POLITÉCNICO DE LEIRIA
ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

www.dei.estg.ipleiria.pt

Systems Integration

Web Services – SOAP

Topics:

- Consume public web services
- Implement a WCF Service (SOAP), with .NET Framework.
- Create, publish, and consume web-services using the development server
- Install and configure IIS
- Publish web-services in IIS
- Consume web-services hosted by IIS
- Publish web-service in cloud services

Requires:

- Visual Studio 2019
- IIS - Internet Information Services installed and supporting WCF services (.svc files)
- AppHarbor account - fully hosted .NET Platform as a Service

Duration: 2 classes

WINDOWS COMMUNICATION FOUNDATION 4.5

"The WCF is a part of the .NET Framework that provides a unified programming model for rapidly building service-oriented applications that communicate across the web and the enterprise."

- PART 1 -
HOW TO CONSUME PUBLIC WEBSERVICES

Create a **Windows Forms Application** project named *WindowsApplicationGlobalClient* to consume public Web Service available in the cloud.

1. Explore the service Holiday available in https://store.services.sapo.pt/en/cat/catalog/utility/free-api-holiday/coverage.
The data contact of this service is available at http://services.sapo.pt/Metadata/Contract/Holiday?culture=PT

1.1. Using the above service, create a Form to show the holidays of a specific year.

- A Service reference with the URL of the web service must be added in your project.
- A **service reference**[1] enables a project to access one or more Web services. Use the **Add Service Reference** dialog box to search for Web services locally, on a local area network, or on the Internet.
- Look at the **endpoints** data generated at the configuration file of your application (App.config).
- Using the Object Explorer View, explore the content of the Service Reference added to the project.

1.2. The year must be defined by the user in a Textbox.

1.3. Show the holidays information in a *ListBox*. Explore the data types (classes) available in the service.

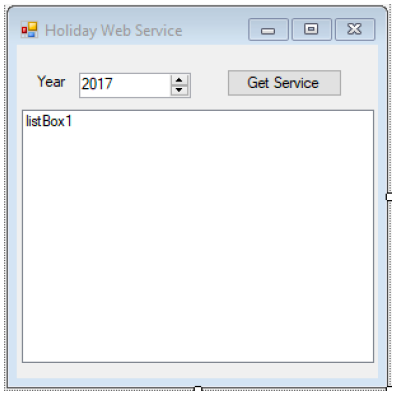


Figure 1 - Holiday web service client

[1] How to: Add, Update, or Remove a Service Reference: https://msdn.microsoft.com/en-us/library/bb628652.aspx

- PART 2 -
IMPLEMENT A WEB SERVICES (WCF SERVICE - SOAP) AND RUN IT IN IIS EXPRESS

2. Create a Web Service named '**WcfServiceBookstore'** (use the template WCF Service Application).

2.1. The template generates several files. For now, analyze and understand the relevance of the following files:

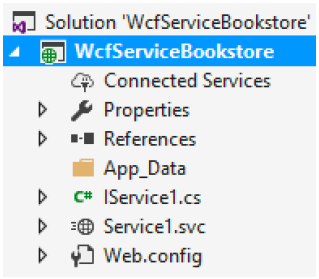


Figure 2 - WCF Service Application template structure. Additionally, the name of the Service was change from Service1 to ServiceBookstore

- **Service1.svc**: contains implementation of the methods specified in the interface.
- **IService1.cs:** contains an interface with the signature of the methods to implement and the definition of composite types.

2.2. Rename the service from Service1 to **ServiceBookstore**.

2.3. Run the service application. You should get a message box warning you about "Debugging Not Enabled". As any other web application, debug should only be enabled in a development environment. For now, any option is acceptable. Notice how the application runs and is hosted by the development server. Why does this happen? Where in "project properties" can one change these settings?

