Web services are web application components.

Web services can be published, found, and used on the Web.

Web Services Benefit

* Reusable application components.
* Interoperability

WSDL stands for Web Services Description Language

WSDL is an XML-based language for describing Web services.

Sample WSDL:

<definitions>

<types>

data type definitions........

</types>

<message>

definition of the data being communicated....

</message>

<portType>

set of operations......

</portType>

<binding>

protocol and data format specification....

</binding>

</definitions>

SOAP stands for Simple Object Access Protocol

SOAP is an XML based protocol for accessing Web Services.

Sample SOAP format:

<soap:Envelope

xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"

soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>

...

</soap:Header>

<soap:Body>

...

<soap:Fault>

...

</soap:Fault>

</soap:Body>

</soap:Envelope>

RDF stands for Resource Description Framework

RDF is a framework for describing resources on the web

RDF is written in XML

RDF is designed to be read and understood by computers

<rdf:RDF

xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

xmlns:si="http://www.w3schools.com/rdf/">

<rdf:Description rdf:about="http://www.w3schools.com">

<si:title>W3Schools</si:title>

<si:author>Jan Egil Refsnes</si:author>

</rdf:Description>

</rdf:RDF>

UDDI stands for Universal Description, Discovery and Integration

UDDI is a directory for storing information about web services

By using Web services, your application can publish its function or message to the rest of the world.

Web services use XML to code and to decode data, and SOAP to transport it (using open protocols).

With Web services, your accounting department's Win 2k server's billing system can connect with your IT supplier's UNIX server.

namespace MyService

{

...

/// <summary>

/// Summary description for WebService1.

/// </summary>

[WebService(Namespace="http://codeproject.com/webservices/",

Description="This is a demonstration WebService.")]

[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]

public class WebService1 : System.Web.Services.WebService

{

public WebService1()

{

//CODEGEN: This call is required by the ASP+ Web Services Designer

InitializeComponent();

}

...

[WebMethod(CacheDuration = 30)]

public string HelloWorld()

{

return "Hello World";

}

}

}

Proxy - To access a Web service from a client application, you first add a Web reference, which is a reference to a Web service. When you create a Web reference, Visual Studio creates a Web service proxy class automatically and adds it to your project. This proxy class exposes the methods of the Web service and handles the marshalling of appropriate arguments back and forth between the Web service and your application.

Marshaling is the process of creating a bridge between managed code and unmanaged code; it is the homer that carries messages from the managed to the unmanaged environment and reverse. An example is the unmanaged DWORD; it is an unsigned 32-bit integer, so we can marshal it in .NET as System.UInt32. Therefore, System.UInt32 is a substitute for the unmanaged DWORD. On the other hand, unmanaged compound types (structures, unions, etc.) do not have counterparts or substitutes in the managed environment. Thus, you’ll need to create your own managed types (structures/classes) that will serve as the substitutes for the unmanaged types you use.

Wsdl.exe – It is used to create proxy for accessing web methods exposed by a web service.

|  |  |  |
| --- | --- | --- |
| **No.** | **SOAP** | **REST** |
| 1) | SOAP is a **protocol**. | REST is an **architectural style**. |
| 2) | SOAP stands for **Simple Object Access Protocol**. | REST stands for **REpresentational State Transfer**. |
| 3) | SOAP **can't use REST** because it is a protocol. | REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP. |
| 4) | SOAP **uses services interfaces to expose the business logic**. | REST **uses URI to expose business logic**. |
| 6) | SOAP **defines standards**to be strictly followed. | REST does not define too much standards like SOAP. |
| 7) | SOAP **requires more bandwidth** and resource than REST. | REST **requires less bandwidth** and resource than SOAP. |
| 8) | SOAP **defines its own security**. | RESTful web services **inherits security measures** from the underlying transport. |
| 9) | SOAP **permits XML** data format only. | REST **permits different** data format such as Plain text, HTML, XML, JSON etc. |

Windows Communication Foundation (Code named Indigo) is a programming platform and runtime system for building, configuring and deploying network-distributed services.

