The road to “Developing Microsoft Azure and Web Services”

# MVA: Developing Windows Azure and Web Services Jump Start

## WCF Web Services

* blogs.msdn.com/brunoterkaly
* bretstateham.com

**Windows Communication Foundation (WCF)**  
The WCF is a runtime and a set of APIs in the .NET Framework for building connected, service-oriented applications.

* Perfomr read-write operations on back end databases
* Perform operations by calling other web services
* Abstract blocking I/O operations affects performance and scalability.

SOAP VS REST(Representational State Transfer)

Advantages of REST

* Automatic support – native http
* Lightweight, Efficient
* Secure – supports https
* Modern – Twitter, Yahoo etc
* No toolkits needed, XML format

Advantages of WCF + SOAP

* provides a lot of support for many bindings

With WCF you can write the service/code and then open up for different types of bindings in the config-file (Data contract)

**Data Contracts:** a formal agreement between a service and a client. It describes the data to be exchanged. To communicate, the client and the service do not have to share the same types, only the same data contracts. A data contract precisely defines, for each parameter or return type, what data is serialized (turned into XML) to be exchanged.

Decorate the service class and/or data members with attributes to tell the compiler and/or runtime to provide additional functionalities to support the code.

**Interfaces:** put the interfaces into a separate assembly (project). That way it is stored centrally and the client can see the interface and the server. Increases the ability to share it. This is useful because the interface can be used by any piece of code that need to know about the interface, but not necessarily about the implementation. The client does not need to know what happens inside the method call, they just need to know the interface, the way it communicates with that service. It is common to have one class implement more than one interface. The attributes can be defined in the interface, and the class will not need them.

**Configuring endpoints in the config (the ABC):** each endpoint consists of four properties. An **address** that indicates where the endpoint can be found (where am I), a **binding** that specifies how a client can communicate with the endpoint (How can you talk to me? http, tcp, basic), a **contract** that identifies the operations available (What can I do for you? interface or class) and a **set of behaviors** that specify local implementation details of the endpoint.

* A duplex service contract is a message exchange pattern in which both endpoints can send messages to the other idependently.

**Hosting WCF**

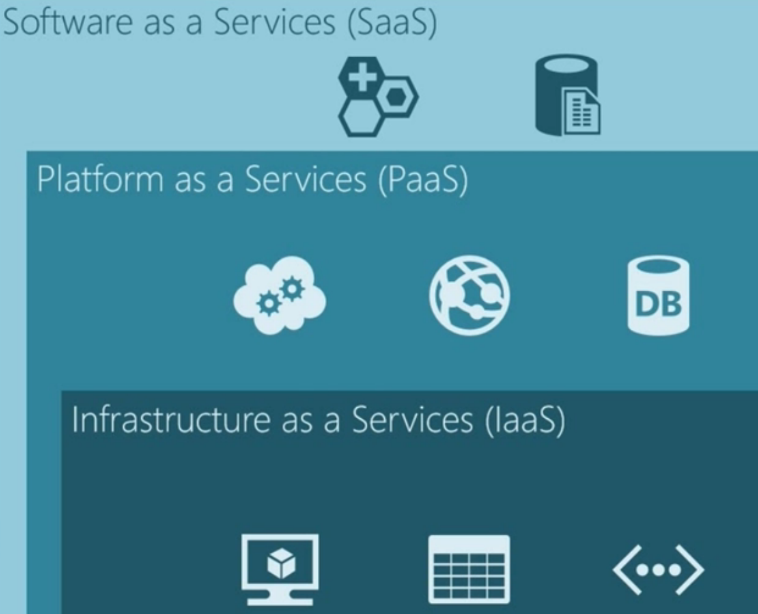
* IIS  
  Makes your service available by all clients that have access to the IIS server. Takes full advantage of IIS features: process recycling, idle shutdown, process health monitoring, message-based activation.
* Windows Service  
  Is broadly supported. Starts up automatically every time your system boots up, and you can control its life time with the service control manager.
* Managed .NET App  
  Hosting a service in a managed application is the most flexible option. It requires the least infrastructure to deploy. It is also the least robust hosting option. Managed apps do not provide the advanced hosting and management features of other hosting options in WCF.

