

## Azure and WCF

To quote right off the Microsoft site “The Azure™ Services Platform is an internet-scale cloud computing and services platform hosted in Microsoft data centers. The Azure Services Platform provides a range of functionality to build applications that span from consumer web to enterprise scenarios and includes a cloud operating system and a set of developer services “. There is much more information at <http://www.microsoft.com/azure/whatisazure.mspx>.



In some respects it is a move back in time to the days of mainframes and dumb terminals as the infrastructure to deliver robust scalable applications is moved into the sky. Microsoft is of course not the only player here as Amazon and Google are also prominent. This article will explore how to host a simple calculator service in Azure and expose its functionality thru Windows Communication Foundation(WCF) . Of course this is a simple example but this will let us build robust highly scalable server applications that are hosted in Azure. I will not go into details on the setup of Azure but will zoom in on the actual code as there are many articles on these issues online.

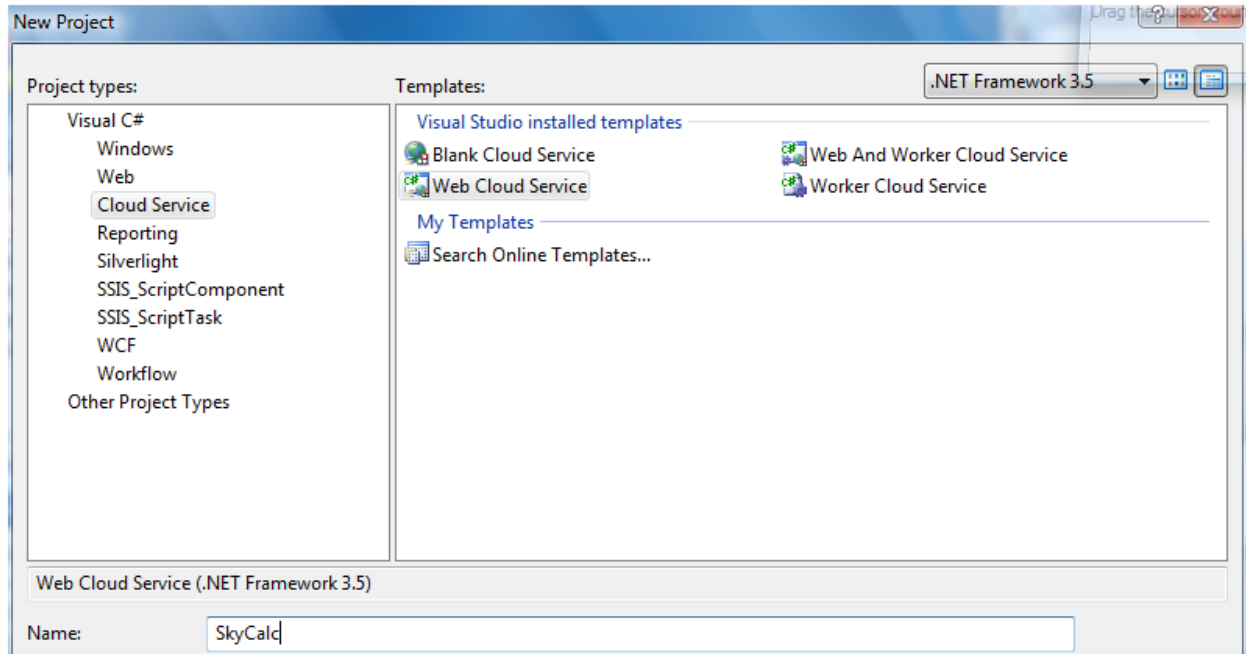
First one must register for Azure services , I will not repeat what others have written about and refer you to an excellent article by Ryan Dunn at <http://dunnry.com/blog/CreateAndDeployYourWindowsAzureServiceIn5Steps.aspx>. Microsoft also supplies what is called the development fabric that simulates both the Windows Azure Fabric, as well as Windows Azure Storage on your local machine so you are able to build and test applications taking advantages of these services without deploying to or accessing the cloud. When you are satisfied with the local version then it is deployed to the sky. To use this local development one must install:

- [The Windows Azure SDK](#)
- [Visual Studio Tools](#)

