max-age=30036000”;
blob.SetProperties();
```

- Uncached URL:
  – `http://`*`account`*`.blob.core.windows.net/`*`container`*`/`*`blob`*
- Cached URL:
  – `http://`*`guid`*`.vo.msecnd.net/`*`container`*`/`*`blob`*

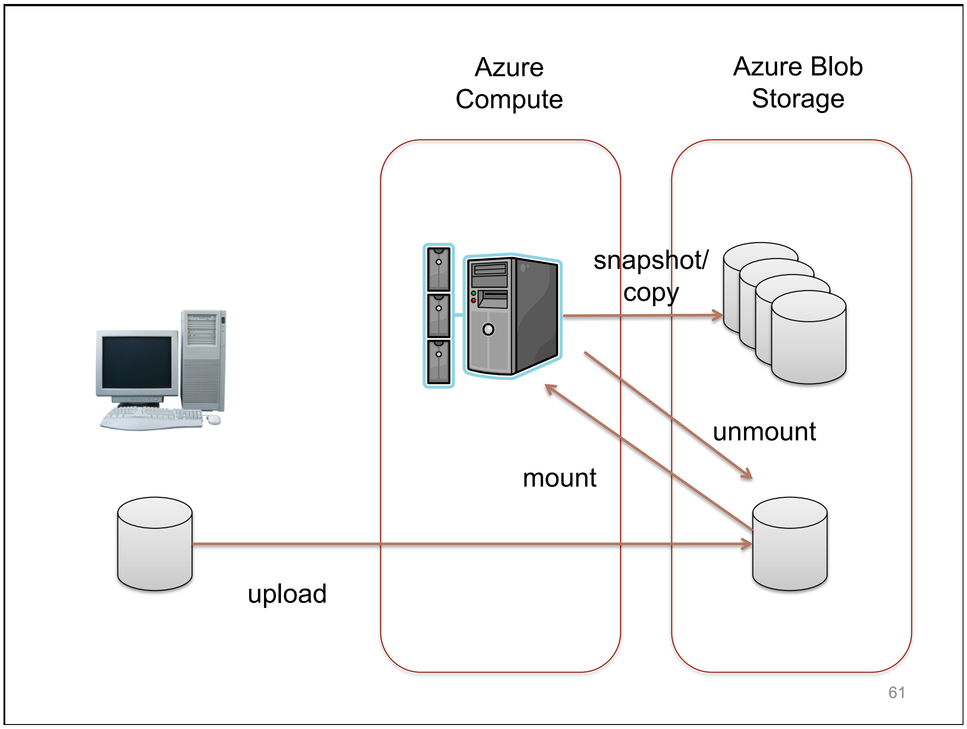


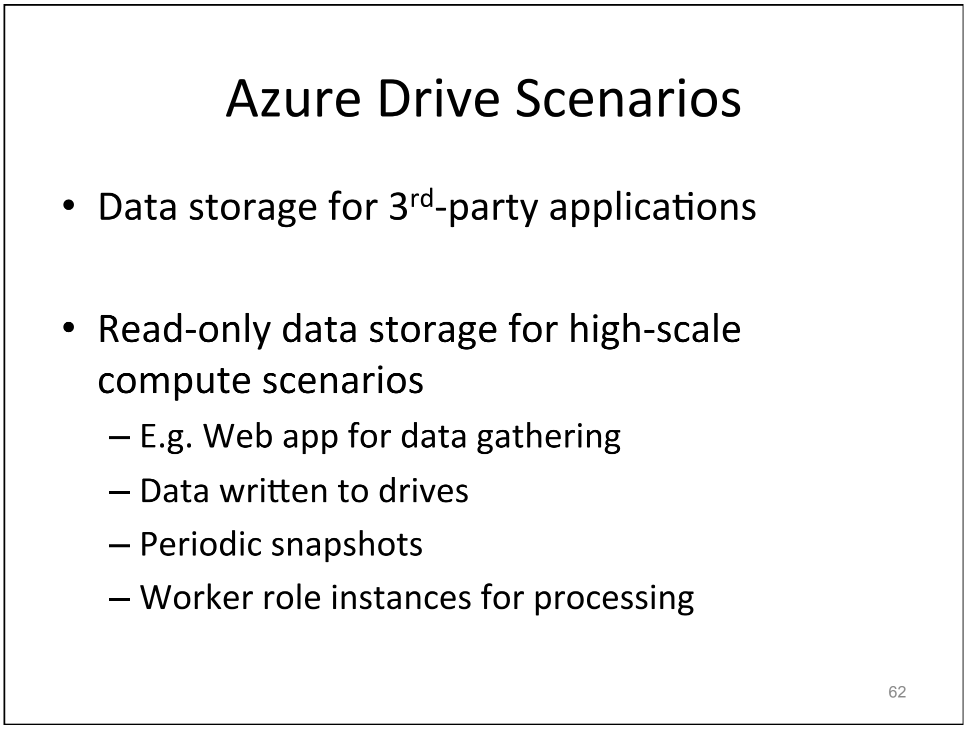
AZURE DRIVES



Azure Drives

- NTFS file system VHDs
- Stored as Page blobs
- Persist role instance state
  – `CloudDrive.Mount(), CloudDrive.Unmount()`
- Provide data for role instances
  – Multiple read-only mounts
- Create snapshots
  – Read-only: `CloudDrive.Snapshot()`
  – Make writable copies: `CloudDrive.Copy()`

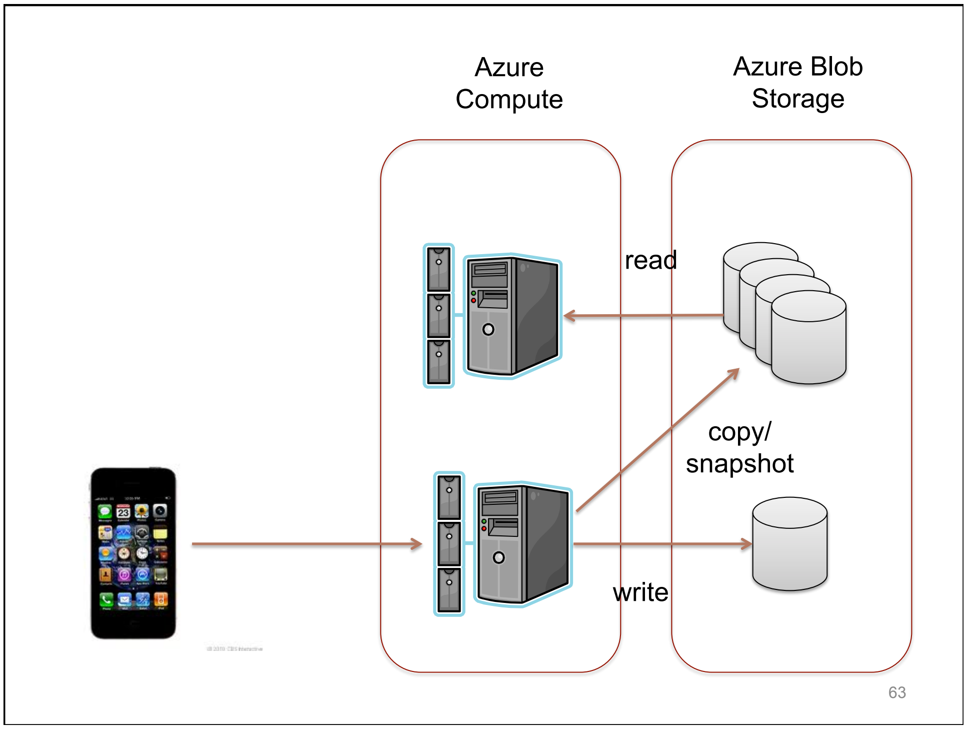




Azure Drive Scenarios

- Data storage for 3rd-party applications
- Read-only data storage for high-scale compute scenarios
  - E.g. Web app for data gathering
  - Data written to drives
  - Periodic snapshots
  - Worker role instances for processing

62

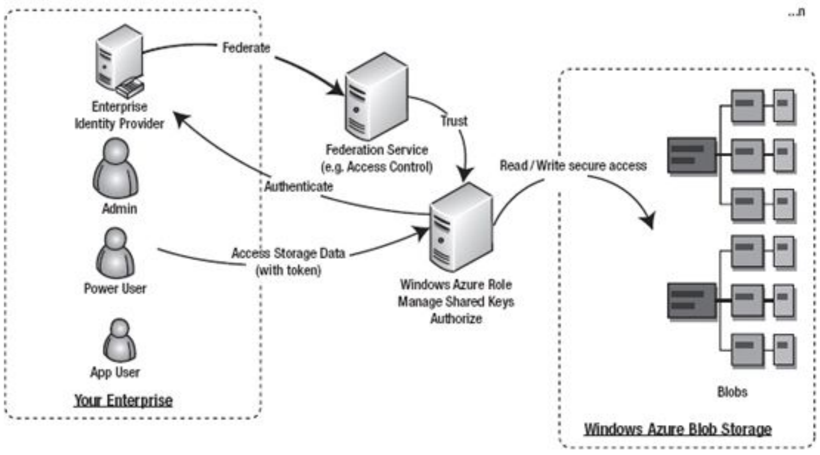


BLOB USAGE SCENARIOS

Blob Storage Scenarios

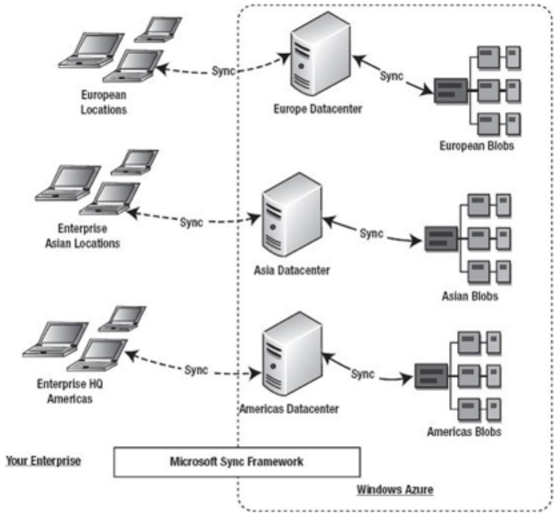
- Massive Data Uploads
  - Courier disks?
- Storage as a Service in the Cloud
  - Integrate with enterprise identity management
- Enterprise File Sync
  - Automatic backup

Storage as a Service


...n
Federate
Enterprise Identity Provider
Trust
Federation Service (e.g. Access Control)
Read / Write secure access
Authenticate
Admin
Access Storage Data (with token)
Power User
Windows Azure Role
Manage Shared Keys
Authorize
App User
Your Enterprise
Blobs
Windows Azure Blob Storage

© Redkar & Guidici

Enterprise Sync



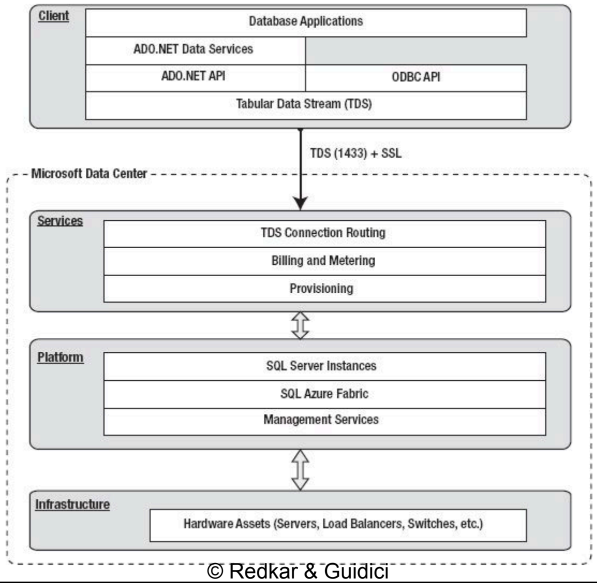
© Redkar & Guidici

Other Scenarios

- Media Streaming
- Disaster Recovery Data Repository
- Application Store Repository
- Storing Mobile Apps and Data

SQL DATABASE

Architecture



Client

Database Applications

ADO.NET Data Services

ADO.NET API

ODBC API

Tabular Data Stream (TDS)

TDS (1433) + SSL

Microsoft Data Center

Services

TDS Connection Routing

Billing and Metering

Provisioning

Platform

SQL Server Instances

SQL Azure Fabric

Management Services

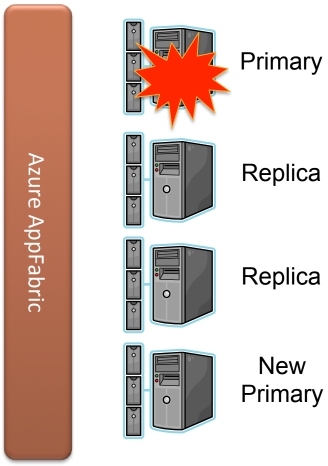
Infrastructure

Hardware Assets (Servers, Load Balancers, Switches, etc.)

© Redkar & Guidici

Platform Layer

- SQL server instances
  - Deployed databases
  - Replicas
  - OS instances
- Azure fabric
  - Replica set-up
  - Automatic failover
- Management service



Services Layer

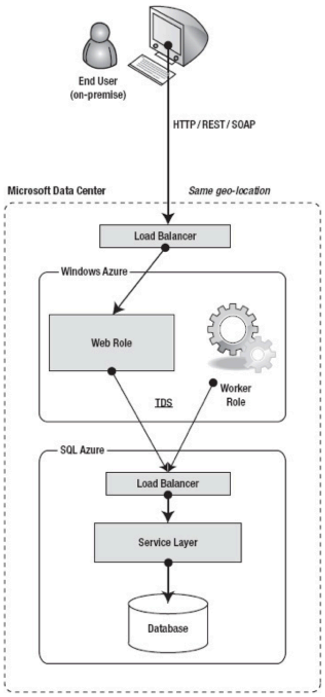
- Gateway to platform layer
- Tabular data stream (TDS)
  - Port #1433 over SSL
  - DB provisioning (via Azure Fabric)
- Billing & metering
- Account provisioning

Client Layer

- Admin API
  - ADO.NET
  - ODBC
- TDS connection
  - Routed to primary

Access Pattern #1: Code Near

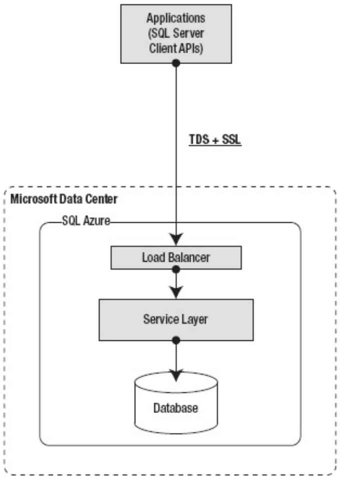
- Advantages:
  - Business logic close to DB
    - Latency
  - Open standards data access
    - HTTP, SOAP, etc
  - No lock-in
    - SQL Server client
- Disadvantage:
  - Performance
    - Azure as middle tier



© Redkar & Guidici

Access Pattern #2: Code Far

- Advantages:
  - Performance
    - Direct app connection
- Disadvantage:
  - Lock-in
    - TDS protocol
    - SQL Server clients
    - ADO.NET, ODBC, etc



© Redkar & Guidici

Connecting using ADO.NET

```
private string GetConnString() {
   SqlConnectionStringBuilder sb =
      new SqlConnectionStringBuilder();
   string server =
      “servername.ctp.database.windows.net”;
   sb.DataSource = server;
   sb.InitialCatalog = “user database”;
   sb.Encrypt = true;
   sb.TrustServerCertificate = true;
   sb.UserID = “userName”;
   sb.Password = “password”;
   return sb.ToString();
}
```

Connecting using ADO.NET

```
using (SqlConnection conn = new SqlConnection(GetConnString())) {
   using (SqlCommand command = conn.CreateCommand()) {
      conn.Open();

      command.CommandText = “create table myTable” +
                            “(Column1 primary key clustered, ...)”;
      command.ExecuteNonQuery();

      command.CommandText = “select ... From MyTable”;
      using (SqlDataReader reader = command.ExecuteReader()) {
         while (reader.read()) {
            ...reader[“Column1”].ToString().Trim()...
         }
      }
   }
}
```

DATABASE REVIEW

- SQL and Natural Join

