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 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

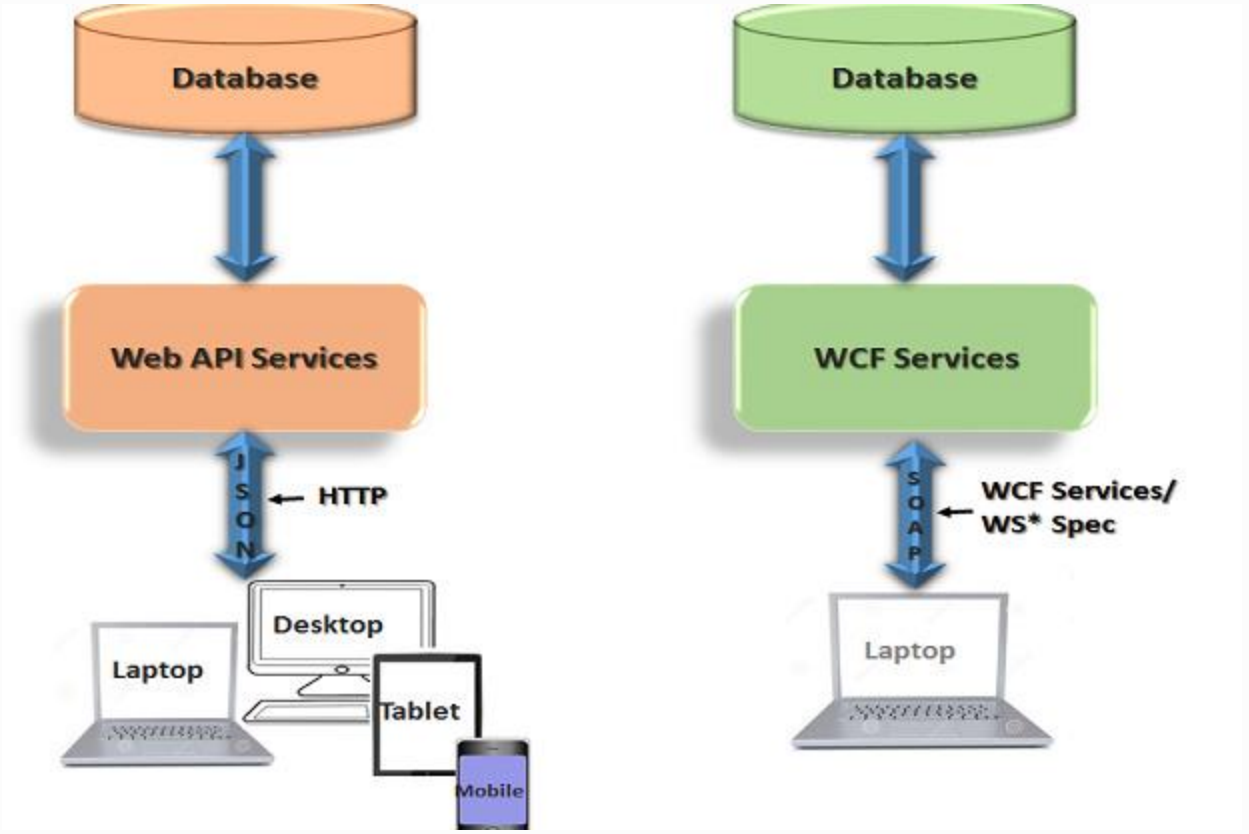
1. This is the new framework for building HTTP services with easy and simple way.
2. Web API is open source an ideal platform for building REST-ful services over the .NET Framework.
3. Unlike WCF Rest service, it use the full featues of HTTP (like URIs, request/response headers, caching, versioning, 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.

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.

Choosing the right technology for building the service layer in .NET



Two prominent contenders you have when designing the service layer in .Net are WCF and Web API. WCF is a development platform for SOA -- it provides many features and supports many different transport protocols. While WCF is a unified framework for building service oriented applications, Web API is a light weight alternative to build RESTful services that can be consumed by many different clients. RESTful services use basic HTTP and are simple with much less payload compared to SOAP services. You can use the WebHttpBinding in WCF to build non-SOAP RESTful services over HTTP. WCF is much more versatile in the sense that it can support many transport protocols -- HTTP, TCP, etc. You can leverage WCF to build secure, reliable, and transactional services that can support messaging, duplex communication and fast transport channels like, TCP, Named Pipes or UDP.

If you need to build lightweight, resource-oriented services over HTTP that can leverage the full features of the HTTP protocol, use versioning, cache control for browsers, and concurrency using Etags, Web API is a good choice. You should choose Web API over WCF in your service layer when you would want to expose your services to a broad range of clients i.e., Web browsers, mobiles, tablets, etc. Web API is light weight and is well suited on devices that have limited bandwidth like smart phones. One of the major constraints I faced while using WCF is its extensive configuration - Web API is much simpler and easy to use. I admit that WCF is much more versatile compared to Web API but, if you don't need the features that WCF provides and all you need is just RESTful services over HTTP, I would always prefer Web API as it is lightweight and simple to use.

I would also like to present a discussion on the differences between Web API and ASP.Net MVC as there are certain misconceptions on when to choose one over the other. The choice between ASP.Net MVC and Web API depends on many factors. There are certain considerations you would need to keep in mind before you decide to use any one of them.

Note that Web API uses HTTP verbs and hence HTTP verb based mapping for mapping methods to respective routes. You cannot have overloaded methods for the same HTTP verb for a particular route. You should be aware of this design constraint (though workarounds are available) when choosing between ASP.Net MVC and Web API. Unlike ASP.Net MVC, Web API uses routing based on HTTP verbs rather than URIs that contain actions. So, you can use Web API to write RESTful services that can leverage the HTTP protocol -- you can design services that are easier to test and maintain. Routing in Web API is much simpler and you can leverage content negotiation seamlessly. The routing model in ASP.Net MVC includes actions in the URIs.

Another point you would want to consider is whether you would like your functionality to be exposed for a specific application or whether the functionality should be generic. If you want to expose your services specific to one application only, you would want to use ASP.Net MVC -- the controller in an ASP.Net MVC application is application specific. On the contrary, you would want a Web API approach if your business needs require you to expose the functionality generically. I would prefer to use the Web API approach if the functionality is more data centric and the ASP.Net MVC approach if the functionality is more UI centric.

You should use Web API over ASP.Net MVC if you would want your controller to return data in multiple formats like, JSON, XML, etc. Also, specifying the data format in Web API is simple and easy to configure. Web API also scores over ASP.Net MVC in its ability to be self-hosted (similar to WCF). You would need ASP.Net MVC controllers to be hosted in the same webserver where the application has been hosted because the ASP.Net MVC controllers are part of the same application. On the contrary, you can host your Web API controllers outside of IIS also -- you can host it in a lightweight custom host and allow the service to be consumed by many different clients.