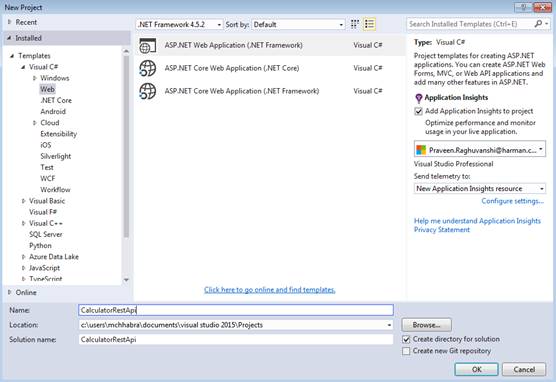
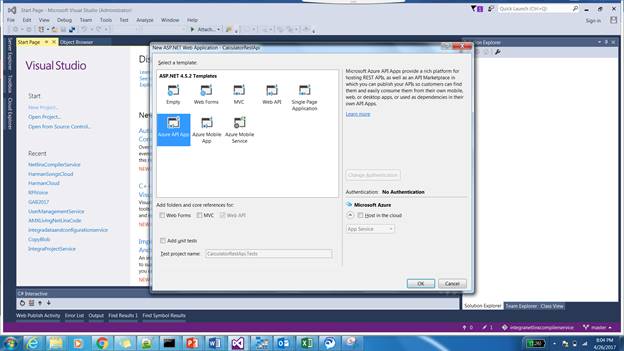
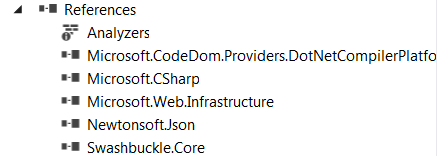
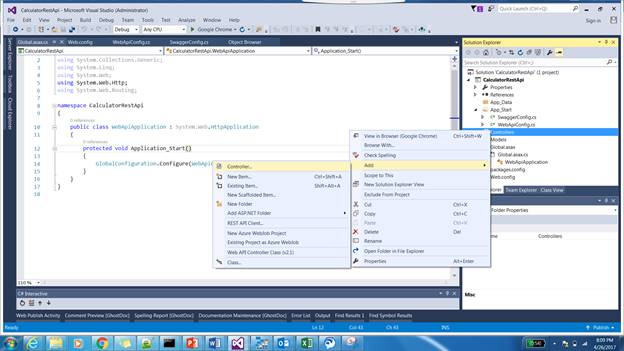
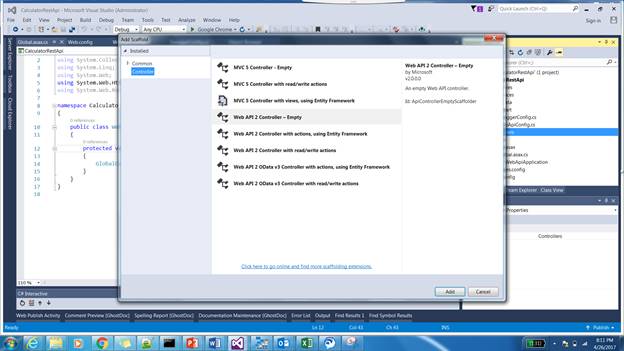
* inbuilt Swagger Integration.
* Ability to push your API APPS into Azure MarketPlace.
* API definition.
* Support for creating an Azure API client from Visual Studio.