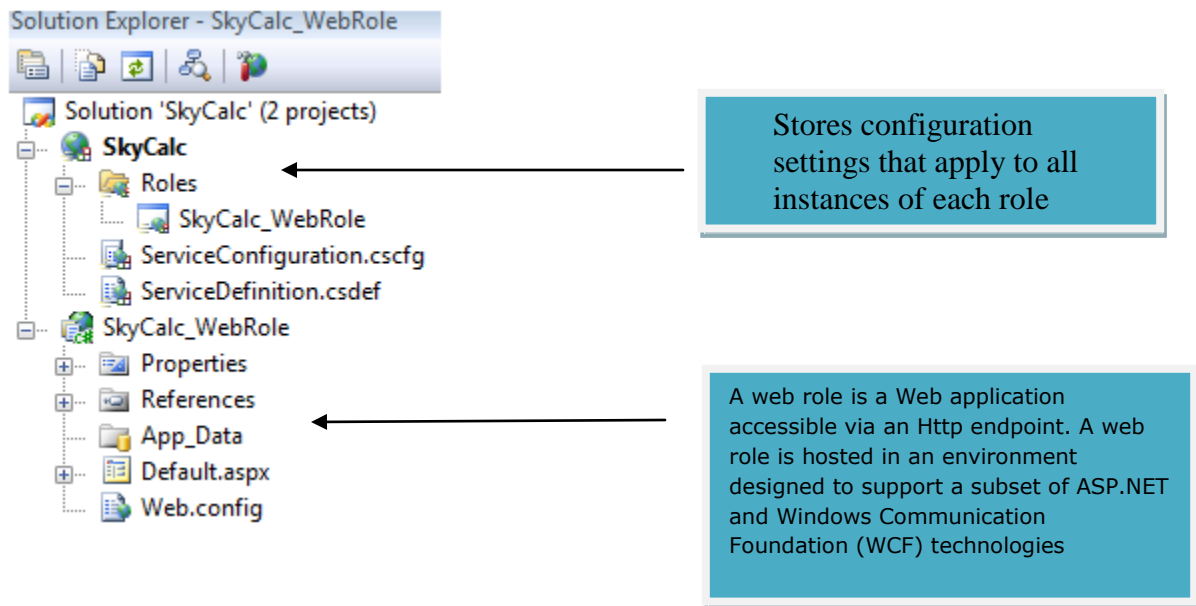
We will now use these to develop a calculator service, I actually used the logic of a Microsoft example for this. I believe that they made the WCF part a little more complex than it needs to be as we will implement a Restful WCF service. **Rest** stands for **Representational State Transfer** and in short a Restful service uses a unique URI to reference every resource, and HTTP verbs to define actions on those resources. For a more complete discussion please see my article at <http://www.sandkeysoftware.com/WCF/Restful/Restful.pdf>. This approach will make the

service accessible by web services hosted in all browsers such as iPhones, unlike a Soap based call. This also gets around some issues that Azure has with Soap at the time of writing. The service will then be consumed by a Silverlight client.

After the Azure SDK has been loaded on your machine, create a new Web cloud service called SkyCalc:



This will produce a solution that looks like:



Now add a WCF service to the SkyCalc\_Webrole by right clicking on “Add New Item” and selecting WCF service and call it Calc. I am assuming that reader is familiar with WCF (if not check out my primer at <http://www.sandkeysoftware.com/WCF/wcfrecepte.pdf>) so I will not bore you with details here. Modify the generated ICalc.cs file so that it looks like:

```
[OperationContract]
MyResult Calc(string op, int op1, int op2);
```

And the implementation of the interface contains the code for a simple calculator and looks like:

```
public MyResult Calc(string op, int op1, int op2)
{
    MyResult r = new MyResult();
    switch (op)
    {
        case "add":
            r.result = op1 + op2;
            break;

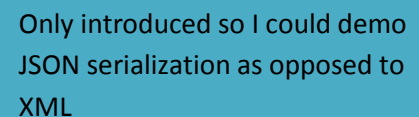
        case "subt":
            r.result = op1 - op2;
            break;

        case "mult":
            r.result = op1 * op2;
            break;

        case "div":
            r.result = op1 / op2;
            break;
    }

    return r;
}
```

```
public class MyResult
{
    public string operation { get; set; }
    public int result { get; set; }
}
```



Only introduced so I could demo JSON serialization as opposed to XML

So far we have a normal Soap based WCF service so now we make the changes to turn this into a Restful service where the parameters are passed though the Uri for the service e.g. [www.mysite.com/Calc?op=add&op1=2&op2=6](http://www.mysite.com/Calc?op=add&op1=2&op2=6). This will result in call to Calc method to add 2 and 6.