```
select name, phone
from people
join phonenumbers
on people.id=phonenumbers.id
```

Contacts		
Id	Name	Phone
1	Joe	5532
1	Joe	2234
1	Joe	3211
2	Jane	3421
3	Chris	2341
3	Chris	6655

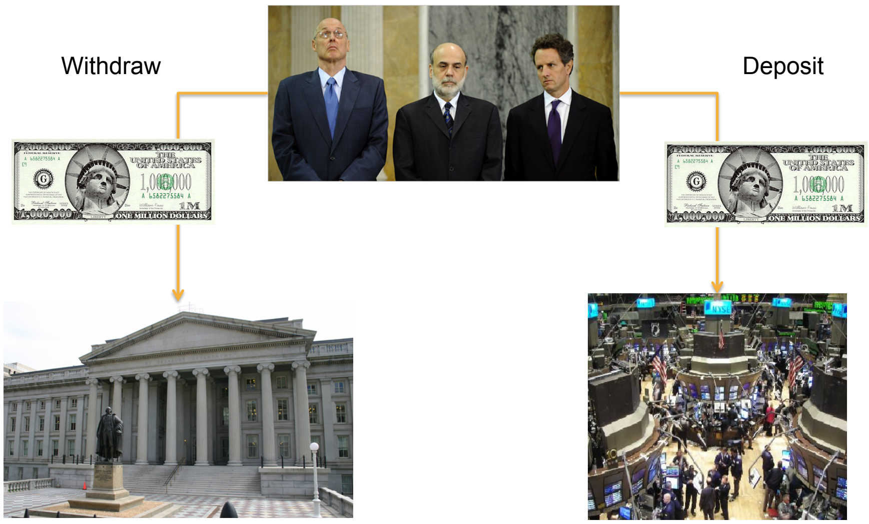
PhoneNumbers		
PhoneID	Id	Phone
1	1	5532
2	1	2234
3	1	3211
4	2	3421
5	3	2341
6	3	6655

People	
Id	Name
1	Joe
2	Jane
3	Chris

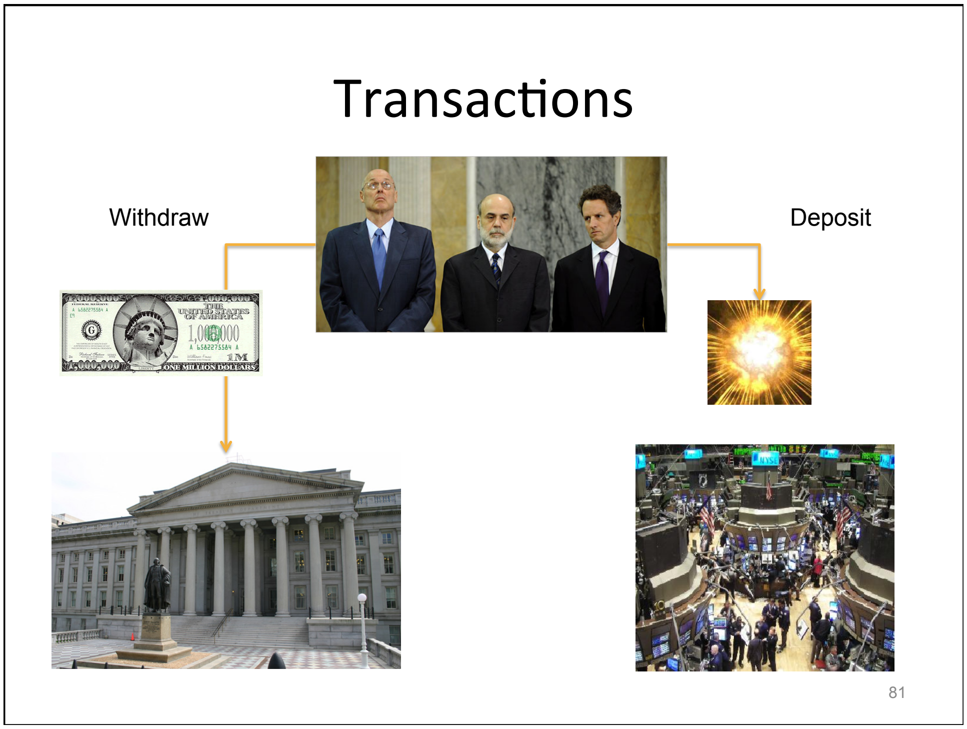
Name	Phone
Joe	5532
Joe	2234
Joe	3211
Jane	3421
Chris	2341
Chris	6655

79

Transactions



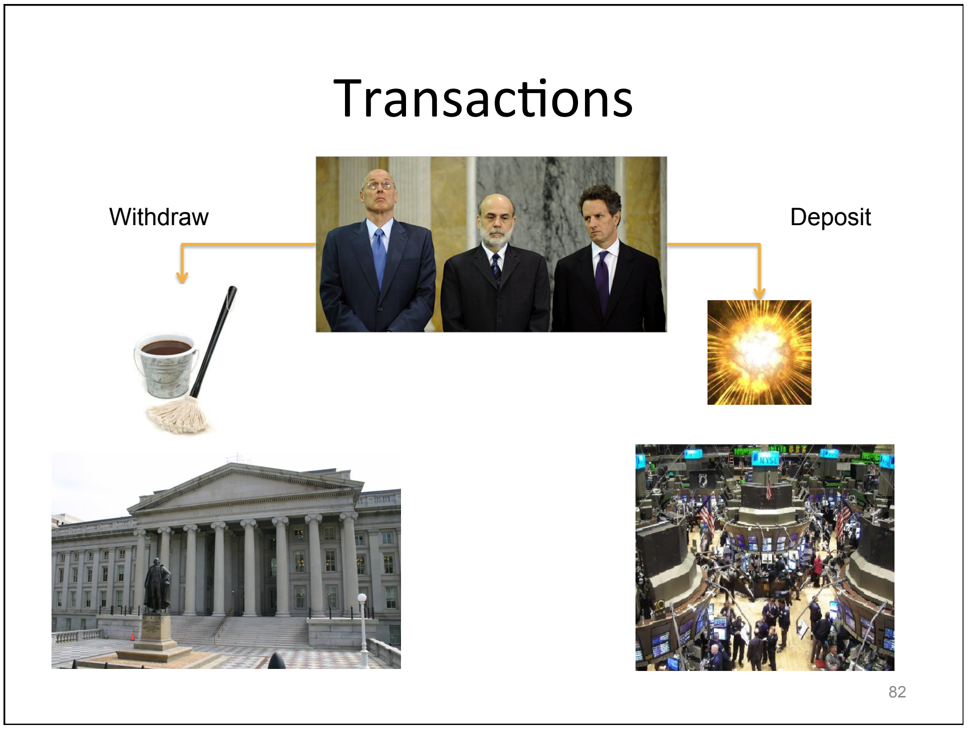
80



Transactions

Withdraw

Deposit



Transactions

Withdraw

Deposit

Transactional Operations

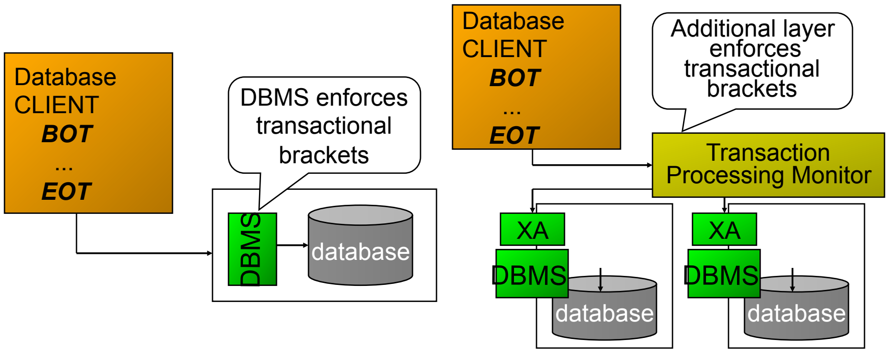
amazon.com

```
Begin Transaction
   AmtOwed = Amazon.Read (CustID);
   Balance = Paypal.Read (AccountID);
   if (AmtOwed ≤ Balance) {
      Paypal.Withdraw (AccountID, AmtOwed);
      Amazon.Deposit (CustID, AmtOwed);
   } else {
      Abort Transaction;
   }
End Transaction
```

PayPal®

Distributed Databases

- Transactions and atomic commitment

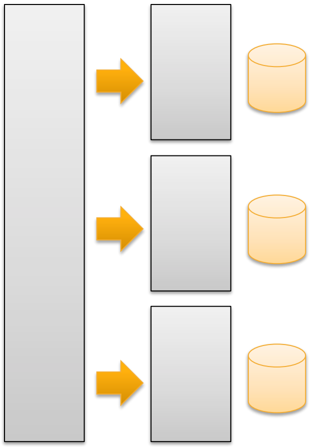

Database CLIENT
BOT
...
EOT
DBMS enforces transactional brackets
DBMS
database
Database CLIENT
BOT
...
EOT
Additional layer enforces transactional brackets
Transaction Processing Monitor
XA
DBMS
database
XA
DBMS
database

Relational Database Summary

- Database Schema
  - Normalized for efficiency
- SQL for ad-hoc queries
- Transactional updates
  - Atomic
  - Consistent
  - Isolated
  - Durable

Challenge: Big Data™

- Historical approach: *vertical scaling*
  - Limited
- Modern approach: *horizontal scaling*
  - *Sharding*
  - Azure: Federated SQL databases
  - Applications see data partitioning
  - No joins across partitions



SQL vs NoSQL

Relational

- Database Schema
  - Business data model
- SQL for ad-hoc queries
- ACID properties
  - Atomic
  - Consistent
  - Isolated
  - Durable

NoSQL

- Unstructured
  - Web server logs
- Map-Reduce
- BASE properties
  - Basically Available
  - Soft state
  - Eventually consistent

Fault-tolerance

So, how do I query the database?
It's not a database. It's a key-value store!
Ok, it's not a database. How do I query it?
You write a distributed map reduce function in Erlang!
Did you just tell me to go f□k myself?
I believe I did, Bob.

TABLE STORAGE

Table Storage

- Property:
  - Name-value pair e.g. `userid="JoeBlow"`
- Entity: group of related properties
  - `PartitionKey`
    - Defines partition
  - `RowKey`
    - Defines order in a partition
  - `Timestamp`
  - Size limitations (1 Mb)
- Table:
  - Partitioned set of key-entity pairs

Table Storage

- Property
- Entity
- Table

- Address:

```
http://account.table.core.windows.net/Tables
http://account.table.core.windows.net/
                                    Table('table-name')
http://acct.table.core.windows.net/table-name()
```

Table Service Architecture

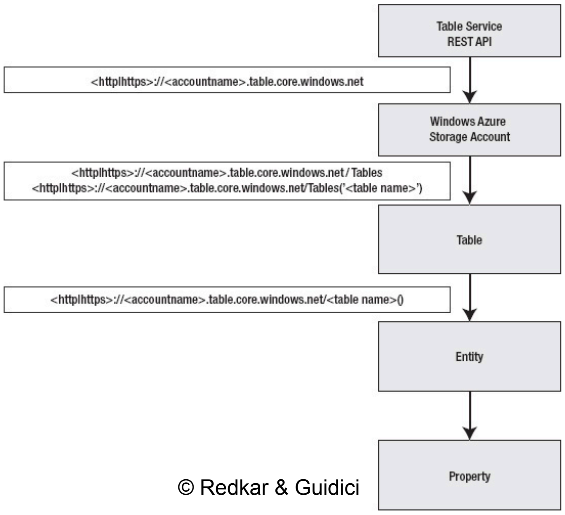
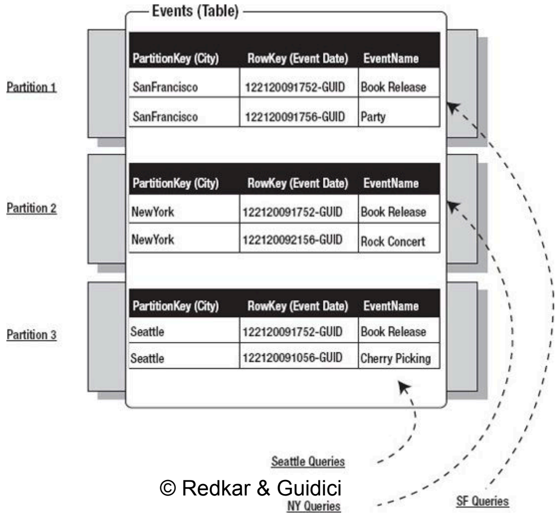


Table Service
REST API
<httplhttps>://<accountname>.table.core.windows.net
Windows Azure
Storage Account
<httplhttps>://<accountname>.table.core.windows.net / Tables
<httplhttps>://<accountname>.table.core.windows.net/Tables('<table name>')
Table
<httplhttps>://<accountname>.table.core.windows.net/<table name>()
Entity
Property

© Redkar & Guidici

Table Service Architecture



PartitionKey (City)	RowKey (Event Date)	EventName
SanFrancisco	122120091752-GUID	Book Release
SanFrancisco	122120091756-GUID	Party

PartitionKey (City)	RowKey (Event Date)	EventName
NewYork	122120091752-GUID	Book Release
NewYork	122120092156-GUID	Rock Concert

PartitionKey (City)	RowKey (Event Date)	EventName
Seattle	122120091752-GUID	Book Release
Seattle	122120091056-GUID	Cherry Picking

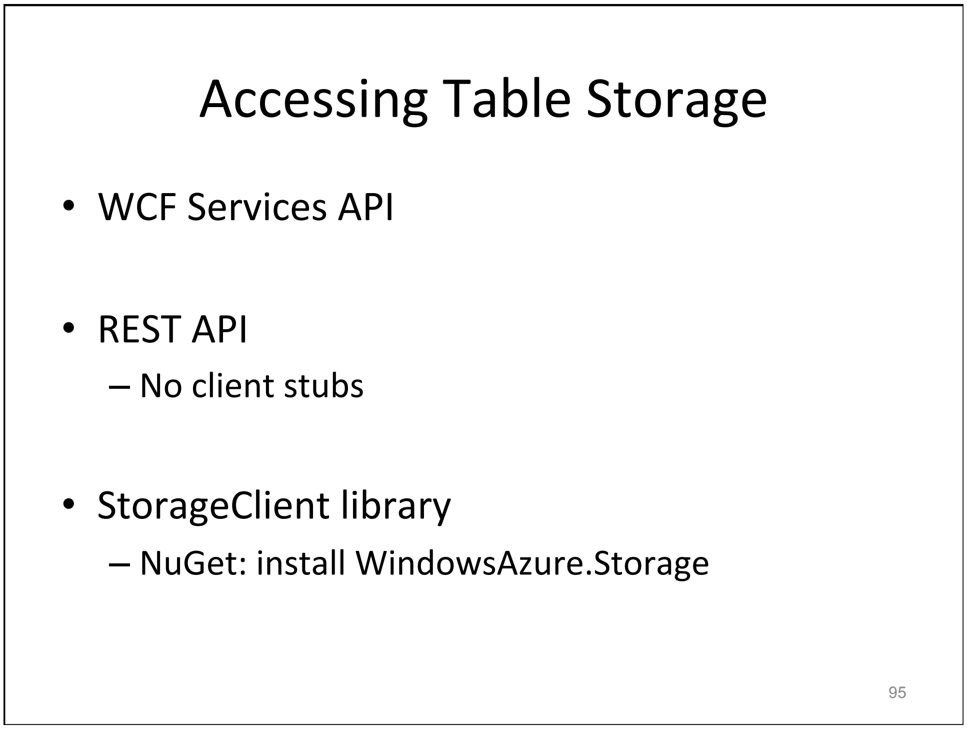
© Redkar & Guidici

Table Names

- Some rules for names
  - Valid DNS name
  - Letters, numbers
  - *All letters must be lowercase*

- Query parameters

```
http://dns-name/table-name()?$top=5
http://dns-name/table-name()?
  $filter=
   Date%20eq%20datetime’2012-09-30-20T00:00:00’
```



Accessing Table Storage

- WCF Services API
- REST API
  - No client stubs
- StorageClient library
  - NuGet: install WindowsAzure.Storage

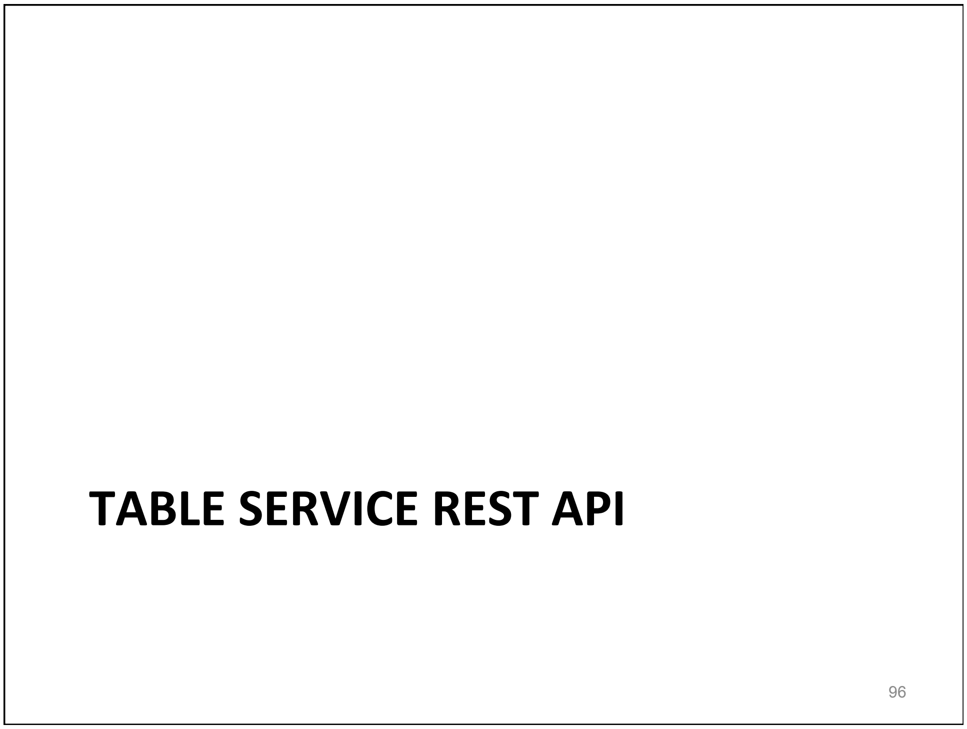


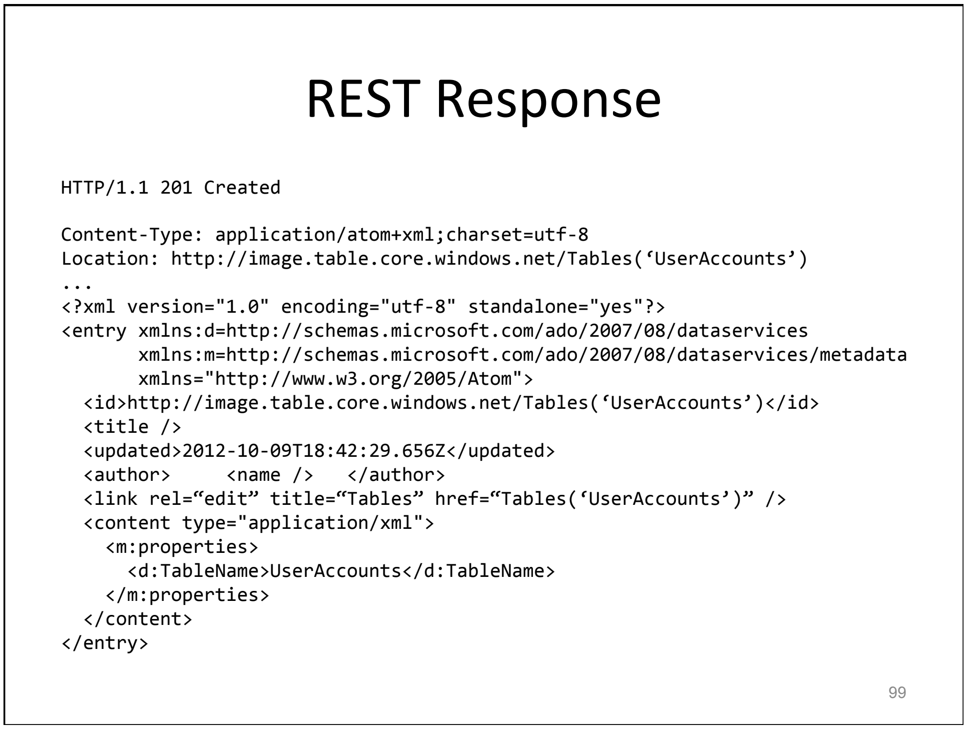
TABLE SERVICE REST API

Table Operations

Operation	HTTP Verb	Cloud URI
Create table	POST	http://*acct*.table.core.windows.net/Tables
Query tables	GET	http://*acct*.table.core.windows.net/Tables()
Delete table	DELETE	http://*acct*.table.core.windows.net/Tables(‘*table-name*’)

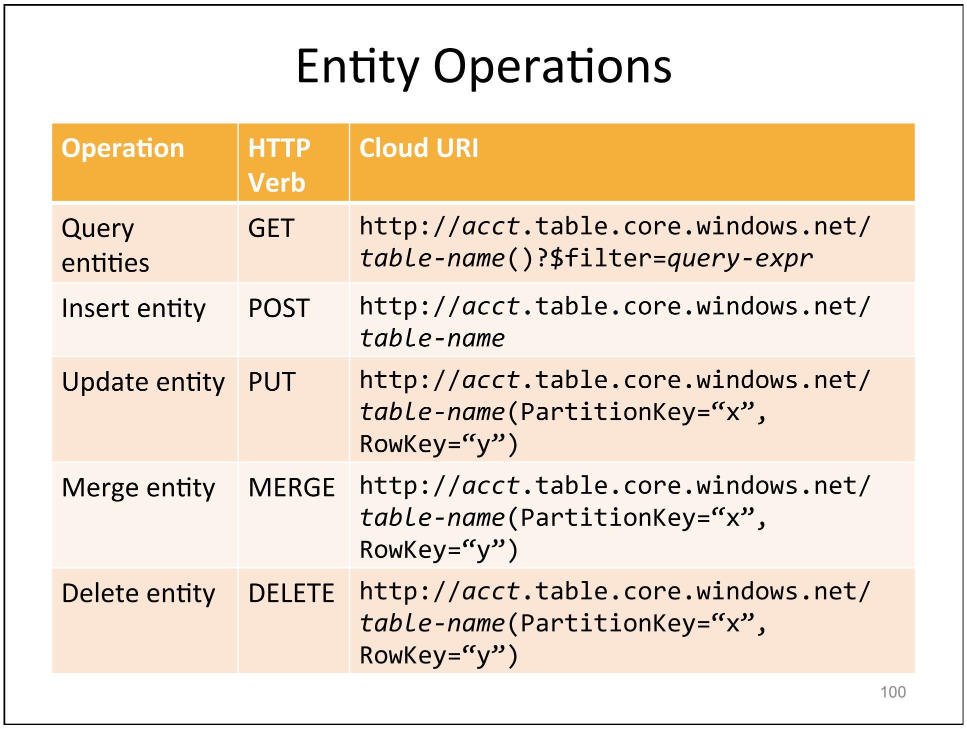
REST Request

```
POST /Tables HTTP/1.1

User-Agent: Microsoft ADO.NET Data Services
x-ms-date: Sun, 9 Oct 2012 18:42:29 GMT
Authorization: SharedKeyLite
               image:pwFouPw+BPWzlaQPyccII+K8zb+v6qygxZhp9fCdqRA=
Content-Type: application/atom+xml
Host: image.table.core.windows.net
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<entry xmlns:d=http://schemas.microsoft.com/ado/2007/08/dataservices
       xmlns:m=http://schemas.microsoft.com/ado/2007/08/dataservices/metadata
       xmlns="http://www.w3.org/2005/Atom">
  <title />
  <updated>2012-10-09T18:42:29.656Z</updated>
  <author>     <name />   </author>   <id />
  <content type="application/xml">
    <m:properties>
      <d:TableName>UserAccounts</d:TableName>
    </m:properties>
  </content>
</entry>
```



REST Response