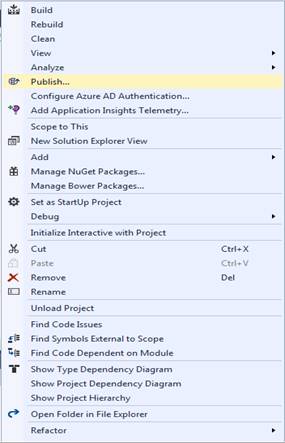
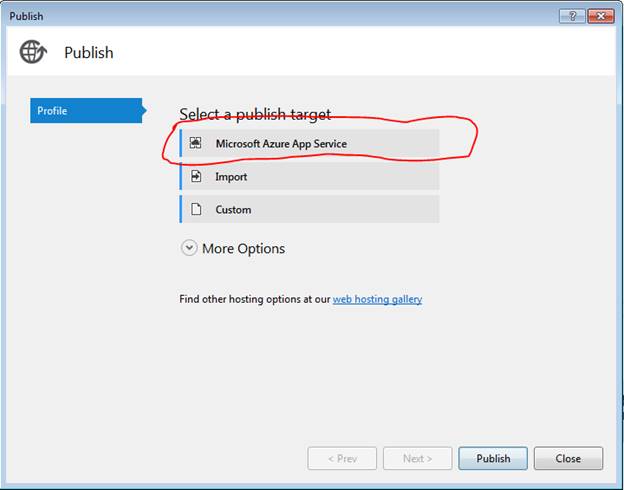
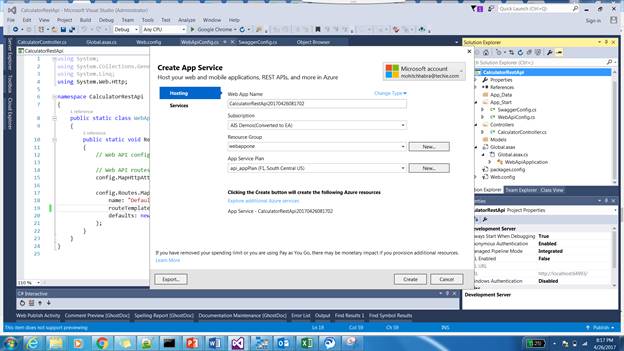
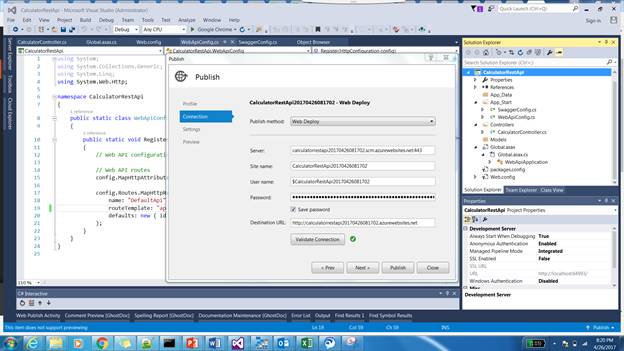
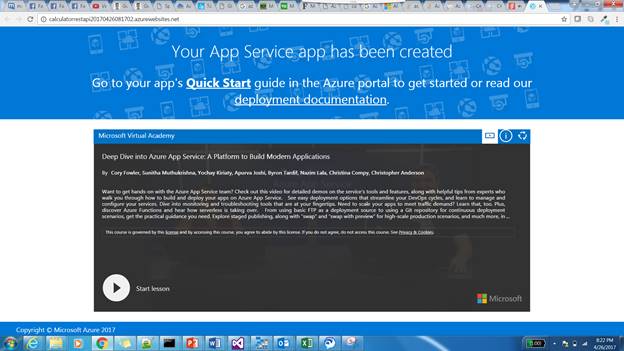
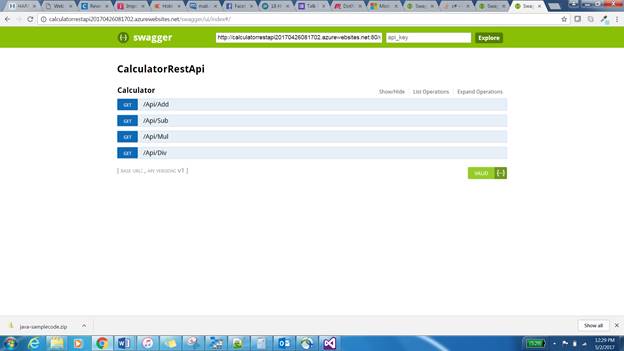
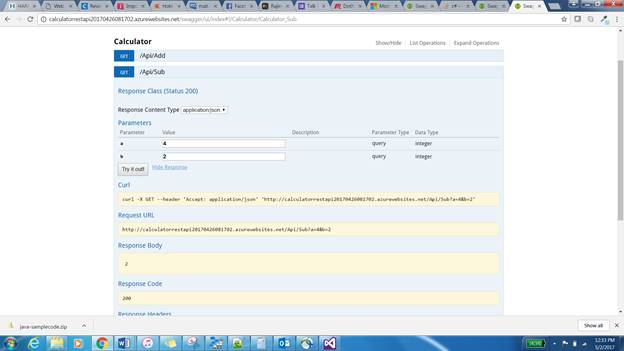
We are going to create a demo and discuss all the 4 main reasons along with the demo.

