Microsoft

Getting Started with Apps for SharePoint

Office 365

Hands-on lab

In this lab you will get hands-on experience working with the new SharePoint App model. Through the exercises in this lab you will learn how to create and test a SharePoint-hosted app as well as a provider-hosted app.

This document is provided for informational purposes only and Microsoft makes no warranties, either express or implied, in this document. Information in this document, including URL and other Internet Web site references, is subject to change without notice. The entire risk of the use or the results from the use of this document remains with the user. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Copyright 2014 © Microsoft Corporation. All rights reserved.

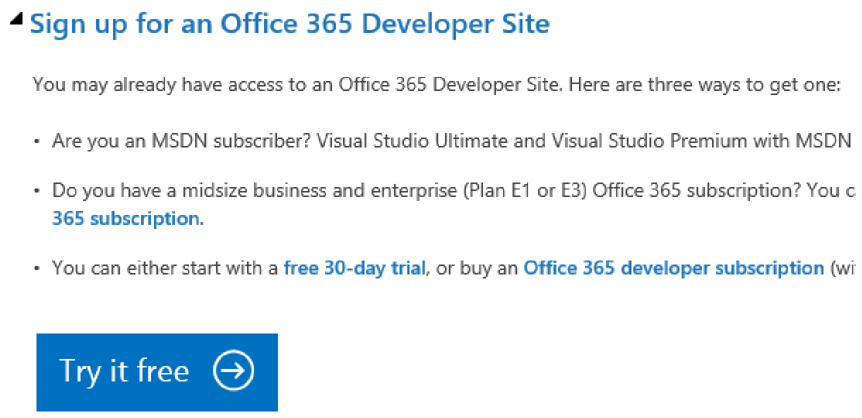
Microsoft, Internet Explorer, Microsoft Azure, Microsoft Office, Office 365, Visual Studio, and Windows are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Exercise 1: Creating and Debugging a SharePoint-hosted App

You must have an Office 365 tenant to complete this lab. To sign up for an Office 365 developer subscription:

1. Navigate to http://msdn.microsoft.com/en-us/library/office/fp179924(v=office.15).aspx.
2. Under the heading **Sign up for an Office 365 Developer Site** click **Try It Free**.



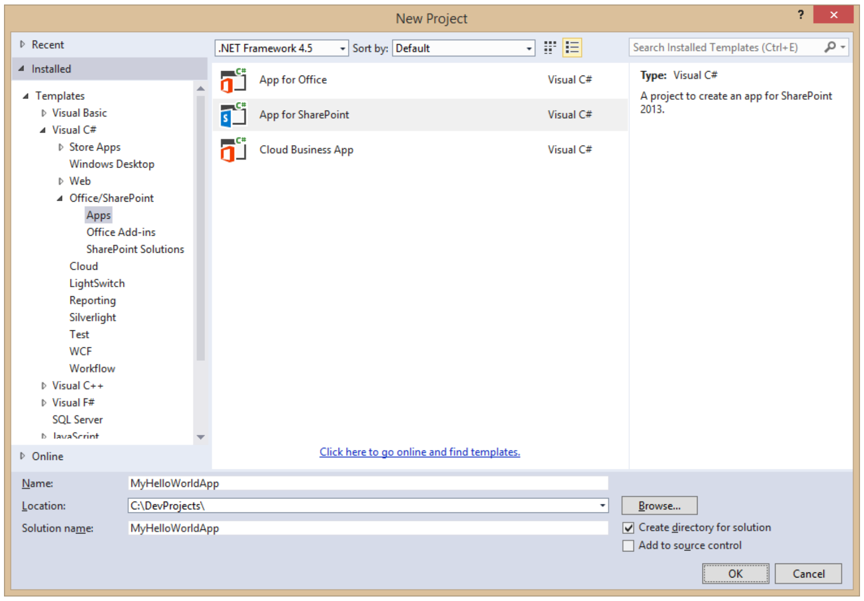
Sign up for an Office 365 Developer Site

You may already have access to an Office 365 Developer Site. Here are three ways to get one:

- Are you an MSDN subscriber? Visual Studio Ultimate and Visual Studio Premium with MSDN
- Do you have a midsize business and enterprise (Plan E1 or E3) Office 365 subscription? You c 365 subscription.
- You can either start with a free 30-day trial, or buy an Office 365 developer subscription (wi

3. Fill out the form to obtain your trial O365 subscription.
4. When completed, you will have a developer site in the [subscription].sharepoint.com domain located at the root of your subscription (e.g. https://mysubscription.sharepoint.com)
5. You must have a Microsoft account to complete this lab. If you do not have one, navigate to https://signup.live.com/signup.aspx?lic=1 and create one.

