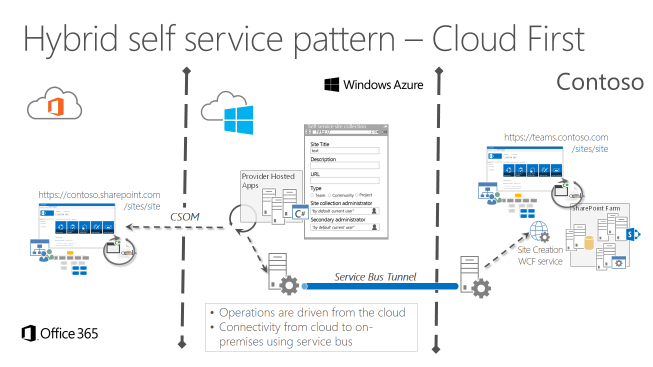
# Office AMS: Site provisioning reference solution

|  |  |
| --- | --- |
| Summary: | Applies to: |
| This sample is a site provisioning reference implementation that shows how to provision sites in SharePoint online and SharePoint on-premises. Sites are provisioned with a custom theme. | * Office 365 Multi Tenant (MT) |
| Solution: | Provisioning.Hybrid, version 1.0 |
| Author: | Vesa Juvonen (Microsoft), 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.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |

# Introduction

This reference implementation shows how one can use the cloud app model to provision site collections in SharePoint Online and in SharePoint on-premises. The same code base is used for both options: depending on the choice the user makes in the provisioning form the solution will either create the site collection in SharePoint Online or in SharePoint on-premises. The SharePoint provisioning app in this sample is designed to run in Azure cloud services which will result in cloud driven solution. In order to provision sites collections on-premises the app running on Windows Azure cloud services uses Windows Azure service bus to make the connection to an on-premises component. Below schema shows the above flows in action:



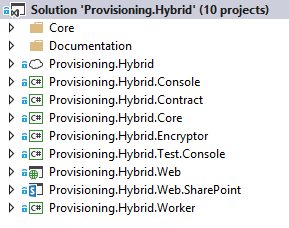
## References to other Office AMS samples

This sample must be seen a reference implementation that makes use of some of the other Office AMS samples. Therefore we would like to refer to the documentation of these samples for the following elements:

* **“Core.CloudServices”**: this sample explains how to setup a Visual Studio 2013 project that uses Azure Cloud services and SharePoint apps. It also elaborates on how to use an app only OAUTH token for the provisioning part and how to handle encryption and decryption of configuration data when running on Windows Azure cloud services
* **“Core.PeoplePicker”**: this sample provides you with a detailed explanation on how to use the people picker component for provider hosted apps
* **“Provisioning.Pages”**: explanation on how to create wiki pages, how to add and remove web parts or html snippets from these pages is shown in this sample
* **“Provisioning.SubSiteCreationApp”**: this sample shows how to hookup sub site provisioning and how to apply branding by uploading and applying a custom theme
* **“Provisioning.Services.SiteManager”**: as there’s no remote API for site collection provisioning for SharePoint On-premises this sample uses the WCF endpoint that’s being provided as part of this sample.

# Solution setup

The solution contains 10 projects:



Below you can find a short description of each of the projects:

* **Provisioning.Hybrid**: this is the Azure cloud services project that holds the configuration data for three cloud services: one web role (Provisioning.Hybrid.Web) and one worker role named Provisioning.Hybrid.Worker
* **Provisioning.Hybrid.Web**: this is ASP.Net web project that will be part of the SharePoint application and that will be hosted as a web role on Azure Cloud services
* **Provisioning.Hybrid.Web.SharePoint**: this is the SharePoint app project. Together with the previous project these 2 projects together are the SharePoint app. Since this a provider hosted app this project only contains the app manifest and a dummy module to trigger the creation of an appweb (required for the people picker control)
* **Provisioning.Hybrid.Worker**: this is the Azure Cloud services worker role project. This project will, by making use of the Provisioning.Hybrid.Core and the Office AMS Core library projects, do the actual site collection creation work
* **Provisioning.Hybrid.Contract**: this project holds the data contract classes which are used to pass information from the SharePoint app to the service endpoint.
* **Provisioning.Hybrid.Core**: this project contains the core provisioning classes and all their supporting artefacts such as themes and mail template files
* **Provisioning.Hybrid.Console**: this project is a console project that uses Windows Azure Service Bus to listen for requests coming from the SharePoint Provisioning solution. This typically is the part that you would run on-premises
* **Provisioning.Hybrid.Test.Console**: this is a test console application that you can use to push a message via service bus to the “Provisioning.Hybrid.Console” process that will be running on-premises
* **Provisioning.Hybrid.Encryptor**: this is a little forms application that can be used to encrypt or decrypt content based on a certificate. You typically use this to do the initial encryption of the sensitive configuration data