|  |  |  |
| --- | --- | --- |
| **Features** | **Web Service(ASMX)** | **WCF** |
| Hosting | It can be hosted in IIS | It can be hosted in IIS, windows activation service, Self-hosting, Windows service |
| Programming | [WebService] attribute has to be added to the class | [ServiceContraact] attribute has to be added to the class |
| Model | [WebMethod] attribute represents the method exposed to client | [OperationContract] attribute represents the method exposed to client |
| Operation | One-way, Request- Response are the different operations supported in web service | One-Way, Request-Response, Duplex are different type of operations supported in WCF |
| XML | System.Xml.serialization name space is used for serialization | System.Runtime.Serialization namespace is used for serialization |
| Encoding | XML 1.0, MTOM(Message Transmission Optimization Mechanism), DIME, Custom | XML 1.0, MTOM, Binary, Custom |
| Transports | Can be accessed through HTTP, TCP, Custom | Can be accessed through HTTP, TCP, Named pipes, MSMQ,P2P, Custom |
| Protocols | Security | Security, Reliable messaging, Transactions |

**EndPoint**

WCF Service is a program that exposes a collection of Endpoints. Each Endpoint is a portal for communicating with the world

### Address

Basically URL, specifies where this WCF service is hosted .Client will use this url to connect to the service. e.g

http://localhost:8090/MyService/SimpleCalculator.svc

### Binding

Binding will describes how client will communicate with service. There are different protocols available for the WCF to communicate to the Client.

A binding has several characteristics, including the following:

* Transport -Defines the base protocol to be used like HTTP, Named Pipes, TCP, and MSMQ are some type of protocols.
* Encoding (Optional) - Three types of encoding are available-Text, Binary, or Message Transmission Optimization Mechanism (MTOM). MTOM is an interoperable message format that allows the effective transmission of attachments or large messages (greater than 64K).
* Protocol(Optional) - Defines information to be used in the binding such as Security, transaction or reliable messaging capability

|  |  |
| --- | --- |
| **Binding** | **Description** |
| BasicHttpBinding | Basic Web service communication. No security by default |
| WSHttpBinding | Web services with WS-\* support. Supports transactions |
| WSDualHttpBinding | Web services with duplex contract and transaction support |
| WSFederationHttpBinding | Web services with federated security. Supports transactions |
| MsmqIntegrationBinding | Communication directly with MSMQ applications. Supports transactions |
| NetMsmqBinding | Communication between WCF applications by using queuing. Supports transactions |
| NetNamedPipeBinding | Communication between WCF applications on same computer. Supports duplex contracts and transactions |
| NetPeerTcpBinding | Communication between computers across peer-to-peer services. Supports duplex contracts |
| NetTcpBinding | Communication between WCF applications across computers. Supports duplex contracts and transactions |

### Contract

Collection of operation that specifies what the endpoint will communicate with outside world. Usually name of the Interface will be mentioned in the Contract, so the client application will be aware of the operations which are exposed to the client.

<system.serviceModel>

<services>

<service name="MathService"

behaviorConfiguration="MathServiceBehavior">

<endpoint

address="http://localhost:8090/MyService/MathService.svc" contract="IMathService"

binding="wsHttpBinding"/>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="MathServiceBehavior">

<serviceMetadata httpGetEnabled="True"/>

<serviceDebug includeExceptionDetailInFaults="true" />

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

the serviceMetadata node with attribute httGetEnabled='true'. This attribute will specifies the publication of the service metadata.

Exposing the metadata using HTTP-GET has a disadvantage, such that there is no guarantee that other platforms you interact will support it. There is other way of exposing the using special endpoint is called as Metadata Exchange Endpoint.

It is basically Uri to identify the metadata. You can specify as address in the endpoint but append with "mex" keyword. For example <http://localhost:9090/MyCalulatorService/mex>

There are four types of bindings supported for metadata exchange. They are mexHttpBinding, mexHttpsBinding, mexNamedPipesBinding, mexTcpBinding.

