WCF Services in Nutshell

Based on the original slides of Michael Arnwine: WCF using Service Oriented Architecture (SOA) and “Restful” Service

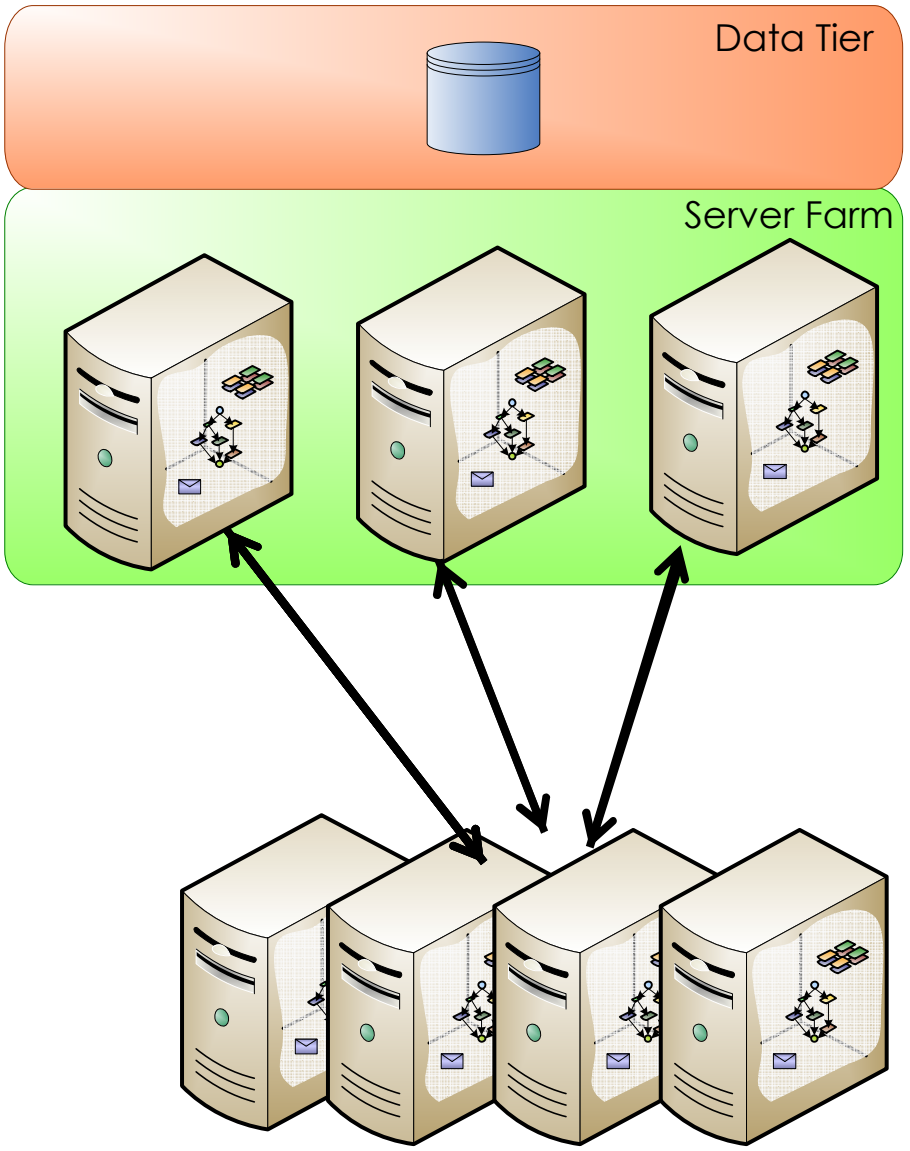
What is WCF

Windows Communication Foundation (WCF) is an SDK for developing and deploying services on Windows. WCF provides a runtime environment for your services, enabling you to expose CLR types as services, and to consume other services as CLR types.

….a part of the .NET Framework that provides a unified programming model for rapidly building service-oriented applications that communicate across the web and the enterprise.

Scalability

- WCF services scale easily.
- Written and deployed as Windows Service, Web Service, Windows Process Activation Service (WAS)
- With Bindings you can change for format of the ending message to JSON, XML, JSOP, or any other format,
- Service Federation and Tokenize you WCF for security



ABC of WCF Services

- "A" stands for Address—as expressed address or uri of the endpoint.
- "B" stands for Binding—as expressed in the binding section, an envelope format and associated policies.
- "C" stands for Contract—as expressed in the portType, message and type sections and describes types, messages, message exchange patterns and operations.