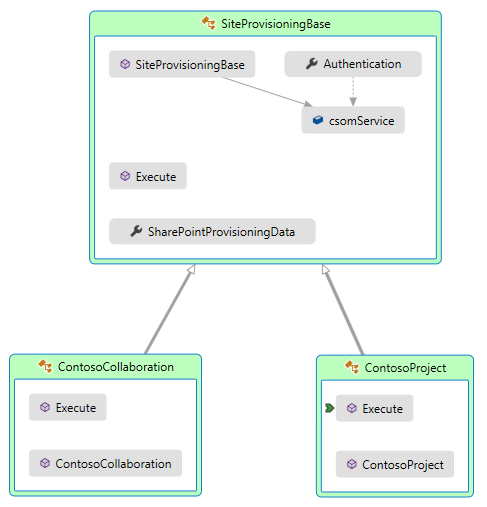
As mentioned this project makes also use of the Office AMS Core library to the actual CSOM work. Check out the documentation of Office AMS Core to learn more.

# Site Provisioning

The actual site provisioning code is running in the ContosoCollaboration and ContosoProject classes. These classes inherit a SiteProvisioningBase abstract base class. The base class is responsible for providing the correct authentication provider and to provide a reference to an instantiated ClientSideSharePointService class. This last class is the class that will hold all the CSOM manipulations such as creating a site collection, enabling a feature, creating a list,…If you want to create additional “templates” then you should take the following steps

1. Add a new template provisioning class that inherits from SiteProvisioningBase (Provisioning.Hybrid.Core project)
2. Define a new a new constant in the SiteProvisioningTypes class (Provisioning.Hybrid.Core project)
3. Update the ProcessMessage method in the WorkerRole class (Provisioning.Hybrid.Worker project)
4. Add an item to the dropdown on default.aspx (Provisioning.Hybrid.Web project)

Below picture shows the important code in the Provisioning.Hybrid.Core project:



If we take a deeper look at such a site provisioning class then we can see only one method named Execute. All the site provisioning logic that you need must be triggered from this method call. A typical Excute method contains the following structure:

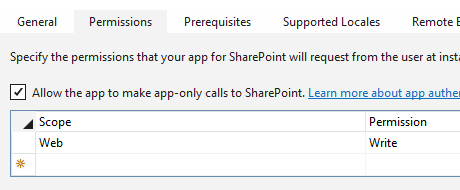
1. Read configuration data, decrypt when needed
2. Determine the site collection URL for the to be created site collection
3. Set the status to “Provisioning” in the site directory so that the users can see that their site collection is being created
4. Complete and verify the received site creation input
5. Create the site collection
6. Enable/Disable site collection and/or site scoped features
7. Add additional lists / configure lists / add data to lists
8. Update existing pages (add/remove web parts and/or html snippets)
9. Create additional pages
10. Adjust the navigation
11. Adjust site permissions (add additional administrators, add other types of access)
12. Set the status to “Available” in the site directory
13. Send a mail to notify the site owners of the site creation success. This mail contains a link to the newly created site collection

Off course these steps can be expanded with other steps such as there are:

