# SharePoint 2013 Demo – SharePoint Cloud Hosted App

## Estimated time to complete this demo

10 minutes

## Objectives

Demonstration of following capabilities in SharePoint 2013

* Create & Deploy a SharePoint Cloud App manifested as a local ASP.NET Web Application.

## Computers in this demo setup

This demo uses virtual machines as described in the following table. Before you begin the demo, you must start the virtual machines and then log on to the computers.

|  |  |
| --- | --- |
| **Virtual Machine** | **Role** |
| {Supplied by Instructor} | Domain Controller |
| {Supplied by Instructor} | Actual SharePoint environment with Office client and other required software. |

All user accounts in this lab use the password {Supplied by Instructor}.

## Getting started with demo

In the files provided with the hands on lab, run the batch file called **SetupModule.bat** by double clicking it. This file will execute a PowerShell script that will create a new site collection at [http://intranet.contoso.com/sites/SpAppsCloudHosted](http://wave15-sp/sites/IntroSpApps).

# Exercise 1: Register High-Trust App using the Server-To-Server Protocol

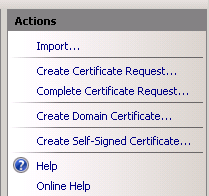
In this exercise you will create the certificate and register the app with SharePoint so it is fully trusted.

## Task 1 – Create a Self-Signed Certificate

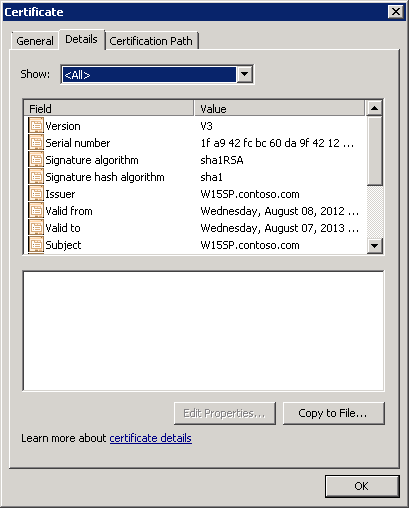
In this task you will create a self-signed certificate using IIS and export it to the desktop for use in creating a high-trust app using the server-to-server protocol.

* Begin this task logged on to **SP** as **CORP\Administrator**.

1. Open Internet Information Services (IIS) Manager (**Start** ⮚ **Administrative** **Tools** ⮚ **Internet Information Services (IIS) Manager**).
2. Select the server in the **Connections** Tool Pane.
3. Double-click the **Server Certificates** icon in the **IIS** section of the center pane.
4. In the **Actions** pane on the right-hand side, click **Create Self-Signed Certificate**.



1. When prompted for a name, enter **HighTrustAppCert** and click **OK**.
2. Select the certificate you just created and in the **Actions** pane, click **Export**.
3. In the **Export Certificate** dialog, export the certificate to the desktop as **HighTrustAppCert.pfx** and use the password **Password1**.
4. Next, double-click the certificate you created in IIS to open the detail view of the certificate.
5. Click the **Details** tab and then the **Copy to File…** button.



1. In the Certificate Export Wizard, choose the following options to export the \*.cer file to the desktop:
   * **Export Private Key:** No, do not export the private key
   * **Export File Format:** DER encoded binary X.509 (.CER)
   * **File Name:** HighTrustAppCert.cer (saved to the desktop)

## Task 2 – Register App with SharePoint Site

In this task you will use PowerShell to create a client ID and register the app with the SharePoint Developer Site created by the script in the pre-steps for this demo.

* Begin this task logged on to **SP** as **CORP\Administrator**.

1. Open a SharePoint 2013 Management Shell (**Start** ⮚ **All Programs** ⮚ **Microsoft SharePoint 2013 Products** ⮚ **SharePoint 2013 Management Shell**).
2. Create a few variables that will be used throughout this demo in registering the app:

$devSite = "http://intranet.contoso.com/sites/SpAppsCloudHosted"

$appName = "Provider Hosted App"

$issuerId = [System.Guid]::NewGuid().ToString()

$certPath = "c:\Users\administrator.corp\Desktop\HighTrustAppCert.cer"

1. Get the target site where the app will be installed and its authentication realm:

$devSiteWeb = Get-SPWeb $devSite

$devSiteRealm = Get-SPAuthenticationRealm –ServiceContext $devSiteWeb.Site

1. Get a reference to the certificate:

$appCert = Get-PfxCertificate $certPath

1. Create a full app identifier:

$fullAppId = $issuerId + '@' + $devSiteRealm

1. Create a new trusted security token issuer:

$secureTokenIssuer = New-SPTrustedSecurityTokenIssuer –Name $appName –Certificate $appCert –RegisteredIssuerName $fullAppId

1. Create a new app principal:

$appPrincipal = Register-SPAppPrincipal –NameIdentifier $fullAppId –Site $devSiteWeb –DisplayName $appName

1. Finally, configure the SharePoint STS to allow OAuth over HTTP

$stsConfig = Get-SPSecurityTokenServiceConfig

$stsConfig.AllowOAuthOverHttp = $true;

$stsConfig.Update()

1. Now that the app has been registered, get a copy of the issuer ID that you created. This will be used in the next part of the demo.

