|  |
| --- |
| Workshop de ALM - Laboratorio 03 |
| Implementar Integración Continua en Visual Studio Online |
|  |

|  |
| --- |
|  |

# Pre requisitos

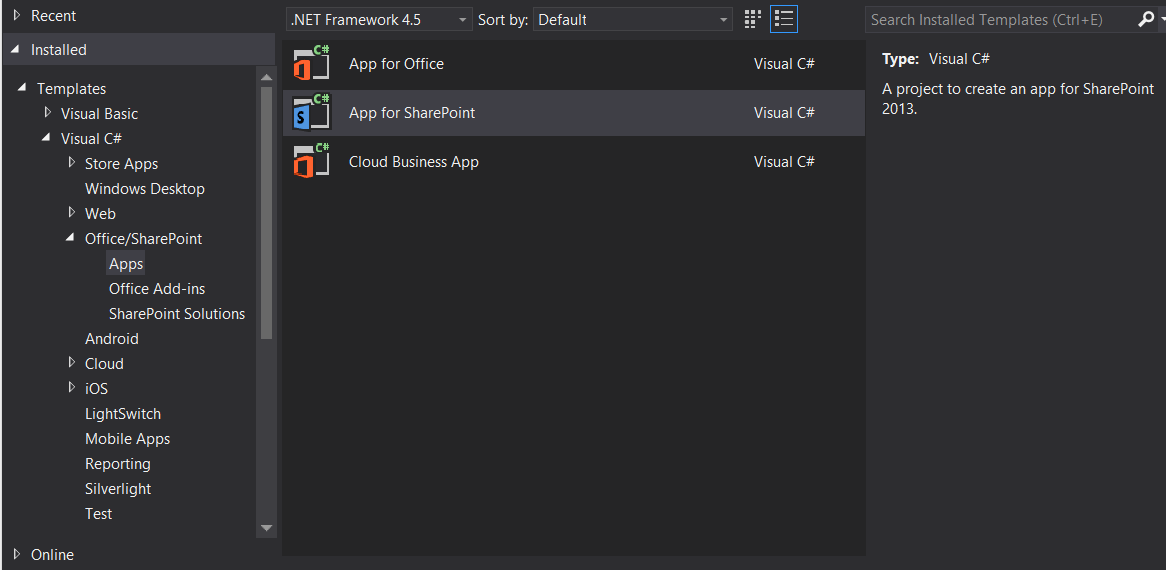
Para este laboratorio se requiere que tenga un Visual Studio Online con permiso de Administrador. Si no la tiene puede crear una subscripción Gratuita hasta 5 desarrolladores desde la URL [https://www.visualstudio.com/products/what-is-visualstudio-online-vs.aspx](https://www.visualstudio.com/products/what-is-visual-studio-online-vs.aspx)

También es necesario un subscripción de Office 365. Podemos obtener una licencia de evaluación de 30 días en la siguiente URL <https://portal.office.com/Signup/Signup.aspx?OfferId=B07A1127-DE83-4a6d-9F85-2C104BDAE8B4&dl=ENTERPRISEPACK&culture=es-ES&country=ES&ali=1#0>

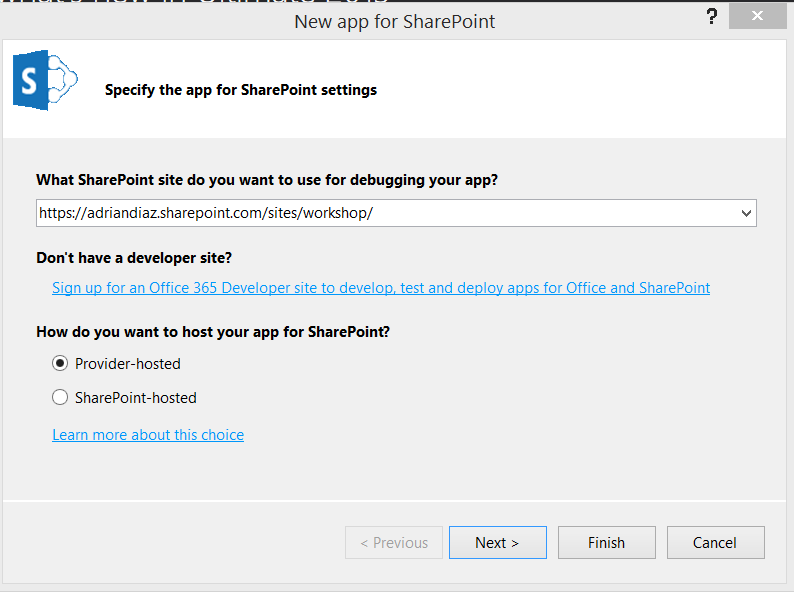
Y una subscripción de Azure. Es posible obtener una licencia trial durante 30 días o un gasto de 150$ en la siguiente URL <http://azure.microsoft.com/es-es/pricing/free-trial/?WT.mc_id=AzureBg_Spain_SEM_GA_DENTSU>