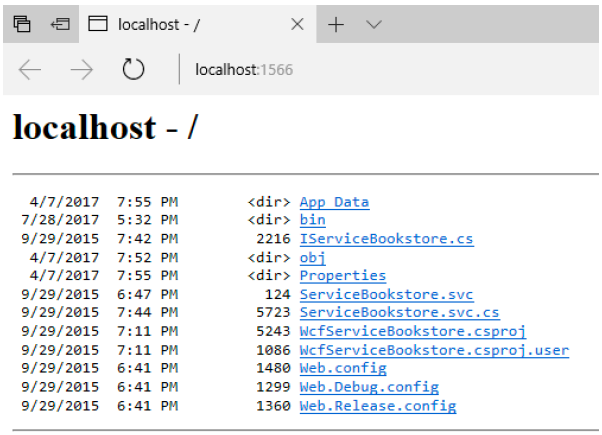


Figure 3 - Service Application running on the development server (IIS Express).

2.4. Click 'ServiceBookstore.svc'. A page with a message 'You have created a service' should appear and a link to the service descriptor.

2.5. Test the service just created using the **WCF Test Client tool**. This tool enables the developer to use a GUI tool to test the input parameters, submit that input to the service and view the response that the service sends back.

2.5.1. Right click '**ServiceBookstore.svc**' in your solution explorer and select "**Set As Start Page**".

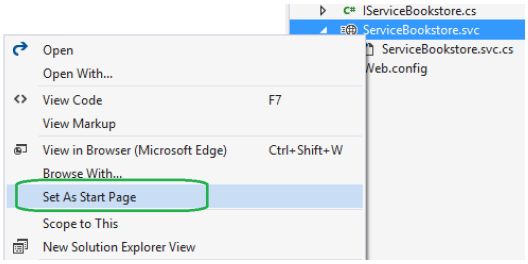


Figure 4 – Set start page of the web service

2.5.2. Test the behavior of the web service methods using the WCF Test Client. See if all methods are returning the correct values.

2.5.3. Additionally, notice that the address (URL) of the web service is shown in the left panel. This URL is available in development mode. You may copy this URL (address) and test it in browser.

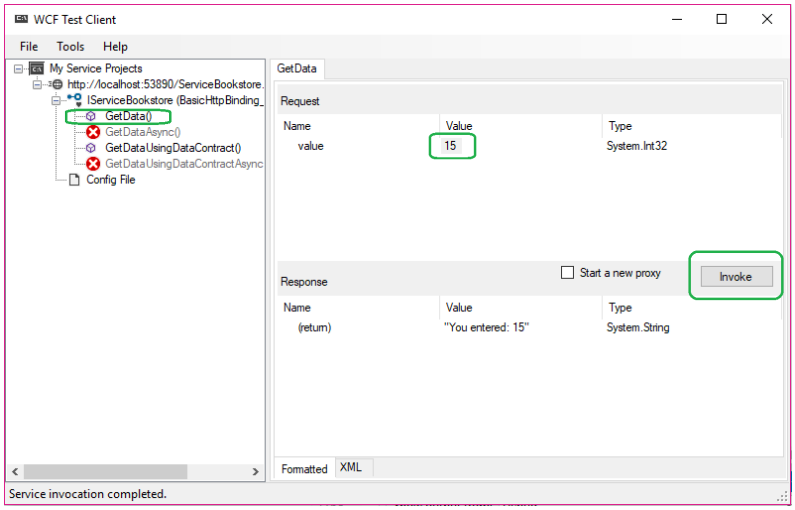


Figure 5 – WCF Test Client

Now that we have tested the generated service templated, we will start implementing our behaviors.

2.6. Add to the **App_Data** folder the xml file available in the course web page (bookstore.xml). This file will persist the data in the web service. Note: In a real scenario, a database would be recommended.

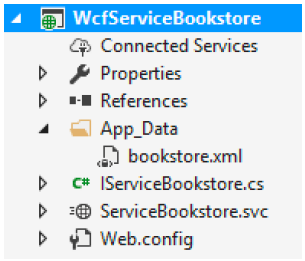


Figure 6 - Bookstore.xml file in the App_Data folder

2.7. Add a **data contract** named **`Book`** with the following data members: Title (string), Author (string), Year (int), Price (double) and Category (enumeration).