$issuerId

# Exercise 2: Create a SharePoint Provider-Hosted App as an ASP.NET IIS Web Application

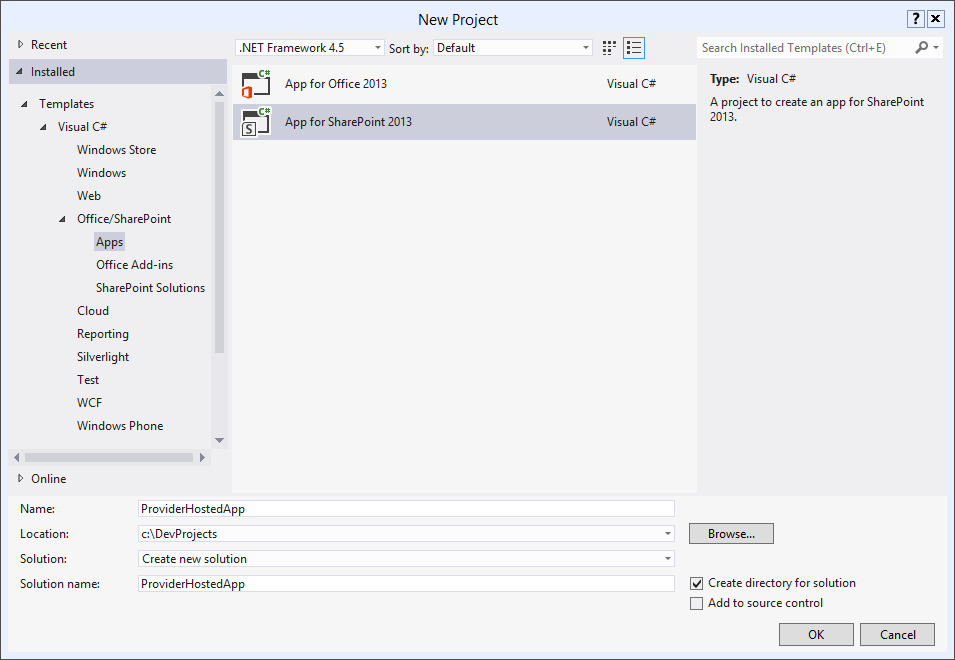
In this exercise you will create a new SharePoint App that is manifested as an IIS Web Application

## Task 1 – Create a SharePoint App

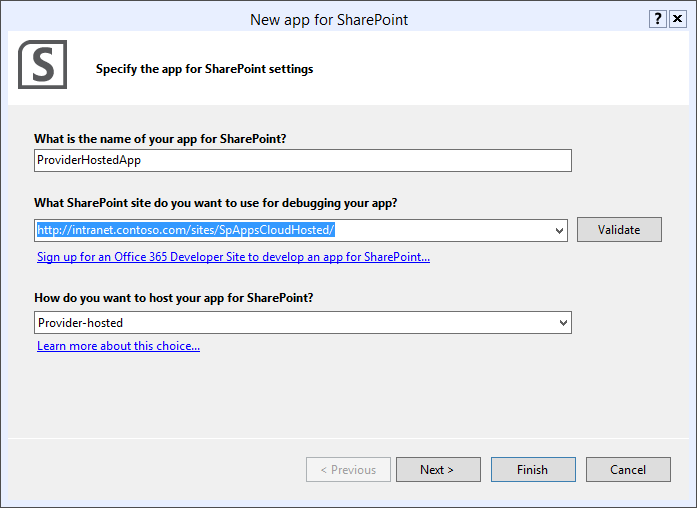
In this task you will create a SharePoint app.

* Begin this task logged on to **SP** as **CORP\Administrator**.

