# Office AMS: Host SP Apps on Azure Cloud Services

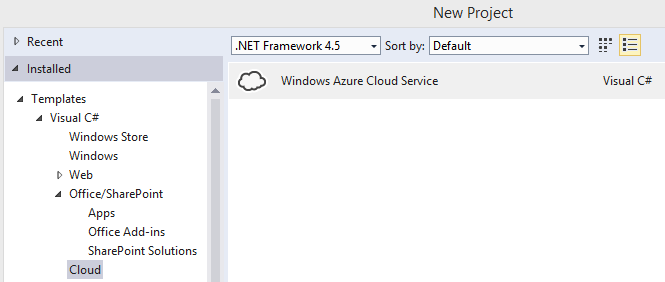
|  |  |
| --- | --- |
| Summary: | Applies to: |
| This sample shows how one can leverage the Azure Cloud Services (web and worker roles) to host SharePoint provider hosted apps. The code running on Azure shows how to make use of the tenant administration CSOM. | * Office 365 Multi Tenant (MT) |
| Solution: | Core.CloudServices, version 2.0 |
| Author: | Bert Jansen, Microsoft |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // THIS CODE IS PROVIDED \*AS IS\* WITHOUT WARRANTY OF  // ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING ANY  // IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR  // PURPOSE, MERCHANTABILITY, OR NON-INFRINGEMENT.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |

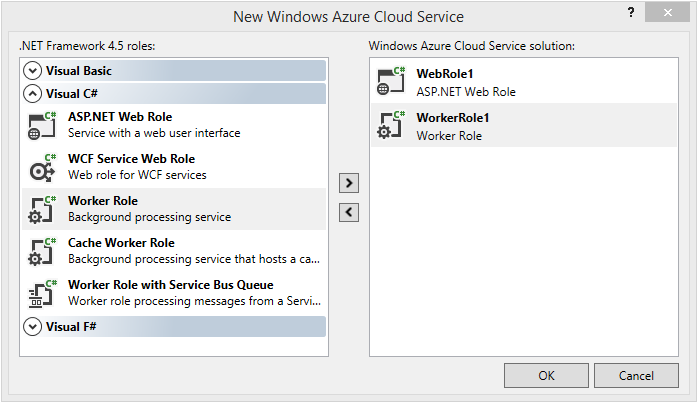
# How to prepare your Visual Studio 2013 environment to combine Azure and SharePoint Apps

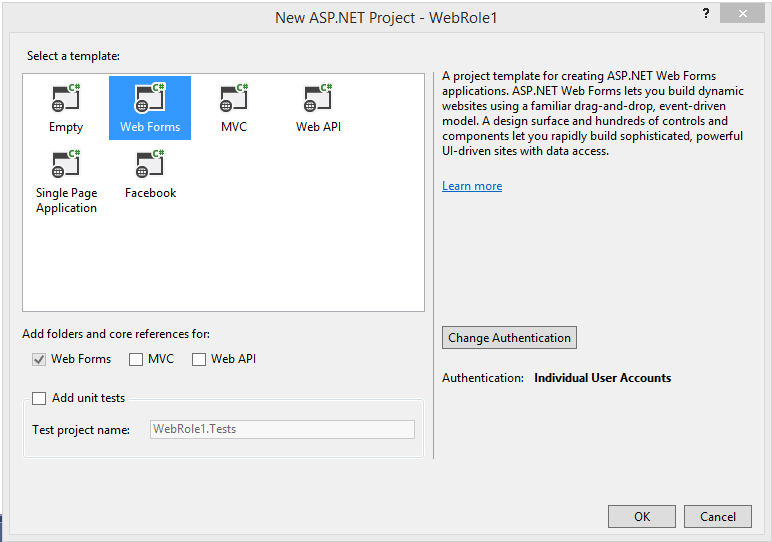
This sample is based on Visual Studio 2013 in combination with the Azure SDK version 2.2. It’s important to at least use these versions in order to get a first class experience in Visual Studio. We do want to have a solution that when pressing F5 spins up the Azure emulator, deploys the solution to your developer site collection and then runs your solution on the emulator. To realize this the following steps need to be followed (note that there are also alternative options, but below approach is used to build this sample).