2.8. Declare an enumeration (enum) named BookCategory that accepts the values: Web, Children, Science, Romance, Biographies and Other.

2.9. Add the operations `AddBook`, `GetAllBooks`, `GetBooks by category`, `GetBook by title` and `DeleteBook`. All operations must use (return) the data contract specified in the previous exercise.

```
[OperationContract]
1 reference
void AddBook(Book newBook);

[OperationContract]
1 reference
List<Book> GetBooks();

[OperationContract]
1 reference
List<Book> GetBookByCategory(BookCategory category);

[OperationContract]
1 reference
Book GetBookByTitle(string title);

[OperationContract]
1 reference
bool DeleteBook(string title);
```

Figure 7 - Suggested operation to be implemented in the web service

2.10. Implement the behavior of the operations, which must read/write from a XML file named *bookstore.xml*. You may use the *bookstore.xml* file created in the previous worksheet. Such behaviors were implemented in the XML exercises worksheet also.

2.11. Test the service just created using the **WCF Test Client tool**. Note: Only the methods marked as [OperationContract] will be exposed in the web service endpoint.

SECTION 2.1 - CLIENT APPLICATION
HOW TO CONSUME YOUR LOCAL WEB SERVICE

First, a **service reference** needs to be added to a project. It enables a project to access one or more Windows Communication Foundation (WCF) services. Use the **Add Service Reference** dialog box to search for WCF services in the current solution, locally, on a local area network.

We will start by testing the local service (local URL of the web service), before publishing it in an external service provider.

3. In a **new Visual Studio instance**, create a new **Windows Forms Application** project and name it '**ClientBookstore'**.

3.1. Add a 'Service Reference' to the created project. A wizard should appear. Copy the URL of your web service to the address fill in the wizard. Select 'ServiceBookstore' service and name it '**ServiceRefBookstore'**. Tip: what you name the reference becomes its namespace within the client application.

Tip: When creating the client project inside the same solution as the web service you may add the service reference by click 'Discover' to obtain a list of services available in the solution.

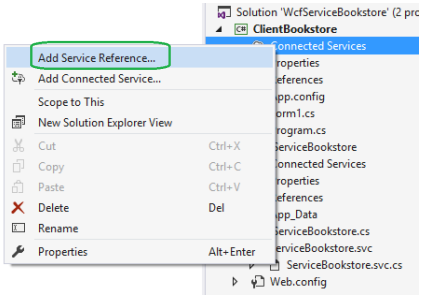


Figure 8 - Windows Forms Application template structure with reference

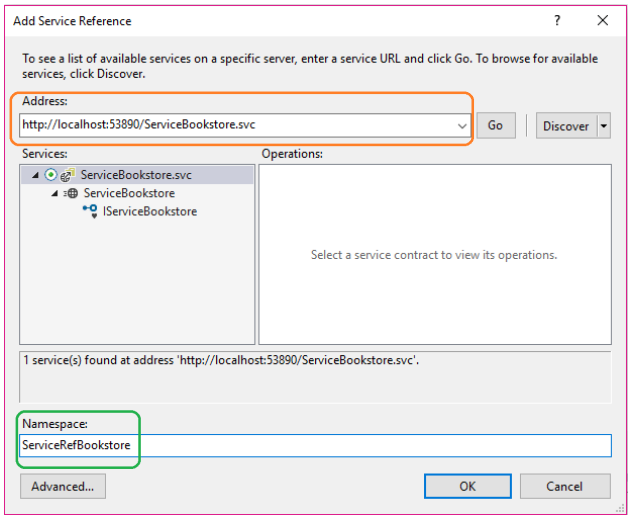
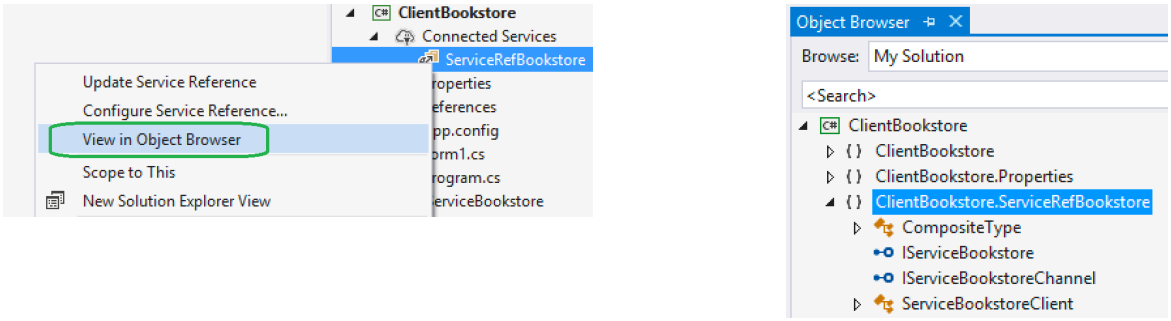