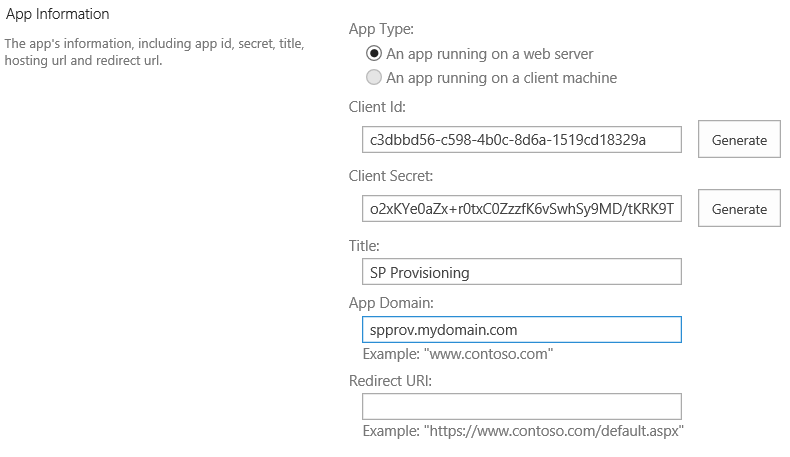
* Inserting of a JavaScript based hook for site customizations that are not possible via CSOM (see Office AMS samples: “**Core.JavaScriptCustomization**”)
* Sub site creation. See Office AMS sample **Provisioning.SubSiteCreationApp**

# Security concepts

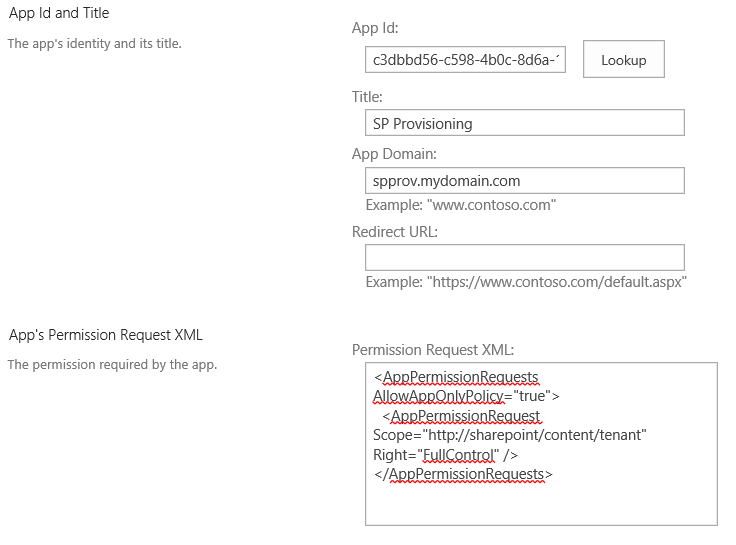
From a security perspective this sample application actually consists out of 2 apps: there the SharePoint App that contains the UI (default.aspx) and there’s the SharePoint app that’s being used for the site collection creation in the Azure worker role. The SharePoint app for the UI is regular SharePoint app with basic permissions as you can see in the below screenshot of the app manifest:



The SharePoint app for the site collection creation however is different: there’s no appmanifest.xml file for this app, so the creation of the app always need to happen via the appregnew.aspx page. Once the app is created via the appregnew.aspx page you can provide it the app with the needed permissions via the appinv.aspx page. Below screenshots show these two pages. First screenshot shows the appregnew.aspx page where you can generate a client ID (aka app ID) and a client secret (aka app Secret), provide a title and domain. Note that the domain you specify here does not have to be a real existing domain, just a string that’s formatted as a domain name is good enough:



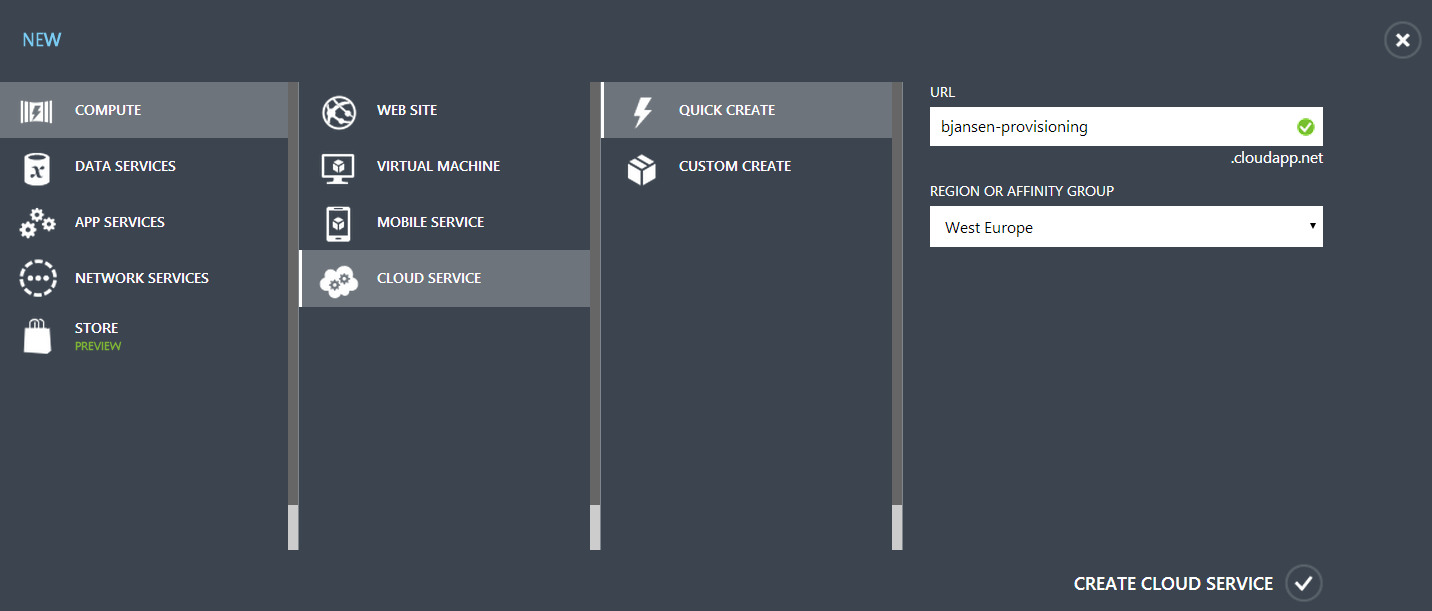
Use the Appinv.aspx page to lookup the app created in the previous step and then specify the permission XML. Given that this app will be used in a worker role it’s important that you don’t forget to set the AllowAppOnlyPolicy to true:



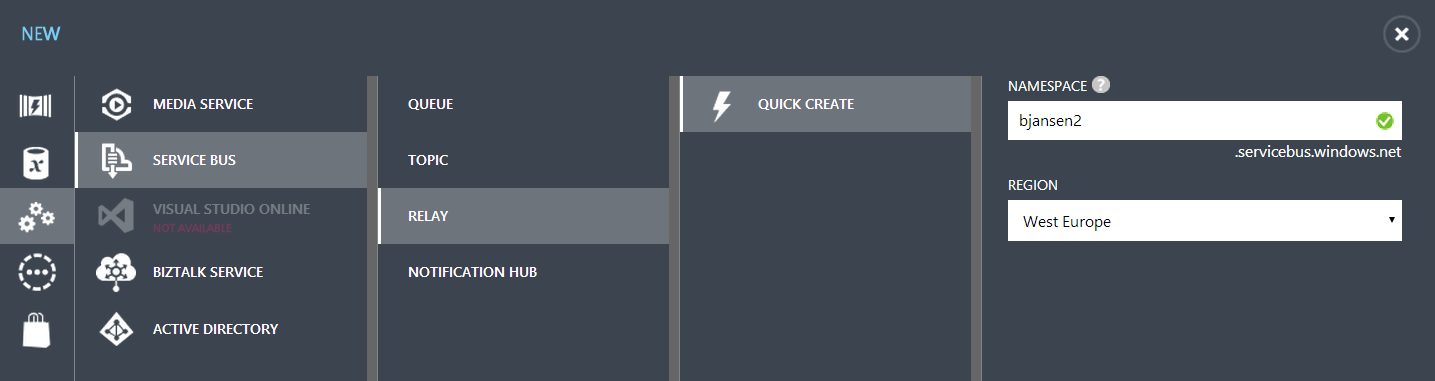
# Deployment

## Preparation of the Azure tenant

### Create a cloud service in your Azure tenant



### Create a service bus namespace in your Azure tenant (or reuse an existing one)



Click on connection information and copy the default issuer (owner = default), the default key and your service bus namespace (bjansen2).