1. Using the browser, navigate to your Office 365 developer site and log on using your credentials.
2. On your developer workstation, launch Visual Studio as administrator.
3. Create a new project in Visual Studio 2013 by selecting the menu command **File > New > Project**.
4. In the **New Project** dialog, find the **App for SharePoint 2013** project template under the **Templates > Visual C# > Office / SharePoint > Apps** section. Enter a name of **My First SharePoint Hosted App**, a location of **C:\DevProjects** and a Solution name of **MyFirstSharePointHostedApp** and then click **OK** button.



Type: Visual C#

A project to create an app for SharePoint 2013.

Name: MyHelloWorldApp

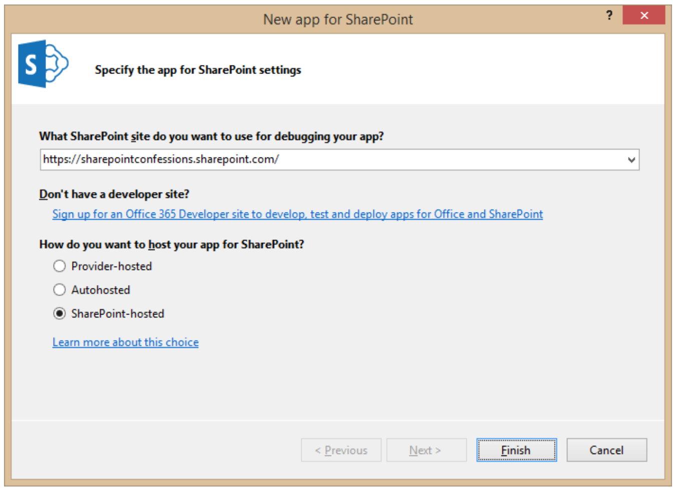
Location: C:\DevProjects\

Solution name: MyHelloWorldApp

[✓] Create directory for solution

[ ] Add to source control

5. In the **New app for SharePoint** wizard, enter the URL for your Office 365 Developer site and select **SharePoint-hosted** for the app hosting model. When done, complete the wizard by clicking the **Finish** button.


New app for SharePoint
Sign up for an Office 365 Developer site to develop, test and deploy apps for Office and SharePoint
Learn more about this choice
< Previous
Next >
Finish
Cancel

Specify the app for SharePoint settings

What SharePoint site do you want to use for debugging your app?

https://sharepointconfessions.sharepoint.com/

Don't have a developer site?

How do you want to host your app for SharePoint?

( ) Provider-hosted

( ) Autohosted

(●) SharePoint-hosted

6. Examine the default project setup for a SharePoint-Hosted app. As you can see, it is like a traditional SharePoint solution-based project because you have a Features and Packages node.
7. Note that there are project folders named Content, Images & Pages are actually SharePoint Project Items (SPI) that are Modules and will provision their contents to the respective folders in the app web that will be generated upon installing the app.
   - **Content/App.css**: main Cascading Style Sheet used for the app.
   - **Images/AppIcon.png**: default image used for the app.
   - **Pages/Default.aspx**: default start page for the app.
   - **Scripts/App.js**: main JavaScript file which is referenced by Default.aspx.

- **AppManifest.xml** app manifest containing app metadata such as its Name, Product ID, App Version Number and minimum version for the SharePoint host environment.

8. Examine the app's start page by right-clicking **Pages/Default.aspx** file and selecting **Open**.
    - Look at the links to other JavaScript libraries inside the PlaceHolderAdditionalPageHead placeholder.
    - There are references to the jQuery library and the App.js file.
    - There is a reference to the App.css file as well.

9. Using the Solution Explorer tool window, right-click the **Scripts/App.js** file and select **Open**.
    - This file has four functions and a few variables.
    - The function $(document).ready(function()){ … } gets a reference to the client object model (CSOM) ClientContext object and then gets a reference to the current site.
    - The getUserName() function is one that will usually be deleted from the project when you get more experience with SharePoint-Hosted apps. It uses the CSOM to get the name of the current user logged in.
    - The last two functions are used as the success and failed callback when the CSOM request completes.