Figure 9 – Add the address of the web service (URL).

If the web service is fully operational you may explore the content of the service (in the left panel) and define the *Namespace* for the web service inside your project.

- After adding the **service reference** in your project look at is the configuration file (App.config). This file contains the **endpoints** of the Web service you have added to your project. Using the Visual Studio IDE, the web service configuration metadata is transparently added to the application that will use it. Over time the metadata for the WCF service may change, requiring that the service reference be updated.



(A)

(B)

Figure 10 – Explorer the object inside the reference (Web Service).

3.2. Implement in the Form a behavior (functions) that allow the user to access and invoke the published operations available in the web service.

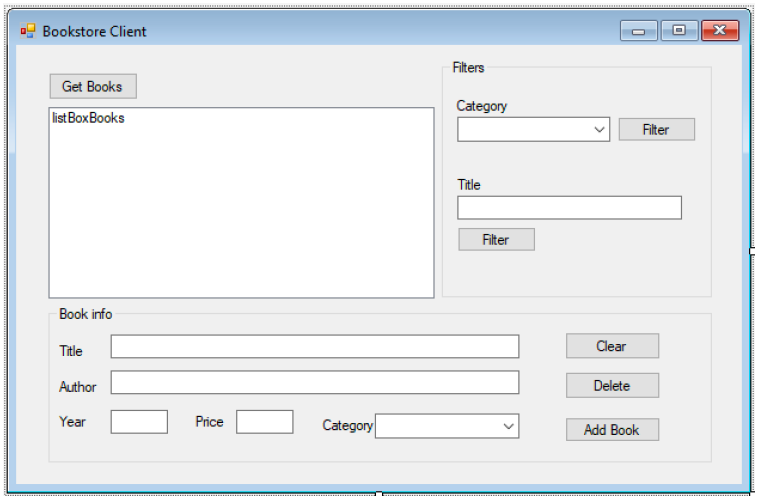


Figure 11 – Bookstore client application interface to test the web service

3.2.1. Add the namespace to Program.cs. Notice how the namespace depends on how you named it in the previous step.

```
using ClientBookstore.ServiceRefBookstore;
```

3.2.2. Invoke the method *GetBooks* with the expected data type: List<Book>. Show the titles in a ListBox. Analyze the results.

```
using (ServiceBookstoreClient service = new ServiceBookstoreClient())
{
    books = service.GetBooks().ToList();

    listBoxBooks.DataSource = books;
    listBoxBooks.DisplayMember = "Title";
    listBoxBooks.ValueMember = "Title";
    service.Close();
}
```

3.2.3. Invoke all the other available methods as suggested in **FIGURE 11**. Note: In the combobox's use the enumeration BookCategory available in the service namespace.