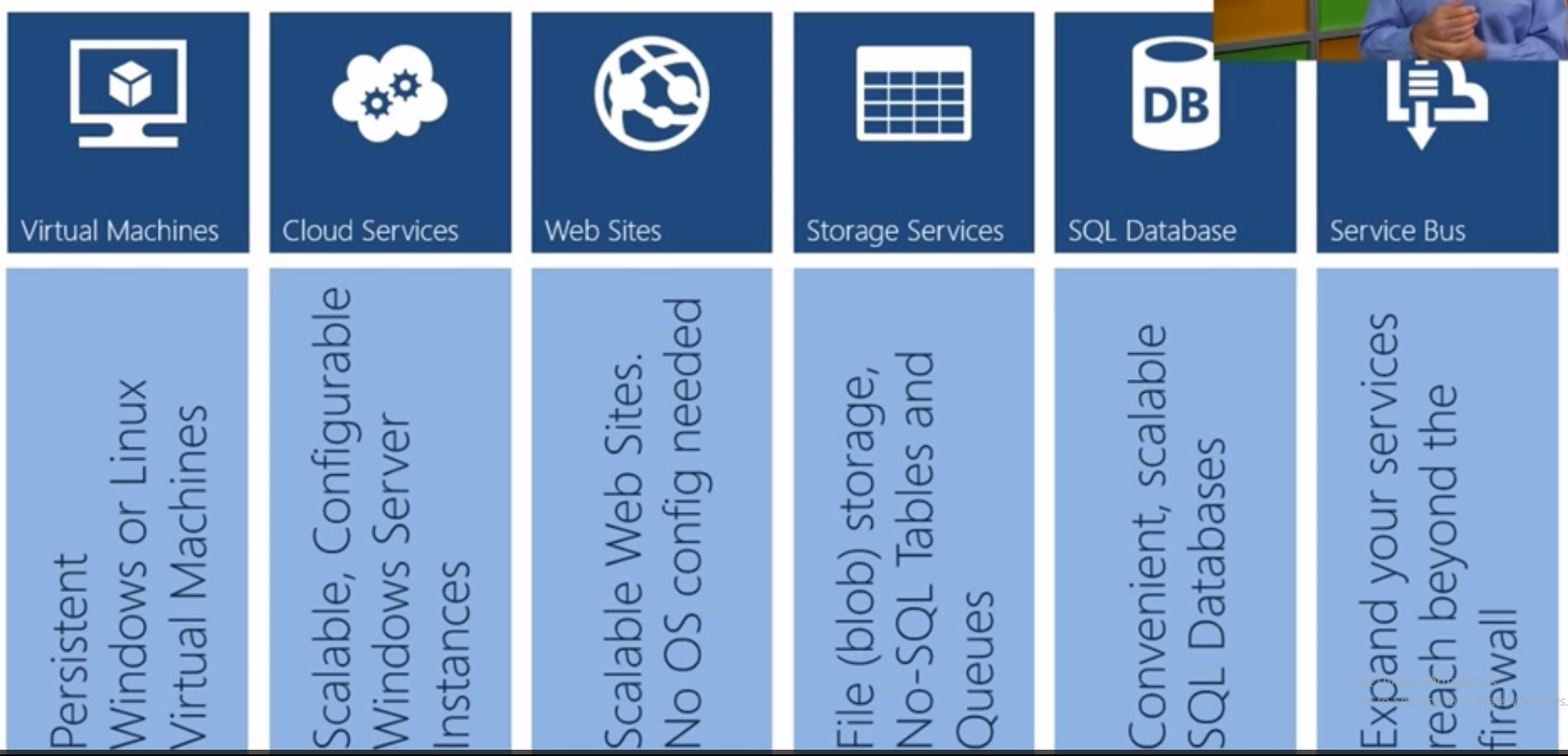
## Windows Azure Services

**Infrastructure as a Service (IaaS):** Azure Virtual Machines, data storage, network, etc is set up for you and you can use it “as your local computer”.

**Platform as a Service (PaaS):** more abstracted. You’re not logging in, you’re not running scripts, you’re just publishing from VS. You get the benefits of “no longer caring for your machine”; Upgrades happen automatically, you do not have to perform hot fixes or service packs etc. Cloud Services (cloud services are still virtual machines), Web Sites, SQL Databases.

**Software as a Service (SaaS):** Either you are providing the SaaS or you are consuming the SaaS. Azure has third party services. Add-Ons2, SQL Reporting



**Azure from a web service perspective**

**Windows Azure Web Sites:** powerful web sites in seconds.  
Supported publishing methods: FTP, TFS, WebDeploy,git, DropBox.

**A cloud service** is a collection of related service roles: web role, worker role. PaaS solution.  
Computer web / worker role VMs are stateless. When a VM is recycled, no data is preserved. Data in local storage (local to the VM) will be lost. Persist data in Table or Blob storage or SQL Azure.

Install Azure SDK to be able to create a cloud service in VS.

**Worker Roles:**

* Pre-configured Windows Server
* Nothing else installed
* No default Azure endpoints
* Run customer workloads that don’t require IIS
* Install additional software, etc using scripts
* Implement Logic in the WorkerRole.cs class

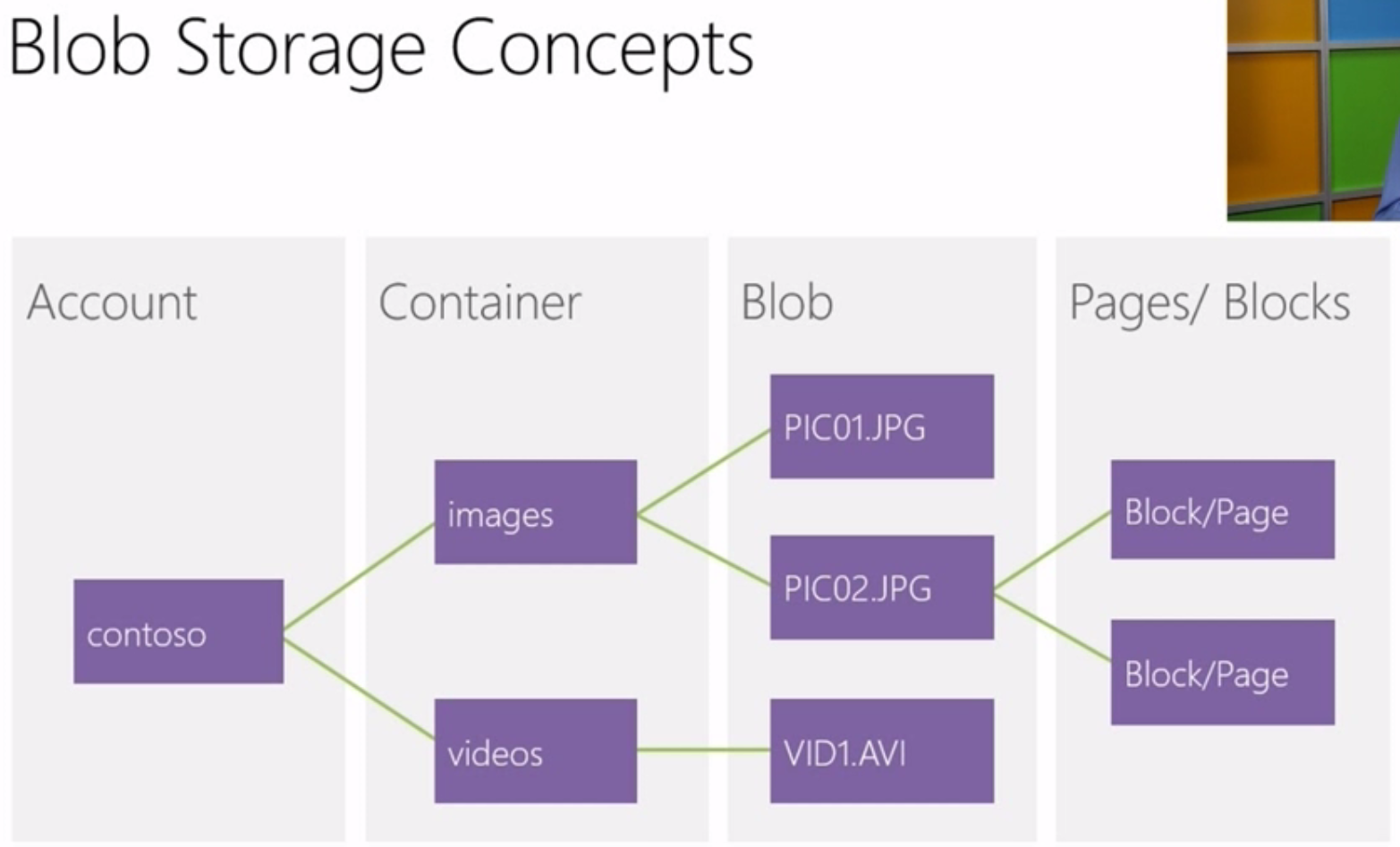
**WCF Web Service in Worker Roles**

* Not as easy as web roles
* You need to write code for the Service Host
* You need to create Azure Endpoints
* You can create the WCF endpoints in config
* Again, WorkerRole.cs is where you code