IMetadataExchange is the contract used for MEX endpoint. WCF service host automatically provides the implementation for this IMetadataExcahnge while hosting the service.

## Contracts

In WCF, all services are exposed as contracts. Contract is a platform-neutral and standard way of describing what the service does.

### Service Contract

Service contracts describe the operation that service can provide. For Eg, a Service provide to know the temperature of the city based on the zip code, this service is called as Service contract.

Service Contract can be define using [ServiceContract] and [OperationContract] attribute. [ServiceContract] attribute is similar to the [WebServcie] attribute in the WebService and [OpeartionContract] is similar to the [WebMethod] in WebService.

[ServiceContract()]

public interface ISimpleCalculator

{

[OperationContract()]

int Add(int num1, int num2);

}

public class SimpleCalculator : ISimpleCalculator

{

public int Add(int num1, int num2)

{

return num1 + num2;

}

}

With out creating the interface, we can also directly created the service by placing Contract in the implemented class. But it is not good practice of creating the service

[ServiceContract()]

public class SimpleCalculator

{

[OperationContract()]

public int Add(int num1, int num2)

{

return num1 + num2;

}

}

# Data Contract

A data contract is a formal agreement between a service and a client that abstractly describes the data to be exchanged.

Data contract can be explicit or implicit. Simple type such as int, string etc has an implicit data contract. User defined object are explicit or Complex type, for which you have to define a Data contract using [DataContract] and [DataMember] attribute.

[ServiceContract]

public interface IEmployeeService

{

[OperationContract]

Employee GetEmployeeDetails(int EmpId);

}

[DataContract]

public class Employee

{

private string m\_Name;

private int m\_Age;

private int m\_Salary;

private string m\_Designation;

private string m\_Manager;

[DataMember]

public string Name

{

get { return m\_Name; }

set { m\_Name = value; }

}

[DataMember]

public int Age

{

get { return m\_Age; }

set { m\_Age = value; }

}

[DataMember]

public int Salary

{

get { return m\_Salary; }

set { m\_Salary = value; }

}

[DataMember]

public string Designation

{

get { return m\_Designation; }

set { m\_Designation = value; }

}

[DataMember]

public string Manager

{

get { return m\_Manager; }

set { m\_Manager = value; }

}

}

### Client side

On client side we can create the proxy for the service and make use of it. The client side code is shown below.

protected void btnGetDetails\_Click(object sender, EventArgs e)

{

EmployeeServiceClient objEmployeeClient = new EmployeeServiceClient();

Employee empDetails;

empDetails = objEmployeeClient.GetEmployeeDetails(empId);

//Do something on employee details

}

# Message Contract

### Message

Message is the packet of data which contains important information. WCF uses these messages to transfer information from Source to destination.

WCF uses SOAP(Simple Object Access Protocol) Message format for communication. SOAP message contain Envelope, Header and Body.SOAP envelope contails name, namespace,header and body element. SOAP Hear contain important information which are not directly related to message.

### Message Pattern

It describes how the programs will exchange message each other. There are three way of communication between source and destination

1. **Simplex** - It is one way communication. Source will send message to target, but target will not respond to the message.
2. **Request/Replay** - It is two way communications, when source send message to the target, it will resend response message to the source. But at a time only one can send a message
3. **Duplex** - It is two way communication, both source and target can send and receive message simultaniouly.

### What is Message contract?

WCF uses SOAP message for communication. Most of the time developer will concentrate more on developing the DataContract, Serializing the data, etc. WCF will automatically take care of message. On Some critical issue, developer will also require control over the SOAP message format. In that case WCF provides Message Contract to customize the message as per requirement.