3.3. Change de WcfServiceBookstore implementation. Add a new operation that returns all the books that have a specific word in the book title.

```
[OperationContract]
1 reference
Book[] GetBooksByTitle(string title);
```

3.4. Update de service reference in the client application and invoke this new method.

SECTION 2.2 - PUBLISHING THE WEB SERVICE IN A CLOUD REPOSITORY – CLOUD SERVICE: APPHARBOR AVAILABLE AT HTTPS://APPHARBOR.COM/

The aim of this task is to publish the web service, that you have developed, into a cloud service. This way any client application may consume it.

The service will be published in **AppHarbor**, which is a fully hosted .NET Platform as a Server.

Mandatory: First you must create an account to be able to use the AppHarbor service. Therefore, go to the https://appharbor.com/ and create an account.

You also need to have installed in Visual Studio 2019 the **GitHub Extension for Visual Studio**. To see if you have it already installed go to *Tools -> Extensions and Updates* menu in VS and search for It in the installed extensions.

4. Publish the Web Service created in exercise #3 in the cloud Service. Follow the next Guidelines.

4.1. Access https://appharbor.com and sign in and create a new application (project repository). Name the application *WcfServiceExample*.

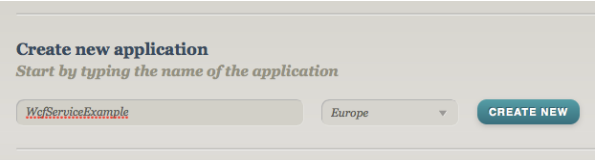


Figure 12 – Option in AppHarbor to create a new application

4.2. Select the Repository URL. The system should display a message "Copied repository URL to clipboard". Paste the content anywhere (notepad file, browser, etc.). The URL look something like this https://username@appharbour.com/mygitremoteapplication.git. (Example: https://USERNAME@appharbor.com/wcfserviceexample.git)

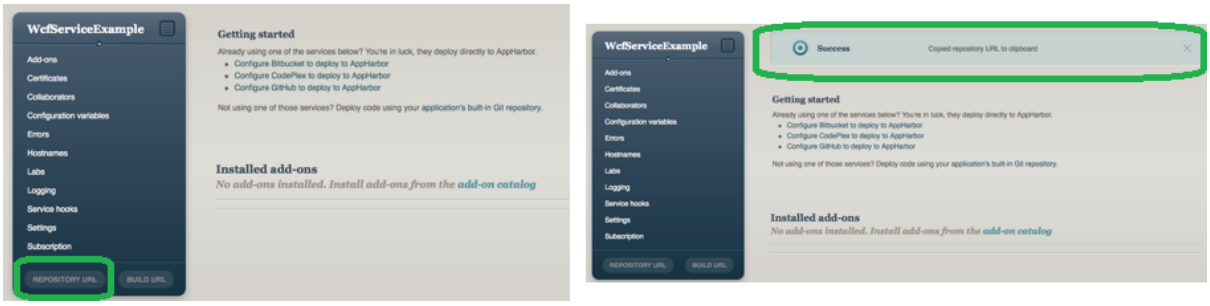


Figure 13 – AppHarbor interface to access Repository URL

4.3. In VS2017, inside your web service project (*WcfServiceExample*) commit the source code of the Web Service to the repository.

4.3.1. In the solution Explorer, choose your **solution** *WcfServiceSolution*. You may now add the solution to the source control. Select "Add Solution to Source Control…". Select a Git source control (see **FIGURE 14**).

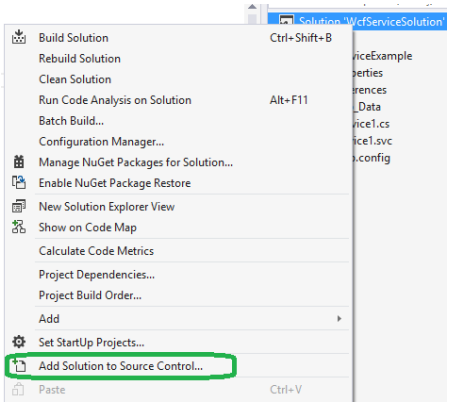


Figure 14 – Add a solution to a source control system.