# Laboratorio 1 Implementar la integración Continua para Apps de SharePoint

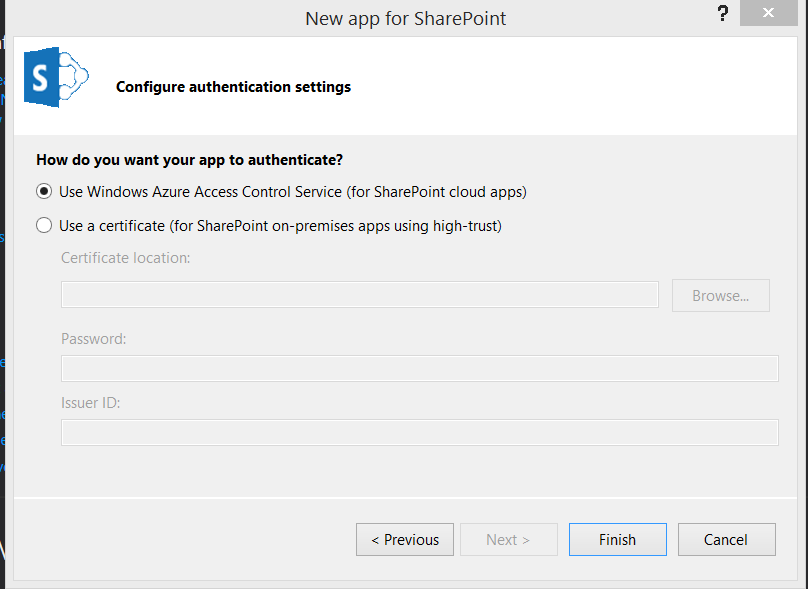
Partimos del supuesto en el que tenemos una APP de SharePoint de Tipo Provider Hosted. Esta Aplicación va a estar alojada en Azure. Para ello abrimos Visual Studio -> Templates -> Visual C# -> Office /SharePoint -> Apps-> App for SharePoint

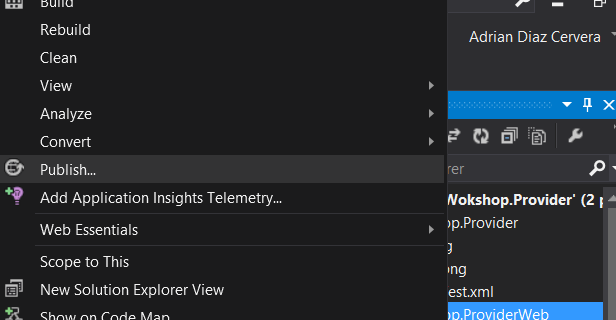


A continuación seleccionamos el tipo de alojamiento Provider Hosted para nuestra aplicación.