10. Now it is time to update the app homepage. Using the Solution Explorer tool window, right-click the **Pages/Default.aspx** file and select Open. After the existing **div**, add the following markup

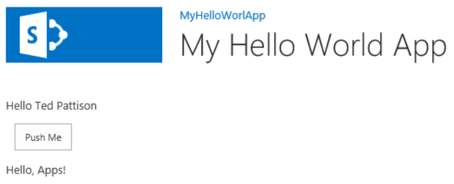
```
↪  <input type="button" value="Push Me" onclick="hello();" />
↪  <div id="displayDiv"></div>
```

11. Inside **default.aspx**, locate the **PlaceHolderPageTitleInTitleArea** placeholder control and replace the content inside with the title **My Hello World App**.
12. In this step you will update the app script file. Using the Solution Explorer tool window, right-click the **Scripts/App.js** file and select **Open**. Add the following function to the bottom of the file that will be called when you click the button.

```
↪  function hello() {
↪    $get("displayDiv").innerHTML = "<p>Hello, Apps!</p>";
↪  }
```

13. Save all changes: File > Save All.
14. Build and Test the Project by pressing **[F5]** or **Debug > Start Debugging**.
15. The installation process for an app will take a moment to complete. If you watch the lower-left corner of Visual Studio, it will tell you what it is currently doing. If you want more information, click the Output tab at the bottom of Visual Studio to see a log of what is going on. If the Output tab isn't present, select the window from the menu in Visual Studio 2013 using the menu command **View > Output**.

16. Once the app has been installed, Internet Explorer will launch and navigate to the app's start page **default.aspx** page.
17. When the page loads, click the **Push me** button to see your text get written to the page:



Hello Ted Pattison

Hello, Apps!

18. Once you have tested the app, close the browser to stop the debugging session and return to Visual Studio.
19. In Visual Studio, save all changes using **File > Save All**.

Exercise 2: Using jQuery in a SharePoint-hosted App

In this lab, you will continue working with the SharePoint-hosted app project you created in the previous lab exercise. However, you will rewrite the JavaScript code to use the jQuery library to initialize the app and create an event handler using best practice techniques.

1. Open **default.aspx** and ensure that the HTML code inside the **PlaceHolderMain** content control looks exactly like the following code listing.

```
<asp:content contentplaceholderid="PlaceHolderMain" runat="server">

  <div>
    <p id="message">
    <!-- The following content will be replaced with the user name when you run the app - see App.js -->
    initializing...
    </p>
  </div>

  <input type="button" value="Push Me" onclick="hello();" />
  <div id="displayDiv"></div>

</asp:content>
```

2. Remove the onclick attribute from the input element and add an **id** of **cmdPushMe** so the element definition looks like this.

```
<input id="cmdPushMe" type="button" value="Push Me" />
```

3. Save your changes and close **default.aspx**.
4. Right-click on **app.js** and select **Open** to open this JavaScript file in an editor window.
5. Delete all the code inside **app.js** except for the 'use strict'; statement at the top.
6. Inside **app.js**, add two new functions into **onPageLoad** and **onButtonClicked**.

```
'use strict';

function onPageLoad() {
}

function onButtonClicked() {
}
```

7. At the top of **App.js** right after the use strict statement, add a jQuery document ready event handler to execute the **onPageLoad** function once the page loads and the JavaScript DOM is available for access within the browser.

```
↪  'use strict';
↪
↪  $(document).ready(onPageLoad);
↪
↪  function onPageLoad() {
↪  }
↪
↪  function onButtonClicked() {
↪  }
```

8. Implement **onPageLoad** with the following code to display a text message on the page when the document ready event handle executes and to register the **onButtonClick** function as an event handler for the input control with the **id** of **cmdPushMe**.

```
↪  function onPageLoad() {
↪      $("#message").text("Hello from the document ready event handler");
↪      $("#cmdPushMe").click(onButtonClicked);
↪  }
```

9. Implement the **onButtonClicked** function to write the text message **"Hello Apps"** into the div element with the **id** of **displayDiv** and to use the jQuery `css` method to style thediv element with a margin div of 16px, a font color of green and a font-size of 32px.