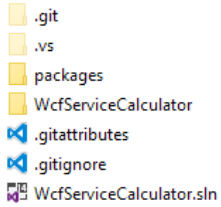
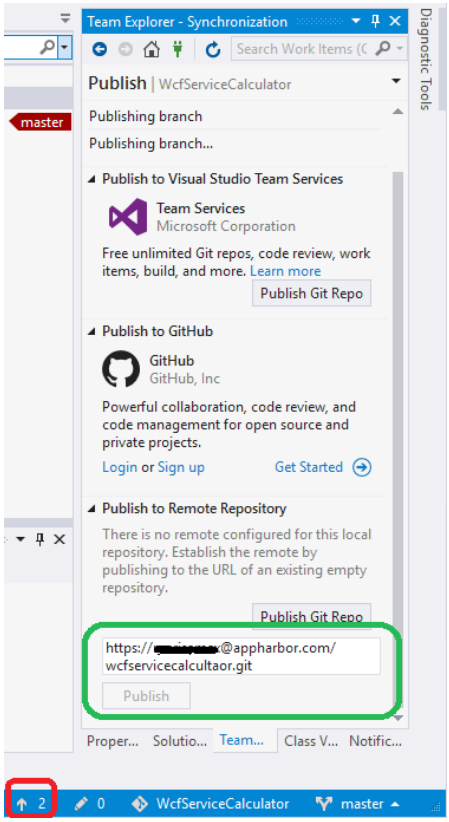


Figure 15 – Project Git source control configuration files (it is showing the service calculator source code, from exercise 10), but the folder structure is identical to all projects.

4.3.2. After selecting Git source control, you may check your solution explorer and you are able to perform VCS operations. Right click your solution and select **Open Folder in File Explorer**. See the files that were created (see and **FIGURE 15**).

4.3.3. In your solution, you must perform the **Commit** operation. If this is your first push, there is no Remote Repository yet. Therefore, you must input the URL of an Empty Git Repository. Use your AppHarbor URL repository (created in exercise 6.2). Therefore, you need to configure the repository URL by selecting Publish on status bar in the lower right-hand corner of Visual Studio. Also, at **first time**, you need to configure your username and email address before commit changes (see **FIGURE 16**). **Team Explorer** displays a resume of changes that will occur.



2 shows the number of outgoing commits. Selecting this will open the Synchronization view in Team Explorer.

3 shows the number of uncommitted file changes. Selecting this will open the Changes view in Team Explorer.

WcfServiceCalculator shows the current Git repository. Selecting this will open the Connect view in Team Explorer.

master shows your current Git branch. Selecting this displays a branch picker to quickly switch between Git branches or create new branches.

Figure 16 – Commit operation in the server explorer and Configure the username and e-mail address for your source control account (AppHarbor)

- Git requires a message to execute a **commit** action (**Commit All** option). Input the message and select commit (see Figure 17). After this step, your changes were locally committed. Remember, Git is a distributed VCS. Therefore, a push action is also performed to synchronize the repositories.

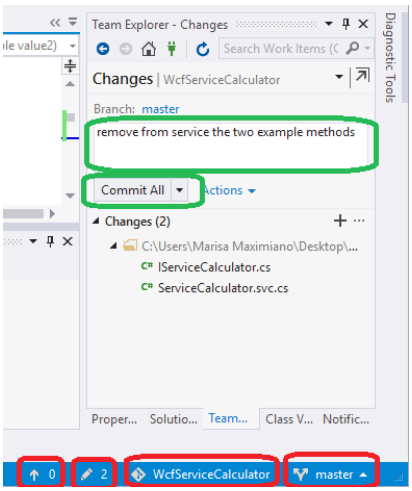


Figure 17 – Commit information required to execute the commit operation

- If the repository is already configured, select **Unsynced Commits**. Now you must select **Push** to inform the other clones that your changes occurred. However, it is recommended to check if the other clones have changes before pushing yours (if you have other collaborators in your repository).

