**PaaS - Development of C# WCF web services with AWS Elastic Beanstalk**

**Objective:** Get practical experience in C# WCF Web services development and deployment on Cloud PaaS with AWS Elastic Beanstalk as well as in creating and running web service clients.

**Tasks:**

1. Development and launching of your own C# WCF Web Service
2. Testing a Web Service using SoapUI tool
3. Developing and running a WCF Web Service C# client
4. Deployment of our own C# WCF Web Service on AWS Elastic Beanstalk PaaS service

**Lab environment**:

* Visual Studio with the following workloads:
  + ASP.NET and web development
  + Azure development
  + AWS Toolkit for Visual Studio (<https://aws.amazon.com/visualstudio/>)
* SoapUI tool

*Contents*

[Step 1: Developing your own WCF Web Service](#_heading=h.3znysh7) 3

[Step 2: Developing a new WCF client](#_heading=h.93vt28e0j41c) 6

[Step 3: Deploying your WCF Web Service on AWS Elastic Beanstalk](#_heading=h.tyjcwt) 10

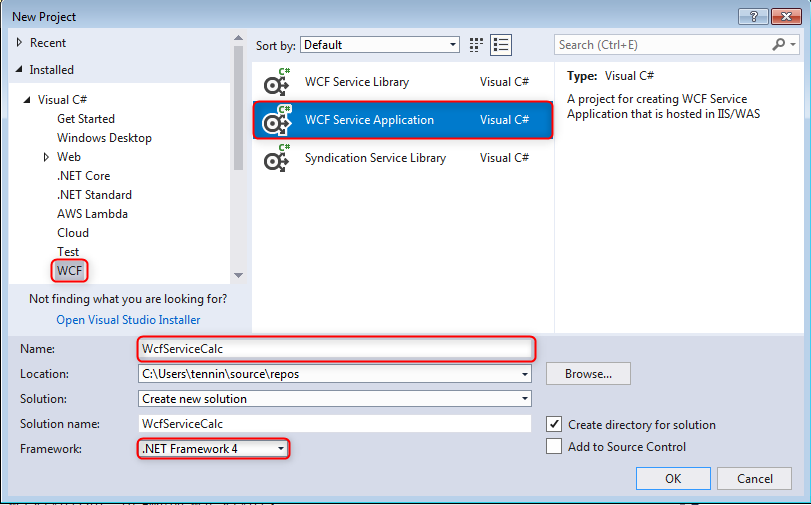
[Step 4: Testing your WCF Web Service](#_heading=h.3dy6vkm) 11

[Step 5: Removing AWS Elastic Beanstalk Resources](#_heading=h.1t3h5sf) 11

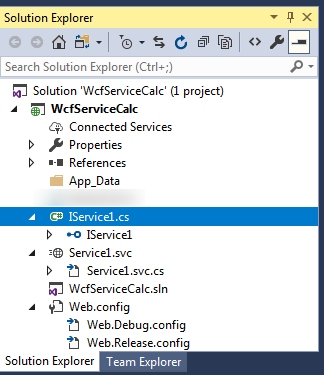
# Step 1: Developing your own WCF Web Service

Using the example considered in the previous lab, let develop a simple WCF Web Service and a Client on your own. It can be an advanced calculator performing an extended set of arithmetic operations, currency exchange rate converter, a converter between metric and imperial units, etc. Do not forget to delete AWS resources after use.

1. Open Visual Studio and create a new WCF project; provide project name and make sure you use .NET Framework 4.0 which is supported by AWS Elastic Beanstalk



1. This is how the project should look in the solution explorer after you hit the “OK” button in the above screen.



1. Replace the operation contracts in the **IService1.cs** with your contract, e.g.:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.ServiceModel.Web;

using System.Text;

namespace WcfServiceCalc

{

[ServiceContract]

public interface IService1

{

[OperationContract]

double Add(double n1, double n2);

[OperationContract]

double Subtract(double n1, double n2);

[OperationContract]

double Multiply(double n1, double n2);

[OperationContract]

double Divide(double n1, double n2);

}

}

The interface or service contract defines the methods that will be exposed when the WCF service is deployed.