```
HTTP/1.1 201 Created

Content-Type: application/atom+xml;charset=utf-8
Location: http://image.table.core.windows.net/Tables(‘UserAccounts’)
...
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<entry xmlns:d=http://schemas.microsoft.com/ado/2007/08/dataservices
       xmlns:m=http://schemas.microsoft.com/ado/2007/08/dataservices/metadata
       xmlns="http://www.w3.org/2005/Atom">
  <id>http://image.table.core.windows.net/Tables(‘UserAccounts’)</id>
  <title />
  <updated>2012-10-09T18:42:29.656Z</updated>
  <author>     <name />   </author>
  <link rel=“edit” title=“Tables” href=“Tables(‘UserAccounts’)” />
  <content type="application/xml">
    <m:properties>
      <d:TableName>UserAccounts</d:TableName>
    </m:properties>
  </content>
</entry>
```



Entity Operations

Operation	HTTP Verb	Cloud URI
Query entities	GET	http://*acct*.table.core.windows.net/*table-name*()?$filter=*query-expr*
Insert entity	POST	http://*acct*.table.core.windows.net/*table-name*
Update entity	PUT	http://*acct*.table.core.windows.net/*table-name*(PartitionKey=“x”, RowKey=“y”)
Merge entity	MERGE	http://*acct*.table.core.windows.net/*table-name*(PartitionKey=“x”, RowKey=“y”)
Delete entity	DELETE	http://*acct*.table.core.windows.net/*table-name*(PartitionKey=“x”, RowKey=“y”)

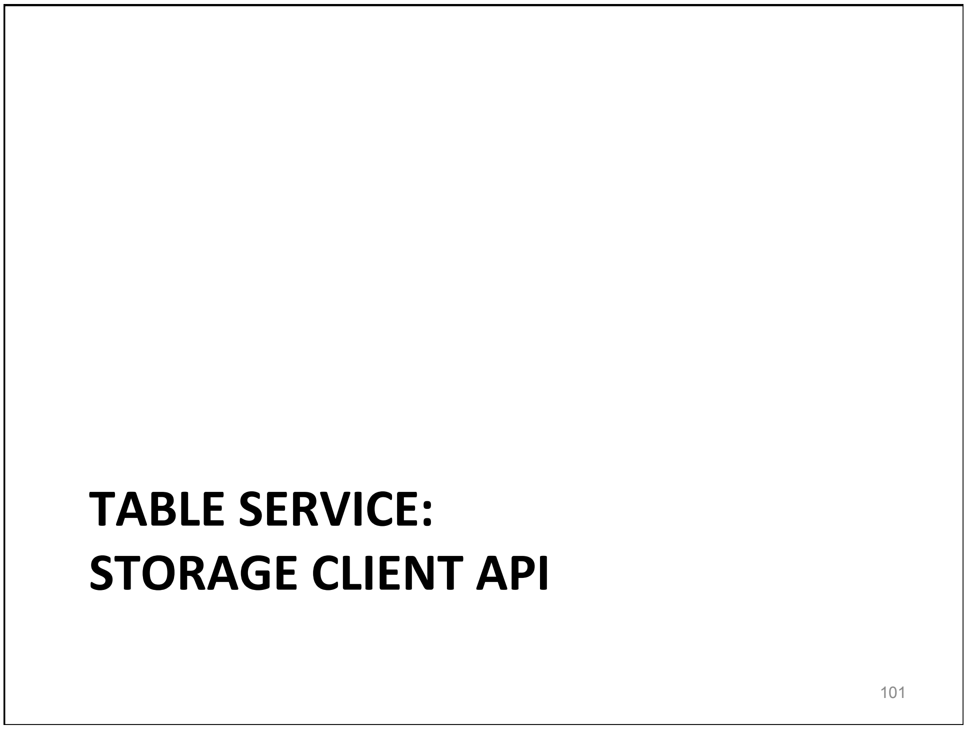
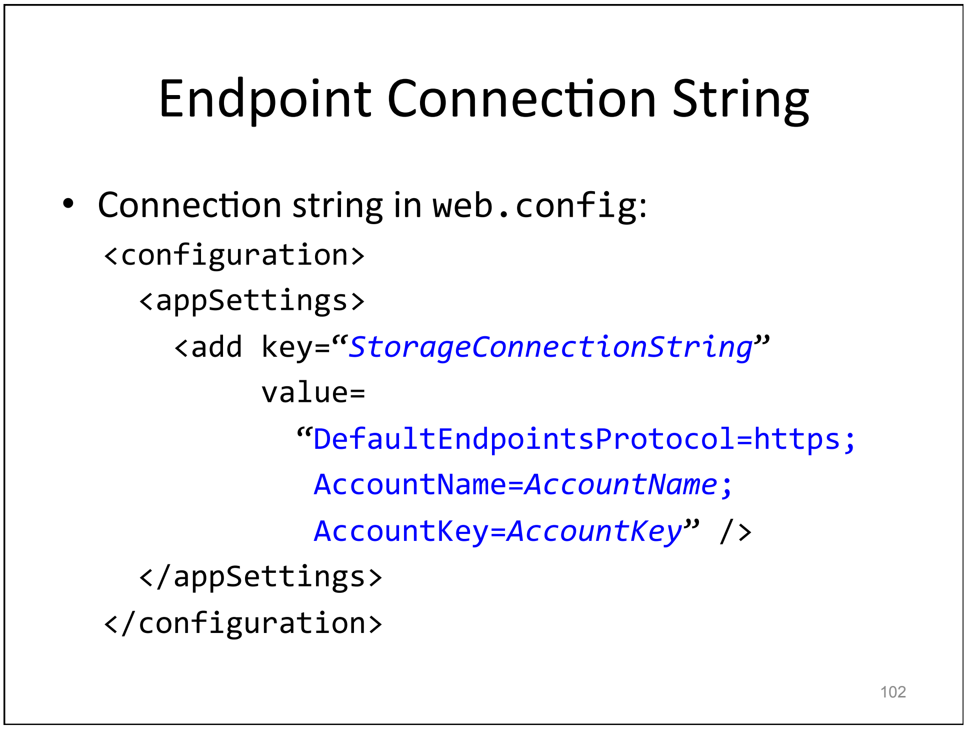


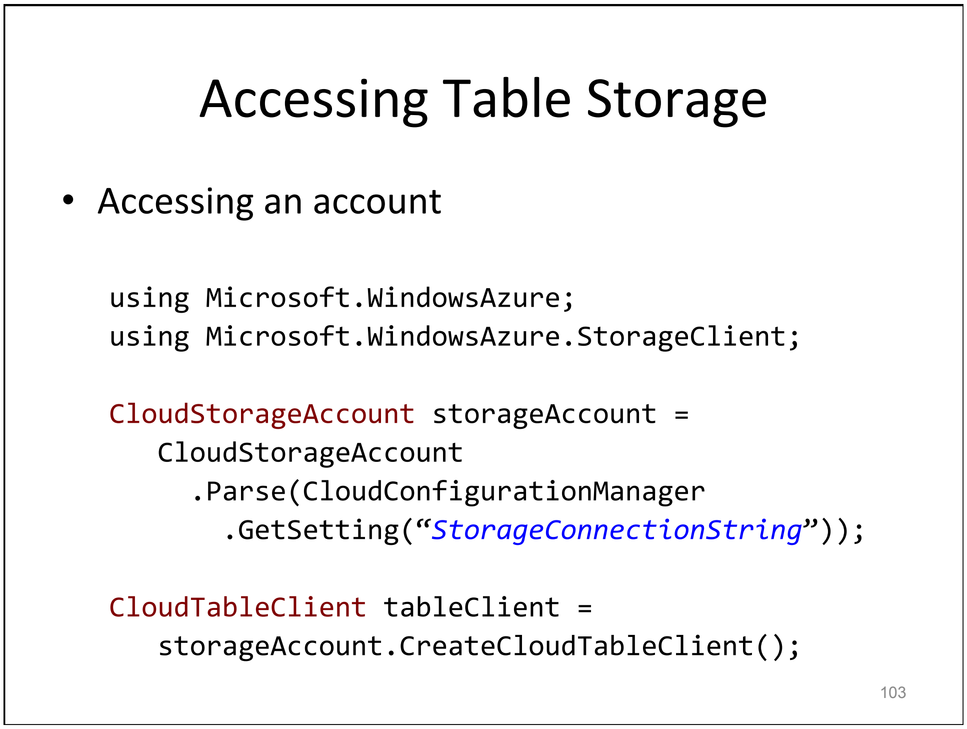
TABLE SERVICE: STORAGE CLIENT API



Endpoint Connection String

- Connection string in `web.config`:

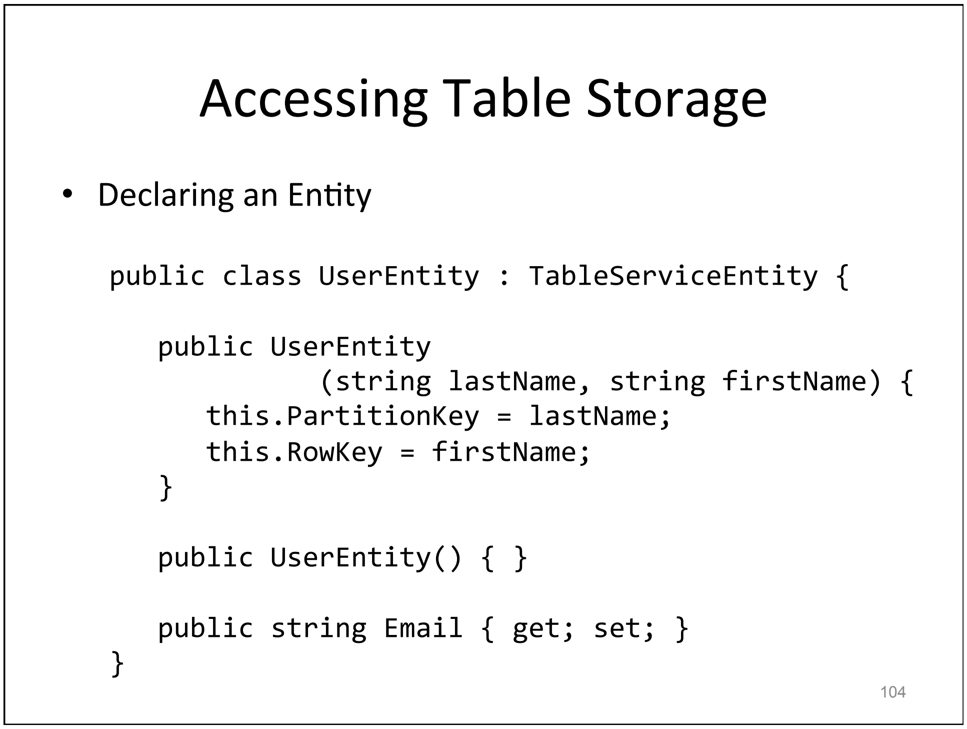
```
<configuration>
  <appSettings>
    <add key="StorageConnectionString"
         value=
           "DefaultEndpointsProtocol=https;
            AccountName=AccountName;
            AccountKey=AccountKey" />
  </appSettings>
</configuration>
```



Accessing Table Storage

- Accessing an account

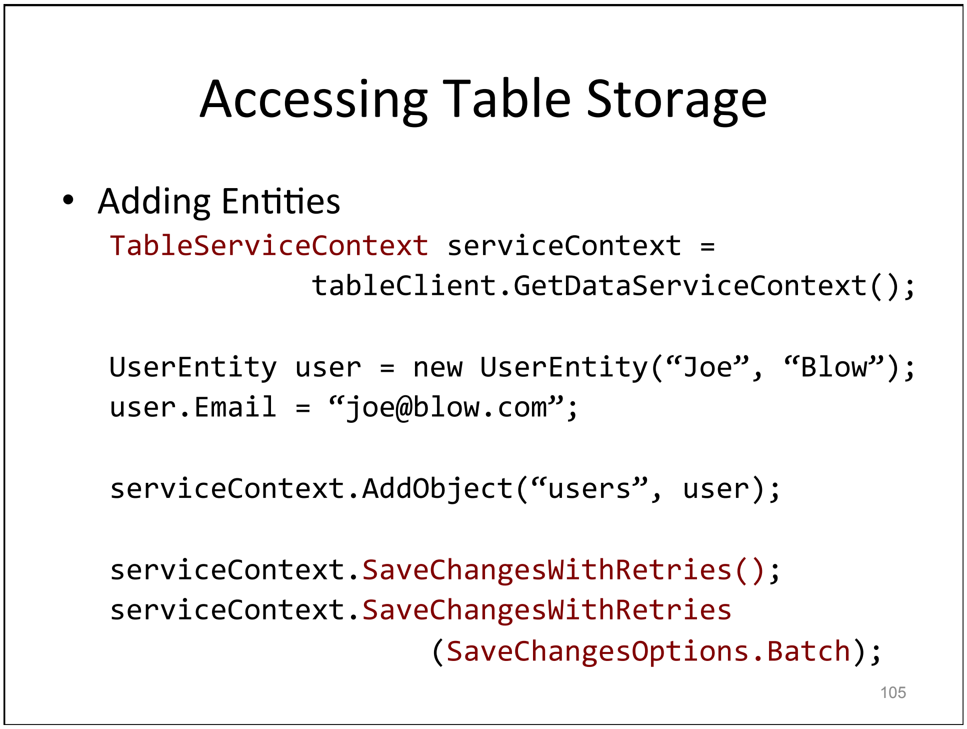
```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;

CloudStorageAccount storageAccount =
   CloudStorageAccount
     .Parse(CloudConfigurationManager
       .GetSetting(“StorageConnectionString”));

CloudTableClient tableClient =
   storageAccount.CreateCloudTableClient();
```



Accessing Table Storage

- Declaring an Entity

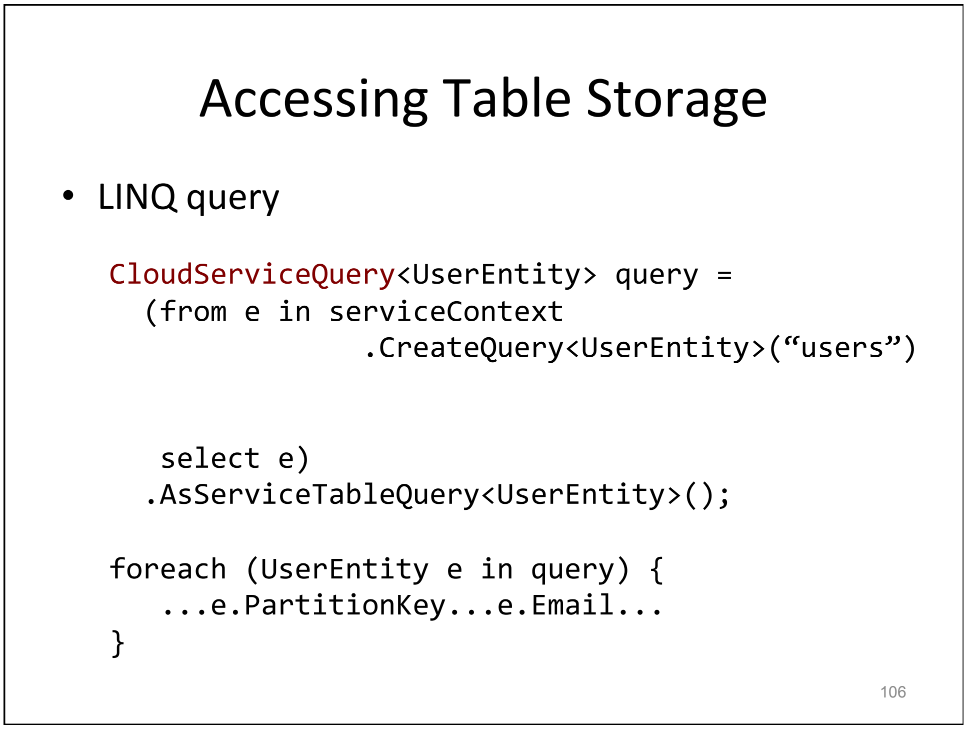
```
public class UserEntity : TableServiceEntity {

   public UserEntity
             (string lastName, string firstName) {
      this.PartitionKey = lastName;
      this.RowKey = firstName;
   }

   public UserEntity() { }

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



Accessing Table Storage

- Adding Entities

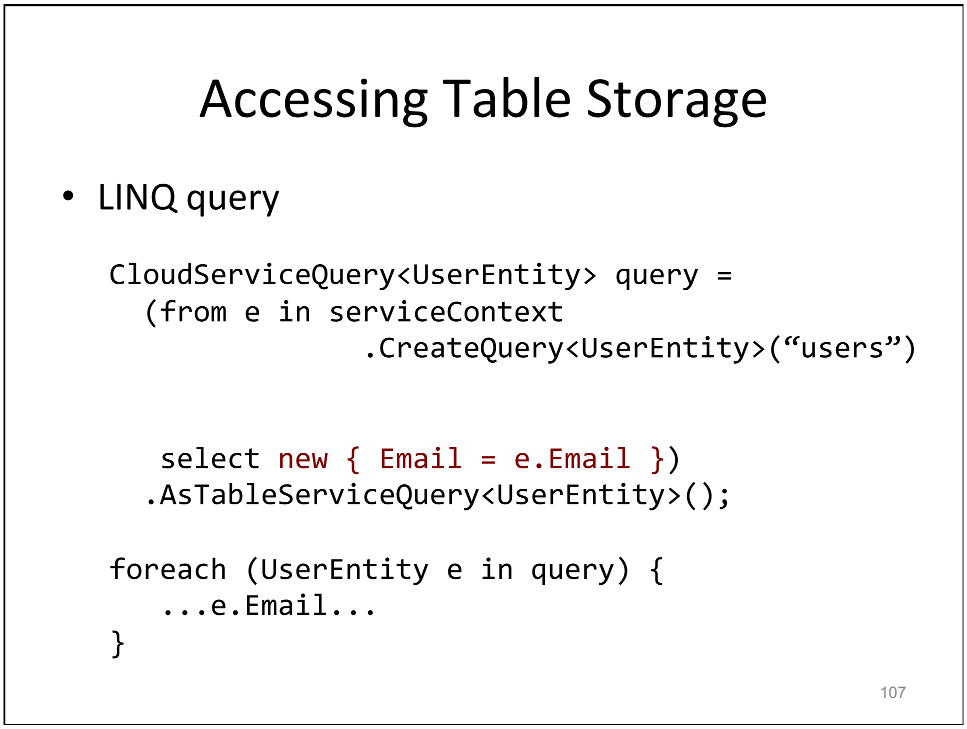
```
TableServiceContext serviceContext =
            tableClient.GetDataServiceContext();

UserEntity user = new UserEntity(“Joe”, “Blow”);
user.Email = “joe@blow.com”;

serviceContext.AddObject(“users”, user);

serviceContext.SaveChangesWithRetries();
serviceContext.SaveChangesWithRetries
                   (SaveChangesOptions.Batch);
```