## Create an Azure Cloud Services project

Open VS2013 and create a new Azure Cloud Services project with a web role and a worker role:

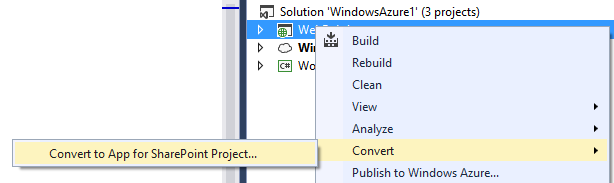




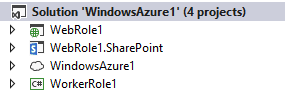


## Convert to Azure Web Role to a SharePoint App project

Right click the Azure web role project and use the convert menu to “transform” it into a SharePoint App project:

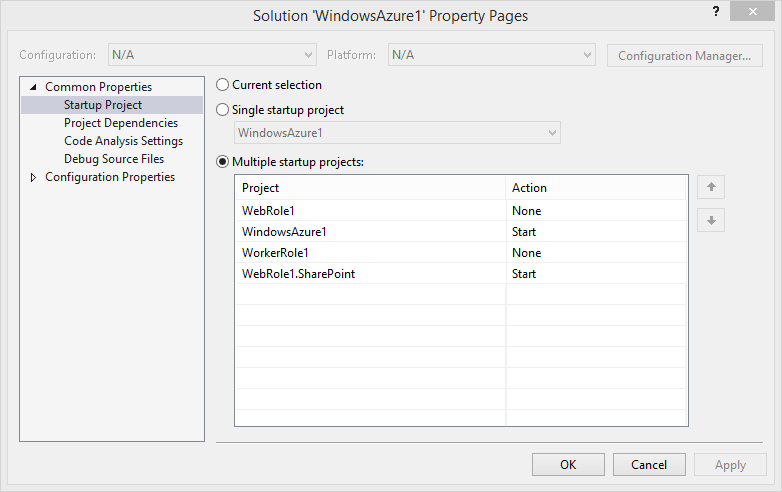


After the conversion completed you should see a similar project structure in your solution. Notice the WebRole1.SharePoint project that has been added:



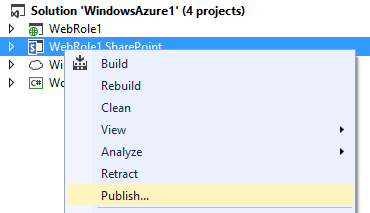
## Ensure both the Azure project as the SharePoint project are startup projects

To make the F5 experience work it’s important that the SharePoint App and the Azure project are starting on F5. This can be configured via the solution properties:



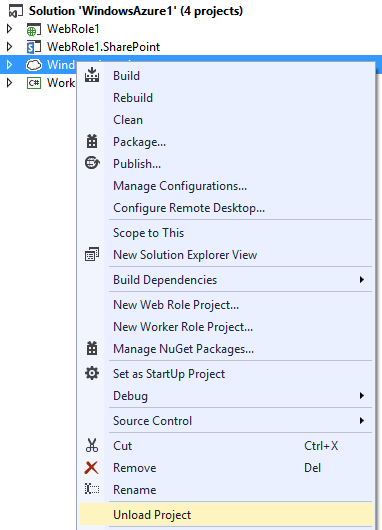
## IMPORTANT: ensure you’ve a SharePoint publishing profile before publishing your SharePoint app

If you’ve created the solution as described above and you do want to “Publish” your SharePoint app then this seems to work, but it’s not working correctly since the publishing wizard will not substitute your client ID and remoteAppUrl token because it’s lacking a SharePoint publishing profile.

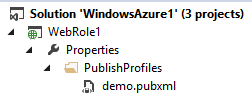


This is due to a bug in Visual Studio 2013, but luckily there’s an easy workaround. The very first time you do want to publish your app you should follow below steps, subsequent publishing actions will just work fine.

First we “unload” the Azure project:



Once this is done publish your SharePoint app by creating a publishing profile and packaging your app. After publishing you should see a folder named PublishProfiles with a pubxml file in the web project of your SharePoint App:



Once this is done you can reload the Azure project and you’re fine for all future publish actions.

# How use the SharePoint from Azure

There are two ways to get a client context that can be used with the both the SharePoint CSOM as the tenant administration CSOM and depending on the method you choose the configuration of your Azure web and worker roles requires additional configuration. A more complex could for example be a site provisioning system that uses the tenant administration CSOM to create a site collection and then use the SharePoint CSOM to activate features, add web parts to the home page,..