## Azure Storage and SQL Database

Storage in the Development Fabric

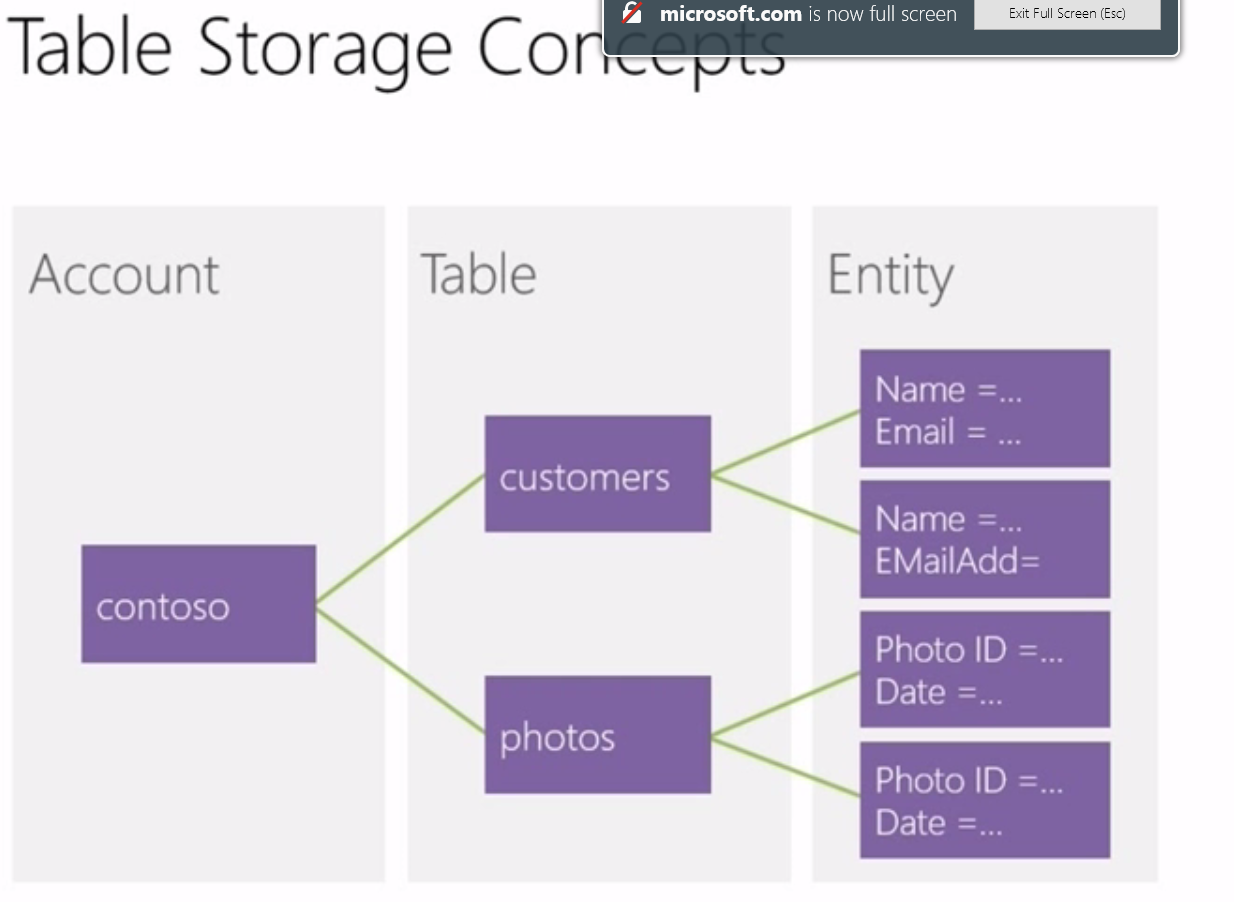
* Provides a local “Mock” storage
* Emulates storage in cloud
* Allows offline development
* Requires SQL Express 2005/2008 or above



**Two types of blobs under the hood**

* **Block Blob**  
  Targeted at streaming workloads  
  Each blob consists of a sequence of blocks  
  Size limit 200GB per blob
* **Page Blob**  
  Targeted at random read/write workloads  
  Each blob consists of an array of pages  
  Size limit 1TB per blob

**Shared access signatures:** fine grain access rights to blobs and containers. Sign URL with storage key – permit elevated rights. Revocation. Use short time periods and re-issue. Use container level policy that can be deleted.



**Entity Properties:** can have up to 255 properties (<=1MB pr entity). Mandatory Properties for every entity (PartitionKey & RowKey; uniquely identifies an entity, defines the sort order. Timestamp; Optimistic concurrency, exposed as an HTTP Etag). No fixed schema for other properties (<name, typed value> pair, can be standard .NET types).

**PartitionKey:** entity locality (entities in the same partition will be stored together). Entity Group Transactions (atomic multiple insert/update/delete in same partition in a single transaction). Table Scalability (target throughput – 500 tps/partition, several thousand tps/account. Automatically load balance partitions).

**Messaging**

**Queue:** Async communication, offline processing, load-balancing.

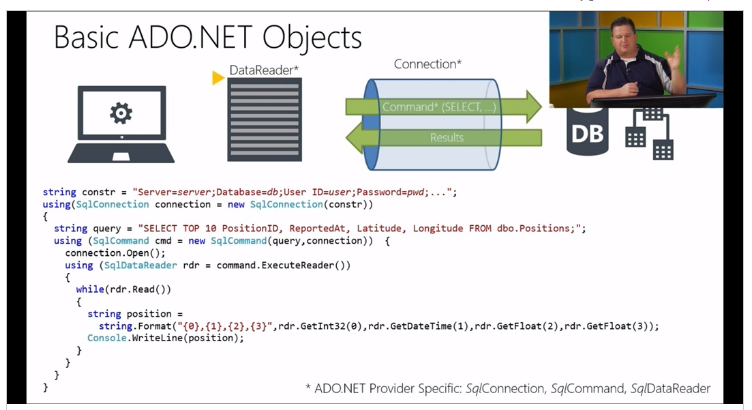
**Topic & Subscription:** Async communication, publish/subscription pattern, message routing