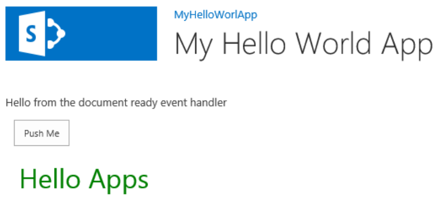
```
↪  function onButtonClicked () {
↪      $("#displayDiv")
↪          .text("Hello Apps")
↪          .css({ "margin": "16px", "color": "green", "font-size": "32px" });
↪  }
```

10. Once the code inside your **app.js** file looks like the following code listing, you are ready to test your work.

```
↪  'use strict';
↪
↪  $(document).ready(onPageLoad);
↪
↪  function onPageLoad() {
↪      $("#message").text("Hello from the document ready event handler");
↪      $("#cmdPushMe").click(onButtonClicked);
↪  }
↪  function onButtonClicked() {
↪      $("#displayDiv")
↪          .text("Hello Apps")
↪          .css( "margin": "16px", "color": "green", "font-size": "32px" });
↪  }
```

11. Save all changes by executing the **File > Save All** menu command.

12. Build and Test the Project by pressing **[F5]** or **Debug** > **Start Debugging**.
13. Once the app has been installed, Internet Explorer will launch and navigate to the app's start page **default.aspx** page.
14. When the page loads, you should see the message **"Hello from the document ready event handler"** on the page.
15. Click the **Push me** button to see your text get written to the page with your custom font styles.



Hello from the document ready event handler

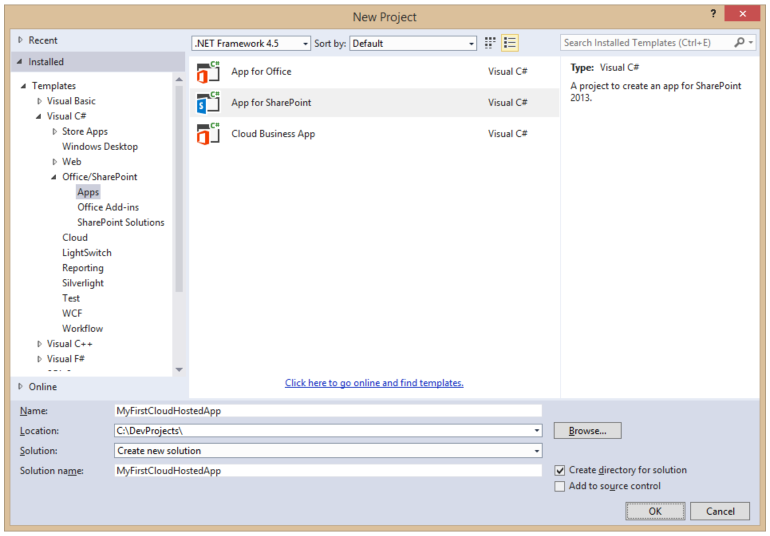
Hello Apps

16. Once you have tested the app, close the browser to stop the debugging session and return to Visual Studio.
17. In Visual Studio, save all changes using **File** > **Save All**.
18. Close the **MyHelloWorldApp** project.

Exercise 3: Creating and Debugging a Provider-Hosted App

In this exercise you will create and test a simple Provider-Hosted App. This will give you opportunity to observe the basic differences between developing SharePoint-hosted apps and cloud-hosted apps using Visual Studio 2013. Note that this lab will not involve security topics such as app authentication. Instead, you will configure the app to use Internal security so that you can get the app up and running without worrying about how to configure app authenhtication.

1. Launch Visual Studio 2013 as administrator if it is not already running.
2. In Visual Studio select **File** > **New** > **Project**.
3. In the New Project dialog select the **App for SharePoint 2013** template under the **Templates** > **Visual C#** > **Office / SharePoint** > **Apps** section.
4. Enter a **Name** of **MyFirstCloudHostedApp** and a **Location** of **C:\DevProjects\** and then click **OK** when you are done.



Type: Visual C#

A project to create an app for SharePoint 2013.

Name: MyFirstCloudHostedApp

Location: C:\DevProjects\

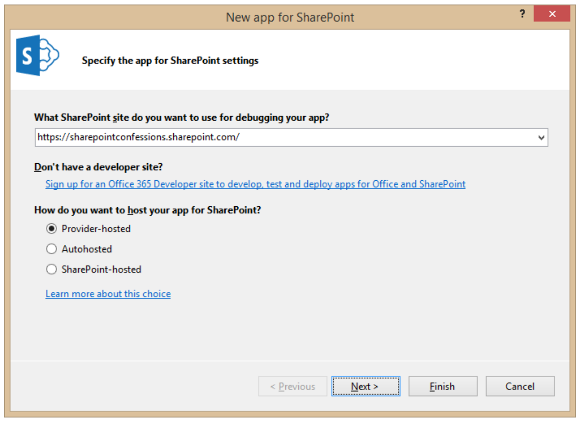
Solution: Create new solution

Solution name: MyFirstCloudHostedApp

☑ Create directory for solution

☐ Add to source control

5. Next, you will see the **New app for SharePoint wizard** which begins by prompting you with the **Specify the App for SharePoint Settings** page. Enter the URL to your Office 365 developer site, configure the app's hosting model to be **Provider-hosted** and click **Next**.