Option 1: using the SharePointOnlineCredentials class:

ClientContext cc = new AuthenticationManager().GetSharePointOnlineAuthenticatedContextTenant(messageParts[1],"TenantAdminUser", "TenantAdminPassword");

Option 2: using an OAuth app only token:

ClientContext cc = new AuthenticationManager().GetAppOnlyAuthenticatedContext(messageParts[1],"Realm","AppId","AppSecret");

Once you’ve a client context you can use it to work with SharePoint:

//Update the site title

cc.Web.Title = messageParts[0];

cc.Web.Update();

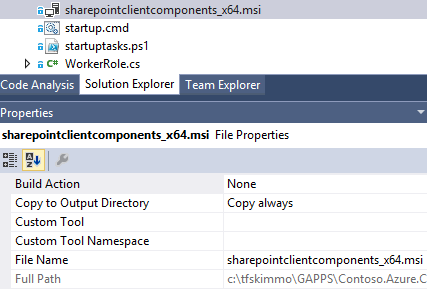
cc.ExecuteQuery();

## ClientContext based on the SharePointOnlineCredentials class

In this model we’re going to work with a SharePoint tenant administrator account and password that is being used to construct an instance of the SharePointOnlineCredentials class. This class is then used to derive the correct ClientContext object. To make this model operate in an Azure web or worker role the below configuration steps are required.

### Installation of ONline Services Sign-In assistant and SharePoint CSOM libraries is required

To install these two components we’ve added the installation packages to the project (notice the build action and copy to output directory settings):



Next we’ve foreseen a startup section in the servicedefinition.csdef file in the Azure project:

<Startup>

<Task commandLine="startup.cmd" executionContext="elevated" taskType="simple">

<Environment>

<Variable name="EMULATED">

<RoleInstanceValue xpath="/RoleEnvironment/Deployment/@emulated" />

</Variable>

</Environment>

</Task>

</Startup>

If we take a look at the startup.cmd we see (simplified) this:

@ECHO off

if "%EMULATED%"=="true" goto :EOF

ECHO "Installing the SharePoint CSOM library" >> log.txt

msiexec.exe /I "sharepointclientcomponents\_x64.msi" /qn

"Installing Microsoft Online Services Sign In Assistant" >> log.txt

msiexec.exe /I "msoidcli\_64bit.msi" /qn

ECHO "Completed SharePoint CSOM + SIA Installation" >> log.txt

### Ensure that the Azure service is running under System Account instead of Network account

The Online Services Sign-In assistant does not operate when being called from a process running under the Network service, so we need to run under the System account. To change this add the Runtime element in the servicedefinition.csdef:

<WorkerRole name="Core.CloudServices.Worker" vmsize="Small">

<Runtime executionContext="elevated" />

This will make an Azure worker role launch its host process under the system account and everything works fine, however for a web role this setting does not affect how the IIS application pools are configured which will host your web project. To fix this you need to change the application pool accounts to run under the system accounts via custom code that’s executed during role start:

// Only change application pool account when running in Azure, no need to change

// this for the emulator as the emulator requires you to run with

// administrative privileges

if (!RoleEnvironment.IsEmulated)

{

// Use the SetAppPoolIdentity method in case you want to use the tenant

// administration CSOM library in combination with specifying credentials

// via the SharePointOnlineCredentials class

SetAppPoolIdentity();

}

## ClientContext based on an OAuth token with tenant level permissions

Using a clientcontext based on an OAuth token actually is easier: there are no additional Azure configurations required! The only thing you need to do is register your app, grant it tenant level permissions and use the TokenHelper.GetAppOnlyAccessToken and TokenHelper.GetClientContextWithAccessToken methods to construct a clientcontext based on an app only OAuth token.

// Use (Get-MsolCompanyInformation).ObjectID to obtain Target/Tenant realm: <guid>

//

// Manually register an app via the appregnew.aspx page and generate an App ID and

// App Secret. The App title and App domain can be a simple string like "MyApp"

//

// Update the AppID in your worker role settings

//

// Add the AppSecret in your worker role settings

//

// Manually set the permission XML for you app via the appinv.aspx page:

// 1/ Lookup your app via its AppID

// 2/ Paste the permission XML and click on create

//

// Sample permission XML:

// <AppPermissionRequests AllowAppOnlyPolicy="true">

// <AppPermissionRequest Scope="http://sharepoint/content/tenant" Right="FullControl" />

// </AppPermissionRequests>

//

// As you're granting tenant wide full control to an app the appsecret is as important

// as the password from your SharePoint administration account!