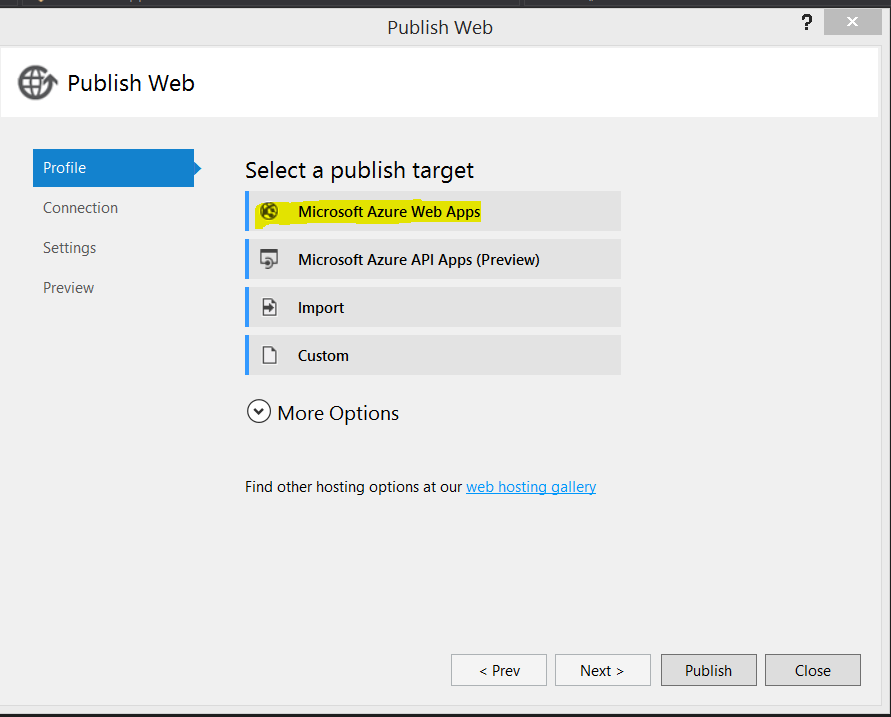


Seleccionamos el tipo de certificado que vamos a utilizar. En caso de Office 365 utilizaremos Windows Azure Access Control Service, si estamos en un entorno On Premise utilizaremos un Certificado de una entidad autorizada.

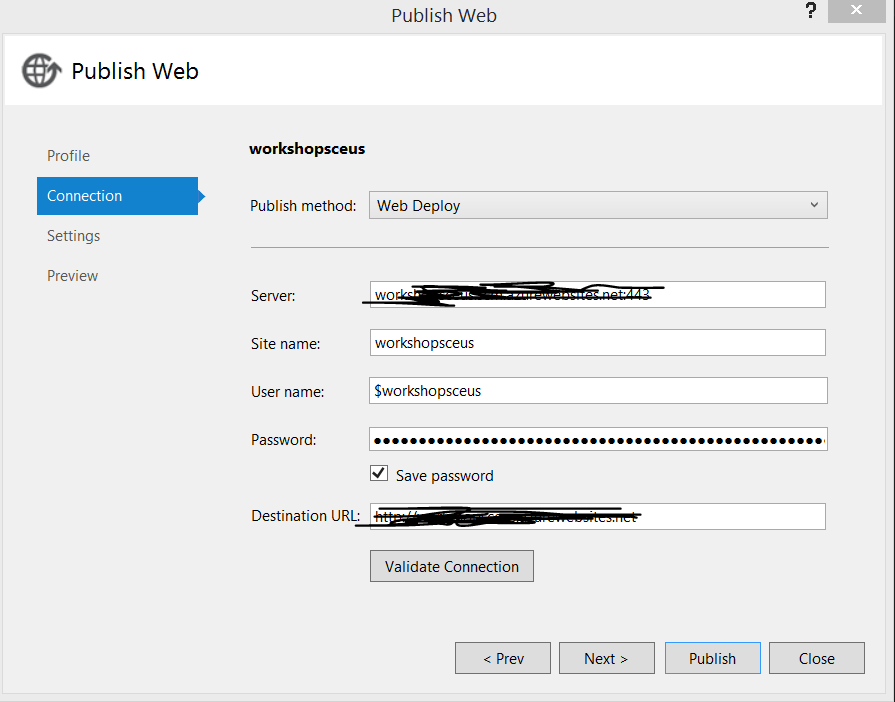


Una vez ya tenemos la estructura del proyecto creada publicaremos la Aplicación MVC en un site de Azure. Para ello en primer lugar iremos al explorador de soluciones, seleccionaremos la aplicación MVC, botón derecho y pulsaremos sobre Publicar 