[MessageContract]

public class EmployeeDetails

{

[MessageHeader(ProtectionLevel=ProtectionLevel.None, Name="ID")]

public string EmpID;

[MessageBodyMember(ProtectionLevel=ProtectionLevel.Sign, Name="EmpName")]

public string Name;

[MessageBodyMember(ProtectionLevel=ProtectionLevel. EncryptAndSign)]

public string Designation;

[MessageBodyMember(Order=2)]

public int Salary;

[MessageBodyMember(Order=1)]

public string Location;

}

<EmployeeDetails>

<ID>45634</ID>

<EmployeeName>Sam</EmployeeName> //Protection Level Digitally Signed

<Designation>Software Engineer</Designation> Protection Level Encrypt&Sign

<Salary>25000</Salary>

</EmployeeDetails>

### ProtectionLevel

You can mention the *MessageHeader* or *MessageBodyMember* to be signed or Encrypted using *ProtectionLevel* property.

### Rules :

You have to follow certain rules while working with Message contract

1. When using Message contract type as parameter, Only one parameter can be used in servicie Operation
2. [OperationContract]
3. void SaveEmployeeDetails(EmployeeDetails emp);
4. Service operation either should return Messagecontract type or it should not return any value
5. [OperationContract]
6. EmployeeDetails GetEmployeeDetails();
7. Service operation will accept and return only message contract type. Other data types are not allowed.
8. [OperationContract]
9. EmployeeDetails ModifyEmployeeDetails(EmployeeDetails emp);

**Note:** If a type has both Message and Data contract, service operation will accept only message contract.

# Fault Contract

By default when we throw any exception from service, it will not reach the client side. WCF provides the option to handle and convey the error message to client from service using SOAP Fault contract.

//Service interface

[ServiceContract()]

public interface ISimpleCalculator

{

[OperationContract()]

int Add(int num1, int num2);

}

//Service implementation

public class SimpleCalculator : ISimpleCalculator

{

public int Add(int num1, int num2)

{

//Do something

throw new FaultException("Error while adding number");

}

}

try

{

MyCalculatorServiceProxy.MyCalculatorServiceProxy proxy

= new MyCalculatorServiceProxy.MyCalculatorServiceProxy();

Console.WriteLine("Client is running at " + DateTime.Now.ToString());

Console.WriteLine("Sum of two numbers... 5+5 =" + proxy.Add(5, 5));

Console.ReadLine();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

These are the steps to be followed to create the fault contract.

* Define a type using the data contract and specify the fields you want to return.
* Decorate the service operation with the FaultContract attribute and specify the type name.
* Raise the exception from the service by creating an instance and assigning properties of the custom exception.

[DataContract()]

public class CustomException

{

[DataMember()]

public string Title;

[DataMember()]

public string ExceptionMessage;

[DataMember()]

public string InnerException;

[DataMember()]

public string StackTrace;

}

[ServiceContract()]

public interface ISimpleCalculator

{

[OperationContract()]

[FaultContract(typeof(CustomException))]

int Add(int num1, int num2);

}

public int Add(int num1, int num2)

{

//Do something

CustomException ex = new CustomException();

ex.Title = "Error Funtion:Add()";

ex.ExceptionMessage = "Error occur while doing add function.";

ex.InnerException = "Inner exception message from serice";

ex.StackTrace = "Stack Trace message from service.";

throw new FaultException(ex,"Reason: Testing the Fault contract") ;

}

try

{

MyCalculatorServiceProxy.MyCalculatorServiceProxy proxy

= new MyCalculatorServiceProxy.MyCalculatorServiceProxy();

Console.WriteLine("Client is running at " + DateTime.Now.ToString());

Console.WriteLine("Sum of two numbers... 5+5 =" + proxy.Add(5, 5));

Console.ReadLine();

}

catch (FaultException<MyCalculatorService.CustomException> ex)

{

//Process the Exception

}

# Service Host

Service Host object is in the process of hosting the WCF service and registering endpoints. It loads the service configuration endpoints, apply the settings and start the listeners to handle the incoming request.

//Creating uri for the hosting the service

Uri uri = new Uri("http://localhost/CategoryService");

//Creating the host object for MathService

ServiceHost host = new ServiceHost(typeof(CategoryService), uri);

//Adding endpoint to the Host object

host.AddServiceEndpoint(typeof(ICategoryService),new WSHttpBinding(), uri);

host.Open(); //Hosting the Service

Console.WriteLine("Waiting for client invocations");

Console.ReadLine();

host.Close();

Instance Management

Instance management refers to the way a service handles a request from a client. Instance management is set of techniques WCF uses to bind client request to service instance, governing which service instance handles which client request.

