**What is ASP.NET Web API ?**  
The term API stands for ‘Application Programming Interface’. ASP.NET Web API is a framework for building Web API’s, i.e. HTTP based services on top of the .NET Framework. The most common use case for using Web API is for building RESTful services. These services can then be consumed by a broad range of clients like  
1. Browsers  
2. Mobile applications  
3. Desktop applications  
4. IOTs  
  
**What are IOTs?**  
The term IOTs stands for Internet Of Things. Internet Of Things are the objects or devices that have an IP address and can communicate over the internet with other internet enabled devices and objects. Examples for IoT include security systems, electronic appliances, thermostats, cars etc..., in addition to desktops, laptops, and smart phones.  
  
One important thing to keep in mind is that, though ASP.NET Web API framework is widely used to create RESTful services, it can also be used to create services that are not RESTful. In short, ASP.NET Web API framework does not dictate any specific architeture style for creating services. In this video, we will discuss creating RESTful services from scratch using ASP.NET Web API framework.  
  
**What are RESTful services?**  
REST stands for Representational State Transfer. REST was first introduced in the year 2000 by Roy Fielding as part of his doctoral dissertation. REST is an architectural pattern for creating an API that uses HTTP as its underlying communication method. The REST architectural pattern specifies a set of constraints that a system should adhere to. Here are the REST constraints.  
  
**Client Server constraint** - This is the first constraint. Client sends a request and the server sends a response. This separation of concerns supports the independent evolution of the client-side logic and server-side logic.  
  
**Stateless constraint** - The next constraint is the stateless constraint. The communication between the client and the server must be stateless between requests. This means we should not be storing anything on the server related to the client. The request from the client should contain all the necessary information for the server to process that request. This ensures that each request can be treated independently by the server.  
  
**Cacheable constraint** - Some data provided by the server like list of products, or list of departments in a company does not change that often. This constraint says that let the client know how long this data is good for, so that the client does not have to come back to the server for that data over and over again.  
  
**Uniform Interface**- The uniform interface constraint defines the interface between the client and the server. To understand the uniform interface constraint, we need to understand what a resource is and the HTTP verbs - GET, PUT, POST & DELETE. In the context of a REST API, resources typically represent data entities. Product, Employee, Customer etc are all resources. The HTTP verb (GET, PUT, POST, DELETE) that is sent with each request tells the API what to do with the resource. Each resource is identified by a specific URI (Uniform Resource Identifier). The following table shows some typical requests that you see in an API

| **Resource** | **Verb** | **Outcome** |
| --- | --- | --- |
| /Employees | GET | Gets list of employees |
| /Employee/1 | GET | Gets employee with Id = 1 |
| /Employees | POST | Creates a new employee |
| /Employee/1 | PUT | Updates employee with Id = 1 |
| /Employee/1 | DELETE | Deletes employee with Id = 1 |

Another concept related to Uniform Interface is **HATEOAS**. HATEOAS stands for Hypermedia as the Engine of Application State. All this means is that in each request there will be set of hyperlinks that let's you know what other actions can be performed on the resource. If this is not clear at the moment, don't worry, we will discuss this in a later video.  
  
There are 2 other constraints as well  
Layered System  
Code on Demand (optional)  
  
Code on Demand constraint is optional. We will discuss these 2 constraints in a later video.  
  
**Difference between WCF and Web API. When to choose one over the other?**  
WCF (Windows Communication Foundation) - One of the choices available in .NET for creating RESTful services is WCF. The problem with WCF is that, a lot of configuration is required to turn a WCF service into a RESTful service. The more natural choice for creating RESTful services is ASP.NET Web API, which is specifically created for this purpose.  
  
WCF is more suited for building services that are transport/protocol independent. For example, you want to build a single service, that can be consumed by 2 different clients - Let's say, a Java client and .NET client. Java client expects transport protocol to be HTTP and message format to be XML for interoperability, where as the .NET client expects the protocol to be TCP and the message format to be binary for performance. For this scenario WCF is the right choice. What we do here is create a single WCF service, and then configure 2 end points one for each client (i.e one for the Java client and the other for the .NET client). If you are new to WCF, please watch our WCF video series. I will have the link available in the description of this video.  
  
There is nothing wrong to use WCF to create RESTful services. It's just that it's a bit more complex and configuration can be a headache. If you are stuck with .NET 3.5 or you have an existing SOAP service you must support but want to add REST to reach more clients, then use WCF.   
  
If you don't have the limitation of .NET 3.5 and you want to a create brand new restful service then use ASP.NET Web API.