1. Replace the functions in the Service1.svc.cs with the code implementing functions from your service contract, e.g.:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.ServiceModel.Web;

using System.Text;

namespace WcfServiceCalc

{

public class Service1 : IService1

{

public double Add(double n1, double n2)

{

return n1 + n2;

}

public double Subtract(double n1, double n2)

{

return n1 - n2;

}

public double Multiply(double n1, double n2)

{

return n1 \* n2;

}

public double Divide(double n1, double n2)

{

return n1 / n2;

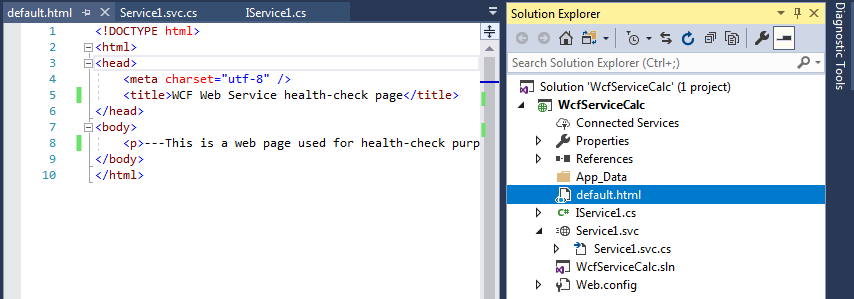
}

}

}

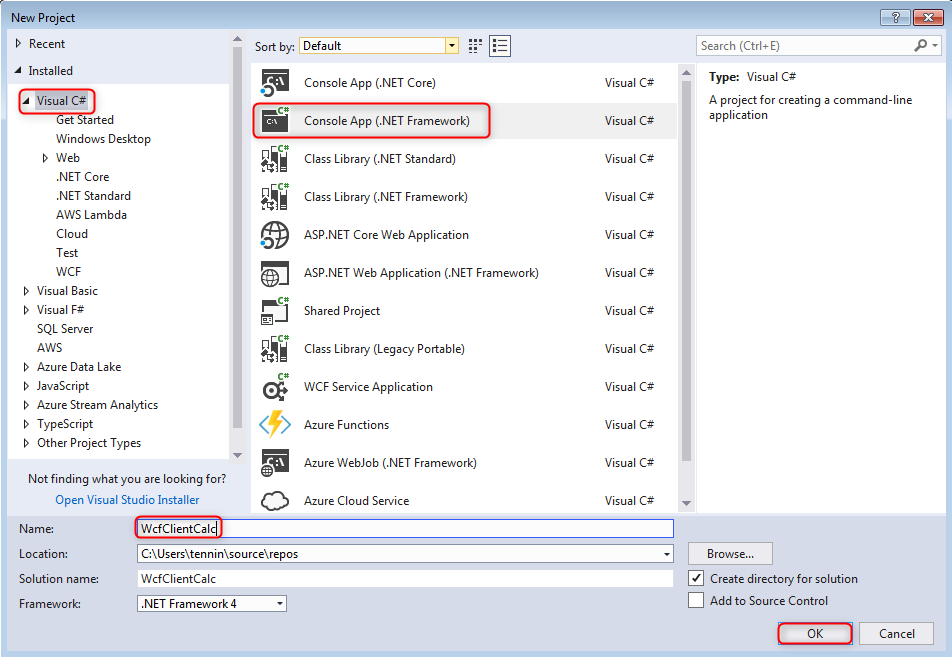
1. Select Service1.svc or Service1.svc.cs at the Solution Explorer and start debugging. The WCF Test Client for testing this service will be launched automatically.
2. Make sure you have added a simple static **default.html** page to your WCF Service project before deploying your project on AWS Amazon Beanstalk.