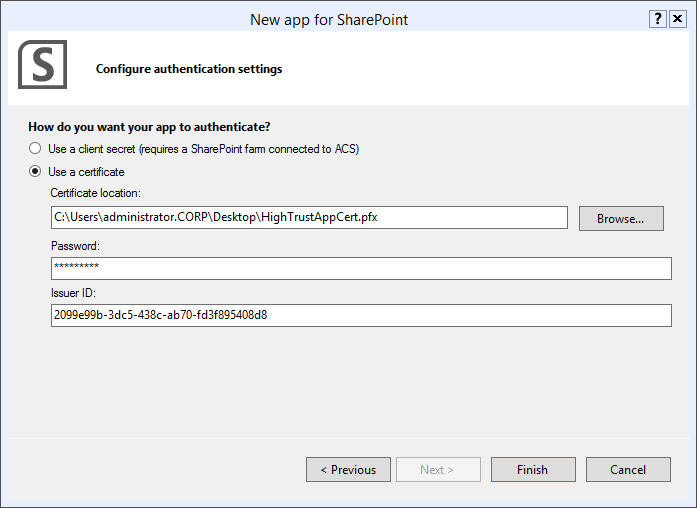
1. Open Visual Studio 2012 (**Start ⮚ Visual Studio 2012 ⮚ Visual Studio 2012**).
2. , select **File ⮚ New ⮚ Project**.
3. In the New Project dialog, select **Visual C# ⮚ Office/SharePoint ⮚ Apps** and select the template **App for SharePoint 2013**. Set the name of the project to **ProviderHostedApp**, finally clicking **OK**:



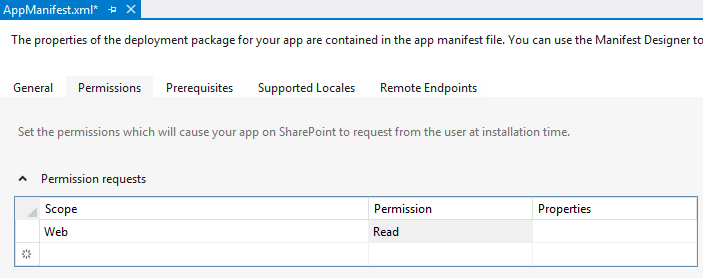
1. In the **New App for SharePoint** dialog, give the app the name of **Provider Hosted App**, a debugging site of [**http://intranet.contoso.com/sites/SpAppsCloudHosted**](http://wave15-sp/sites/SpAppsCloudHosted) and select **Provider-Hosted** for the type of app host. Click **Next**.



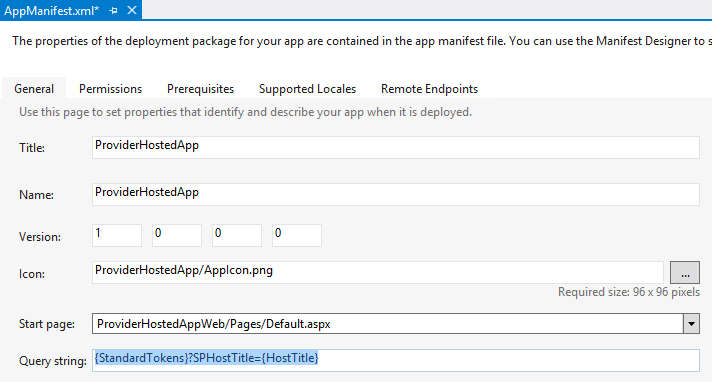
1. In the **Configure authentication settings** screen, choose the option to **Use a certificate.** Use the Browse button to point to the HighTrustAppCert.pfx file created in the previous section. For the password, use the password **Password1.** For the Issuer ID, use the GUID that you created in the previous step (the **$issuerId** value from PowerShell).



1. Click **Finish** to create the project. Visual Studio 2012 will create two projects: a SharePoint app project (**ProviderHostedApp**) and an ASP.NET Web Application project (**ProviderHostedAppWeb**).
2. After the project is added to the solution, within the **Solution Explorer** tool window, select the **ProviderHostedApp** project and double-click the **AppManifest.xml** file.
3. In the **Permission Requests** section, add a new **Read** permission for the hosting SharePoint site:



1. Next, ensure SharePoint includes the name of the current site to the app by appending **?SPHostTitle={HostTitle}** to the **Query string** property:



## Task 2 – Update the ASP.NET Web Application

In this task you will update the ASP.NET Web Application to contain code that retrieves items from the SharePoint site hosting the app… the Host Web.

* Begin this task logged on to **SP** as **CORP\Administrator**.