Accessing Table Storage

- LINQ query

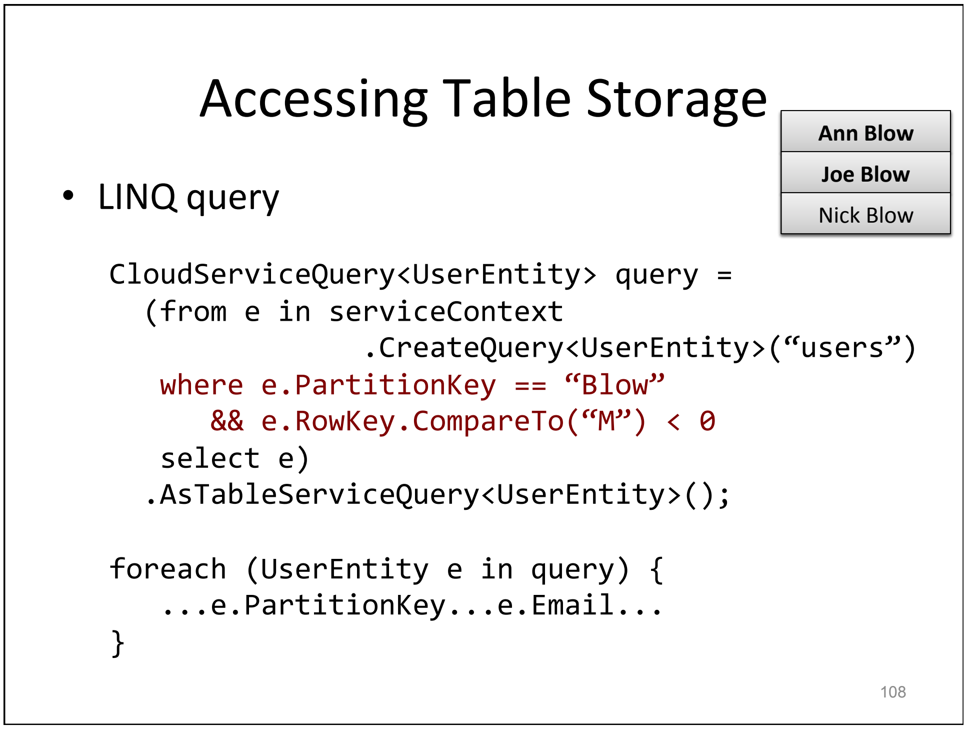
```
CloudServiceQuery<UserEntity> query =
  (from e in serviceContext
               .CreateQuery<UserEntity>(“users”)


   select e)
  .AsServiceTableQuery<UserEntity>();

foreach (UserEntity e in query) {
   ...e.PartitionKey...e.Email...
}
```



Accessing Table Storage

- LINQ query

```
CloudServiceQuery<UserEntity> query =
  (from e in serviceContext
               .CreateQuery<UserEntity>(“users”)


   select new { Email = e.Email })
  .AsTableServiceQuery<UserEntity>();

foreach (UserEntity e in query) {
   ...e.Email...
}
```



Accessing Table Storage

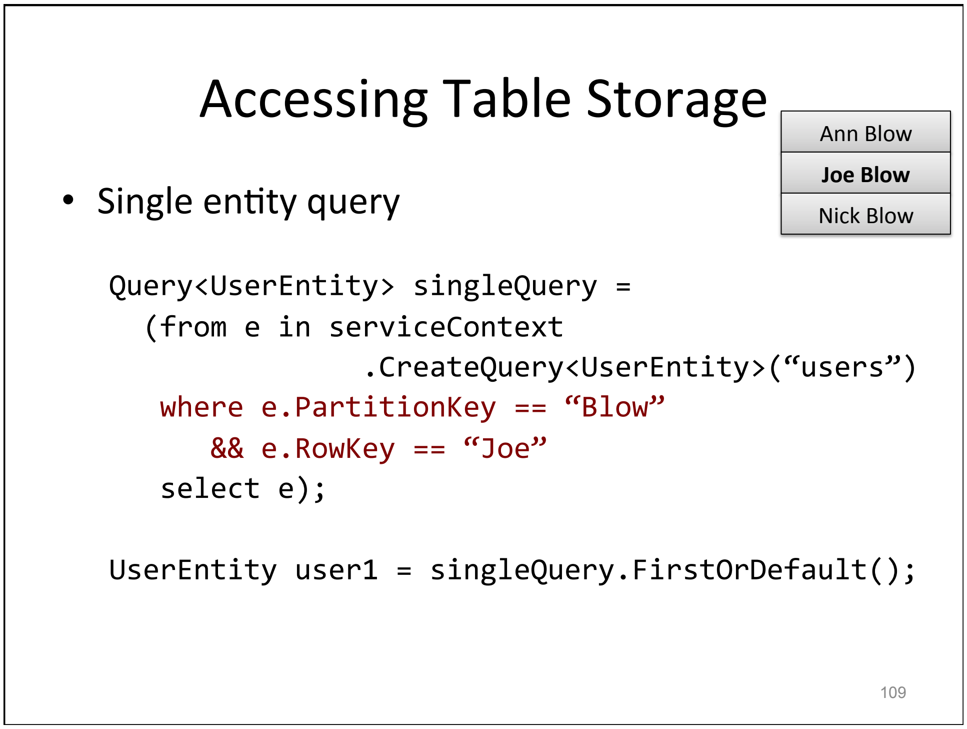
Ann Blow

Joe Blow

Nick Blow

- LINQ query

```
CloudServiceQuery<UserEntity> query =
  (from e in serviceContext
               .CreateQuery<UserEntity>(“users”)
   where e.PartitionKey == “Blow”
      && e.RowKey.CompareTo(“M”) < 0
   select e)
  .AsTableServiceQuery<UserEntity>();

foreach (UserEntity e in query) {
   ...e.PartitionKey...e.Email...
}
```



Accessing Table Storage

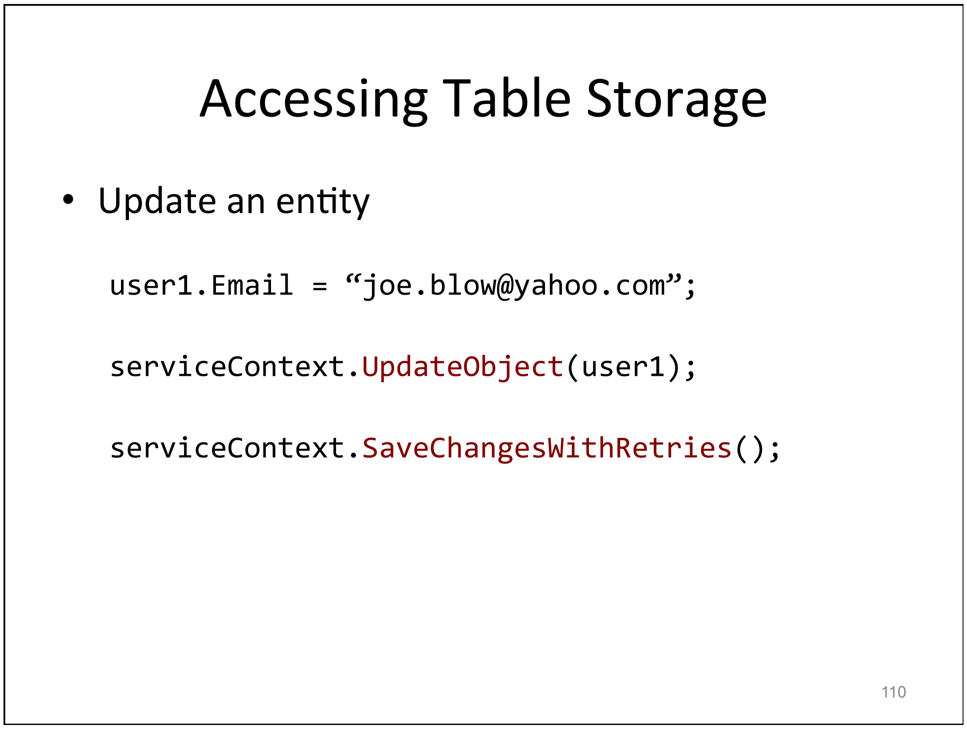
Ann Blow

Joe Blow

Nick Blow

- Single entity query

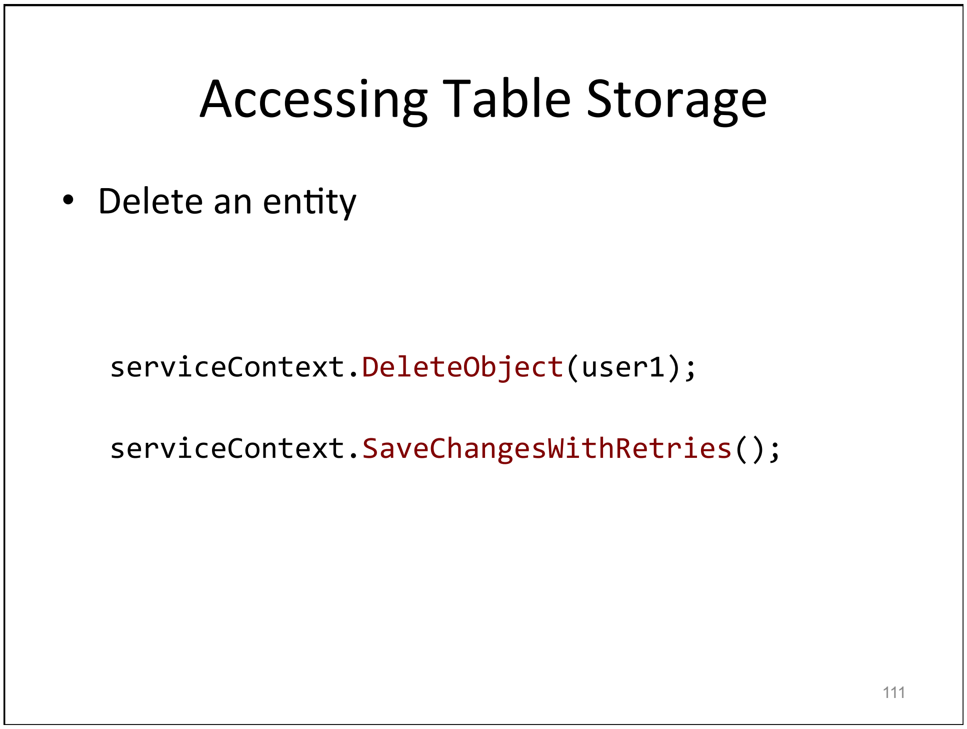
```
Query<UserEntity> singleQuery =
  (from e in serviceContext
                .CreateQuery<UserEntity>(“users”)
   where e.PartitionKey == “Blow”
      && e.RowKey == “Joe”
   select e);

UserEntity user1 = singleQuery.FirstOrDefault();
```



Accessing Table Storage

- Update an entity

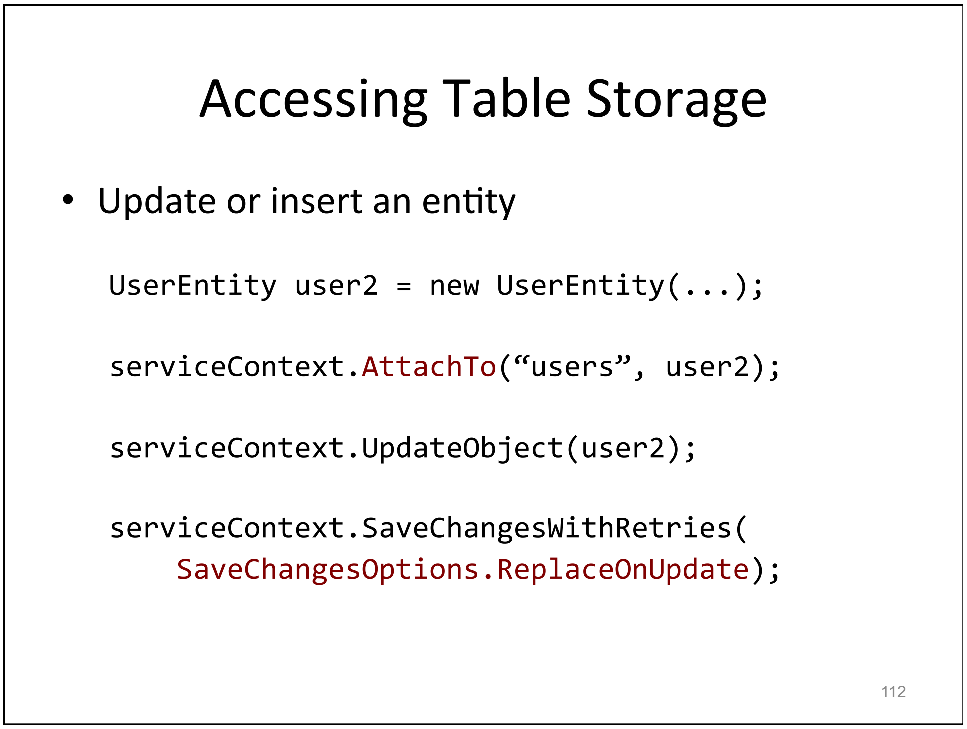
```
user1.Email = “joe.blow@yahoo.com”;

serviceContext.UpdateObject(user1);

serviceContext.SaveChangesWithRetries();
```



Accessing Table Storage

- Delete an entity

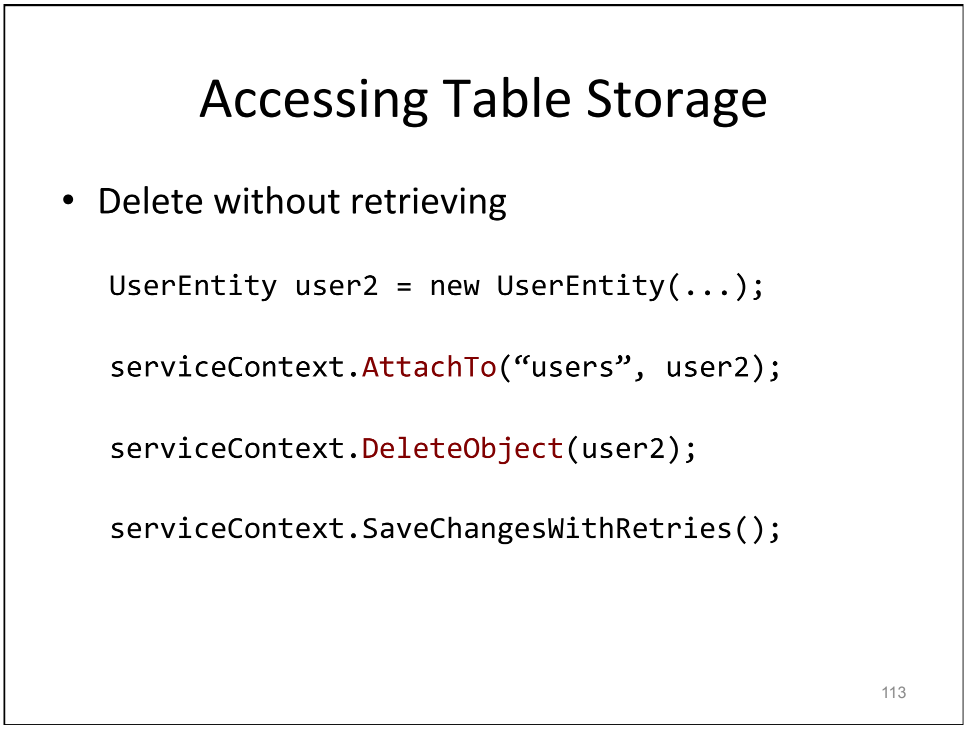
```
serviceContext.DeleteObject(user1);

serviceContext.SaveChangesWithRetries();
```



Accessing Table Storage

- Update or insert an entity

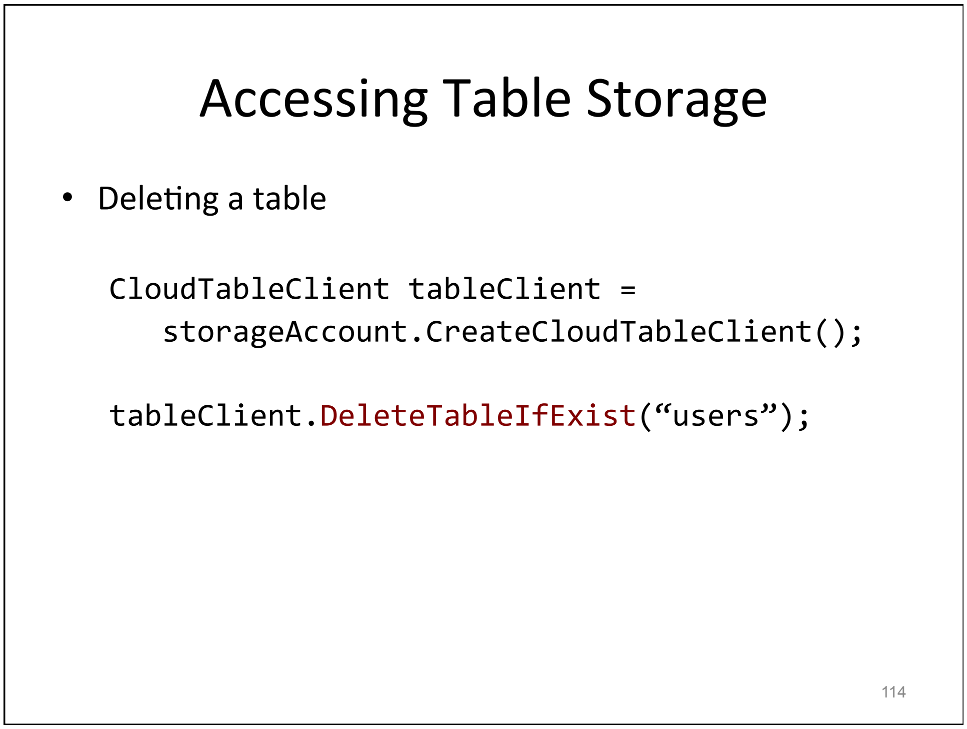
```
UserEntity user2 = new UserEntity(...);

serviceContext.AttachTo(“users”, user2);

serviceContext.UpdateObject(user2);

serviceContext.SaveChangesWithRetries(
    SaveChangesOptions.ReplaceOnUpdate);
```



Accessing Table Storage

- Delete without retrieving

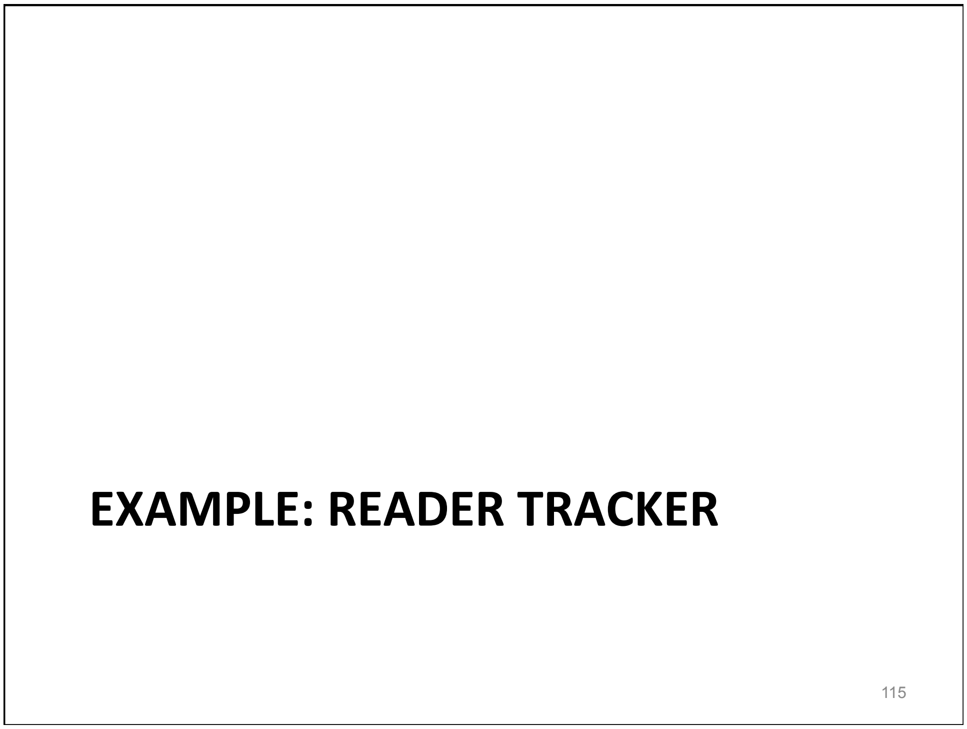
```
UserEntity user2 = new UserEntity(...);

serviceContext.AttachTo(“users”, user2);

serviceContext.DeleteObject(user2);

serviceContext.SaveChangesWithRetries();
```