This static page is needed to path the ‘health check’ when deploying the service with the Cloud provider.

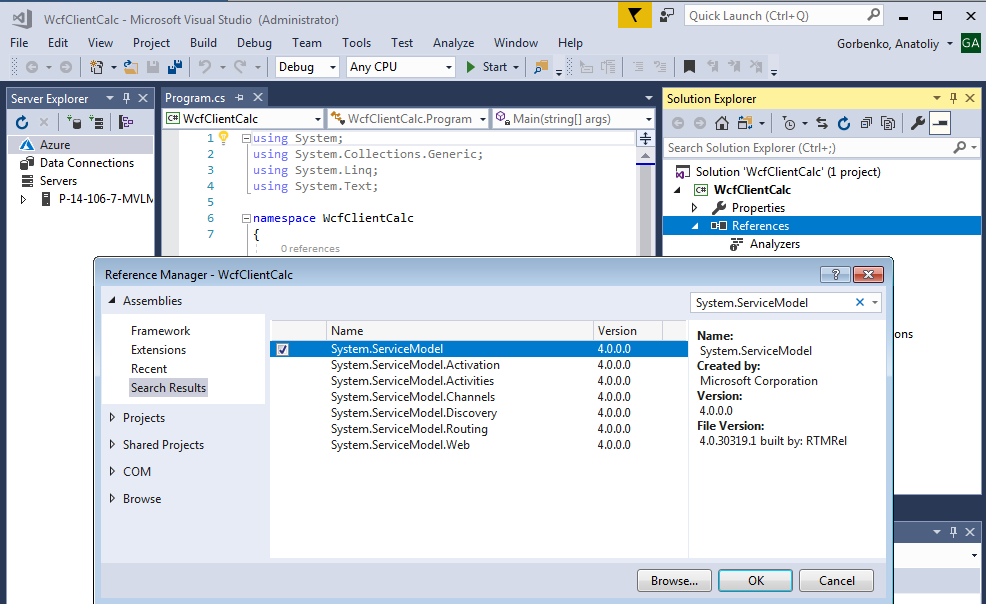


# Step 2: Developing a new WCF client

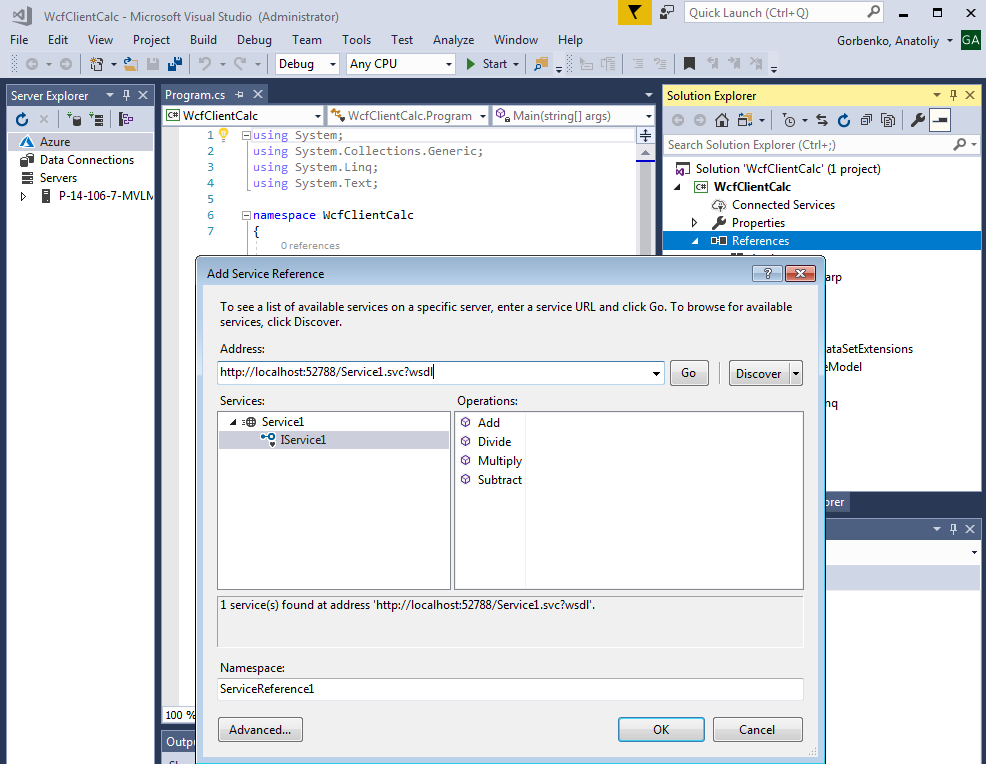
1. Open a new Visual Studio.
2. Create a new console application project by right-clicking on the Getting Started solution, selecting, **Add**, **New Project**. In the **Add New Project** dialog on the left hand side of the dialog select **Visual C#**. In the center section of the dialog select **Console Application (.NET framework)**. Name the project (e.g. WcfClientCalc).