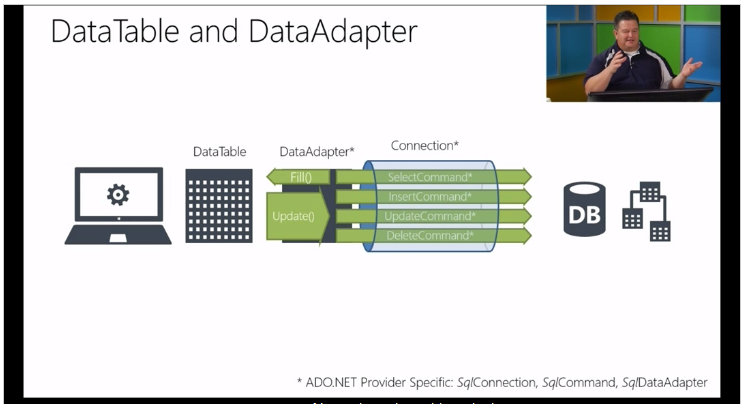
## Data Access Technologies

* ADO.NET, LINQ, XML, LINQ to XML.
* Connectionstrings.com

**ADO.NET**

* Use a using-statement to make sure that when you are done working with that SQL connection, it closes up. Most of the objects in ADO.NET are disposable. They implement the IDisposible interface and the using pattern is recommended practice when working with disposable objects.
* You have cache (store) the data yourself on the client

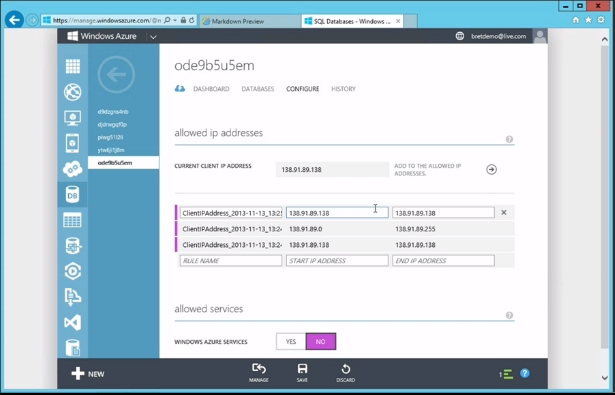




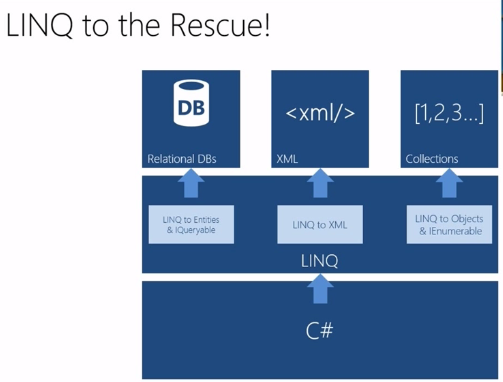
DataTable and DataSet is not provider specific

**DataSets:** almost like an in-memory database. The whole dataset is shipped to the client. This minimizes the network traffic, BUT it will pull a lot more data then what you need. This consumes a lot of resources on the client and therefore the DataSets is going out of fashion.

**SQL Databases in Azure:** same as with storage; you want the databases close to the applications that uses them to eliminate latency and egress data. You can only log on to the db server with SQL Authentication for now.  
Also, this server is available on the internet. It can be wise to turn off “allowed services” in addition to add your personal IP/IP-range.



**LINQ**



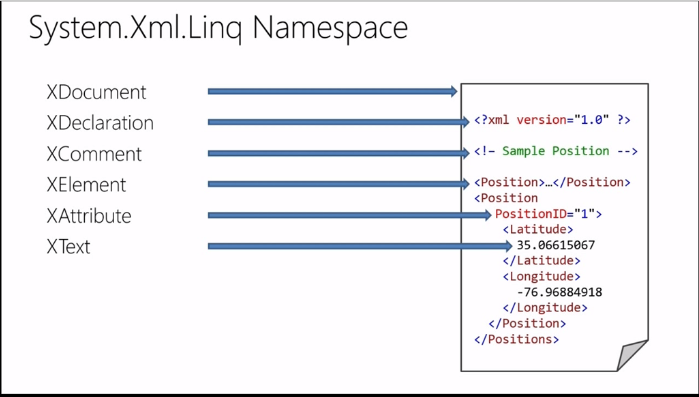
* LINQ is a way to query data in memory.
  + A collection of extension methods to IEnumerable
  + Queries are “composable” and don’t execute until data is accessed.
  + Allows querying, filtering, aggregating, joining, collections in memory
* LINQ is a way to ship queries across application layers
  + IQueriable represents the intention of the query (you only get what you ask for)
  + Can be shipped between application tiers (service <> client)
    - Service can provide the initial query
    - Client can further “compose” the query (filter, sort, etc)
    - Client passes the composed query back to the service
    - The service returns the data as specified by the query

**XML**

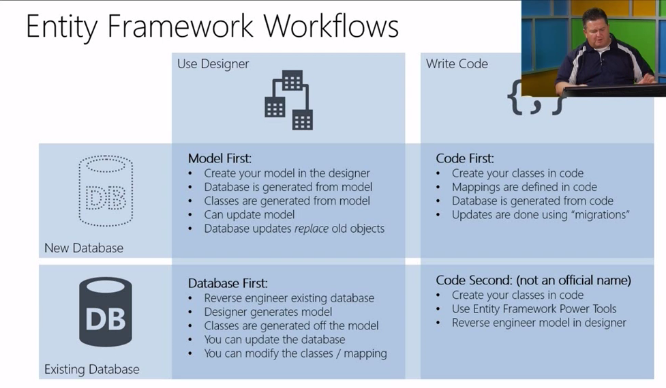
Working with XML in code

* System.Xml
  + XMLDocument for Document Object Model (DOM) based operations
  + XmlReader for SAX (streaming) based operations
* Create XML with XmlDocument, XmlElement, XmlAttribute…
* Can save/load XML from files
* Can parse XML from strings