1.Per Call -When WCF service is configured for Per-Call instance mode, Service instance will be created for each client request. This Service instance will be disposed after response is sent back to client.

2. Per Session(Default Value) - When WCF service is configured for Per-Session instance mode, logical session between client and service will be maintained. When the client creates new proxy to particular service instance, a dedicated service instance will be provided to the client. It is independent of all other instance.

3. Singleton - When WCF service is configured for Singleton instance mode, all clients are independently connected to the same single instance. This singleton instance will be created when service is hosted and, it is disposed when host shuts down.

[ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)]

public class MyService:IMyService

{

public int MyMethod()

{

//Do something

}

}

Instance Deactivation

 WCF provides the option of separating the two lifetimes and deactivating the instance separately from its context.

*ReleaseInstanceMode* property of the *OberationalBehavior* attribute used to control the instance in relation to the method call.

1. RealeaseInstanceMode.None(Default Value)
2. RealeaseInstanceMode.BeforeCall - This property means that it will create new instance before a call is made to the operation.If the instance is already exist,WCF deactivates the instance and calls Dispose() before the call is done.This is designed to optimize a method such as Create()
3. RealeaseInstanceMode.AfterCall - This property means that it will deactivate the instance after call is made to the method.This is designed to optimize a method such a Cleanup()
4. RealeaseInstanceMode.BeforeAndAfterCall - This is means that it will create new instance of object before a call and deactivates the instance after call. This has combined effect of using *ReleaseInstanceMode.BeforeCall* and *ReleaseInstanceMode.AfterCall*

*Explicit Deactivation-*

[ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)]

public class MyService:IMyService

{

public void MyMethod()

{

//Do something

OperationContext.Current.InstanceContext.ReleaseServiceInstance();

}

}

[ServiceContract()]

public interface ISimpleCalculator

{

[OperationContract()]

int Add(int num1, int num2);

}

[OperationBehavior(ReleaseInstanceMode=ReleaseInstanceMode.BeforeCall]

public int Add(int num1, int num2)

{

return num1 + num2;

}

Durable Service - Durable services are WCF services that persist service state information even after service host is restarted or Client. It means that durable services have the capability to restore their own state when they are recycled. It can use data store like SQL database for maintain instance state.

* **CanCreateInstance = true:** Calling this operation results in creating the serialization and inserting it into the datastore.
* **CompletesInstance = true:** Calling this operation results in deleting the persisted instance from the datastor

[Serializable]

[DurableService()]

public class MyService :IMyservice

{

[DurableOperation(CanCreateInstance = true)]

public int StartPersistance()

{

//Do Something

}

[DurableOperation(CompletesInstance = true)]

public void EndPersistence()

{

//Do Something

}

}

# Throttling

WCF throttling provides some properties that you can use to limit how many instances or sessions are created at the application level.

maxConcurrentCalls- Limits the total number of calls that can currently be in progress across all service instances. The default is 16.

maxConcurrentInstances - The number of InstanceContext objects that execute at one time across a ServiceHost. The default is Int32.MaxValue.

maxConcurrentSessions - A positive integer that limits the number of sessions a ServiceHost object can accept. The default is 10.

<behaviors>

<serviceBehaviors>

<behavior name="ServiceBehavior">

<serviceMetadata httpGetEnabled="true"/>

<serviceDebug includeExceptionDetailInFaults="true "/>

<serviceThrottling maxConcurrentCalls="500"

maxConcurrentInstances ="100"

maxConcurrentSessions ="200"/>

</behavior>

</serviceBehaviors>

</behaviors>

**Transfer Security modes in a WCF service**  
  
In security mode WCF makes a secure communication channel, encrypting messages when communicating with clients. The following are the security modes.  
  