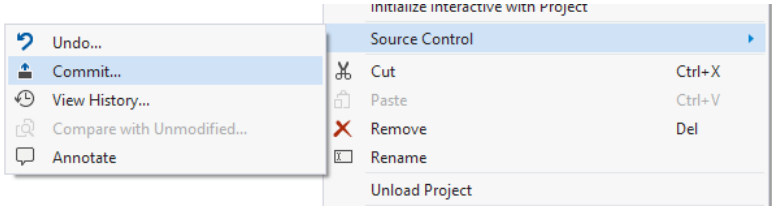


Figure 18 – Commit option from Solution Explorer

4.4. After committing the changes to AppHarbor you need to wait a few seconds before deploying. It is automatic. After that, you can see the active option and then select Go to Your application to see the Service URL in the browser.

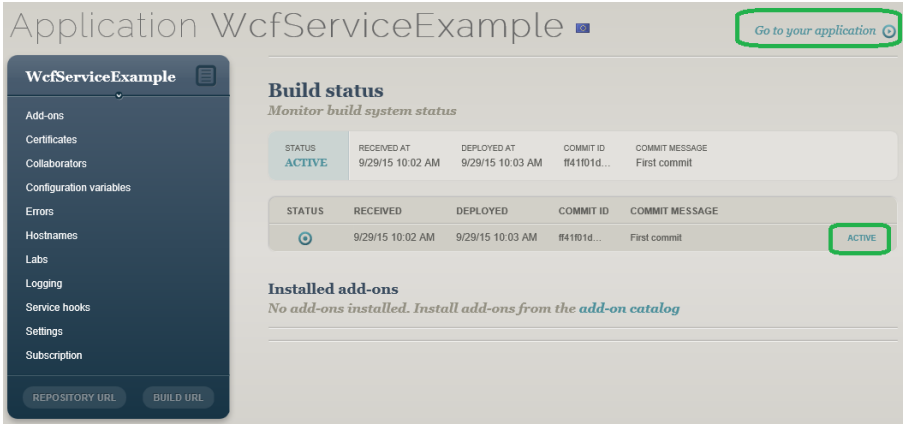


Figure 19 – AppHarbor interface after publishing the service. Deploy is automatic

Figure 20 - WCF Service public address

4.5. You may open the URL in the browser to test if the service is responding. See **FIGURE 19** and **FIGURE 20** that show the published web service.

5. Test the web service hosted in the cloud server using a client application.

5.1. Create a new Windows Forms Application named **ClientCloud**.

5.2. Add the service reference of the web service URL.

5.3. Invoke the methods available in the service.

6. In the project created in exercise 4 (**ClientBookstore**), Implement an option that allows the user to select which is the endpoint used by the client application (the one available in IIS express or to the one in the cloud).

[ADDITIONAL EXERCISES]

7. Publish the Web Service (using your local IIS express).

8. Test the web service hosted in the IIS using a new client application. Therefore, create a new Windows Forms Application named **ClientLocalhostIIS**. Add the service reference of the web service URL in your *localhost* to this application.

8.1. Invoke the methods available in the service.

9. Create a Web Service named 'WcfServiceCalculator'.

9.1. Add the operations Sum, Subtract, Multiply and Divide. Tip: pay attention to `DivideByZero` Exceptions. Note: It is recommended that you throw the exception as a **FaultException.**

9.2. Test the service using the WCF Test Client tool.

9.3. Publish this web service in AppHarbor (cloud service).

9.4. Create a new project to develop a client application (**Windows Form Application** is recommended) named **ClientCalculator** to access and invoke the published operations.

9.4.1. Don't forget to add a service reference of URL were the web service is available.

9.4.2. Invoke the methods available in the service.