# How to make the App ID and App secret configurable from the Azure Management portal

This example shows how to make the relevant SharePoint app parameters (ID, secret, Realm) configurable from within the Azure management portal. To realize this a slightly adapted version of TokenHelper.cs is used in the Office AMS Core project in which the relevant properties can be read as properties. Office AMS Core provides us with an AuthenticationManager class that’s used to obtain the client context objects for both the sample using the SharePointOnlineCredentials class and the OAuth App Only token sample.

ClientContext cc = new AuthenticationManager().GetSharePointOnlineAuthenticatedContextTenant(messageParts[1],"TenantAdminUser", "TenantAdminPassword");

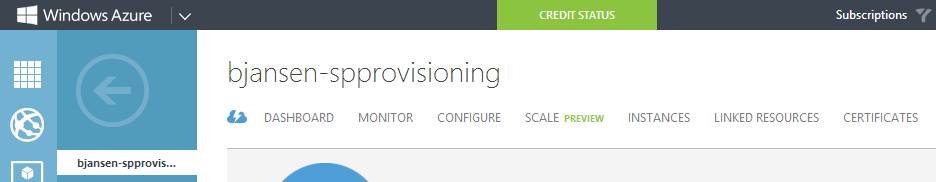
ClientContext cc = new AuthenticationManager().GetAppOnlyAuthenticatedContext(messageParts[1],"Realm","AppId","AppSecret");

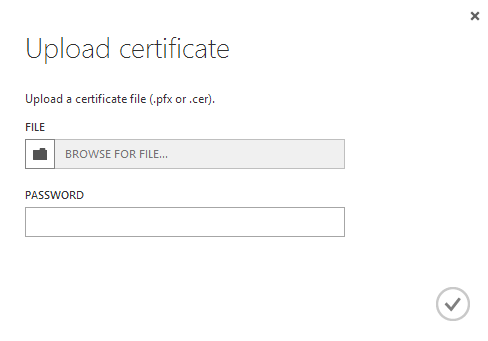
# How to encrypt configuration data in Azure

In the above samples you really would want to secure your password or clientsecret since these both contain tenant level permissions. Since we’re running on Azure and can have multiple instances of web and worker roles we need an encryption mechanism that is machine independent and therefore we’ve opted use certificate based encryption. This means that data is encrypted based on the public key of the certificate, but in order to decrypt the data one requires the private key of the certificate.

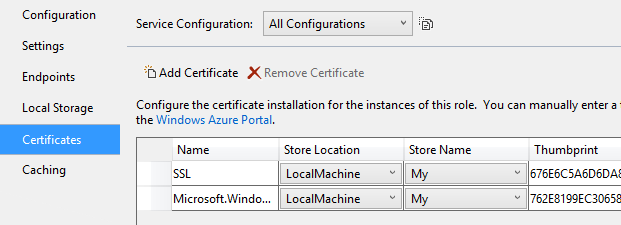
### Setup the certificate in Azure and in your solution

First you need to have a certificate in PFX format which contains a private key. You’ll need to go to your cloud service in the Azure management portal (<https://manage.windowsazure.com>), click on certificates and upload your certificate:



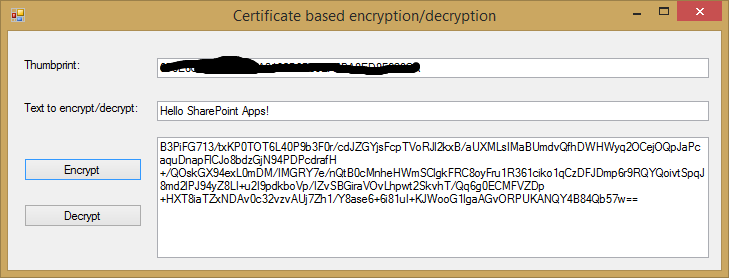


Next to that you should also install the certificate on your local machine. Once that’s done you can link your azure web and worker projects to the certificate by updating the relevant settings (notice the SSL entry):



### Encrypt/Decrypt data using the certificate

To encrypt data the sample contains a very simple winforms application (Core.CloudServices.Encryptor) that allows you to specify the thumbprint of the certificate to use, the value to encrypt and then shows the resulting encrypted text:



Important:

To decrypt the data the process doing the decryption needs to have access to the private key of the certificate. Inside a deployed Azure role this is taken care of by Azure, but in your development environment this means that you **do need to Visual Studio as administrator**.

Encryption/decryption is handled via the EncryptionUtiltity class in the from the Office AMS Core project.