1. Ensure you have the **ProviderHostedApp** solution open that you created in the previous step in Visual Studio 2012.
2. Open the **Pages/default.aspx** page from the **ProviderHostedAppWeb**.
3. Add the following to the **<head></head>** part of the page. This will create the chrome control so the page will look & feel like the hosting SharePoint site:

<script type="text/javascript" src="../Scripts/jquery-1.7.1.min.js"></script>

<script type="text/javascript">

var hostWebUrl;

var hostLayoutsUrl;

// Load the SharePoint resources.

$(document).ready(function () {

// Get the URI decoded app web URL.

hostWebUrl = decodeURIComponent(getQueryStringParameter("SPHostUrl"));

hostLayoutsUrl = hostWebUrl + "/\_layouts/15/";

// Load the js file and continue to the success handler.

$.getScript(hostLayoutsUrl + "SP.UI.Controls.js", renderChromeControl)

});

// Function to prepare the options and render the control.

function renderChromeControl() {

var options = {

"appTitle": "Simple Cloud Hosted App"

};

// activate the chrome control

var nav = new SP.UI.Controls.Navigation("SharePointChromeControl", options);

nav.setVisible(true);

}

// Function to retrieve a query string value.

function getQueryStringParameter(paramToRetrieve) {

var params = document.URL.split("?")[1].split("&");

var strParams = "";

for (var i = 0; i < params.length; i = i + 1) {

var singleParam = params[i].split("=");

if (singleParam[0] == paramToRetrieve)

return singleParam[1];

}

}

</script>

1. Next, add the following HTML just after the opening **<body>** tag. This is where the chrome control will be created:

<div id="SharePointChromeControl"></div>

1. Finally, add the following markup within the **<form></form>** tag:

<div style="margin: 20px;">

<div class="contentRow">

<h2 class="ms-accentText">Lists in the App's Host Web:</h2>

<div class="ms-textSmall">Retrieved via server-side .NET CSOM.</div>

<asp:DataList ID="AppWebLists" runat="server" RepeatDirection="Horizontal" RepeatColumns="3" CellPadding="5">

<ItemTemplate>

&raquo;

<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl='<%=Request.QueryString["SPHostUrl"].ToString(); %>/<%# Eval("DefaultViewUrl") %>' Text='<%# Eval("Title") %>' /></asp:HyperLink>

</ItemTemplate>

</asp:DataList>

</div>

</div>

1. Save all changes.
2. Now, open the code-behind for **default.aspx** by right-clicking it in the **Solution Explorer** and selecting **View Code**.
3. Remove everything inside the **Page\_Load()** method.
4. Add the following using statement to the top of the file:

using Microsoft.SharePoint.Client;

1. Now, within the **Page\_Load()**, add the following code. This will use the client side object model (CSOM) to reach into the hosting SharePoint site and retrieve all lists in the site. It will then bind them to the **DataList** control added to the **default.aspx** page:

Uri hostWeb = new Uri(Request.QueryString["SPHostUrl"]);

using (var clientContext = TokenHelper.GetS2SClientContextWithWindowsIdentity(hostWeb, Request.LogonUserIdentity))

{

var web = clientContext.Web;

clientContext.Load(web.Lists,

lists => lists.Include(

list => list.Title,

list => list.DefaultViewUrl

)

);

clientContext.ExecuteQuery();

AppWebLists.DataSource = web.Lists;

AppWebLists.DataBind();

}

1. Save all changes.

## Task 3 – Configure the Web Project Security Settings

In this task you will modify the security settings for the app’s Web project.

* Begin this task logged on to **SP** as **CORP\Administrator**.

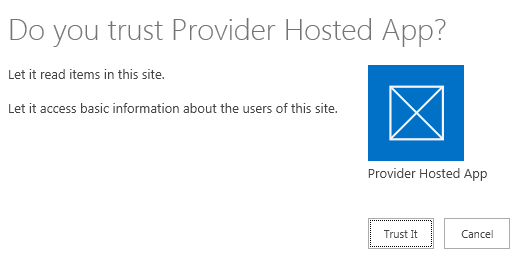
1. Ensure you have the **ProviderHostedApp** solution open that you created in the previous step in Visual Studio 2012.
2. Select the **ProviderHostedAppWeb** project and press **F4** to open the **Property** tool window.
3. Change two of the properties on the project:
   * **Anonymous Authentication:** Disabled
   * **Windows Authentication:** Enabled
4. Save all changes.