Dentro de las opciones disponibles Seleccionaremos Microsoft Azure Web Apps



A continuación creamos una nueva Web App o utilizamos una Web Apps ya existente



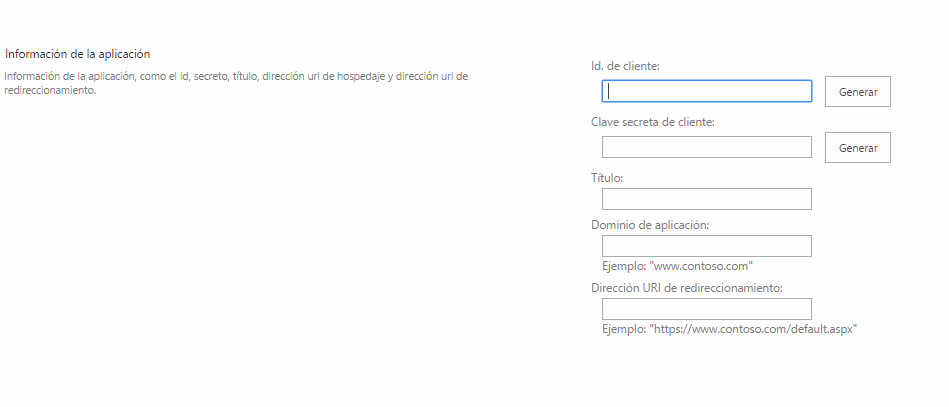
Haga clic en Siguiente y seleccione la forma en la que se va a desplegar bien en modo depuración bien (debug) o en producción (reléase). A continuación se pulsa sobre Publish.

Una vez ya tenemos el entorno de nuestra aplicación listo. El siguiente paso es configurar nuestra Build para poder empezar con la integración continua. El punto de partida para empezar con la integración continua de las Aplicaciones es un paquete de Codeplex. <https://officesharepointci.codeplex.com/>

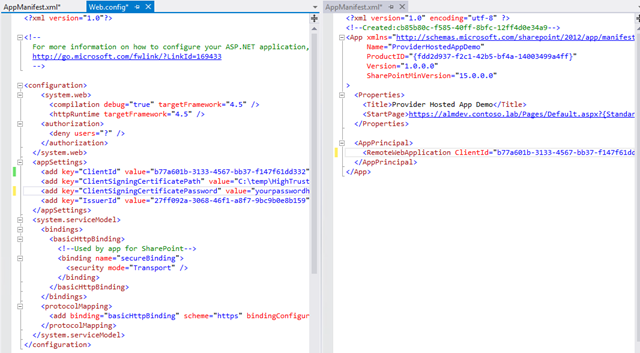
Este paquete tiene dos carpetas:

1. *BuildProcessTemplates*: Ficheros xml que tienen la serie de requisitos que va a realizar la build.
2. *DeployScript*: Una serie de Script PowerShell para poder instalar la apliación en los entornos necesarios. Estos Scripts no asumen que la aplicación ya existe

El proyecto de CodePlex que estamos utilizando asume que la aplicación ya está registrada en tu entorno. Para las pruebas en primer lugar vamos a registrar la aplicación para ello iremos a https://sitiodeveloper/\_layouts /appregnew.aspx y registrar la aplicación.

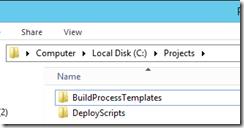


Con el ID Cliente y la clave Secreta la introducimos en la aplicación. Por un lado dentro de los parámetros del WebConfig y por otro lado dentro de la APP de SharePoint tal y como visualizamos en la siguiente imagen:

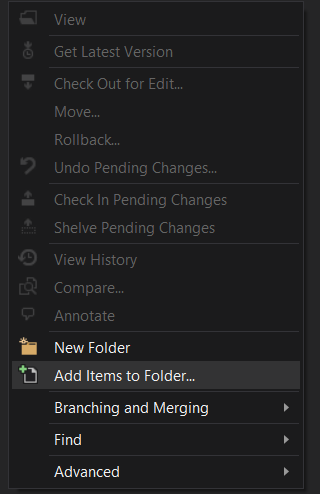


**Agregar el paquete de Codeplex dentro de nuestro TFS**

Copiar las carpetas “DeployScripts” y “BuildProcessTemplates” descargadas del Proyecto de Codeplex dentro de la carpeta del Proyecto.