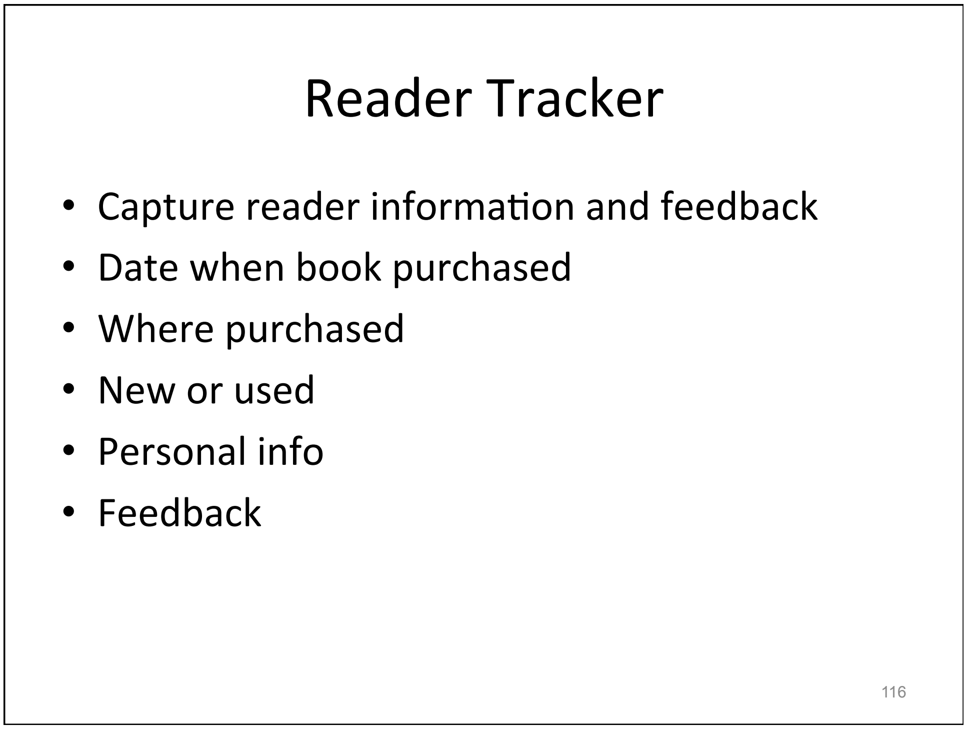
Accessing Table Storage

- Deleting a table

```
CloudTableClient tableClient =
    storageAccount.CreateCloudTableClient();

tableClient.DeleteTableIfExist(“users”);
```



EXAMPLE: READER TRACKER



Reader Tracker

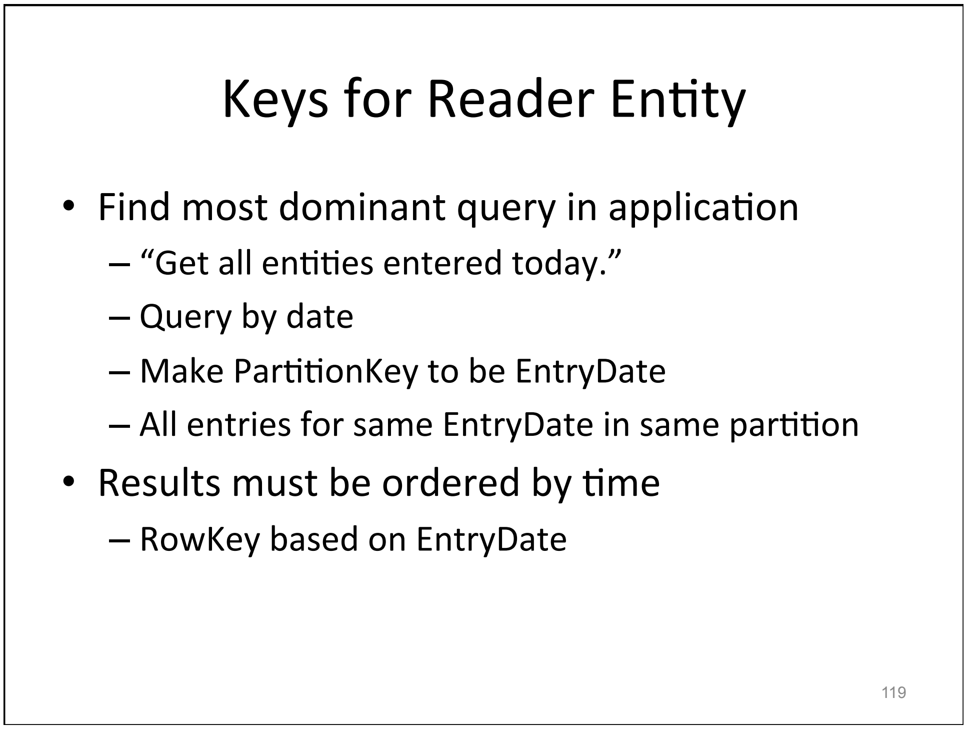
- Capture reader information and feedback
- Date when book purchased
- Where purchased
- New or used
- Personal info
- Feedback

Reader Entity

- Define a class for reader entities:
  `public class Reader : TableServiceEntity { ... }`
- Properties:
  - Purchase date
  - Entry date
  - Country, State, City, Zip
  - Purchase location
  - Purchase type
  - Reader name
  - Reader URL
  - Feedback

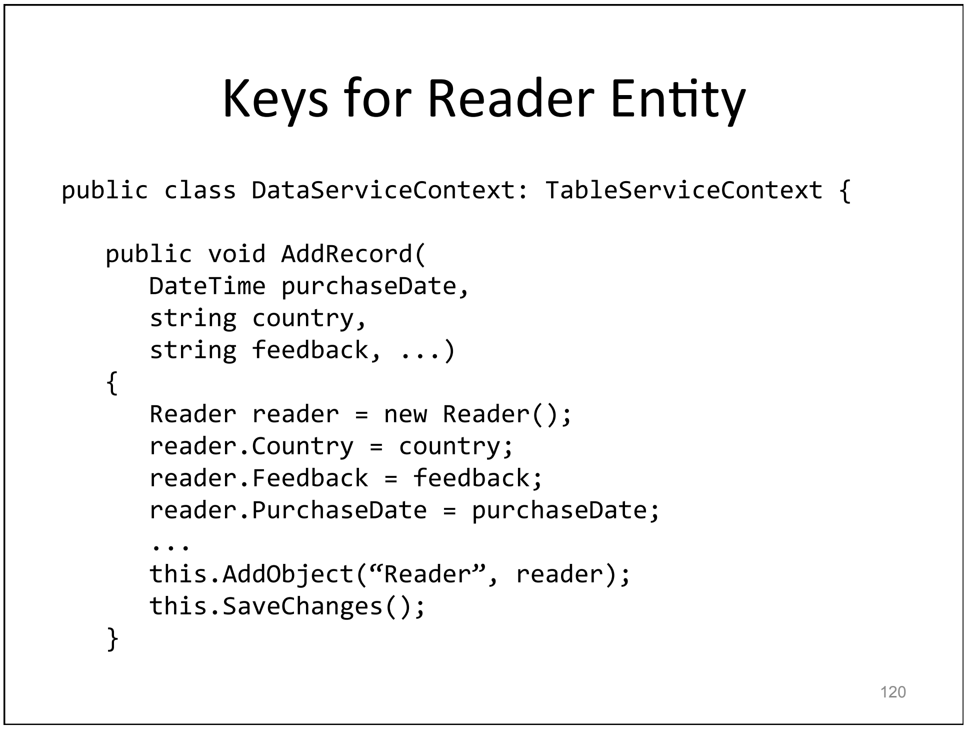
Reader Data Source

- `DataServiceContext`:
  - ADO.NET client-side state of invocations
- `ReaderDataSource`
  - Utility class for data queries
  - `CreateTable, DeleteTable, ListTables`
  - `AddReader, UpdateFeedback`
  - `Select`
  - `SelectByCity, SelectByCountry, SelectByState`
  - `SelectByPurchaseDate`
  - `SelectTopN`



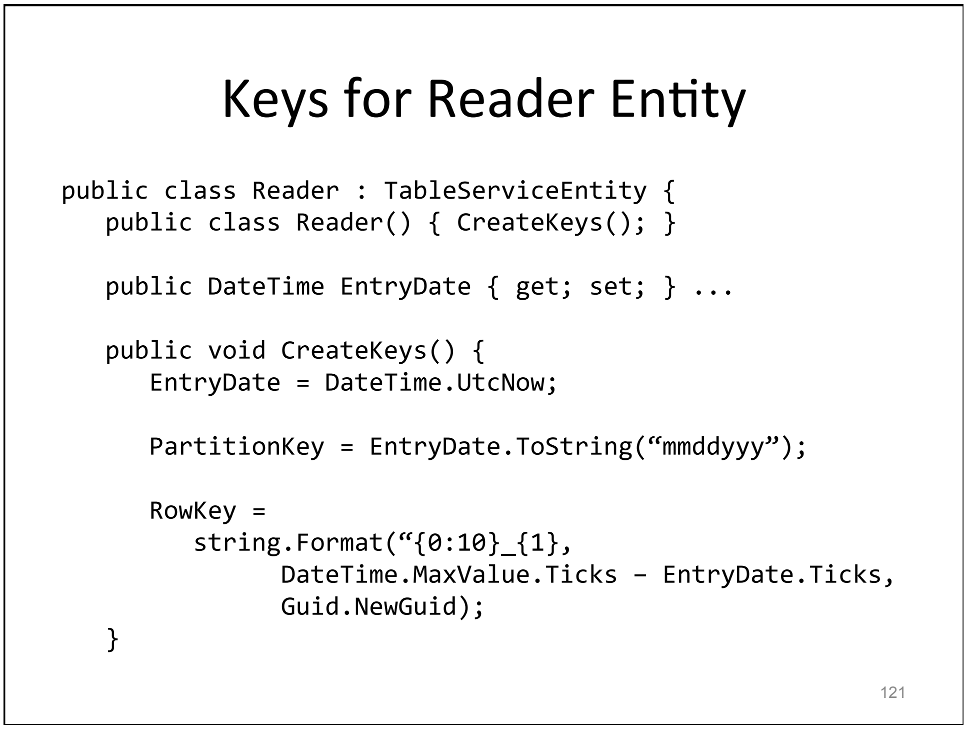
Keys for Reader Entity

- Find most dominant query in application
  - “Get all entities entered today.”
  - Query by date
  - Make PartitionKey to be EntryDate
  - All entries for same EntryDate in same partition
- Results must be ordered by time
  - RowKey based on EntryDate



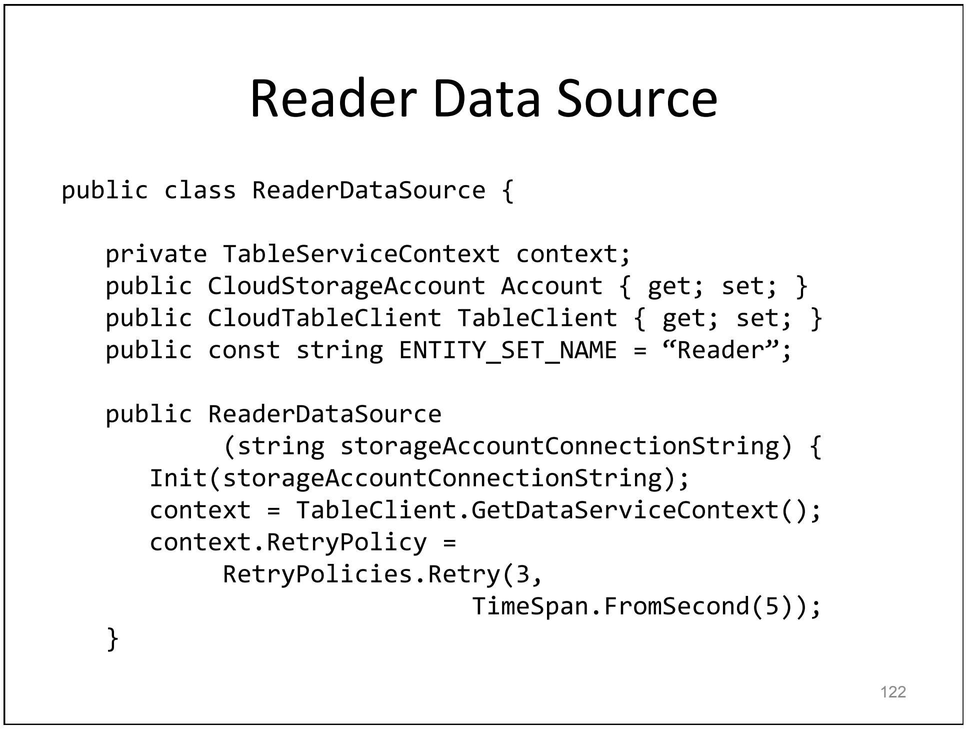
Keys for Reader Entity

```
public class DataServiceContext: TableServiceContext {

   public void AddRecord(
      DateTime purchaseDate,
      string country,
      string feedback, ...)
   {
      Reader reader = new Reader();
      reader.Country = country;
      reader.Feedback = feedback;
      reader.PurchaseDate = purchaseDate;
      ...
      this.AddObject(“Reader”, reader);
      this.SaveChanges();
   }
```



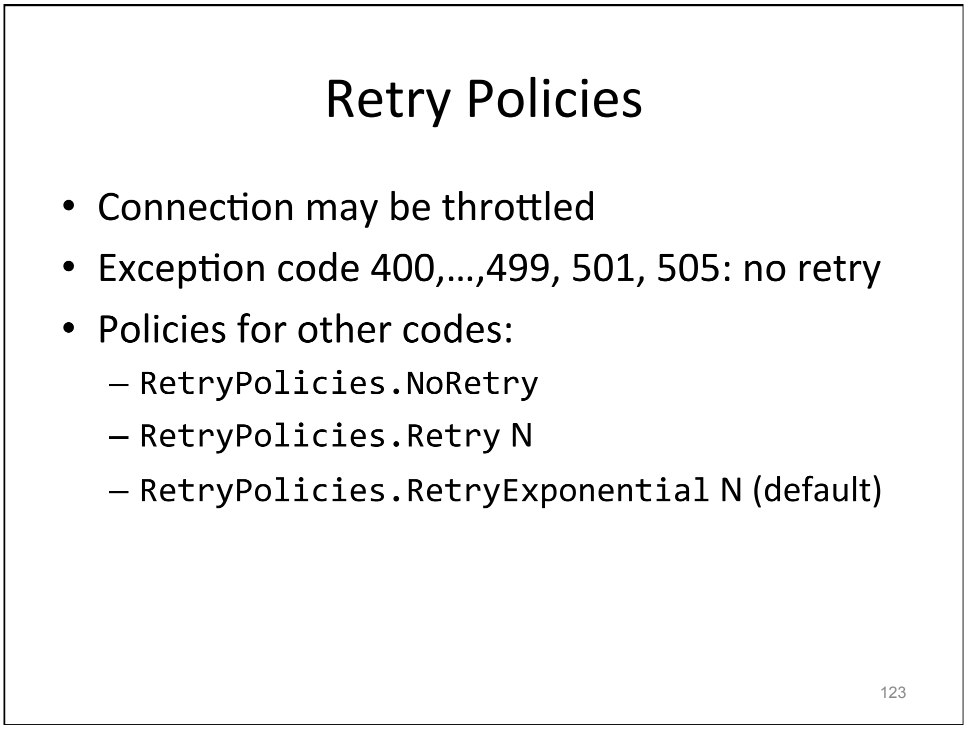
Keys for Reader Entity

```
public class Reader : TableServiceEntity {
   public class Reader() { CreateKeys(); }

   public DateTime EntryDate { get; set; } ...

   public void CreateKeys() {
      EntryDate = DateTime.UtcNow;

      PartitionKey = EntryDate.ToString(“mmddyyy”);

      RowKey =
         string.Format(“{0:10}_{1},
               DateTime.MaxValue.Ticks – EntryDate.Ticks,
               Guid.NewGuid);
   }
```



Reader Data Source

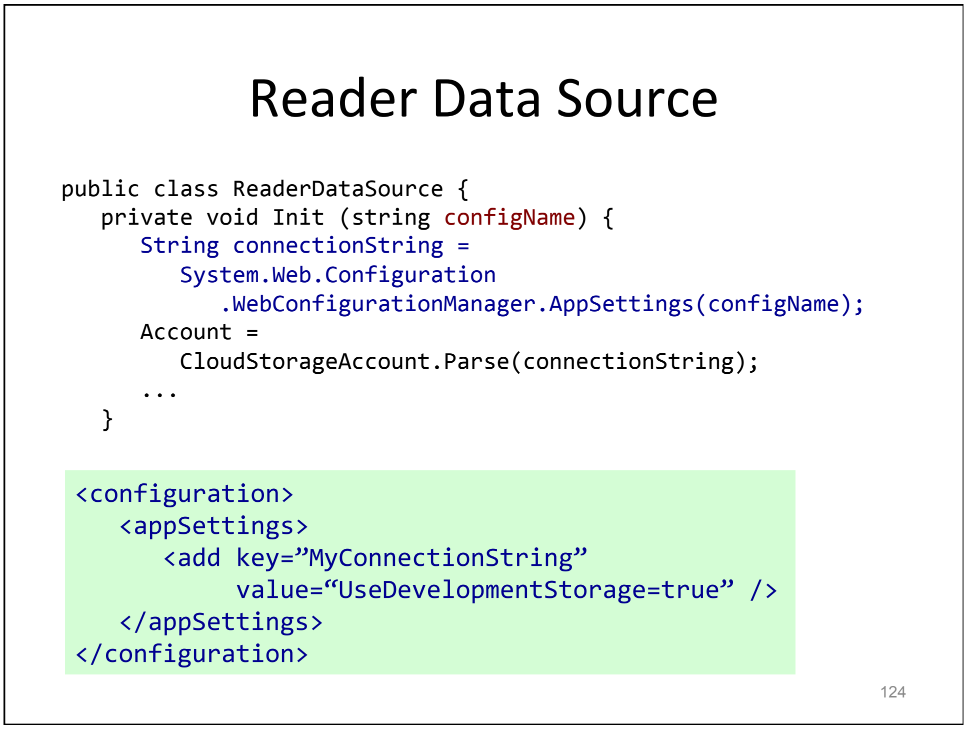
```
public class ReaderDataSource {

   private TableServiceContext context;
   public CloudStorageAccount Account { get; set; }
   public CloudTableClient TableClient { get; set; }
   public const string ENTITY_SET_NAME = “Reader”;

   public ReaderDataSource
           (string storageAccountConnectionString) {
      Init(storageAccountConnectionString);
      context = TableClient.GetDataServiceContext();
      context.RetryPolicy =
           RetryPolicies.Retry(3,
                            TimeSpan.FromSecond(5));
   }
```



Retry Policies

- Connection may be throttled
- Exception code 400,…,499, 501, 505: no retry
- Policies for other codes:
  - `RetryPolicies.NoRetry`
  - `RetryPolicies.Retry` N
  - `RetryPolicies.RetryExponential` N (default)