New app for SharePoint

Specify the app for SharePoint settings

What SharePoint site do you want to use for debugging your app?

https://sharepointconfessions.sharepoint.com/

Don't have a developer site?

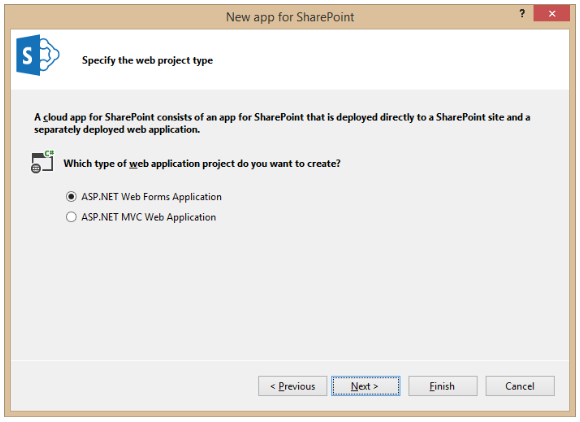
Sign up for an Office 365 Developer site to develop, test and deploy apps for Office and SharePoint

How do you want to host your app for SharePoint?

- [✓ Provider-hosted]
- Autohosted
- SharePoint-hosted

Learn more about this choice

6. On the **Specify the web project type** page, select the **ASP.NET Web Forms Application** setting and click **Next**.


New app for SharePoint

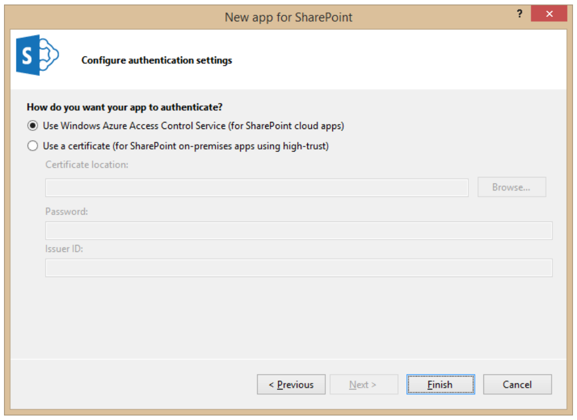
Specify the web project type

A cloud app for SharePoint consists of an app for SharePoint that is deployed directly to a SharePoint site and a separately deployed web application.

Which type of web application project do you want to create?

- [✓ ASP.NET Web Forms Application]
- ASP.NET MVC Web Application

7. On the **Configure authentication settings** page, accept the default settings and click **Finish**.


New app for SharePoint

Configure authentication settings

How do you want your app to authenticate?

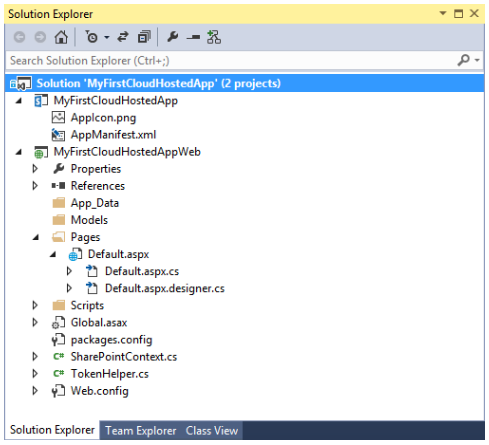
- [✓ Use Windows Azure Access Control Service (for SharePoint cloud apps)]
- Use a certificate (for SharePoint on-premises apps using high-trust)

Certificate location:

Password:

Issuer ID:

8. Examine the structure of the Visual Studio solution that has been created. As you can see, the Visual Studio solution created for a Provider-Hosted app has two projects and is very different from the Visual Studio solution for a SharePoint-Hosted app which has only one project.