**LINQ to XML**



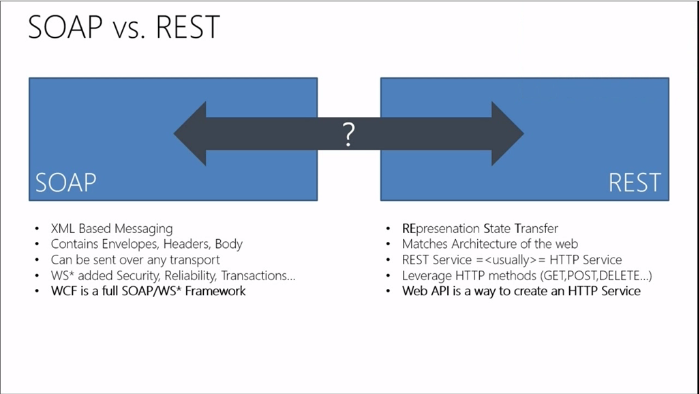
## Data Access with Entity Framework

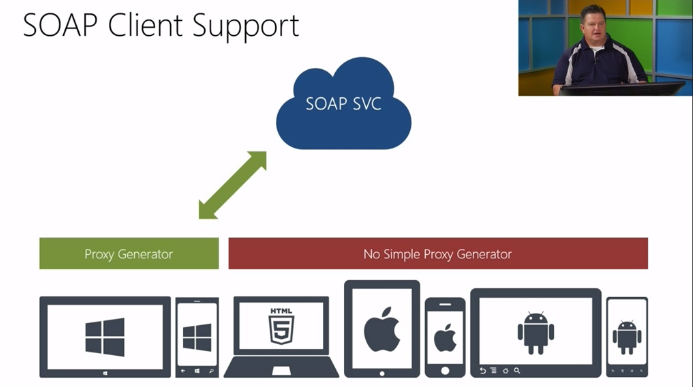


**OData:** a protocol for creating and consuming data APIs. Provides a standard way to use REST to expose & consume data. OData is a formalized REST API. It can be written on any platform, it can be against any set of data and it can be consumed from any client.

**WCF Data Services:** A convenient way of creating an OData service on top of a data model. Access data using URIs or a client library. NuGet: WCF Data Services Entity Framework

## Web API Services





* <http://www.asp.net/web-api>

**Creating a web API**

* Derive form ApiController
* Implement actions
* Actions map to HTTP methods (GET, POST, PUT, DELETE)
  + Either prefix your action method name with HTTP verb: **Get**Customer
  + Or use attributes if you want to name your action methods differently
    - [HttpGet]
    - [HttpPost]
    - [HttpPut]
    - [HttpDelete]

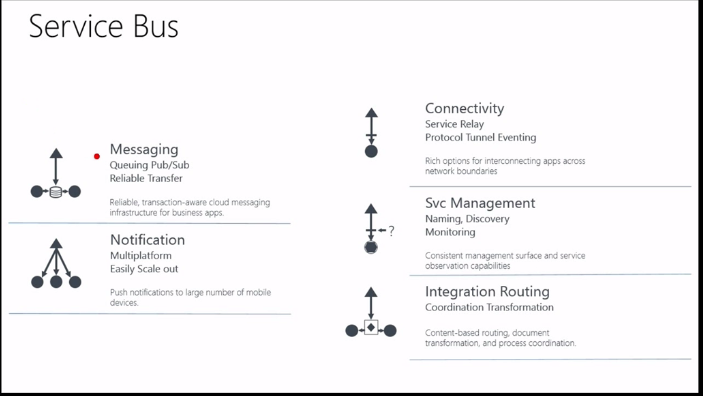
**Consuming Web API services / consuming HTTP clients**

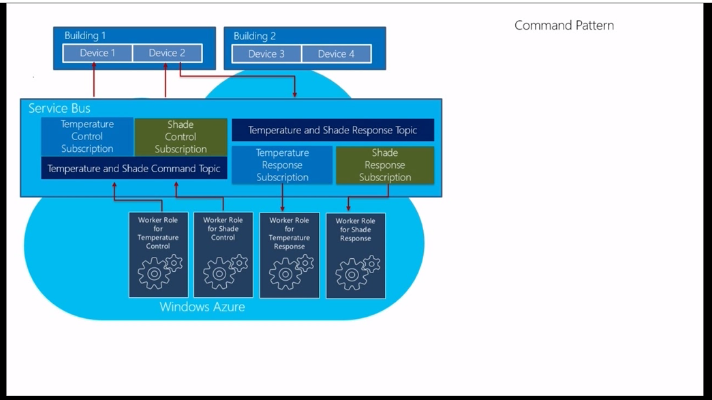
* Build your client side models (optinal)
* Use the HttpClient to call REST service URLs
  + Set headers
  + Query string
  + Body (for POST and PUT)
* User a JSON Serializer / Deserializer (like Json.NET)
  + Map objects to JSON (serializer)
  + Map JSON to objects (deserializer)

**WCF Service Security**

* WCF Security basically boils down to
  + Transport security (SSL/HTTPS)
  + Message Security (encrypting the payload)
* There are a number of ways to accomplish the above using
  + Bindings
  + Transports
  + Authentication mechanisms
  + Certificates…

**Windows Azure Service Bus**





* The answer is Azure Storage Queues until it is not, an then you move on to service bus.

## Advanced Topics

# Notes

## WCF Services vs WCF Data Services

WCF Services are operation centric - the main thing you do with these is define operations (functions). Data comes as a secondary thing in a way. On the other hand they can work over lot of different transports/protocols. You have the ability to stricly define what the client can and can not do. Client can't really make up new queries/operations, the server must implement everything the client needs.