Address

- The <endpoint address> is where the application endpoint lives. So when you create the client and it talks to the service using the specified contract, it does so at that address.
- Put another way, the base address is the address of the 'get wsdl' metadata endpoint, and the endpoint address is the address of the application endpoint

```
<services>
 <service behaviorConfiguration="DefaultBehavior" name="Test.TestService">
  <clear />
  <host>
   <baseAddresses>
    <add baseAddress=http://localhost:8081/Test/ />
   </baseAddresses>
  </host>
  <endpoint address=http://localhost:8082/Test/
  binding="basicHttpBinding" bindingConfiguration="" name="Web"
  contract="Test.ITestService" />
 </service>
</services>
```

Binding

- Bindings are used to specify the t protocol, transport and encoding and details required for clients and services to communicate with each other.
- WCF uses bindings to generate the underlying wire representation of the endpoint, so most of the binding details must be agreed upon by the parties that are communicating

```
<services>
 <service behaviorConfiguration="DefaultBehavior" name="Test.TestService">
  <clear />
  <host>
   <baseAddresses>
    <add baseAddress=http://localhost:8081/Test/ />
   </baseAddresses>
  </host>
  <endpoint address=http://localhost:8082/Test/
  binding="basicHttpBinding" bindingConfiguration="" name="Web"
  contract="Test.ITestService" />
 </service>
</services>
```

Binding Types

- **wsHttpBinding: SOAP encoding over HTTP**
- **netTcpBinding: Binary encoding over TCP**
- **netNamedPipeBinding: Binary encoding using IPC**
- netPeerTcpBinding
- ***basicHttpBinding: JSON or POX over HTTP**
- *custombinding

Contract

- A WCF Contract is a collection of Operations that specifies what the Endpoint communicates to the outside world. Each operation is a simple message exchange, for example one-way or request/reply message exchange.
- Contract can specify a Behavior
- Behaviors are types that modify or extend Service or Client functionality. Like the ServiceMetadata, ServiceDebug

Contract Types

- **Service contracts**
  - OperationsContract - attribute used to define the methods or functions called in the service
- **Data contracts**
  - Define which data types are passed to and from the service. WCF defines implicit contracts for built-in types such as int and string, but we can easily define explicit opt-in data contracts for custom types.
  - There are two types of Data Contracts.
  - DataContract - attribute used to define the class
  - DataMember - attribute used to define the properties.
- **Fault contracts** - Define which errors are raised by the service, and how the service handles and propagates errors to its clients.
- **Message contracts** - Allow the service to interact directly with messages. Message contracts can be typed or untyped, and are useful in interoperability cases and when there is an existing message format we have to comply with.

A WCF Code

```
[ServiceContract(Name = "DataService", Namespace =
  “edu.wmich.Service")]
public interface IMathematics
{
  [OperationContract]
  double Add( double a, double b);

    [OperationContract]
      double Subtract( double a, double b);
}

[ServiceBehavior]
public class Mathematics : IMathematics
{
  public double Add( double a, double b)
  {...}

  public double Subtract( double a, double b)
  {...}
}
```

EXAMPLES:

Publishing and Consuming WCF Services

REST Concept

- Actually only the difference is how clients access our service. Normally, a WCF service will use SOAP, but if you build a REST service, clients will be accessing your service with a different architectural style (calls, serialization like JSON, etc.).

- REST uses some common HTTP methods to insert/delete/update/retrieve information which is below:

- **GET** - Requests a specific representation of a resource
- **PUT** - Creates or updates a resource with the supplied representation
- **DELETE** - Deletes the specified resource
- **POST** - Submits data to be processed by the identified resourc

REST

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use.“

Why REST?

•Less overhead (no SOAP envelope to wrap every call in)
•Less duplication (HTTP already represents operations
like DELETE, PUT, GET, etc. that have to otherwise be represented in a SOAP envelope).
•More standardized - HTTP operations are well understood and operate consistently. Some SOAP implementations can get finicky.
•More human readable and testable (harder to test SOAP with just a browser).
•Don't need to use XML (well, you kind of don't have to for SOAP either but it hardly makes sense since you're already doing parsing of the envelope).
•Libraries have made SOAP (kind of) easy. But you are abstracting away a lot of redundancy underneath as I have noted. Yes, in theory, SOAP can go over other transports so as to avoid riding atop a layer doing similar things, but in reality just about all SOAP work you'll ever do is over HTTP.