**Create an API from Visual Studio and host it in Azure API app**

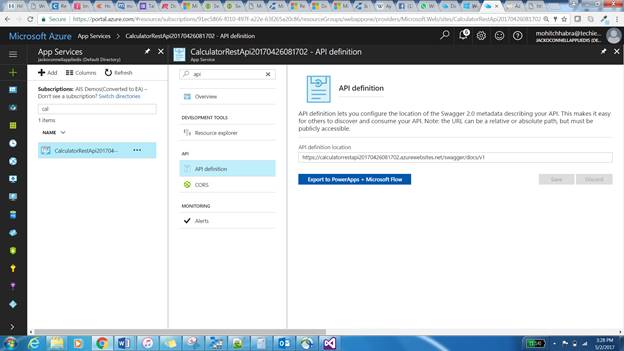
* Go to Visual Studio -> Visual C# -> Web -> ASP.NET Web Application, enter the name of the API and click OK button.  
    
  
* Now, select Azure API app from the dialog box. We can select Web API as well and then publish it as Azure API app, which will also serve the same purpose.  
    
  

**Inbuilt Swagger Integration**

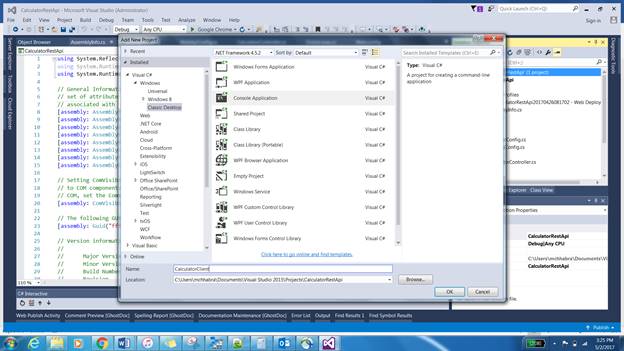
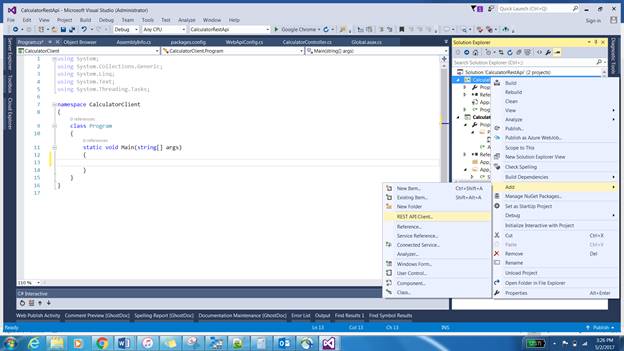
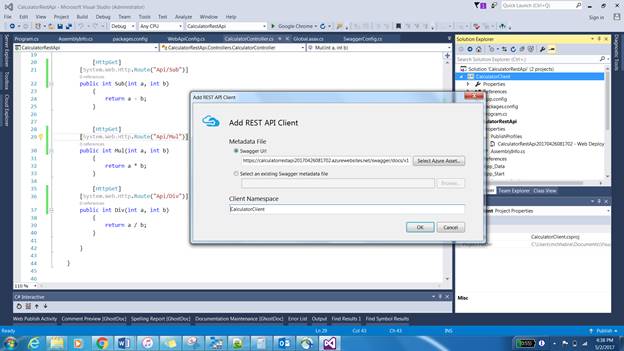
* As we have selected an Azure API app, some of the common Web API used packages like Newtonsoft.Json and Swasbuckle.core (Swagger) come directly in the template.  
    