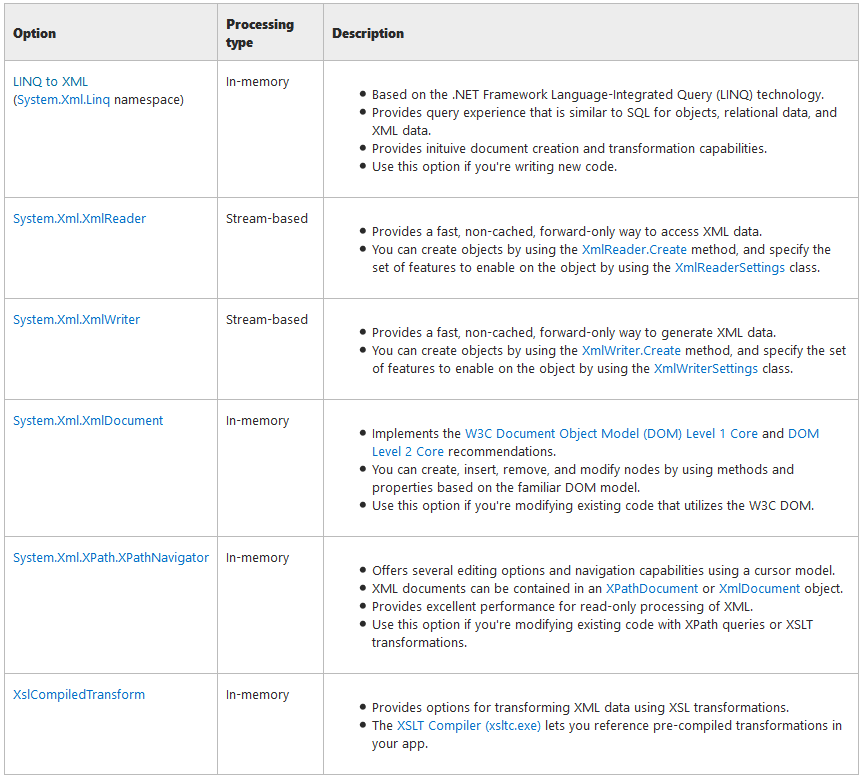
WCF Data Services are data centric - the main thing you do with these is define a data model to be exposed. Operations on the data are "predefined" (Create/Delete/Update/Read). It only supports HTTP protocol and uses REST. You usually allow clients to decide what they want to do (in some boundaries). One of the main differences from WCF Service is that the client can issue a query against the data model (think SQL like stuff) which the client constructs. This means that the server doesn't need to know exactly what the client needs up front, the client will be able to tell it runtime.

## XML Processing Options

See the following table for a list of Microsoft technologies you can use to process XML data.

Table is copied from [this](https://msdn.microsoft.com/en-us/library/bb669131%28v=vs.110%29.aspx) page.

### .Net Framework Options



### Load Data from a Reader

By default, XmlDocument.[Load](https://msdn.microsoft.com/en-us/library/a8ta6tz4%28v=vs.110%29.aspx) does not verify whether the XML is valid using document type definition (DTD) or schema validation. It only verifies whether the XML is well-formed. In order for validation to occur, you need to create an [XmlReader](https://msdn.microsoft.com/en-us/library/system.xml.xmlreader%28v=vs.110%29.aspx) using the [XmlReaderSettings](https://msdn.microsoft.com/en-us/library/system.xml.xmlreadersettings%28v=vs.110%29.aspx) class. The [XmlReader](https://msdn.microsoft.com/en-us/library/system.xml.xmlreader%28v=vs.110%29.aspx) class can enforce validation using a DTD or Schema definition language (XSD) schema. The [ValidationType](https://msdn.microsoft.com/en-us/library/system.xml.validationtype%28v=vs.110%29.aspx) property on the [XmlReaderSettings](https://msdn.microsoft.com/en-us/library/system.xml.xmlreadersettings%28v=vs.110%29.aspx) class determines whether the [XmlReader](https://msdn.microsoft.com/en-us/library/system.xml.xmlreader%28v=vs.110%29.aspx) instance enforces validation. For more information about validating XML data, see the Remarks section of the [XmlReader](https://msdn.microsoft.com/en-us/library/system.xml.xmlreader%28v=vs.110%29.aspx) reference page.

### Modifying Nodes, Content and Values in an XML Document

There are many ways you can modify the nodes and content in a document. You can:

* Change the value of nodes using the [Value](https://msdn.microsoft.com/en-us/library/system.xml.xmlnode.value%28v=vs.110%29.aspx) property.
* Modify an entire set of nodes by replacing the nodes with new nodes. This is done using the [InnerXml](https://msdn.microsoft.com/en-us/library/system.xml.xmlnode.innerxml%28v=vs.110%29.aspx) property.
* Replace existing nodes with new nodes using the [RemoveChild](https://msdn.microsoft.com/en-us/library/system.xml.xmlnode.removechild%28v=vs.110%29.aspx) method.
* Add additional characters to nodes that inherit from the [XmlCharacterData](https://msdn.microsoft.com/en-us/library/system.xml.xmlcharacterdata%28v=vs.110%29.aspx) class using the [AppendData](https://msdn.microsoft.com/en-us/library/system.xml.xmlcharacterdata.appenddata%28v=vs.110%29.aspx), [InsertData](https://msdn.microsoft.com/en-us/library/system.xml.xmlcharacterdata.insertdata%28v=vs.110%29.aspx), or [ReplaceData](https://msdn.microsoft.com/en-us/library/system.xml.xmlcharacterdata.replacedata%28v=vs.110%29.aspx) methods.
* Modify the content by removing a range of characters using the [DeleteData](https://msdn.microsoft.com/en-us/library/system.xml.xmlcharacterdata.deletedata%28v=vs.110%29.aspx) method on node types that inherit from [XmlCharacterData](https://msdn.microsoft.com/en-us/library/system.xml.xmlcharacterdata%28v=vs.110%29.aspx).

### The XML Document Object Model (DOM)

The XML Document Object Model (DOM) treats XML data as a standard set of objects and is used to process XML data in memory. The System.Xml namespace provides a programmatic representation of XML documents, fragments, nodes, or node-sets. It is based on the World Wide Web Consortium (W3C) DOM Level 1 Core and the DOM Level 2 Core recommendations. The XmlDocument class represents an XML document. It includes members for retrieving and creating all other XML objects. Using the XmlDocument, and its related classes, you can construct XML documents, load and access data, modify data, and save changes.

If an application does not require the structure or editing capabilities provided by the DOM, the XmlReader and XmlWriter classes provide non-cached, forward-only stream access to XML. For fast, non-cached, forward-only stream access to XML, use the XmlReader and XmlWriter.

### The XPath Data Model

The [XPathDocument](https://msdn.microsoft.com/en-us/library/system.xml.xpath.xpathdocument%28v=vs.110%29.aspx) class provides a fast, read-only, in-memory representation of an XML document using the XPath data model. Instances of the [XPathDocument](https://msdn.microsoft.com/en-us/library/system.xml.xpath.xpathdocument%28v=vs.110%29.aspx) class are created using one of its six constructors. These constructors allow you to read an XML document using a [Stream](https://msdn.microsoft.com/en-us/library/system.io.stream%28v=vs.110%29.aspx), [TextReader](https://msdn.microsoft.com/en-us/library/system.io.textreader%28v=vs.110%29.aspx), or [XmlReader](https://msdn.microsoft.com/en-us/library/system.xml.xmlreader%28v=vs.110%29.aspx) object, as well as the **string** path to an XML file.