123

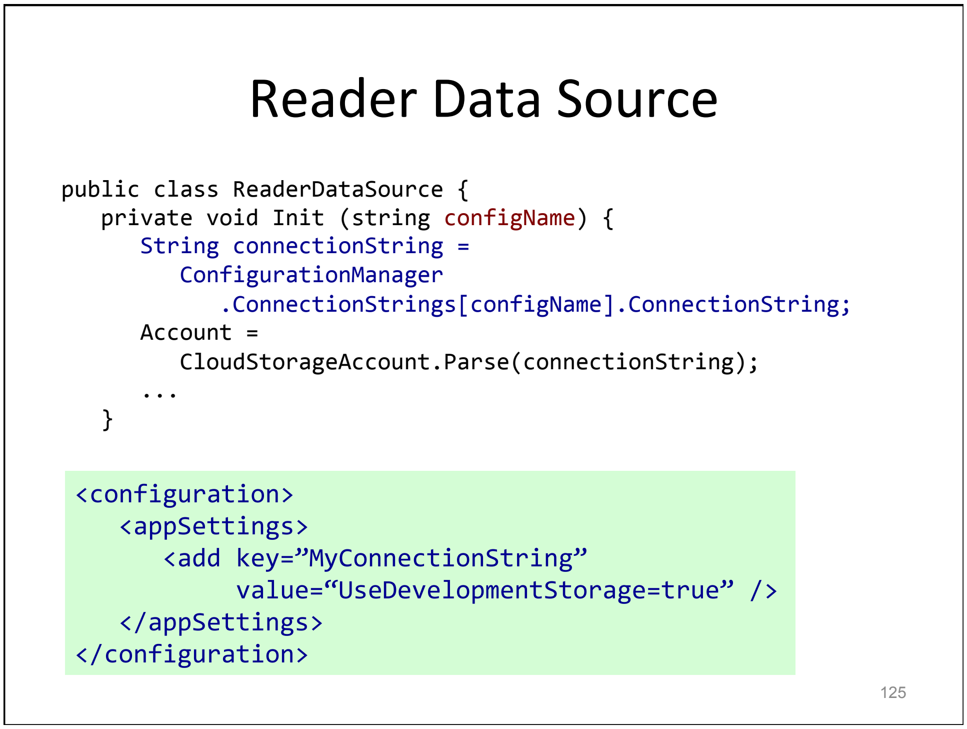


Reader Data Source

```
public class ReaderDataSource {
   private void Init (string configName) {
      String connectionString =
         System.Web.Configuration
            .WebConfigurationManager.AppSettings(configName);
      Account =
         CloudStorageAccount.Parse(connectionString);
      ...
   }
```

```
<configuration>
   <appSettings>
      <add key=”MyConnectionString”
           value=“UseDevelopmentStorage=true” />
   </appSettings>
</configuration>
```

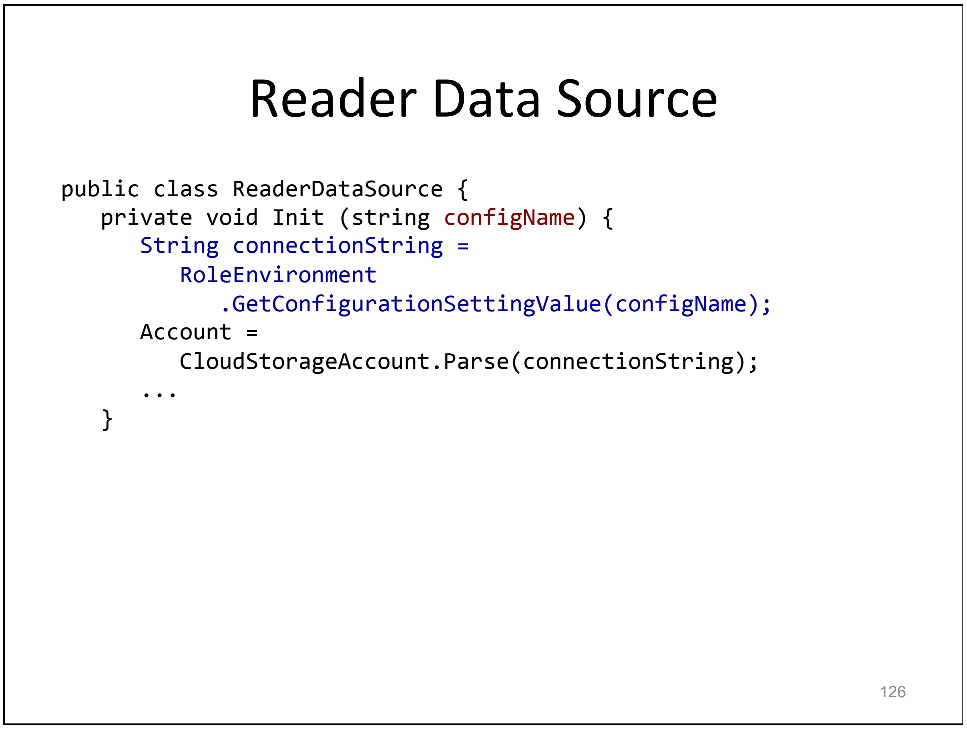
124



Reader Data Source

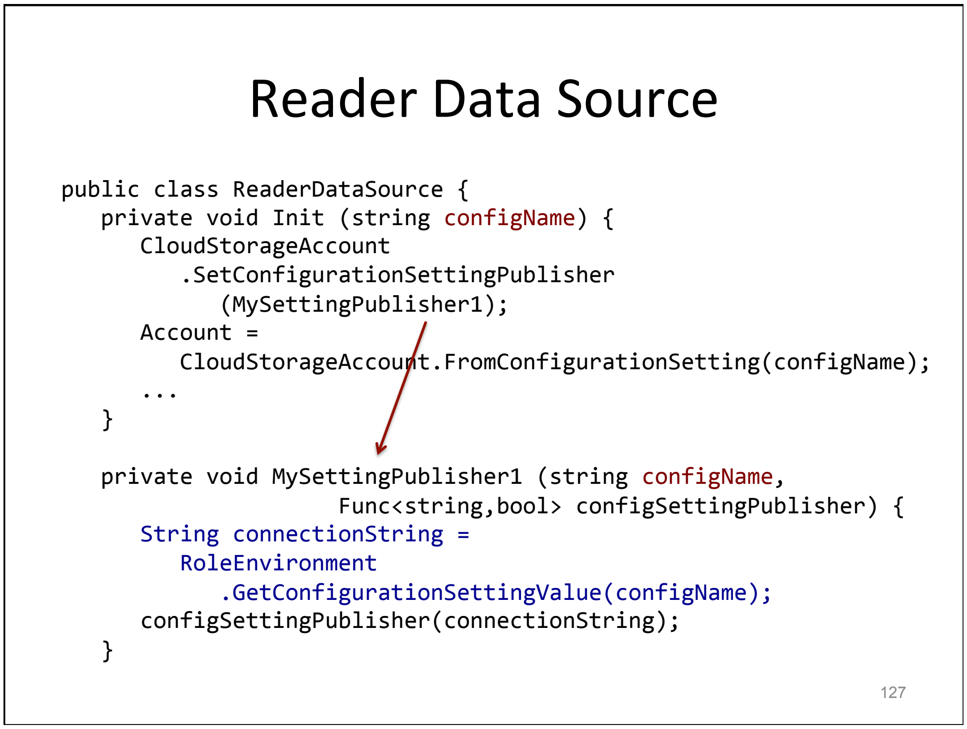
```
public class ReaderDataSource {
   private void Init (string configName) {
      String connectionString =
         ConfigurationManager
            .ConnectionStrings[configName].ConnectionString;
      Account =
         CloudStorageAccount.Parse(connectionString);
      ...
   }
```

```
<configuration>
   <appSettings>
      <add key=”MyConnectionString”
           value=“UseDevelopmentStorage=true” />
   </appSettings>
</configuration>
```



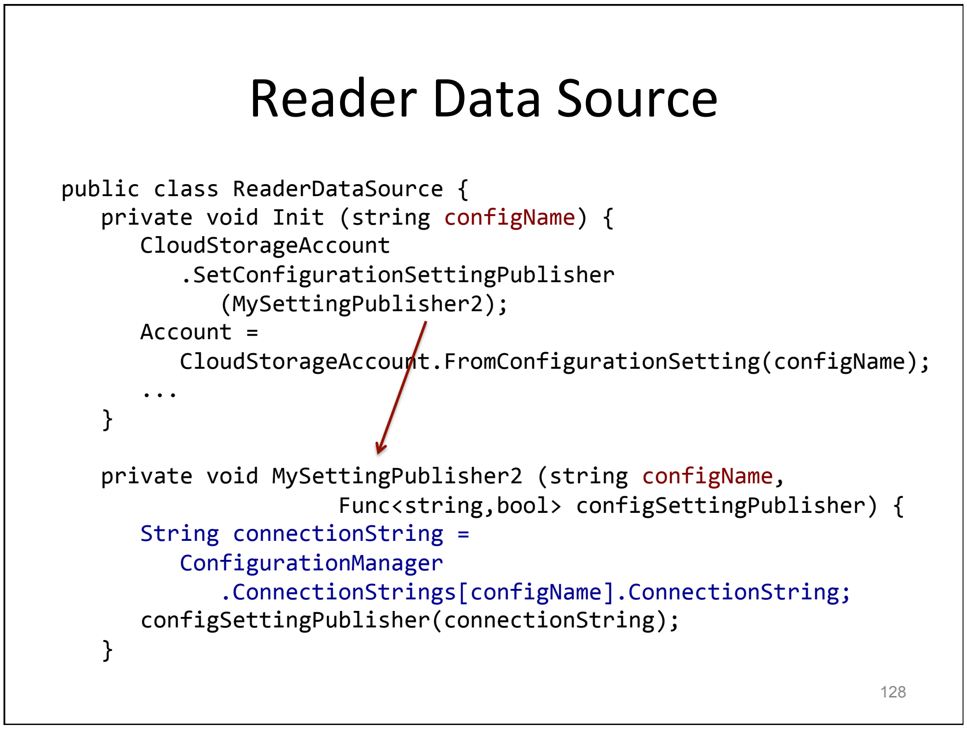
Reader Data Source

```
public class ReaderDataSource {
   private void Init (string configName) {
      String connectionString =
         RoleEnvironment
            .GetConfigurationSettingValue(configName);
      Account =
         CloudStorageAccount.Parse(connectionString);
      ...
   }
```



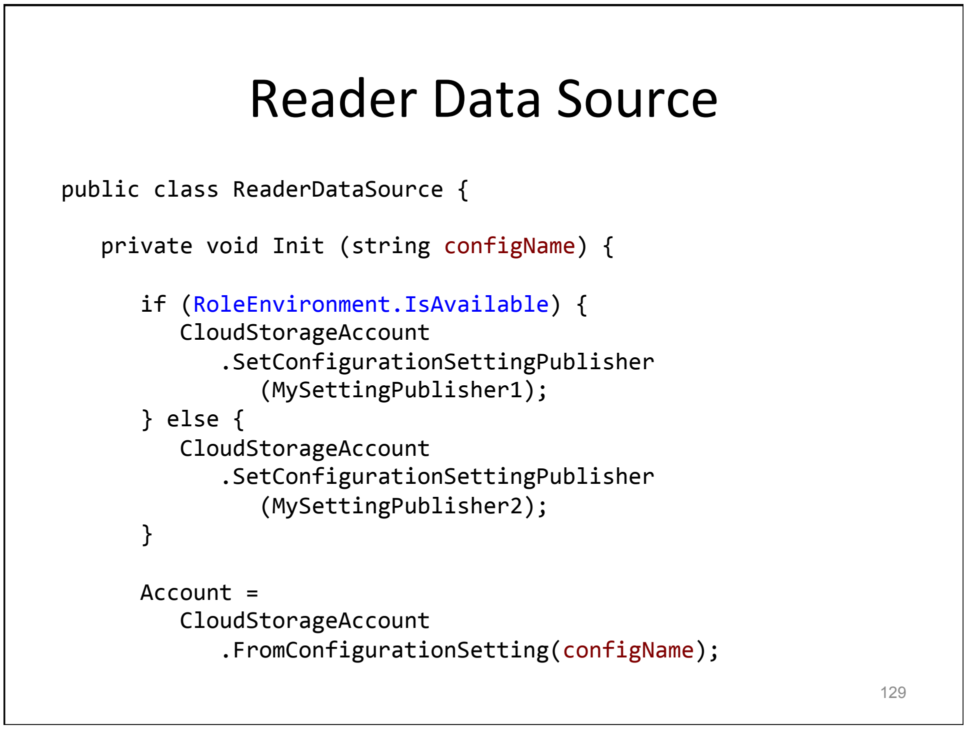
Reader Data Source

```
public class ReaderDataSource {
   private void Init (string configName) {
      CloudStorageAccount
         .SetConfigurationSettingPublisher
            (MySettingPublisher1);
      Account =
         CloudStorageAccount.FromConfigurationSetting(configName);
      ...
   }

   private void MySettingPublisher1 (string configName,
                        Func<string,bool> configSettingPublisher) {
      String connectionString =
         RoleEnvironment
            .GetConfigurationSettingValue(configName);
      configSettingPublisher(connectionString);
   }
```



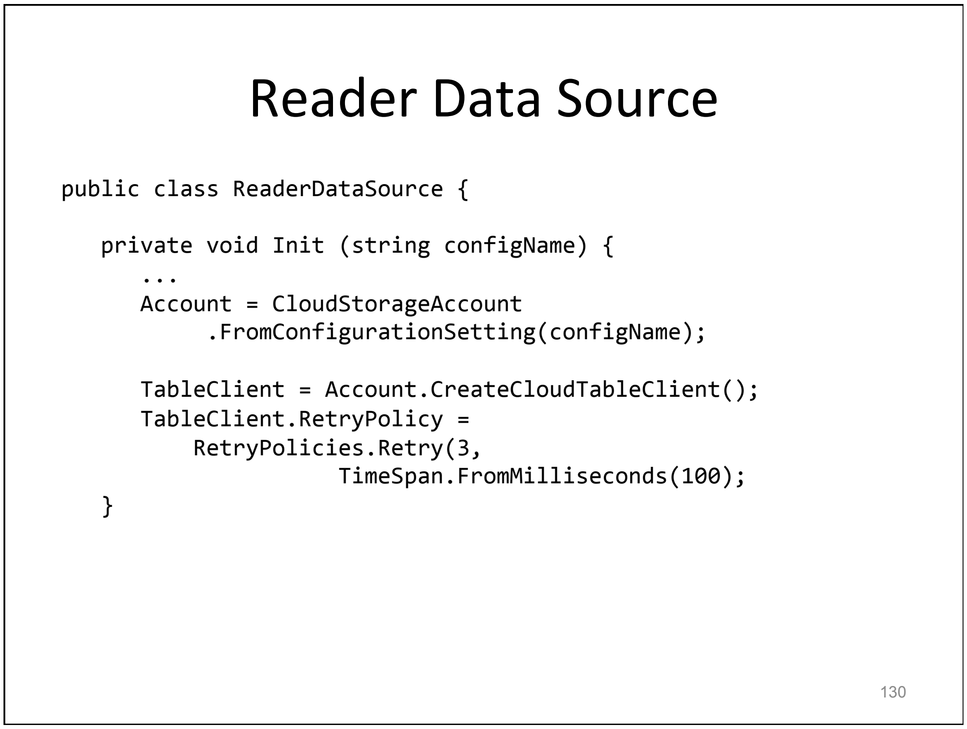
Reader Data Source

```
public class ReaderDataSource {
   private void Init (string configName) {
      CloudStorageAccount
         .SetConfigurationSettingPublisher
            (MySettingPublisher2);
      Account =
         CloudStorageAccount.FromConfigurationSetting(configName);
      ...
   }

   private void MySettingPublisher2 (string configName,
                        Func<string,bool> configSettingPublisher) {
      String connectionString =
         ConfigurationManager
            .ConnectionStrings[configName].ConnectionString;
      configSettingPublisher(connectionString);
   }
```



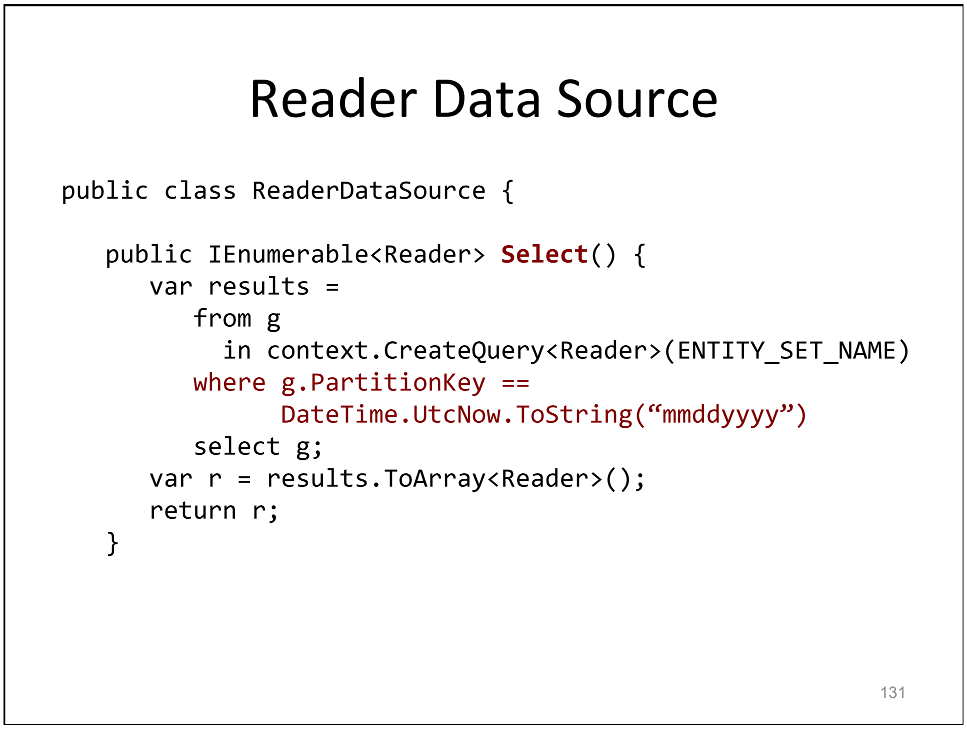
Reader Data Source

```
public class ReaderDataSource {

   private void Init (string configName) {

      if (RoleEnvironment.IsAvailable) {
         CloudStorageAccount
            .SetConfigurationSettingPublisher
               (MySettingPublisher1);
      } else {
         CloudStorageAccount
            .SetConfigurationSettingPublisher
               (MySettingPublisher2);
      }

      Account =
         CloudStorageAccount
            .FromConfigurationSetting(configName);
```



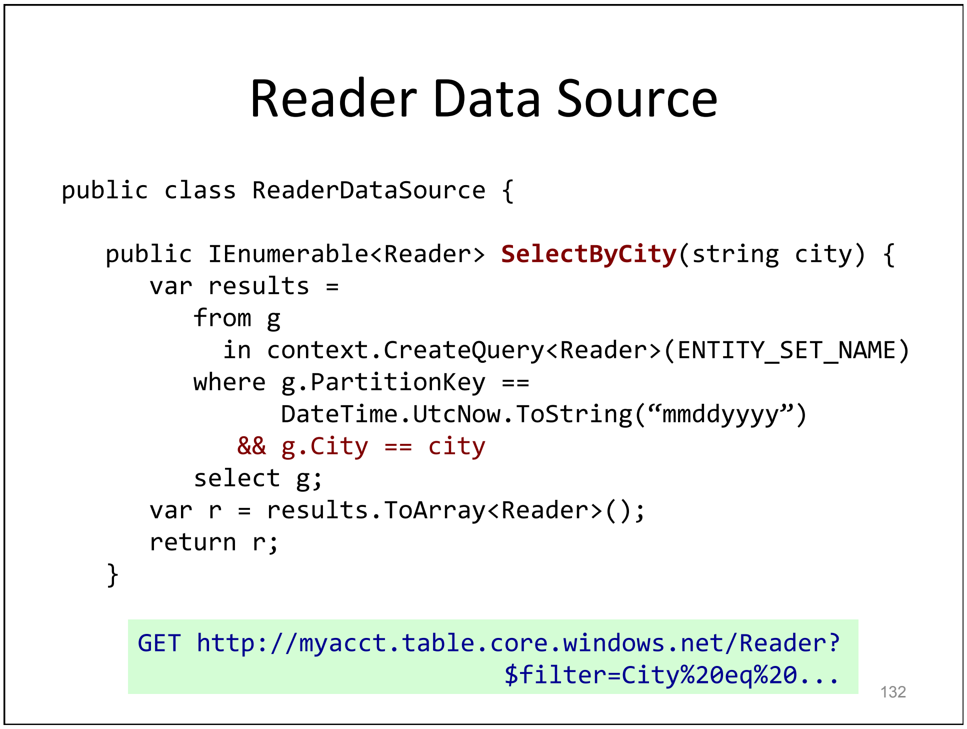
Reader Data Source

```
public class ReaderDataSource {

   private void Init (string configName) {
      ...
      Account = CloudStorageAccount
           .FromConfigurationSetting(configName);

      TableClient = Account.CreateCloudTableClient();
      TableClient.RetryPolicy =
          RetryPolicies.Retry(3,
                     TimeSpan.FromMilliseconds(100);
   }
```