### Certificate

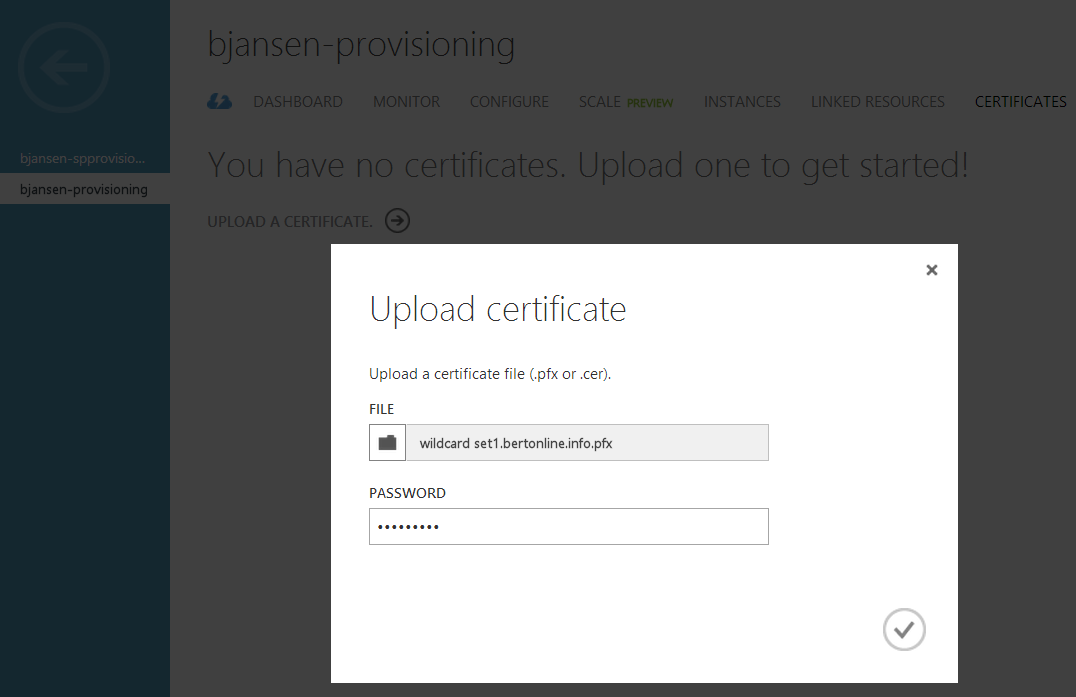
Ensure you either have a self-signed certificate for your cloud app name (bjansen-provisioning.cloudapp.net in the sample) or that you have public trusted certificate linked to your own DNS (e.g. \*.set1.bertonline.info certificate linked to bertonline.info domain name). The latter option is the preferred one and will be used in the remainder of the deployment steps.

### DNS

If you’re using a non cloudapp.net certificate then you’ll need to setup DNS so that there’s a CNAME of your custom domain (e.g. prov.set1.bertonline.info pointing) pointing to your Azure cloud service (e.g. bjansen-provisioning.cloudapp.net). If you don’t have DNS you can do this also by putting the IP address of bjansen-spprovisioning.cloudapp.net in the hosts file on the machines that need it. Minimally these are your test box and the Azure Cloud Service web client (only possible after deployment).

### Deploy the certificate to your Azure cloud service

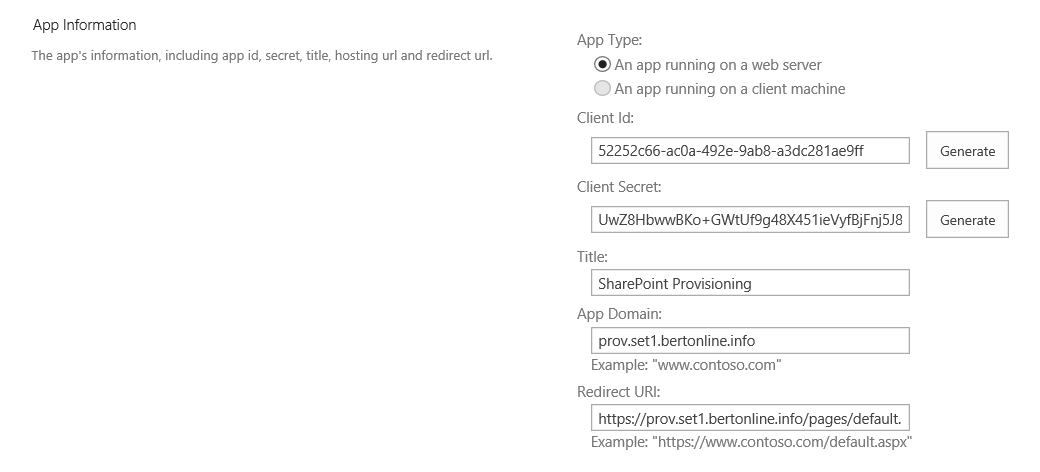
Use the Upload button in the Certificates section of the Azure cloud service you’ve created.