Firstly we modify the ICalc interface so it looks like:

Using System.ServiceModel.Web;

```
[OperationContract]
[WebInvoke(Method = "GET",
    ResponseFormat = WebMessageFormat.Json)]
MyResult Calc(string op, int op1, int op2);
```

Accessible by HTTP "Get" we also add

Use Javascript Notation serialization

Now we are no longer a SOAP based call and now we have to satisfy the WCF hosting requirements by modifying the Calc.svc file to add a hosting Factory that is Web enabled as follows: ??????

```
<%@ ServiceHost Language="C#" Debug="true"
Service="SkyCalc_WebRole.MySkyCalc" CodeBehind="MySkyCalc.svc.cs"
```

The WebServiceHost provides the hosting of non-Soap services, creates a default Service endpoint and disables WSDL-Get Support (SOAP)

```
Factory="System.ServiceModel.Activation.WebServiceHostFactory" %>
```

This will setup a default endpoint and HTTP behavior so we can remove the **ServiceModel** definition from the web.config as the WebServiceHost is providing the WCF plumbing that we need.

Do not miss this step

Now this is real important as the Silverlight client will be making a crossdomain call from our Silverlight client so we must have a Crossdomain file or **clientaccesspolicy.xml** in place or else the call will fail with a security exception. The file contents our file which will let everyone call us looks like:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE cross-domain-policy SYSTEM
"http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
  <allow-http-request-headers-from domain="*" headers="*" />
</cross-domain-policy>
```

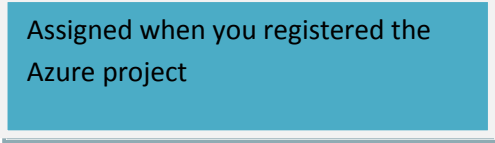
## Silverlight Client

Now our calculator service is good to go and we will first run and test it in the local Development Fabric by developing our Silverlight client that will consume the service as a WCF Restful call. The downside of this approach is that we cannot use a service reference to define the

WCF service as we have no WSDL to generate the service from so we drop back to using WebClient . However first I set up a couple of variables for the local and Azure server address so we can run against the local or remote fabric.

```
string LocalServer = "http://127.0.0.1:81";  
string CloudServer =  
"http://3296beb9d6384d16858099e1ab6f84c8.cloudapp.net/";
```

Assigned when you registered the Azure project



We then form the rest of the URI by appending the service operation which is Calc , the operation and operands that are entered from the UI. So we end up with a URI that looks like:

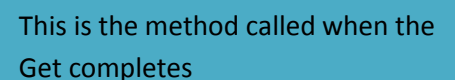
```
http://127.0.0.1:81/MySkyCalc.svc/Calc?op=add&op1=5;op2=10
```

```
if (newOperation == "=")  
{  
    string servicemethod = LocalServer + "/MySkyCalc.svc/Calc?";  
    switch (previousOperation)  
    {  
        case "+":  
            servicemethod = servicemethod + "op=add&op1=" +  
                value.ToString() + "&op2=" + number.ToString();  
            break;  
        case "-":  
            servicemethod = servicemethod + "op=subt&op1=" +  
                value.ToString() + "&op2=" + number.ToString();  
            break;  
        . . .  
    }  
}
```

As mentioned above we then use WebClient to issue the call to the WCF Restful service, it is asynchronous like all Silverlight communications.

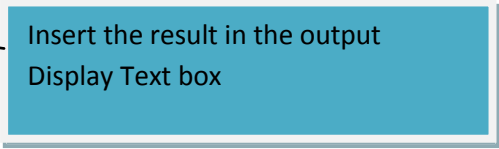
```
Uri serviceuri = new Uri(servicemethod);  
WebClient client = new WebClient();  
client.OpenReadCompleted += new  
OpenReadCompletedEventHandler(client_OpenReadCompleted);  
client.OpenReadAsync(serviceuri);
```

This is the method called when the Get completes



The operation has completed and we now Deserialize the result using the JsonSerializer, we could have used the standard XML serializer here but the JSON format is much more compact which is important in real world application,

```
void client_OpenReadCompleted(object sender,
OpenReadCompletedEventArgs e)
{
    Stream stm = e.Result;
    DataContractJsonSerializer ser = new
DataContractJsonSerializer(typeof(MyResult));
    MyResult valueComputed = ser.ReadObject(stm) as MyResult;
    display.Text = valueComputed.result.ToString();
}
```



Insert the result in the output  
Display Text box

Now that we have run in the local development fabric lets move the calculator into the sky! I again refer you to Ryan Dunn's article Step 5. The important part is that we are copying the service package to our hosted service project in Azure. Now we can change to local server address to our hosted address and we are in business.

My next article will be on using the Data store. I also refer you to a complete list of my articles at <http://www.sandkeysoftware.com/Silverlight/ArticleNavigator/ArticleNavigator.Web/ArticleNavigatorTestPage.html>. You will need Silverlight 3 to access the article navigator.