Reader Data Source

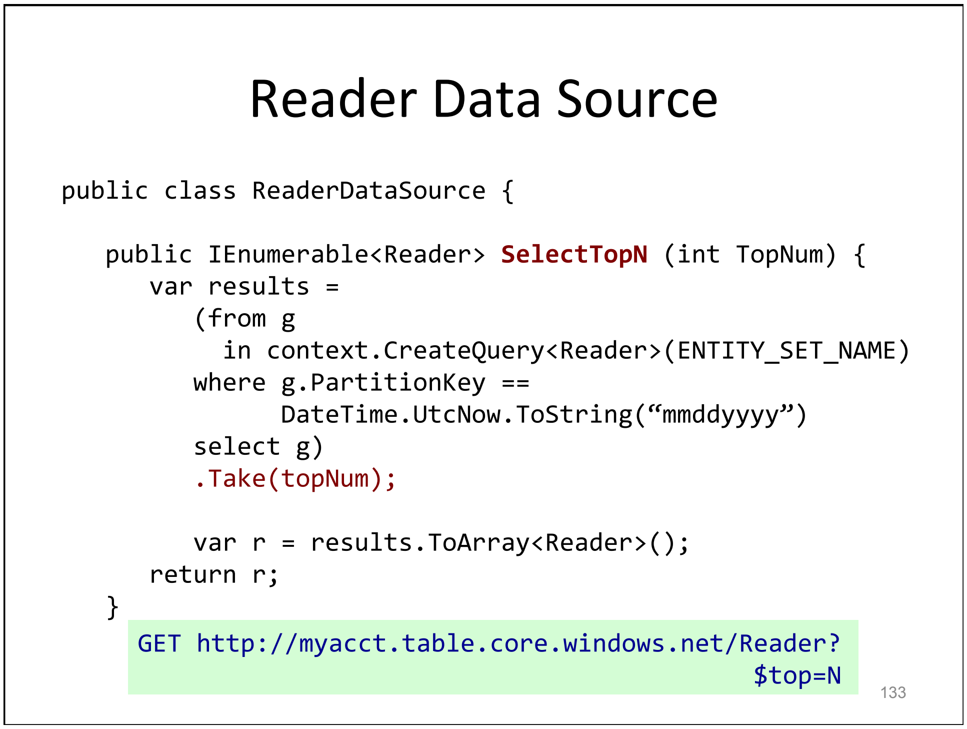
```
public class ReaderDataSource {

   public IEnumerable<Reader> Select() {
      var results =
         from g
           in context.CreateQuery<Reader>(ENTITY_SET_NAME)
         where g.PartitionKey ==
               DateTime.UtcNow.ToString(“mmddyyyy”)
         select g;
      var r = results.ToArray<Reader>();
      return r;
   }
```



Reader Data Source

```
public class ReaderDataSource {

   public IEnumerable<Reader> SelectByCity(string city) {
      var results =
         from g
           in context.CreateQuery<Reader>(ENTITY_SET_NAME)
         where g.PartitionKey ==
               DateTime.UtcNow.ToString(“mmddyyyy”)
            && g.City == city
         select g;
      var r = results.ToArray<Reader>();
      return r;
   }
```

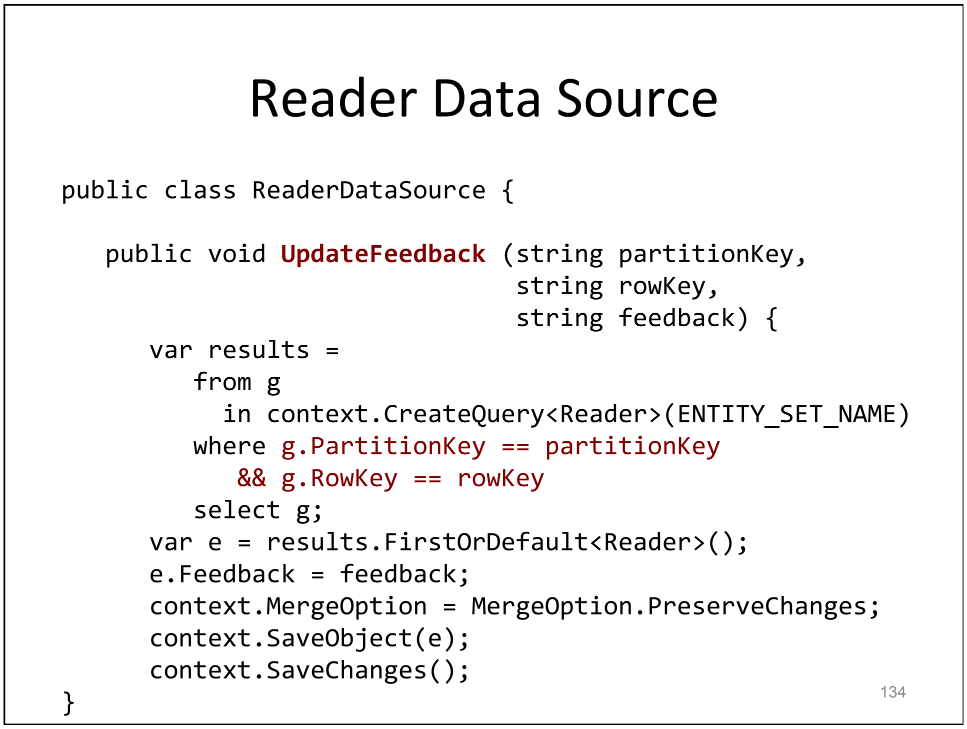
```
GET http://myacct.table.core.windows.net/Reader?
                         $filter=City%20eq%20...
```



Reader Data Source

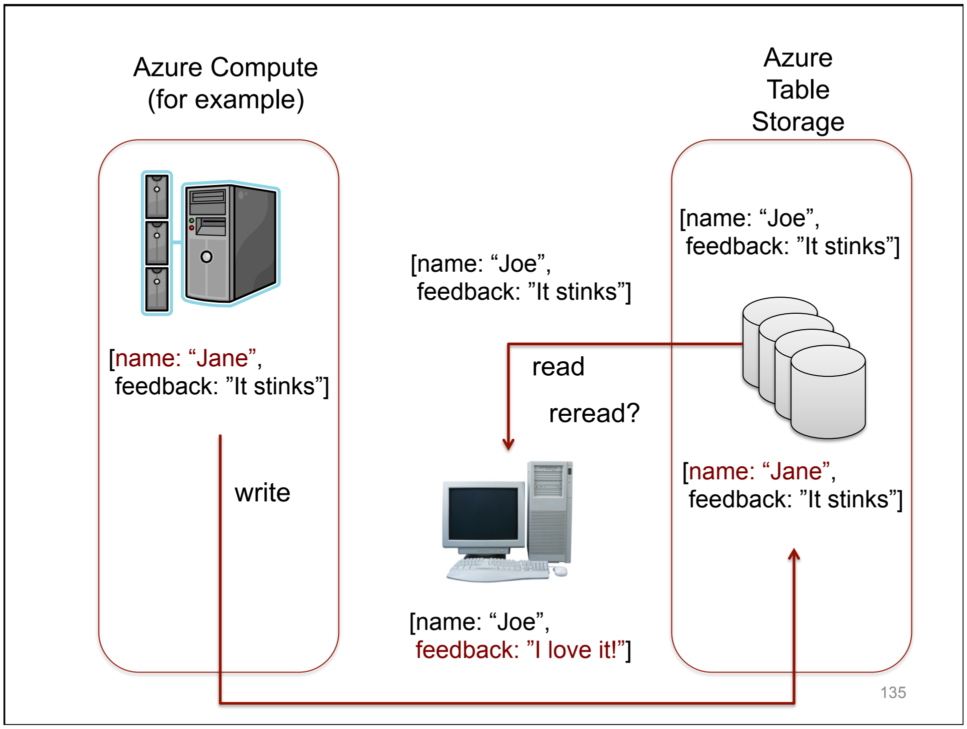
```
public class ReaderDataSource {

   public IEnumerable<Reader> SelectTopN (int TopNum) {
      var results =
         (from g
           in context.CreateQuery<Reader>(ENTITY_SET_NAME)
         where g.PartitionKey ==
               DateTime.UtcNow.ToString(“mmddyyyy”)
         select g)
         .Take(topNum);

         var r = results.ToArray<Reader>();
      return r;
   }
```

```
GET http://myacct.table.core.windows.net/Reader?
                                         $top=N
```



Reader Data Source

```
public class ReaderDataSource {

   public void UpdateFeedback (string partitionKey,
                               string rowKey,
                               string feedback) {
      var results =
         from g
           in context.CreateQuery<Reader>(ENTITY_SET_NAME)
         where g.PartitionKey == partitionKey
            && g.RowKey == rowKey
         select g;
      var e = results.FirstOrDefault<Reader>();
      e.Feedback = feedback;
      context.MergeOption = MergeOption.PreserveChanges;
      context.SaveObject(e);
      context.SaveChanges();
}
```



Azure Compute
(for example)

[name: “Jane”,
feedback: ”It stinks”]

write

[name: “Joe”,
feedback: ”It stinks”]

read

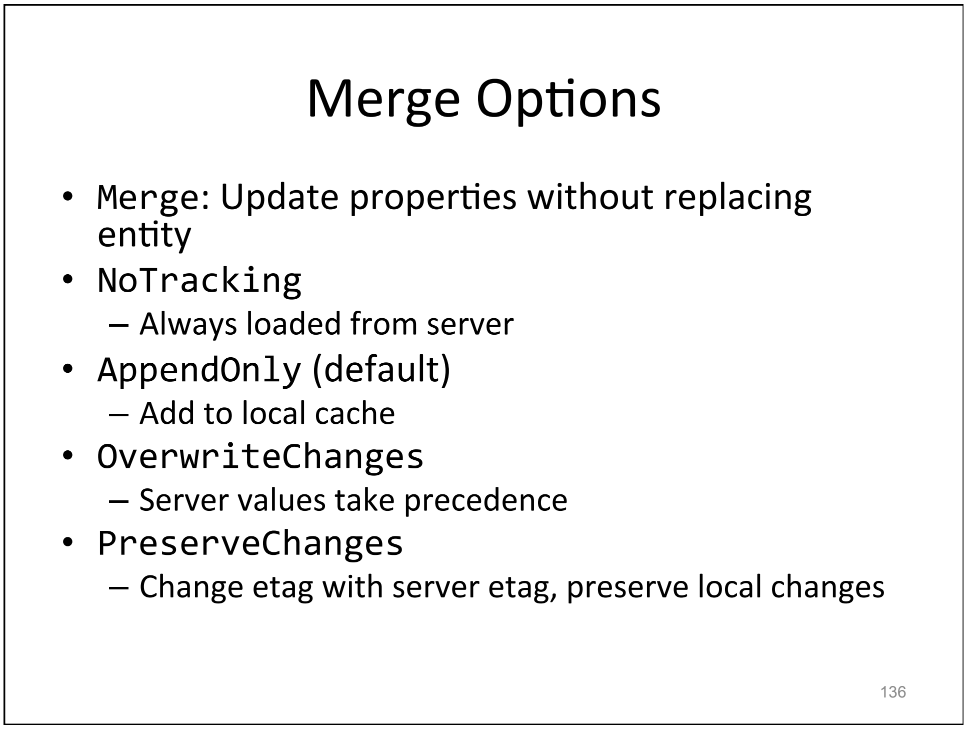
reread?

[name: “Joe”,
feedback: ”I love it!”]

Azure
Table
Storage

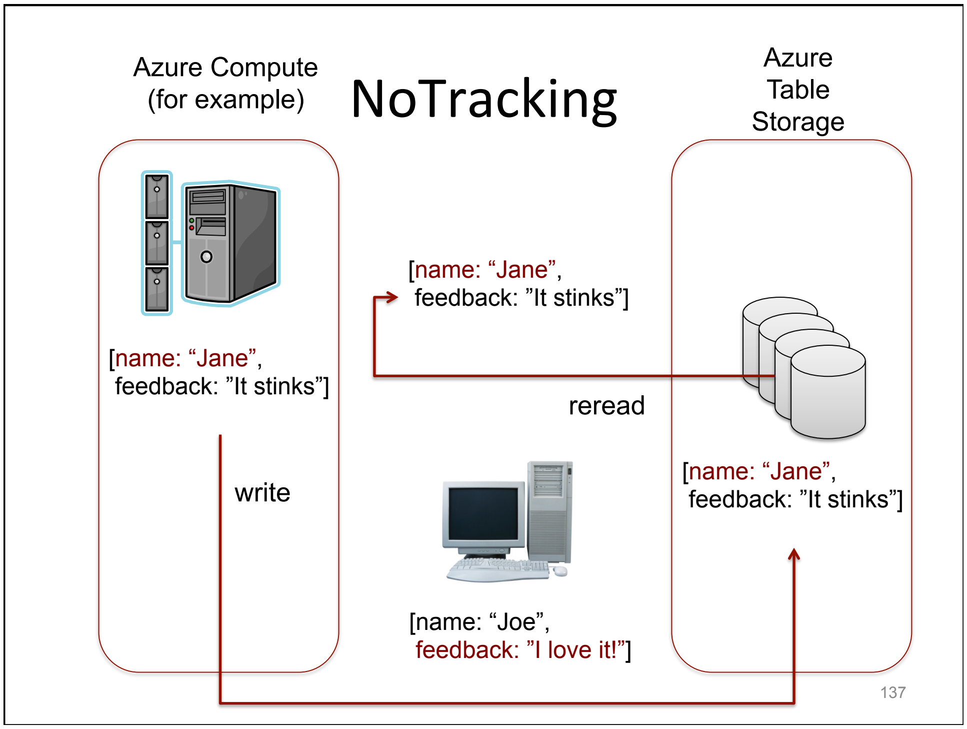
[name: “Joe”,
feedback: ”It stinks”]

[name: “Jane”,
feedback: ”It stinks”]



Merge Options

- Merge: Update properties without replacing entity
- NoTracking
  - Always loaded from server
- AppendOnly (default)
  - Add to local cache
- OverwriteChanges
  - Server values take precedence
- PreserveChanges
  - Change etag with server etag, preserve local changes



NoTracking

Azure Compute
(for example)

Azure
Table
Storage

[name: “Jane”,
feedback: ”It stinks”]

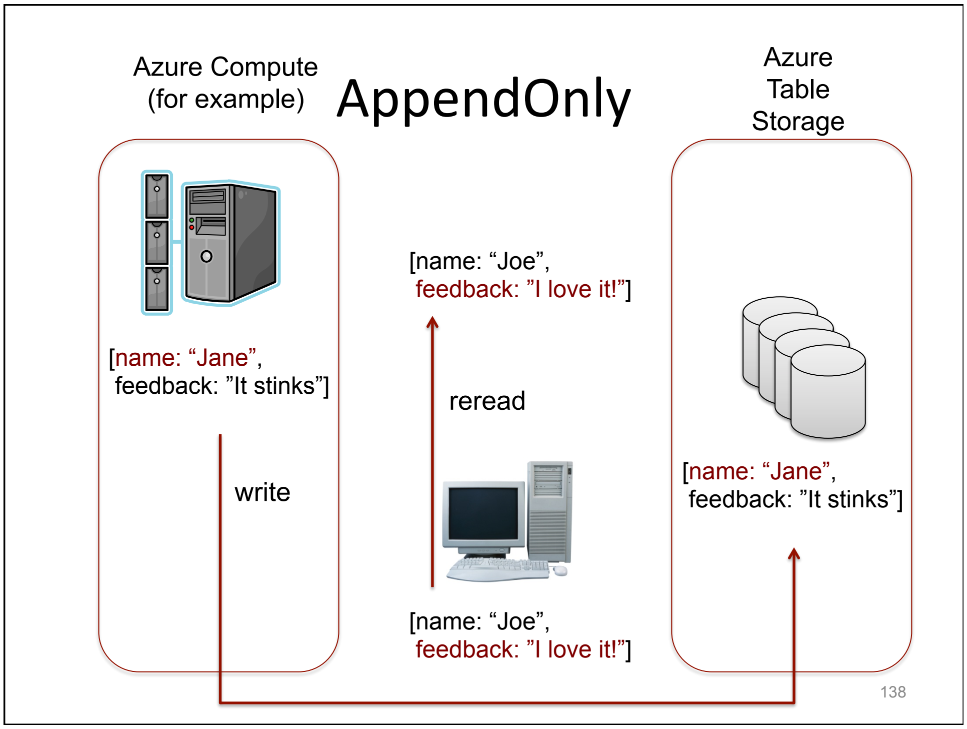
write

[name: “Jane”,
feedback: ”It stinks”]

reread

[name: “Jane”,
feedback: ”It stinks”]

[name: “Joe”,
feedback: ”I love it!”]



AppendOnly

Azure Compute
(for example)

Azure
Table
Storage

[name: “Jane”,
feedback: ”It stinks”]

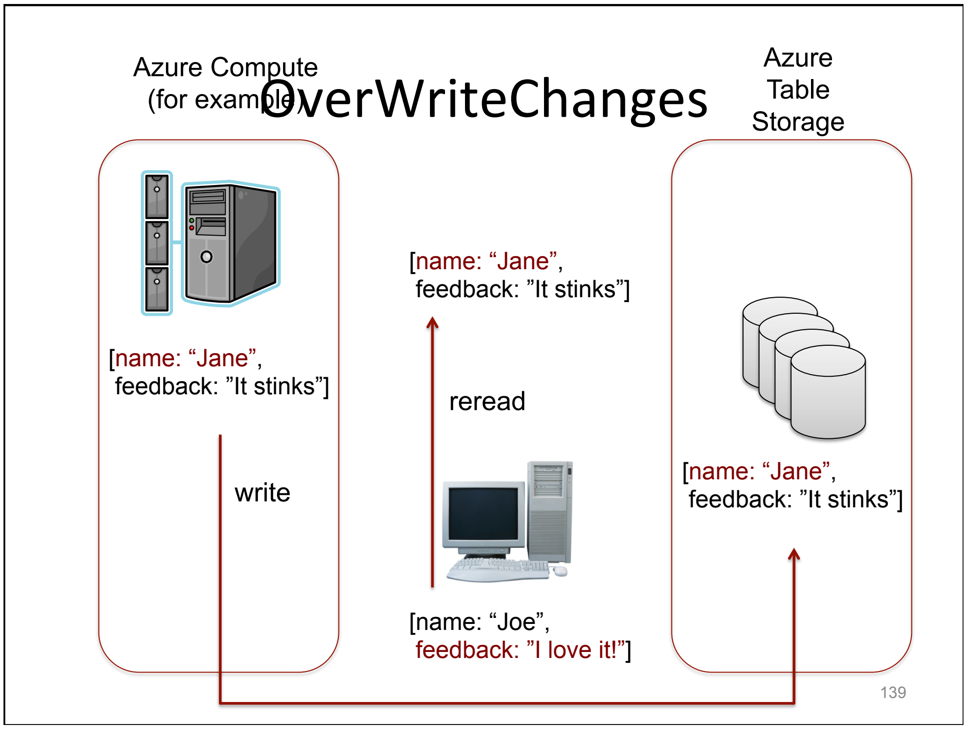
write

[name: “Joe”,
feedback: ”I love it!”]

reread

[name: “Joe”,
feedback: ”I love it!”]