The [XPathNavigator](https://msdn.microsoft.com/en-us/library/system.xml.xpath.xpathnavigator%28v=vs.110%29.aspx) class provides a set of methods used to select a set of nodes in an [XPathDocument](https://msdn.microsoft.com/en-us/library/system.xml.xpath.xpathdocument%28v=vs.110%29.aspx) or [XmlDocument](https://msdn.microsoft.com/en-us/library/system.xml.xmldocument%28v=vs.110%29.aspx) object using an XPath expression. Once selected, you can iterate over the selected set of nodes.

## Caching

### Cache.Add vs Cache.Insert

Insert will overwrite an existing cached value with the same Key; Add fails (does nothing) if there is an existing cached value with the same key. Add also returns the existing value in the cache, meaning if you are adding a new value it returns null. If you attempt to overwrite an existing value, nothing happens and the value in the cache is returned.

### CacheDependency

If you add an item to the cache that is dependent on another object, such as a file or array of files, the dependent item is automatically removed from the cache when the object changes. For example, suppose you create a [DataSet](https://msdn.microsoft.com/en-us/library/system.data.dataset%28v=vs.110%29.aspx) object based on data in an XML file. You can add the [DataSet](https://msdn.microsoft.com/en-us/library/system.data.dataset%28v=vs.110%29.aspx) to the cache with a CacheDependency object that makes the [DataSet](https://msdn.microsoft.com/en-us/library/system.data.dataset%28v=vs.110%29.aspx) dependent on the XML file. If the XML file changes, the [DataSet](https://msdn.microsoft.com/en-us/library/system.data.dataset%28v=vs.110%29.aspx) is removed from the cache.

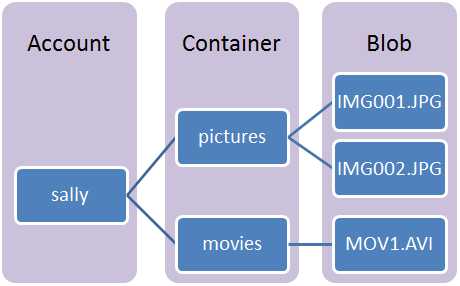
## Azure Storage

### Blobs, tables, queues, and files

Azure Storage provides the flexibility and hyper-scale needed to store and retrieve large amounts of data. Use [Azure Blob Storage](https://azure.microsoft.com/en-us/services/storage/blobs/) to store unstructured data, such as documents and media files. Use [Azure Table Storage](https://azure.microsoft.com/en-us/services/storage/tables/) for structured NoSQL data. Use [Azure Queue Storage](https://azure.microsoft.com/en-us/services/storage/queues/) to reliably store messages. And use SMB-based [Azure File Storage](https://azure.microsoft.com/en-us/services/storage/files) for existing or new applications—no code changes are required.

### Blob storage

The Blob service contains the following components:



**Storage Account:** All access to Azure Storage is done through a storage account. See [Azure Storage Scalability and Performance Targets](https://azure.microsoft.com/en-us/documentation/articles/storage-scalability-targets/) for details about storage account capacity.

**Container:** A container 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:** A file of any type and size. Azure Storage offers three types of blobs: block blobs, page blobs, and append blobs.

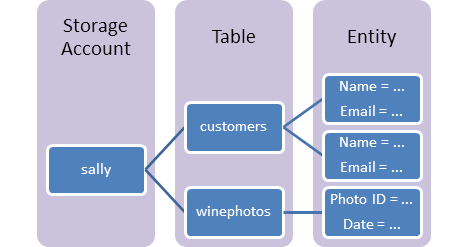
Block blobs are ideal for storing text or binary files, such as documents and media files. Append blobs are similar to block blobs in that they are made up of blocks, but they are optimized for append operations, so they are useful for logging scenarios. A single block blob or append blob can contain up to 50,000 blocks of up to 4 MB each, for a total size of slightly more than 195 GB (4 MB X 50,000).

Page blobs can be up to 1 TB in size, and are more efficient for frequent read/write operations. Azure Virtual Machines use page blobs as OS and data disks.

### Table Storage

Azure Table storage is a service that stores unstructured NoSQL data in the cloud. Table storage is a key/attribute store with a schemaless design. Because Table storage is schemaless, it's easy to adapt your data as the needs of your application evolve. Access to data is fast and cost-effective for all kinds of applications. Table storage is typically significantly lower in cost than traditional SQL for similar volumes of data.

The Table service contains the following components:



**URL format:** Code addresses tables in an account using this address format:  
http://<storage account>.table.core.windows.net/<table>

You can address Azure tables directly using this address with the OData protocol. For more information, see [OData.org](http://www.odata.org/)

**Storage Account:** All access to Azure Storage is done through a storage account. See [Azure Storage Scalability and Performance Targets](https://azure.microsoft.com/en-us/documentation/articles/storage-scalability-targets/) for details about storage account capacity.

**Table**: A table is a collection of entities. Tables don't enforce a schema on entities, which means a single table can contain entities that have different sets of properties. The number of tables that a storage account can contain is limited only by the storage account capacity limit.

**Entity**: An entity is a set of properties, similar to a database row. An entity can be up to 1MB in size.

**Properties**: A property is 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.

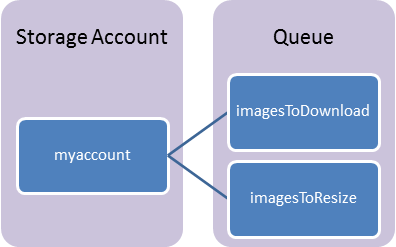
### Azure Queue storage

Azure Queue storage is a service for storing large numbers of messages that can be accessed from anywhere in the world via authenticated calls using HTTP or HTTPS. A single queue message can be up to 64 KB in size, and a queue can contain millions of messages, up to the total capacity limit of a storage account. A storage account can contain up to 500 TB of blob, queue, and table data. See [Azure Storage Scalability and Performance Targets](http://msdn.microsoft.com/library/azure/dn249410.aspx) for details about storage account capacity.

Common uses of Queue storage include:

* Creating a backlog of work to process asynchronously
* Passing messages from an Azure Web role to an Azure worker role

The Queue service contains the following components:



**URL format:** Queues are addressable using the following URL format:  
http://<storage account>.queue.core.windows.net/<queue>

The following URL addresses a queue in the diagram:

http://myaccount.queue.core.windows.net/imagesToDownload

**Storage Account:** All access to Azure Storage is done through a storage account. See [Azure Storage Scalability and Performance Targets](https://azure.microsoft.com/en-us/documentation/articles/storage-scalability-targets/) for details about storage account capacity.