REST Data Elements

- Resources and Resource Identifiers
  - Uniform Interface (GET, PUT, POST, DELETE)
  - Resource Oriented
  - Simple and simple is beautiful

HTTP	**Method**	**CRUD**	**Desc.**
POST	CREATE	Create	-
GET	RETRIEVE	Retrieve	Safe,Idempotent,Cacheable
PUT	UPDATE	Update	Idempotent
DELETE	DELETE	Delete	Idempotent

REST Core Idiologies

- Simple is better
- The web works and works well
- Some web services should follow the “way of the web”.

WCF RESTful?

- Resources as URI
  - Use unique URI to reference every resource on your API
- Operations as HTTP Methods
  - GET – Queries
  - POST – Queries
  - PUT, DELETE – Inset, Update and delete
- Connectedness and Discoverability
  - Like the Web, HTTP Responses contains links to other resources

REST API EXAMPLE: Delicious

URL	http://del.icio.us/api/[username]/bookmarks/	
Method	GET	
Querystring	tag=	Filter by tag
	dt=	Filter by date
	start=	The number of the first bookmark to return
	end=	The number of the last bookmark to return
Returns	200 OK & XML (delicious/bookmarks+xml)	
	401 Unauthorized	
	404 Not Found	

REST API EXAMPLE: Delicious

URL	http://del.icio.us/api/[username]/bookmarks/
Method	POST
Request Body	XML (delicious/bookmark+xml)
Returns	201 Created & Location
	401 Unauthorized
	415 Unsupported Media Type

REST API EXAMPLE: Delicious

URL	http://del.icio.us/api/[username]/bookmarks/[hash]
Method	DELETE
Returns	204 No Content
	401 Unauthorized
	404 Not Found

System.UriTemplate

- Represents Uniform Resource Identifier (URI) template
- Composed of two parts: path and query
- Creating a URI:

```
Uri baseAddress = new Uri("http://localhost:8010");

UriTemplate template =
    new UriTemplate("HistoricalPrices/{market}?ticket={ticket}");

// set the values of the URI segments
Uri boundUri =
    template.BindByPosition(baseAddress, "Nasdaq"  "MSFT");

// http://localhost:8010/HistoricalPrices/Nasdaq?ticket=MSFT
```

System.UriTemplateMatch

- Match a URI to a UriTemplate
- Parsing a URI:

```
Uri baseAddress = new Uri("http://localhost:8010");

UriTemplate template =
    new UriTemplate("HistoricalPrices/{market}?ticket={ticket}");

Uri boundUri =
    new Uri("http://localhost:8010/HistoricalPrices/Nasdaq?ticket=MSFT");

UriTemplateMatch match = template.Match(baseAddress, boundUri);

String market = match.BoundVariables["market"];
String ticket = match.QueryParameters["ticket"];
```

WebGet & WebInvoke

- Maps URI segments and query string parameters into application functionality
- WebGet – GET
- WebInvoke – POST, PUT, DELETE, other
- Other parameters
  - Body Style
  - RequestFormat (JSON / XML)
  - ResponseFormat (JSON / XML)
  - UriTemplate

WebGet & WebInvoke Examples

```
[OperationContract]
[WebGet(BodyStyle = WebMessageBodyStyle.Bare,
        RequestFormat = WebMessageFormat.Json,
        UriTemplate = "product/{productId}")]
Product GetProduct(Int32 productId)
```

```
[OperationContract]
[WebInvoke(Method = "Delete",
           BodyStyle = WebMessageBodyStyle Bare,
           RequestFormat = WebMessageFormat.Json,
           ResponseFormat = WebMessageFormat.Xml
           UriTemplate = "product/{productId}")]
Product DeleteProduct(Int32 productId)
```

What is JSON?

- JavaScript Object Notation
- Lightweight syntax for representing data
- Easier to “parse” for JavaScript client code
- Alternative to XML in AJAX applications

[{"Email":"bob@example.com","Name":"Bob"},{"Email
":"mark@example.com","Name":"Mark"},{"Email":"j
ohn@example.com","Name":"John"}]

EXAMPLES:

Publish and Consume REST Services Using WCF

EXAMPLES of REST APIs:

Facebook Graph API
Google Custom Search API
Yahoo
… and a lot more …

n-Tiers Architecture

- SOAP and REST based web-services enable the 3-tier architecture to be extended into n-tiers.