[name: “Jane”,
feedback: ”It stinks”]



OverWriteChanges

Azure Compute
(for example)

Azure
Table
Storage

[name: “Jane”,
feedback: ”It stinks”]

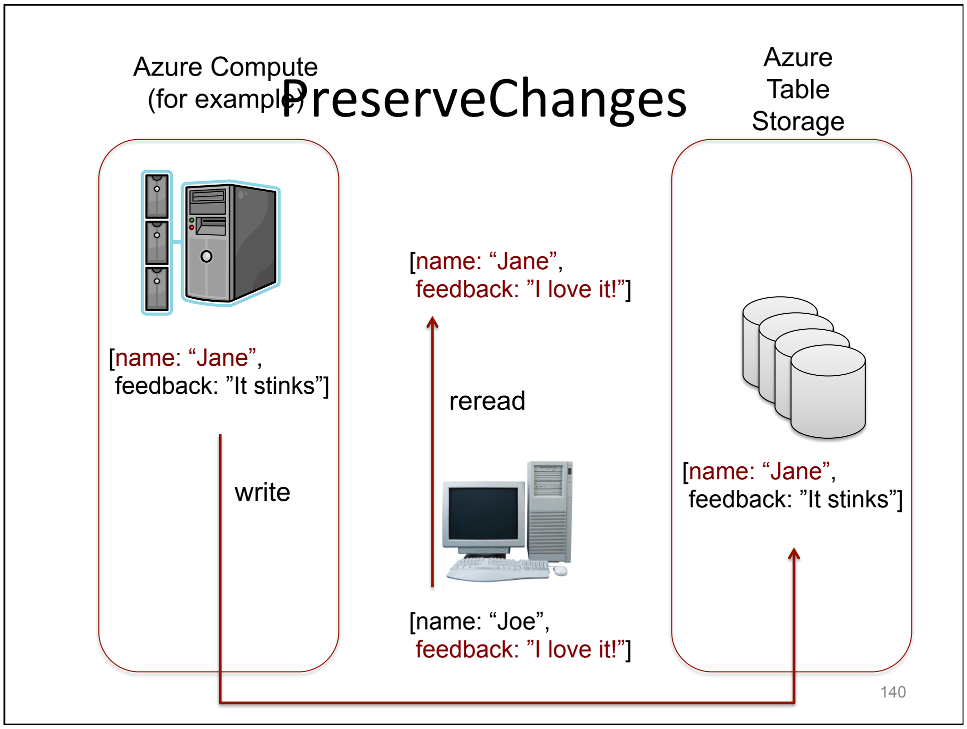
write

[name: “Jane”,
feedback: ”It stinks”]

reread

[name: “Joe”,
feedback: ”I love it!”]

[name: “Jane”,
feedback: ”It stinks”]



PreserveChanges

Azure Compute
(for example)

Azure
Table
Storage

[name: “Jane”,
feedback: ”It stinks”]

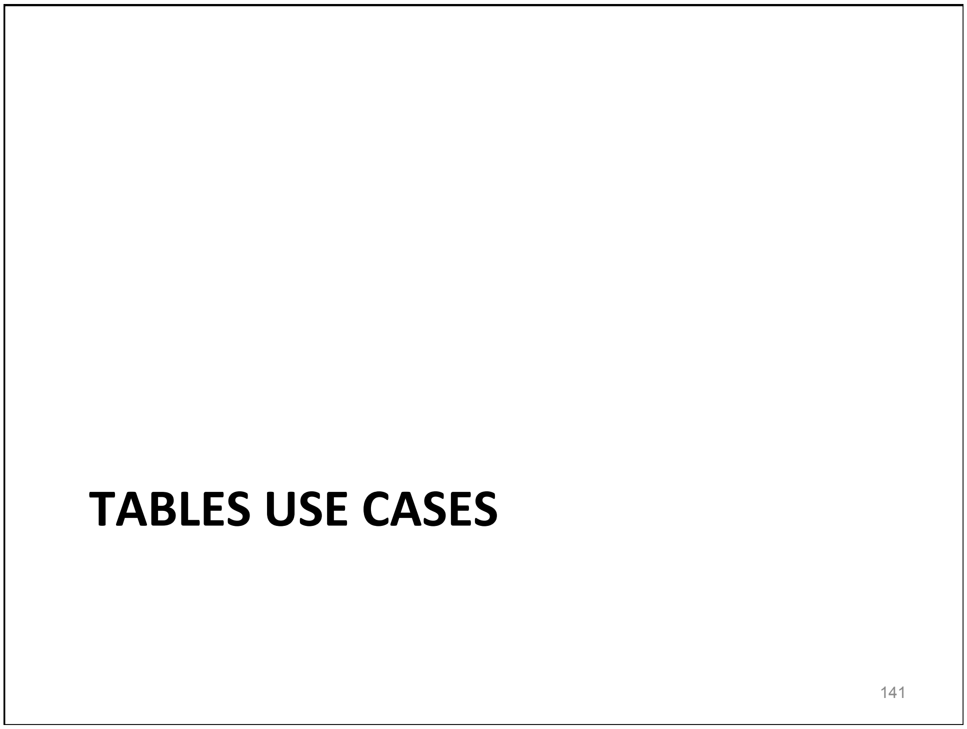
write

[name: “Jane”,
feedback: ”I love it!”]

reread

[name: “Joe”,
feedback: ”I love it!”]

[name: “Jane”,
feedback: ”It stinks”]



TABLES USE CASES

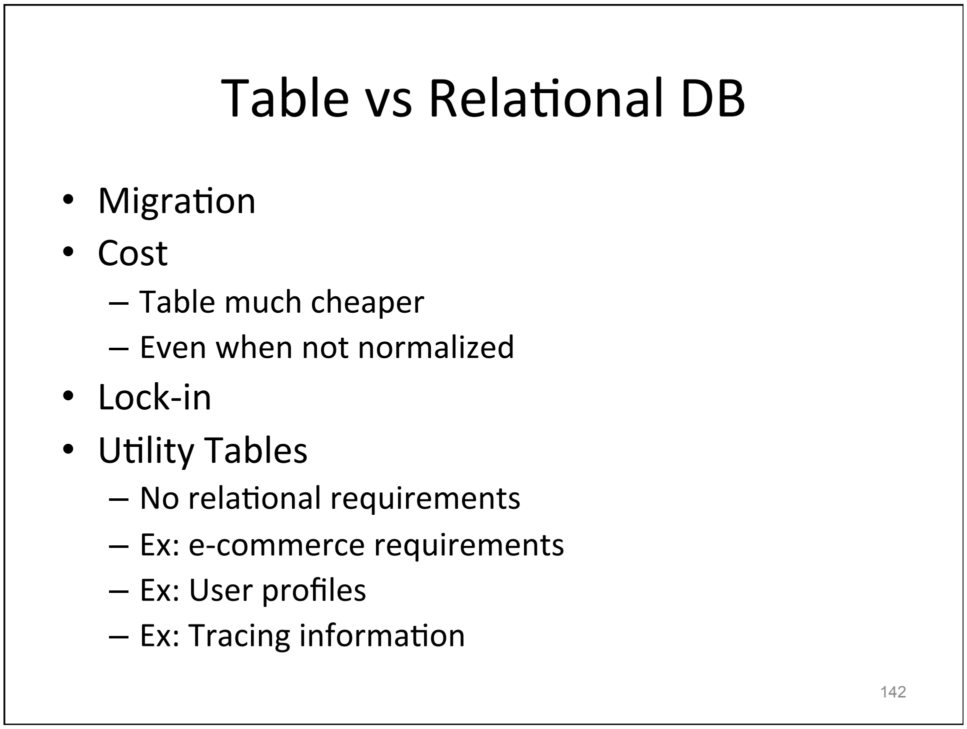


Table vs Relational DB

- Migration
- Cost
  - Table much cheaper
  - Even when not normalized
- Lock-in
- Utility Tables
  - No relational requirements
  - Ex: e-commerce requirements
  - Ex: User profiles
  - Ex: Tracing information

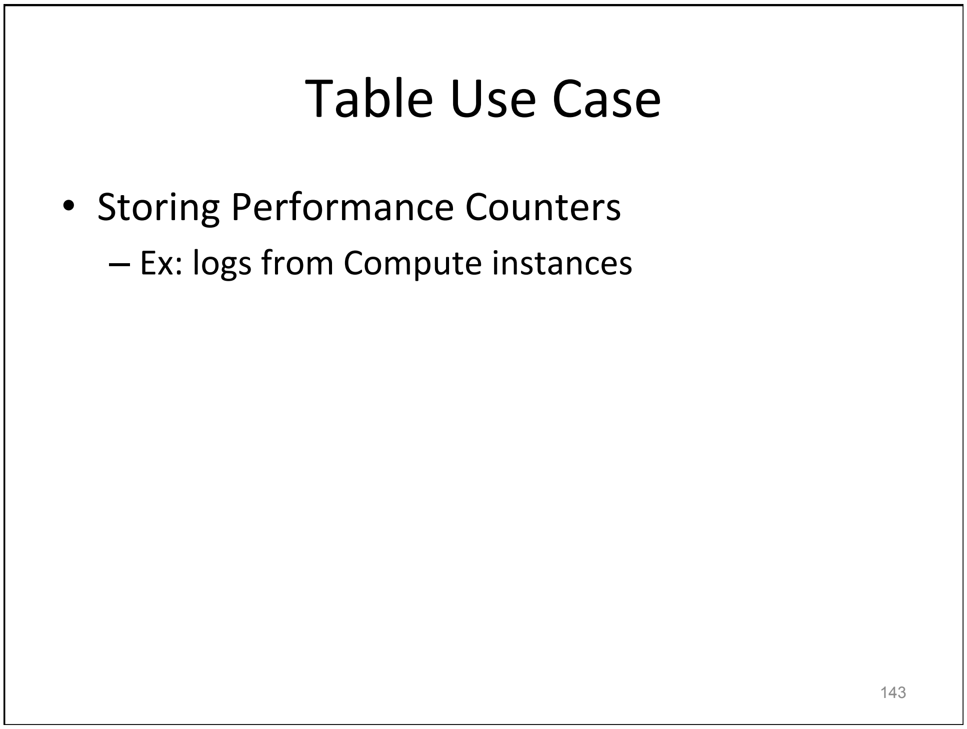
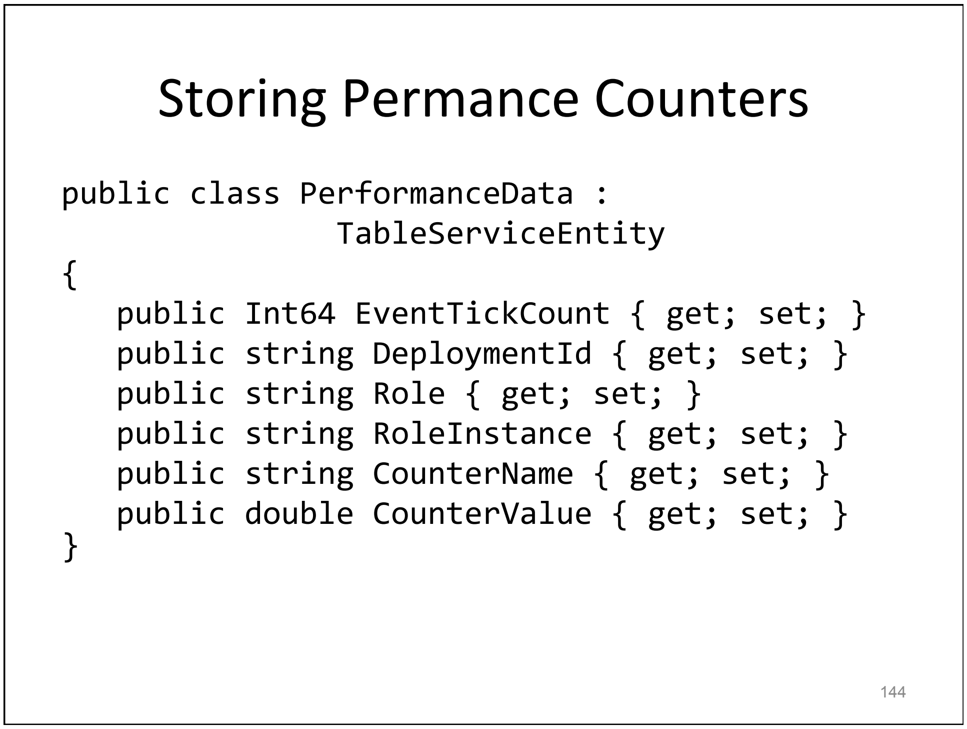


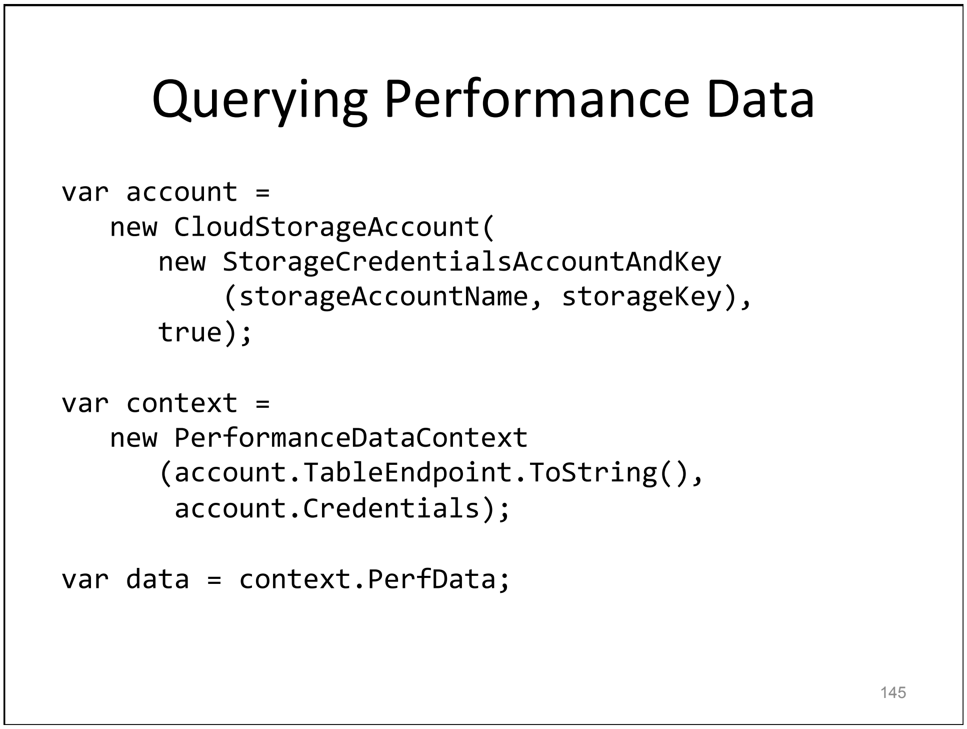
Table Use Case

- Storing Performance Counters
  - – Ex: logs from Compute instances



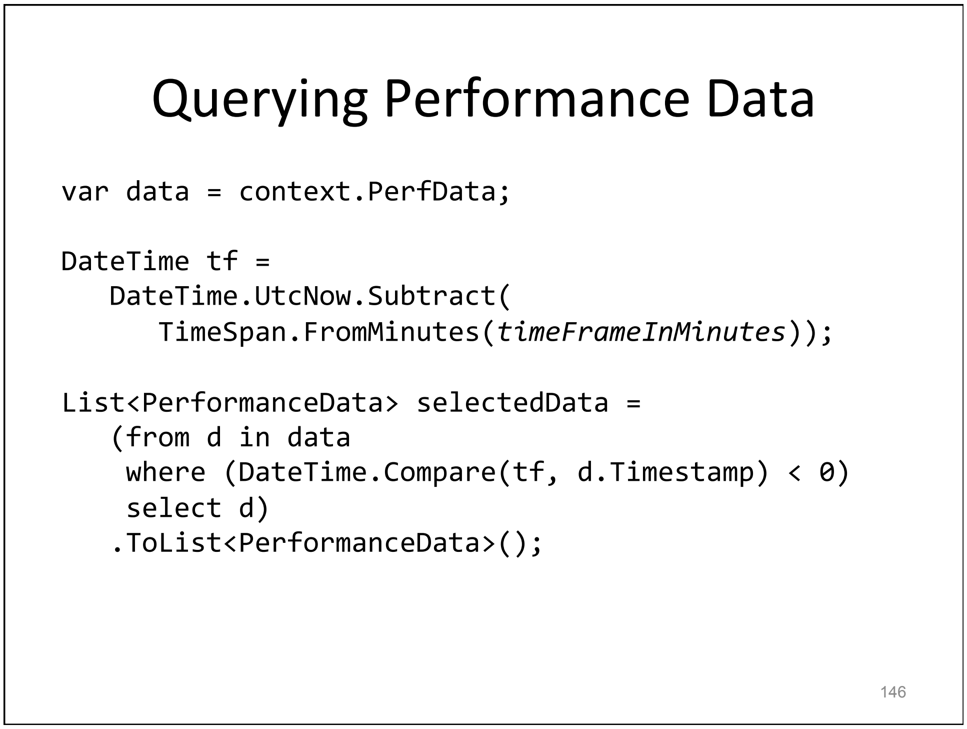
Storing Permance Counters

```
public class PerformanceData :
               TableServiceEntity
{
   public Int64 EventTickCount { get; set; }
   public string DeploymentId { get; set; }
   public string Role { get; set; }
   public string RoleInstance { get; set; }
   public string CounterName { get; set; }
   public double CounterValue { get; set; }
}
```



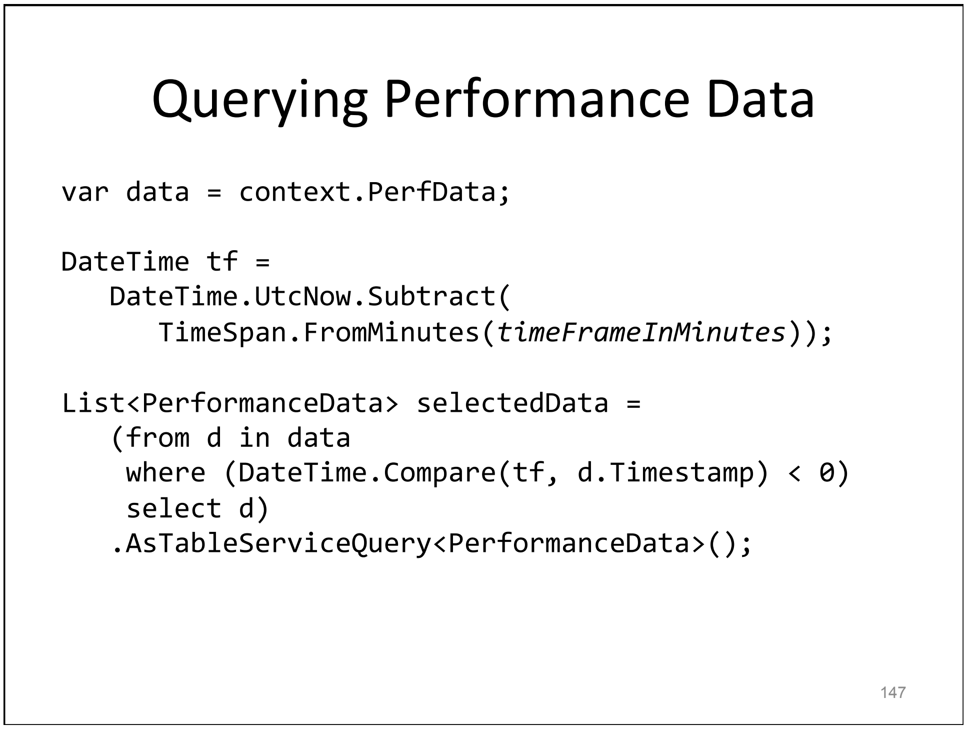
Querying Performance Data

```
var account =
   new CloudStorageAccount(
      new StorageCredentialsAccountAndKey
          (storageAccountName, storageKey),
      true);

var context =
   new PerformanceDataContext
      (account.TableEndpoint.ToString(),
       account.Credentials);

var data = context.PerfData;
```



Querying Performance Data

```
var data = context.PerfData;

DateTime tf =
   DateTime.UtcNow.Subtract(
      TimeSpan.FromMinutes(timeFrameInMinutes));

List<PerformanceData> selectedData =
   (from d in data
    where (DateTime.Compare(tf, d.Timestamp) < 0)
    select d)
   .ToList<PerformanceData>();
```



Querying Performance Data

```
var data = context.PerfData;

DateTime tf =
   DateTime.UtcNow.Subtract(
      TimeSpan.FromMinutes(timeFrameInMinutes));

List<PerformanceData> selectedData =
   (from d in data
    where (DateTime.Compare(tf, d.Timestamp) < 0)
    select d)
   .AsTableServiceQuery<PerformanceData>();
```