## Task 4 - Deploy and Test the SharePoint App

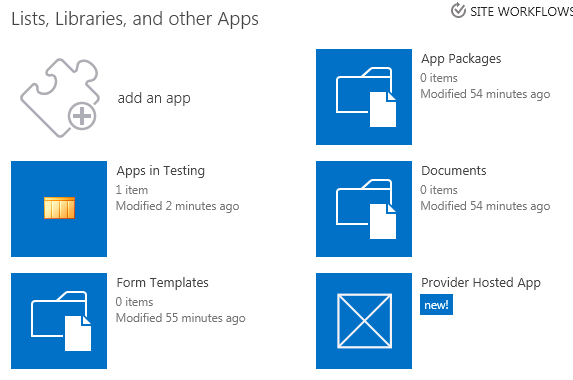
In this task you will deploy and test the SharePoint cloud app built in the previous task.

* Begin this task logged on to **SP** as **CORP\Administrator**.

1. Ensure you have the **ProviderHostedApp** solution open that you created in the previous step in Visual Studio 2012.
2. With the **ProviderHostedApp** solution open in Visual Studio, press **F5**.
3. After deploying the App, Visual Studio 2012 will launch Internet Explorer and take you to the App Installation confirmation page. This page will prompt you to trust the app as it is requesting permissions on the host site. Click the **Trust It** button below the app icon:



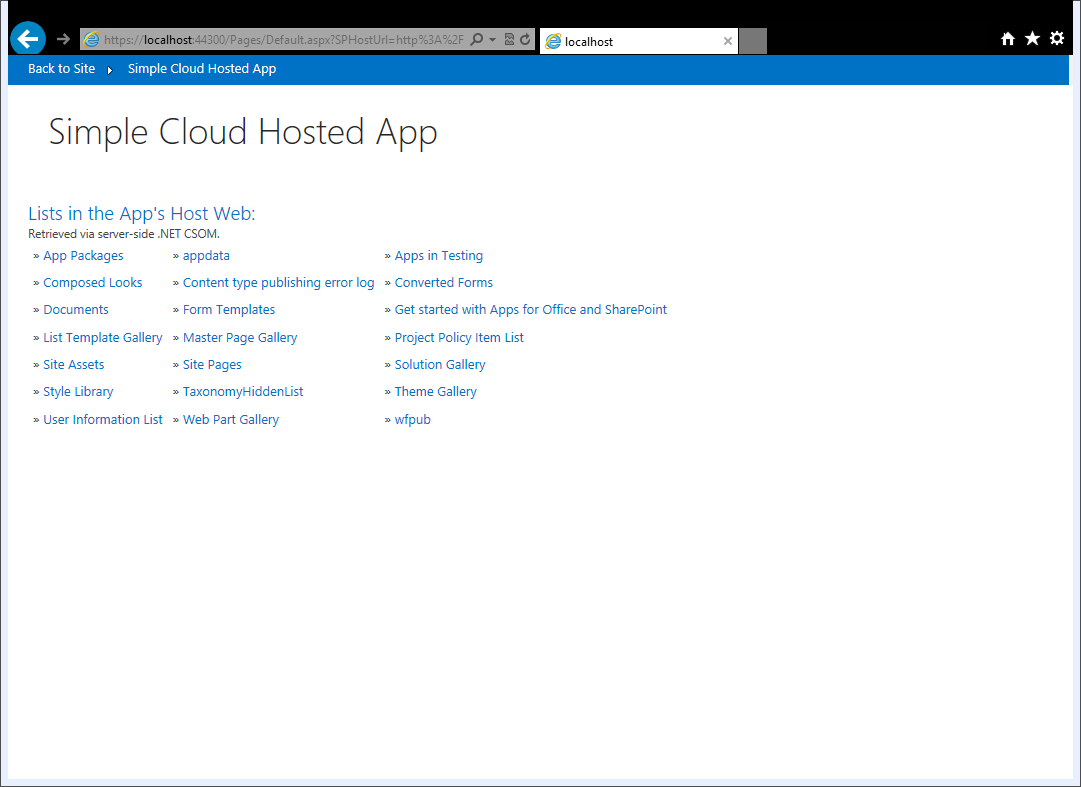
1. The browser will launch and take you to the **Lists, Libraries, and other Apps** page in the hosting SharePoint site.



1. Click the **Provider Hosted App** on the icon.
2. IE will display a message about a problem with the certificate used by the site. This is because the self-signed certificate is not from a trusted root authority.
3. Click the **Continue to this website (not recommended).** link.
4. When the page loads, notice that it lacks any styles. Internet Explorer prompts to show all content on the page. This is because we are using a self-signed certificate that is not trusted in IIS Express. Click “Show all content”.



1. When the page loads, notice that it looks very similar in style to the hosting SharePoint site and it is displaying a list of all the lists in the hosting SharePoint site.



* This is the end of the demo