## Preparation of the SharePoint Online tenant

### Register the provisioning screen app in SharePoint using appregnew.aspx

Your app domain will be the tied to the certificate as it needs to match. In this case the app domain is prov.set1.bertonline.info which matches the \*.set1.bertonline.info certificate. If you use a self-signed cert for bjansen-provisioning.cloudapp.net then that’s your app domain.



Take note of the generated clientID and clientSecret

### Register the “actual” provisioning app in SharePoint using appregnew.aspx and appinv.aspx

Do the same as in previous step, but in this case when entering appregnew.aspx a dummy appdomain ([www.contoso.com](http://www.contoso.com)) will be ok. Redirect URL can be left empty. Copy the ClientID and ClientSecret. Once that’s done use the appinv.aspx page to perform look of the created app and grant it tenant level permissions by pasting the following permission XML and press create to confirm:

<AppPermissionRequests AllowAppOnlyPolicy="true">

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

</AppPermissionRequests>

### Create a site directory site collection

Create a site collection that will hold the site directory and in which the app will need to be installed. In the remaining documentation the site collection used is <https://bertonline.sharepoint.com/sites/spc>.

### Create a site directory

The site directory is not (yet) created automatically as part of the app as the app. Quick solution is to import the sitedirectorytemplate.stp list template and create a list of it with as name **Site overview**.

## Preparation of the SharePoint on-premises solution

### Deploy the contoso.services.sitemanager solution package

This solution package can be obtained from Office AMS sample “Provisioning.Services.SiteManager”:

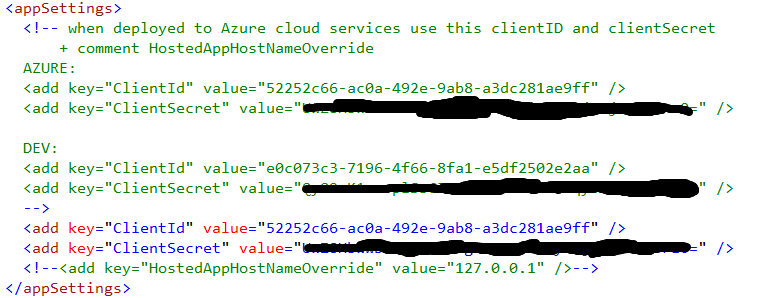


## Preparation of the Visual Studio solution

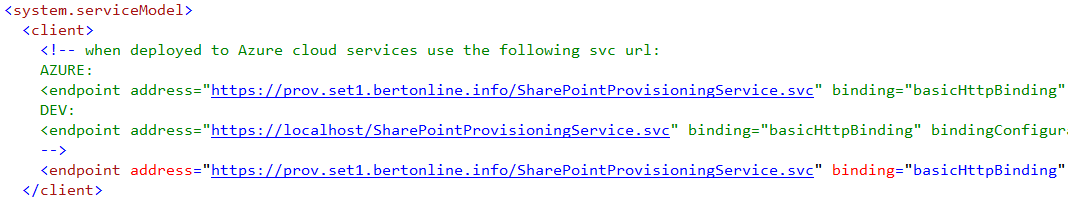
Before the solution can be deployed a number of settings have to be set correctly

### Web.config of Provisioning.Hybrid.Web

Ensure the right **provisioning screen app** clientID and clientSecret are set. Also comment HostedAppHostNameOverride before you deploy.