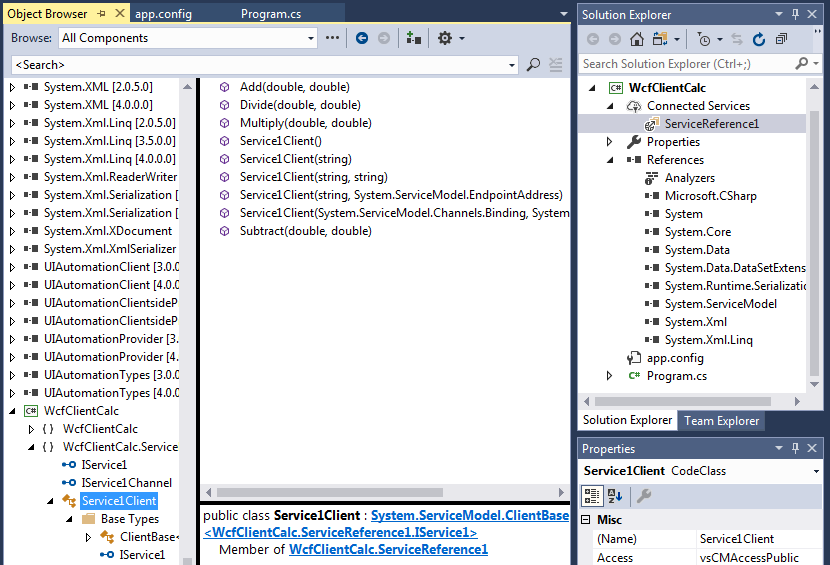
1. Add a reference to System.ServiceModel to the project by right-clicking the **Reference** folder under the project name in Solution Explorer and select **Add** Reference. In the **Add Reference** dialog select **Framework** on the left-hand side of the dialog. In the Search Assemblies textbox, type in System.ServiceModel. In the center section of the dialog pick up **System.ServiceModel**, and click the **OK** button.