9. Observe that top project named **MyFirstCloudHostedApp** contains only two files: **AppManifest.xml** and **AppIcon.png**. This effectively means the app will not install any resources into the SharePoint host such as pages. This project only contains app metadata and an image file that get added to the SharePoint host when the app gets installed.
10. Take a look at the project below named **MyFirstCloudHostedAppWeb** which will provide the implementation of the app's remote web. This project is a standard ASP.NET Web application but it contains a little extra stuff in it:
    - **TokenHelper.cs**: This is a code file provided by Microsoft to make it easier to obtain the user identity, the OAuth token or the token provided by highly trusted apps. You will ignore this for now.
    - **Default.aspx.cs**: (Inside the Pages folder expand out the Default.aspx file to see this) the code behind file for the page contains logic to call back into SharePoint to obtain the title of the host web. The code is written to assume this app will use OAuth authentication.
    - **SharePointContext.cs**: This is a code file provided by Microsoft to encapsulate all the information from SharePoint. You will ignore this for now.
    - **Scripts**: A common folder to place JavaScript files.
11. By default a Provider-Hosted app is expecting to use external authentication with either OAuth or S2S which are topics covered in later modules. In the following steps you will disable the configuration for external authentication to eliminate security requirements which would complicate building and testing our first Provider-Hosted app.
12. In the Solution Explorer within the **MyFirstCloudHostedApp** project, right-click **AppManifest.xml** and select **View Code**.
13. Inside **AppManifest.xml**, locate the `<AppPrincipal>` node.
14. Replace the contents with `<Internal/>` so it looks like the following markup:

```
↪   <AppPrincipal>
↪    <Internal/>
↪  </AppPrincipal>
```

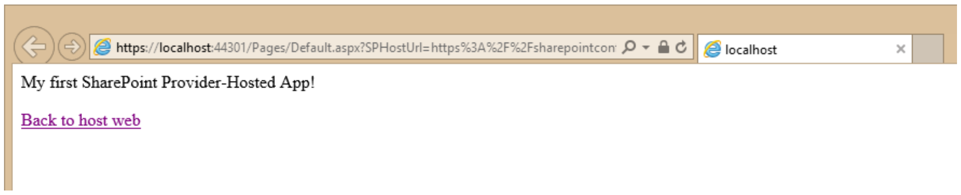
15. Now it's time to write a bit of server-side C# code which will run in the remote web of the app which is something that isn't possible to do in a SharePoint-Hosted app. In the Solution Explorer tool window within the **MyFirstCloudHostedAppWeb** project, right-click **Default.aspx** and select **Open**.
16. Replace the existing `<body>` element on the page with an ASP.NET literal control with an **id** of **Message** and a hyperlink control with an **id** of **HostWebLink** so the body of the page looks like the following markup:

```
↪  <body>
↪      <form id="form1" runat="server">
↪      <asp:Literal ID="Message" runat="server" />
↪      <p><asp:HyperLink ID="HostWebLink" runat="server" /></p>
↪      </form>
↪  </body>
```

17. In the Solution Explorer within the **MyFirstCloudHostedAppWeb** project, right-click the code beind files named **Default.aspx.cs** file and select **Open** to open the file in a code editor window.
18. Delete the existing `Page_PreInit` method and all the code inside.
19. Replace the contents of the `Page_Load` method with the following code:

```
↪  this.Message.Text = "My first SharePoint Provider-Hosted App!";
↪
↪  var hostWeb = Page.Request["SPHostUrl"];
↪  this.HostWebLink.NavigateUrl = hostWeb;
↪  this.HostWebLink.Text = "Back to host web";
```

20. Save all changes by using the **File > Save All** menu command.
21. Build and Test the Project by pressing **[F5]** or **Debug > Start Debugging**.
22. Visual Studio 2013 may prompt you with a Security Alert to trust a self-signed certificate. You are not using a certificate in this solution, so just click Yes (and again to get past the extra confirmation prompt) to continue.
23. Once the solution has been deployed, Internet Explorer will launch and navigate to the start page of the app in the remote web.
24. Notice when the page loads it is just a plain white page with the text you added and a link back to the hosting site.


https://localhost:44301/Pages/Default.aspx?SPHostUrl=https%3A%2F%2Fsharepointcon
localhost

My first SharePoint Provider-Hosted App!

Back to host web

25. Test the **Back to host web** link to make sure it correctly redirects you back to the host web which should be your Office 365 developer site.
26. Close the browser to stop the debugger and go back to Visual Studio.

In this exercise you created a simple SharePoint Provider-Hosted app. As in the last exercise, you didn’t do much in this exercise beyond creating and testing the simplest cloud-hosted app possible. In later labs you will build on this foundation to add more capabilities to SharePoint apps.