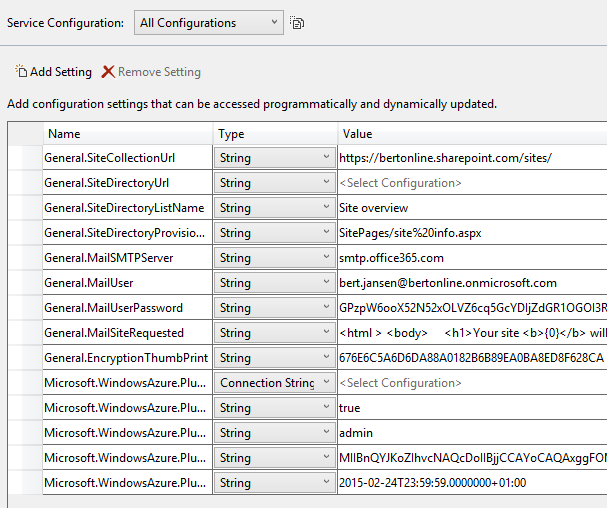
Set the correct WCF endpoint URL:



### “Provisioning.Hybrid” cloud service settings

#### Provisioning.Hybrid.Web

Go to the settings of the cloud service where you’ll see this:



Note:

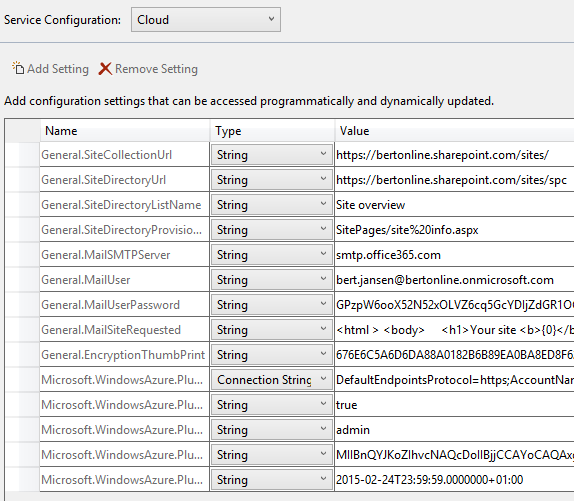
Some of the values are encrypted in the settings file. To encrypt your own take the following steps:

* Compile the solution
* Launch Provisioning.Hybrid.Encryptor.exe as an admin
* Define the thumbprint of the certificate you want to use for encryption. You can take a dedicated cert or use the SSL cert. Assumption is that the SSL cert is used, if you use a dedicated cert then don’t forget to also deploy it to the cloud service as mentioned in the certificate chapter of the Azure preparation steps
* Enter the text you want to encrypt, encrypt it and copy the encrypted text

Update following values:

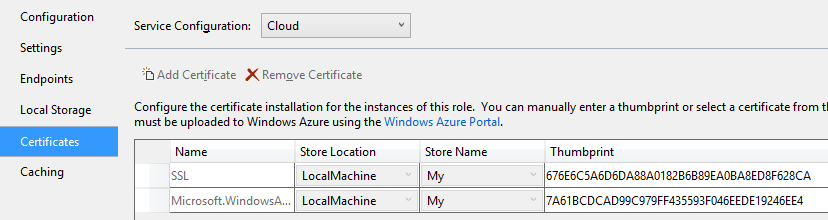
* General.siteCollectionUrl: this should point to your tenant managed path that will hold the newly created site collections
* General.MailUser: a user in your tenant that has a mailbox. Mails will be sent from this user’s mailbox.
* General.MailUserPassword: password of the mail user. This value must be encrypted
* General.EncryptionThumPrint: thumbprint of the cert used to encrypt the sensitive data

Switch the service configuration to cloud:



Update the general.sitedirectoryListName to point to the site collection you’ve created earlier.