  
* Create an API Controller by right clicking on the Controller and Add -> Controller.  
    
  
* Now, select the Web API 2 Empty Controller. You can use any of the controller sbut for this demo, we are going to use the empty controller.  
    
  
* Name the controller as Calculator Controller and as we selected the WEB API 2 template, its going to be derived from the APIController. Add the code given below in it.
  1. **public** **class** CalculatorController : ApiController
  2. {
  3. // GET api/values/5
  4. [HttpGet]
  5. [System.Web.Http.Route("Api/Add")]
  6. **public** **int** Add(**int** a, **int** b)
  7. {
  8. **return** a + b;
  9. }
  11. [HttpGet]
  12. [System.Web.Http.Route("Api/Sub")]
  13. **public** **int** Sub(**int** a, **int** b)
  14. {
  15. **return** a - b;
  16. }

  19. [HttpGet]
  20. [System.Web.Http.Route("Api/Mul")]
  21. **public** **int** Mul(**int** a, **int** b)
  22. {
  23. **return** a \* b;
  24. }
  26. [HttpGet]
  27. [System.Web.Http.Route("Api/Div")]
  28. **public** **int** Div(**int** a, **int** b)
  29. {
  30. **return** a / b;
  31. }
  33. }
* The code given above is going to hit with the route API/Controller/Action/ID given below, so we need to go to WebAPI Config and update it accordingly.  
    
  **WebApiConfig**
  1. config.Routes.MapHttpRoute(
  2. name: "DefaultApi",
  3. routeTemplate: "api/{controller}/{action}/{id}",
  4. defaults: **new** { id = RouteParameter.Optional }
  5. );
* Now, we need to publish the API to Azure API app instance. Right click on the project and click Publish.  
    
  
* Select Azure API app and enter your credentials to authenticate and login to your subscription. If you have an existing API app, you can directly skip to #.  
    
  
* Enter the name and select a Resource Group, AppService plan in the approproiate subscription.  
    
  If you don’t have the Resource Group or an app Service plan, you can create from the same wizard by clicking the respective new button and pass the appropriate values. Once all the values are passed, click Create button. It’s going to create an Azure API Web app in your Azure account.  
    
  
* Once completed, the published metadata file will be downloaded and subsequently you can click Publish button to push your binaries.  
    
  
* Once the publish is complete, it’s going to open up the URL in your Browser.  
    
  
* Append the swagger to the URL and you can see all the Methods, which we have created in our code above.  
    
  
* Swagger UI also allows you to test the methods by acting as a Client. We are trying the Sub method and passed two params to it and clicked the “TRY IT NOW” button. Now, we will contact the API and return the result. This is very useful in case of APIs, as you can straightaway test your APIs and see if it's working fine or not without writing a single line of code.  
    
  

**API definition**

* Now, go to an Azure portal and go to your resource. Select the API definition and you can see that it gives you an option to export the metadata to the PowerApps and Microsoft Flow.  
    
  

**Support for creating an Azure API client from Visual Studio**

* Now, add a new project and select a console Application.  
    
  
* Right click on the project and add REST API client.  
    
  
* Now, add the metadata URL or select an Azure asset, which we have created and click OK button to download the metadata associated with it. Now, consume the Service, as we do in any other client with the supported metadata.  
    
  

**Summary**

Azure API app is a platform as a Service solution under the umbrella of Web Apps. It gives you the inbuilt features to work with APIs seamlessly like inbuilt support for the metadata using swagger, along with the API definition support on the portal. It also gives us the option of publishing the API into an Azure marketplace, which can also be consumed by Logic App as a connector.