**1. Message security mode:**In this mode the message will be encrypted and pass over a non-secure channel so that nobody can read the message.  
  
**2. Transport security mode:** In this mode the communication channel will be encrypted and also provide integrity, privacy and so on.  
  
**3. Mixed transfer security mode:**This mode provides transport security for message privacy and it uses message security for secure credentials.  
  
**4. Both security mode:**This mode uses both transport and message security. So the message will be encrypted using message security and will pass over a secure channel using transport security. It provides more security than others but it degrades performance.

<security mode="Message">

        <transport clientCredentialType="Windows"

                   proxyCredentialType="None"

                   realm="" />

        <message clientCredentialType="Windows"

                 negotiateServiceCredential="true"

                 algorithmSuite="Default"

                 establishSecurityContext="true" />

      </security>

Default Secuirty mode in wsHttpBinding is Message.

**Authentications in WCF service:**In authentication process WCF verifies the caller (who calls the services) and checks whether they are authorized or not to get the service.  
  
**Windows authentication:** In this mode the caller must provide his/her Windows credential for authentication.  
  
**Username/Password:** In this mode the caller must provide username and password for authentication.  
  
**X509 certificates:**In this mode the caller must send certificate information and the service will check whether the certificate is valid.  
  
**Custom mechanism:** In this mode the user must use their own protocol and credentials type instead of built-in authentication.  
  
**Issue token:** In this mode both the caller and the service rely on a secure token service to issue the client a token for the service identity. It uses a Windows card space.  
  
**No authentication:** WCF service doesn't implement any authentication in this mode.

Custom Authentication and Authorization in WCF

<https://www.codeproject.com/Articles/698862/Custom-Authentication-and-Authorization-in-WCF>

WCF REST Example

https://www.codeproject.com/Articles/571813/A-Beginners-Tutorial-on-Creating-WCF-REST-Services

**WCF Rest** To use WCF as WCF Rest service you have to enable webHttpBindings.  
It support HTTP GET and POST verbs by [WebGet] and [WebInvoke] attributes respectively.  
To enable other HTTP verbs you have to do some configuration in IIS to accept request of that particular verb on .svc files  
Passing data through parameters using a WebGet needs configuration. The UriTemplate must be specified  
It support XML, JSON and ATOM data format.

ASP.Net Web API

ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. ASP.NET Web API is an ideal platform for building RESTful applications on the .NET Framework.

REST stands for Representational State Transfer. This is a protocol for exchanging data over a distributed environment. The main idea behind REST is that we should treat our distributed services as a resource and we should be able to use simple HTTP protocols to perform various operations on that resource.

When we talk about the Database as a resource we usually talk in terms of CRUD operations. i.e. Create, Retrieve, Update and Delete. Now the philosophy of REST is that for a remote resource all these operations should be possible and they should be possible using simple HTTP protocols.

Now the basic CRUD operations are mapped to the HTTP protocols in the following manner:

* **GET**: This maps to the R(Retrieve) part of the CRUD operation. This will be used to retrieve the required data (representation of data) from the remote resource.
* **PUT**: This maps to the U(Update) part of the CRUD operation. This protocol will update the current representation of the data on the remote server.
* **POST**: This maps to the C(Create) part of the CRUD operation. This will create a new entry for the current data that is being sent to the server.
* **DELETE**: This maps to the D(Delete) part of the CRUD operation. This will delete the specified data from the remote server.

The problem with using WCF restful services is that we need to do a lot of configurations in a WCF service to make it a RESTful service. WCF is suited for he scenarios where we want to create a services that should support special scenarios such as one way messaging, message queues, duplex communication or the services that need to conform to WS\* specifications.

But using WCF for creating restful services that will provide fully resource oriented services over HTTP is a little complex. Still WCF is the only option for creating the RESTful services if there is a limitation of using .NET 3.5 framework.

## Web Service

1. It is based on SOAP and return data in XML form.
2. It support only HTTP protocol.
3. It is not open source but can be consumed by any client that understands xml.
4. It can be hosted only on IIS.