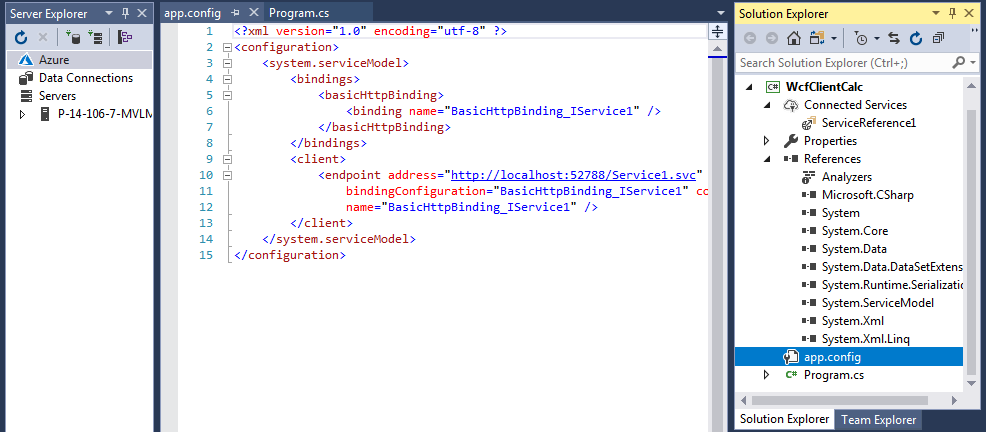


1. Next you will add a service reference to the Service. Before you can do that, be sure that the service is already running and its WSDL description can be viewed on-line (e.g. <http://localhost:52788/Service1.svc?wsdl> ).
2. Right click the References folder under the project name in the Solution Explorer and select Add Service Reference and type in the URL to the service WSDL description in the address box of the Add Service Reference dialog and click the **Go** button. The Service name should then be displayed in the Services list box, Double click Service name and it will expand and show the service contracts implemented by the service. Leave the default namespace (ServiceReference1) as is and click the **OK** button.



1. When you add a reference to a service using Visual Studio a new item (generated proxy in C#) will appear in the Solution Explorer under the Connected Services folder under the project name. Now you can inspect it as well as changes made in app.config file.





1. Open the Program.cs from the project and replace the existing code with your code, e.g.:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using WcfClientCalc.ServiceReference1;

namespace WcfClientCalc {

class Program {

static void Main(string[] args) {

Service1Client client = new Service1Client();

// Ener arguments from console.

Console.Write("Enter value 1: ");

double value1 = Convert.ToDouble(Console.ReadLine());

Console.Write("Enter value 2: ");

double value2 = Convert.ToDouble(Console.ReadLine());

// Call the Add service operation.

double result = client.Add(value1, value2);

Console.WriteLine("\nAdd({0},{1}) = {2}", value1, value2, result);

// Call the Subtract service operation.

result = client.Subtract(value1, value2);

Console.WriteLine("Subtract({0},{1}) = {2}", value1, value2, result);

// Call the Multiply service operation.

result = client.Multiply(value1, value2);

Console.WriteLine("Multiply({0},{1}) = {2}", value1, value2, result);

// Call the Divide service operation.

result = client.Divide(value1, value2);

Console.WriteLine("Divide({0},{1}) = {2}", value1, value2, result);

Console.WriteLine("\nPress <ENTER> to terminate client.");

Console.ReadLine();

//Closing the client releases all communication resources.

client.Close();

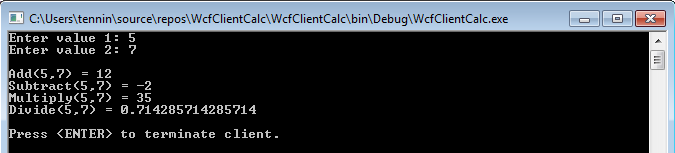
}

}

}

Notice the using or imports statement that imports the **<project\_name>.ServiceReference1**. This imports the code generated by Add Service Reference in Visual Studio. The code instantiates the WCF proxy and then calls each of the service operations exposed by the calculator service, closes the proxy and terminates. Do not forget to close a client’s connection to the service before terminating the program.

1. Run the program



# Step 3: Deploying your WCF Web Service on AWS Elastic Beanstalk

Deploy your WCF Web Service on AWS Elastic Beanstalk following an instruction from the previous lab.

# Step 4: Testing your WCF Web Service

Test your WCF Web Service deployed on AWS Elastic Beanstalk using SoapUI and a desktop client by following an instruction from the previous lab. Do not forget to update target URL in your client’s code.

# Step 5: Removing AWS Elastic Beanstalk Resources

Remove AWS Elastic Beanstalk resources after use.