Go to certificates and ensure that the entry SSL points to the thumbprint of the certificate you’re using for SSL and encryption. If you’ve a separate encryption certificate then you’ll need to add a line here and also reference the encryption certificate:



#### Provisioning.Hybrid.Worker

This is identical to the previous chapter with additional of following “All Configurations” settings:

* AppId: clientID of your “actual” provisioning app
* AppSecret: encrypted client secret of your “actual” provisioning app
* Realm: realm of your tenant (use MSOL PowerShell (Get-MsolCompanyInformation).ObjectID to get this for your tenant)
* General.SBServiceNameSpace: you’re service bus namespace (e.g. bjansen2)
* General.SBIssuerName: the service bus issuer name (owner)
* General.SBIssuerSecret: the **encrypted** service bus issuer secret

### App.config of Provisioning.Hybrid.Console

The settings in this config file are pretty identical to the once set for the Azure worker project with some on-premises specific additions:

* General.OnPremUserName: name of an account that can create site collections
* General.OnPremUserPassword: **encrypted** password of that account
* General.OnPremUserDomain: domain of that account
* General.OnPremWebApplication: url of the web app that will host the on-premises site collections (e.g. https://sp2013.set1.bertonline.info)

## Publish the SharePoint app

### Create the app package

Ensure you’ve a correct publishing profile: best to make a new one with your provisioning screen app client ID and client secret. Right click the Provisioning.Hybrid.Web.SharePoint project and choose “Publish”. Use the “Package the app” option to create an app package. When you create the app package use the same domain name as used in your ssl certificate (e.g. prov.set1.bertonline.info).

### Upload the app package to your app catalog

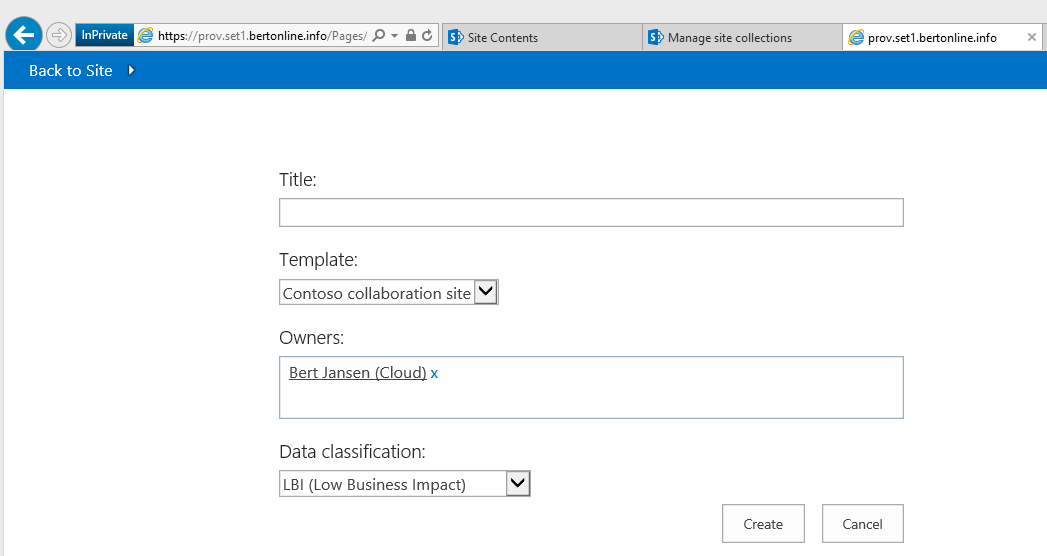
Upload the created .app file to the app catalog of your tenant. If you don’t know how to find the app catalog then use tenant administration to find out.

## Publish to Azure Cloud Services

Right click the “Provisioning.Hybrid” project and choose “Publish”. Create a target profile that matches your Azure tenant and select the cloud service you’ve created before. Press Publish to have Visual Studio do the deployment. This will take around 15 minutes the first time. Subsequent runs are faster.

## Installation of the SharePoint app and testing

Go the site collection created earlier on (<https://bertonline.sharepoint.com/sites/spc>) and install the provisioning app you’ve added to the app catalog. After installation clicking on the app should give this:



## Hooking up the on-premises part

For testing purposes the on-premises farm should also have VS2013 installed. In that case the solution you’ve just created can be copied to the on-premises SharePoint 2013 server. Open the solution in VS2013 and set the Provisioning.Hybrid.Console project as start project.

Note:

Ensure that the certificate used to encrypt sensitive data is deployed on the machine hosting the on-premises component. Deployment via PFX file as we need to the private key.

Press F5 to run. The Provisioning.Hybrid.Console project will now connect to Service Bus and wait for a “HBI” site creation from the SharePoint Online provisioning app.