## WCF

1. It is also based on SOAP and return data in XML form.
2. It is the evolution of the web service(ASMX) and support various protocols like TCP, HTTP, HTTPS, Named Pipes, MSMQ.
3. The main issue with WCF is, its tedious and extensive configuration.
4. It is not open source but can be consumed by any client that understands xml.
5. It can be hosted with in the applicaion or on IIS or using window service.

## WCF Rest

1. To use WCF as WCF [Rest service](http://kellabyte.com/2011/09/04/clarifying-rest/) you have to enable webHttpBindings.
2. It support HTTP GET and POST verbs by [WebGet] and [WebInvoke] attributes respectively.
3. To enable other HTTP verbs you have to do some configuration in IIS to accept request of that particular verb on .svc files
4. Passing data through parameters using a WebGet needs configuration. The UriTemplate must be specified
5. It support XML, JSON and ATOM data format.

## Web API

1. This is the new framework for building HTTP services with easy and simple way.
2. Web API is open source an ideal platform for building REST-ful services over the .NET Framework.
3. Unlike WCF Rest service, it use the full featues of HTTP (like URIs, request/response headers, caching, versioning, various content formats)
4. It also supports the MVC features such as routing, controllers, action results, filter, model binders, IOC container or dependency injection, unit testing that makes it more simple and robust.
5. It can be hosted with in the application or on IIS.
6. It is light weight architecture and good for devices which have limited bandwidth like smart phones.
7. Responses are formatted by Web API’s MediaTypeFormatter into JSON, XML or whatever format you want to add as a MediaTypeFormatter.

## To whom choose between WCF or WEB API

1. Choose WCF when you want to create a service that should support special scenarios such as one way messaging, message queues, duplex communication etc.
2. Choose WCF when you want to create a service that can use fast transport channels when available, such as TCP, Named Pipes, or maybe even UDP (in WCF 4.5), and you also want to support HTTP when all other transport channels are unavailable.
3. Choose Web API when you want to create a resource-oriented services over HTTP that can use the full features of HTTP (like URIs, request/response headers, caching, versioning, various content formats).
4. Choose Web API when you want to expose your service to a broad range of clients including browsers, mobiles, iphone and tablets.

|  |  |
| --- | --- |
| **ASP.NET MVC** | **ASP.NET WebAPI** |
| used to create web applications that returns both view and data | used to create HTTP services, It returns only data. |
| return data in json format using jsonResult | return data in JSON, XML |
| Requests are mapped to actions name. | Requests are mapped to the actions based on HTTP verbs |

More new features introduced in ASP.NET Web API framework v2.0 are as follows:

* Attribute Routing
* External Authentication
* CORS (Cross-Origin Resource Sharing)
* OWIN (Open Web Interface for .NET) Self Hosting
* IHttpActionResult
* Web API OData

|  |
| --- |
|  |
| Web API REST | WCF REST |
| HTTP only. Suitable for access from various browsers, mobile devices etc enabling wide reach. | Support multiple transport protocols (HTTP, TCP, UDP, and custom transports) and allows switching between them. |
| Enables building Web APIs that support wide variety of media types including XML, JSON etc. | Support multiple encodings (Text, MTOM, and Binary) of the same message type and allows switching between them. |
| Uses basic protocol and formats such as HTTP, WebSockets, SSL, Equerry, JSON, and XML. There is no support for higher level protocols. | Supports building services with WS-\* standards like Reliable Messaging, Transactions, Message Security. |
| HTTP is request/response, for additional need SignalR and WebSockets integration. | Supports Request-Reply, One Way, and Duplex message exchange patterns |
| Ships with .NET framework but is open-source | Ships with the .NET framework. |

Web API can be self hosted or ASP.Net hosted

If using ASP.net hosting then we can have asp.net authentication and authorization mechanism,other wise we can use WCF security binding settings.

Authentication in ASP.Net Web API

Allow Anonymous – attribute on the action

Form Authentication

Basic Authentication

Windows Authentication

Authorization in ASP.Net Web API

Authorize attribute on Actions

Web config using <authorization>