**Queue:** A queue contains a set of messages. All messages must be in a queue.

**Message:** A message, in any format, of up to 64KB.

### CDN for Azure

The Azure Content Delivery Network (CDN) is the fundamental building block to scale any HTTP application in Azure. It offers Azure customers a global solution by caching and delivering content close to end users. As a result, instead of hitting origin every single time, user requests get intelligently routed to the best performed CDN edge POP. This significantly increases the performance and user experience.

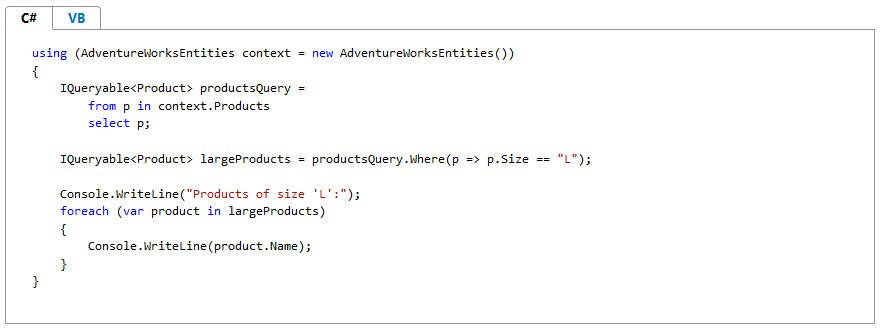
Azure CDN can distrubute content from a variety of origins. Integrated origins within Azure include App Service, Cloud Services, blob storage, and Media Service. You can also define a custom origin using any publicly accessible web address.

## Entity Framework

### Deferred query execution

In a query that returns a sequence of values, the query variable itself never holds the query results and only stores the query commands. Execution of the query is deferred until the query variable is iterated over in a **foreach** or **For Each** loop. This is known as deferred execution; that is, query execution occurs some time after the query is constructed. This means that you can execute a query as frequently as you want to. This is useful when, for example, you have a database that is being updated by other applications. In your application, you can create a query to retrieve the latest information and repeatedly execute the query, returning the updated information every time.

Deferred execution enables multiple queries to be combined or a query to be extended. When a query is extended, it is modified to include the new operations, and the eventual execution will reflect the changes. In the following example, the first query returns all the products. The second query extends the first by using **Where** to return all the products of size "L":



### Lazy Loading

With lazy loading enabled, related objects are loaded when they are accessed through a navigation property. You can still eagerly load objects with the [Include](https://msdn.microsoft.com/en-us/library/bb738708%28v=vs.100%29.aspx) method or explicitly load them with the [LoadProperty](https://msdn.microsoft.com/en-us/library/dd382880%28v=vs.100%29.aspx) method.

In the Entity Framework runtime, the default value of the [LazyLoadingEnabled](https://msdn.microsoft.com/en-us/library/system.data.objects.objectcontextoptions.lazyloadingenabled%28v=vs.100%29.aspx) property in an instance of [ObjectContext](https://msdn.microsoft.com/en-us/library/system.data.objects.objectcontext%28v=vs.100%29.aspx) is **false**. However, if you use the Entity Framework tools to create a new model and the corresponding generated classes, the generated code will set **LazyLoadingEnabled** to **true** in the constructor of the generated object context.

### Compiled Queries

When you have an application that executes structurally similar queries many times in the Entity Framework, you can frequently increase performance by compiling the query one time and executing it several times with different parameters. For example, an application might have to retrieve all the customers in a particular city; the city is specified at runtime by the user in a form. LINQ to Entities supports using compiled queries for this purpose.

The [CompiledQuery](https://msdn.microsoft.com/en-us/library/system.data.objects.compiledquery%28v=vs.100%29.aspx) class provides compilation and caching of queries for reuse. Conceptually, this class contains a **CompiledQuery**'s **Compile** method with several overloads. Call the **Compile** method to create a new delegate to represent the compiled query. The **Compile** methods, provided with a [ObjectContext](https://msdn.microsoft.com/en-us/library/system.data.objects.objectcontext%28v=vs.100%29.aspx) and parameter values, return a delegate that produces some result (such as an [IQueryable](https://msdn.microsoft.com/en-us/library/bb351562%28v=vs.100%29.aspx) instance). The query compiles once during only the first execution. The merge options set for the query at the time of the compilation cannot be changed later. Once the query is compiled you can only supply parameters of primitive type but you cannot replace parts of the query that would change the generated SQL.

### Entity SQL Language

Entity SQL is a storage-independent query language that is similar to SQL. Entity SQL allows you to query entity data, either as objects or in a tabular form. You should consider using Entity SQL in the following cases:

* When a query must be dynamically constructed at runtime. In this case, you should also consider using the query builder methods of [ObjectQuery](https://msdn.microsoft.com/en-us/library/bb345303%28v=vs.100%29.aspx) instead of constructing an Entity SQL query string at runtime.
* When you want to define a query as part of the model definition. Only Entity SQL is supported in a data model. For more information, see [QueryView Element (MSL)](https://msdn.microsoft.com/en-us/library/cc716798%28v=vs.100%29.aspx)
* When using EntityClient to return read-only entity data as rowsets using a [EntityDataReader](https://msdn.microsoft.com/en-us/library/system.data.entityclient.entitydatareader%28v=vs.100%29.aspx). For more information, see [EntityClient Provider for the Entity Framework](https://msdn.microsoft.com/en-us/library/bb738561%28v=vs.100%29.aspx).
* If you are already an expert in SQL-based query languages, Entity SQL may seem the most natural to you.

## NuGet

### Creating Remote Feeds

You can also host a remote (or internal) feed on a server that runs IIS. There are two alternatives from the NuGet team here 1. NuGet.Server 2. NuGet Gallery

For relatively small projects with a small set of packages go with NuGet.Server, it is basically a view of a network share or local folder through http, and as such is easy to setup and works quite well when the number of packages is small. However if the package count is high, say in the thousands, the server will take a very long time to boot. The alternative is to use the [NuGet Gallery Project](https://github.com/NuGet/NuGetGallery/wiki/Hosting-the-NuGet-Gallery-Locally-in-IIS) it is more complex to set up and host, but offers a lot more nuget.org like features.

### Creating and publishing a package

<https://docs.nuget.org/create/creating-and-publishing-a-package>

### Hosting your own NuGet feed

<https://docs.nuget.org/create/hosting-your-own-nuget-feeds>

# TODO

* Create a cloud service in Azure and publish a WCF service. Use both worker role and web role.
* Create and use service bus in azure