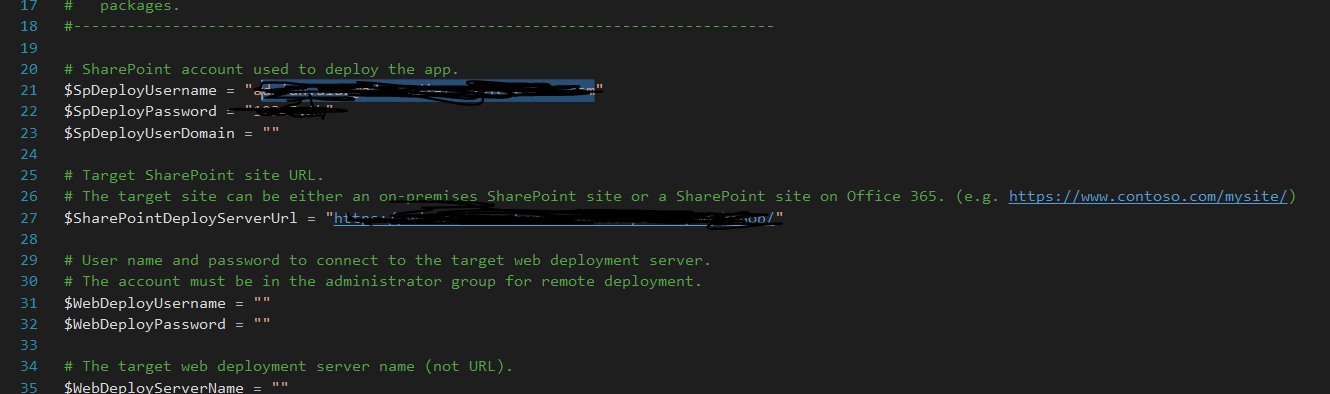


Ahora que la estructura del archivo coincide con lo que quiero en control de código fuente, que lo hace más fácil para mapear todo en TFS. Haga clic derecho en el nodo del proyecto en el Explorador de control de código fuente y elegir “Add Item to Folder”



TFS nos advierte que hay dos elementos que han cambiado en el proyecto existente. Aceptamos y elegimos todos los elementos a añadir.

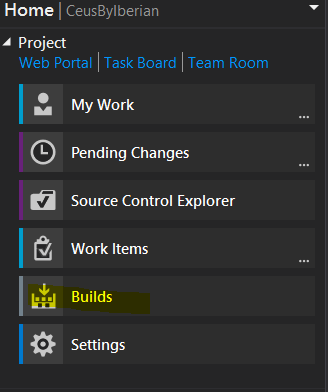
A continuación llega la hora de indicar los valores en los que nosotros queremos realizar la modificación. Para ello vamos a la carpeta DeployScripts/SharePointApp/Parameters Aquí es donde añadimos los valores de nuestro Office 365, SharePoint Onpremise o Web Deployment (en caso de que no sea una Web APP de Azure).



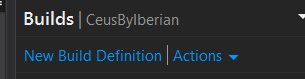
A continuación guardamos los cambios en el TFS

**Crear la definición de la Build**

En el explorador de soluciones pulsar sobre la pestaña Team Explorer y seleccionar sobre el icono de Builds.

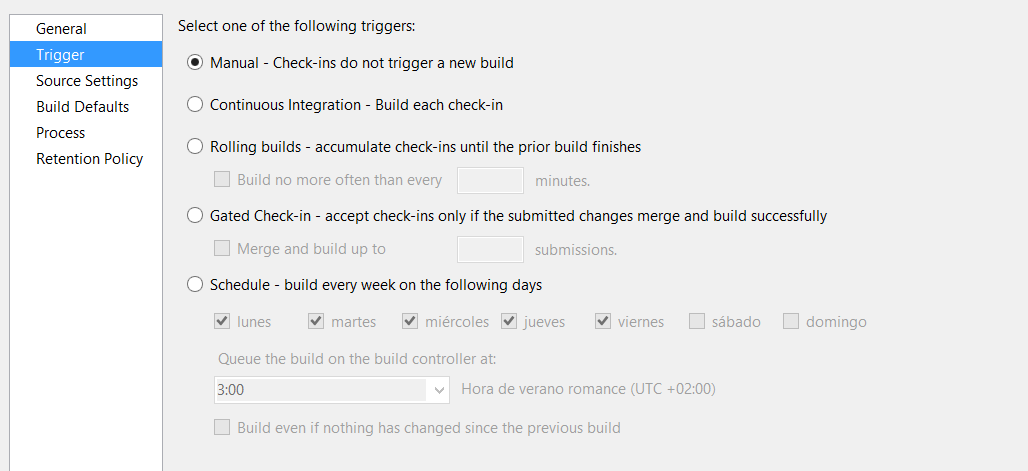


Pulsamos sorbe la Opción de New Build Definition:



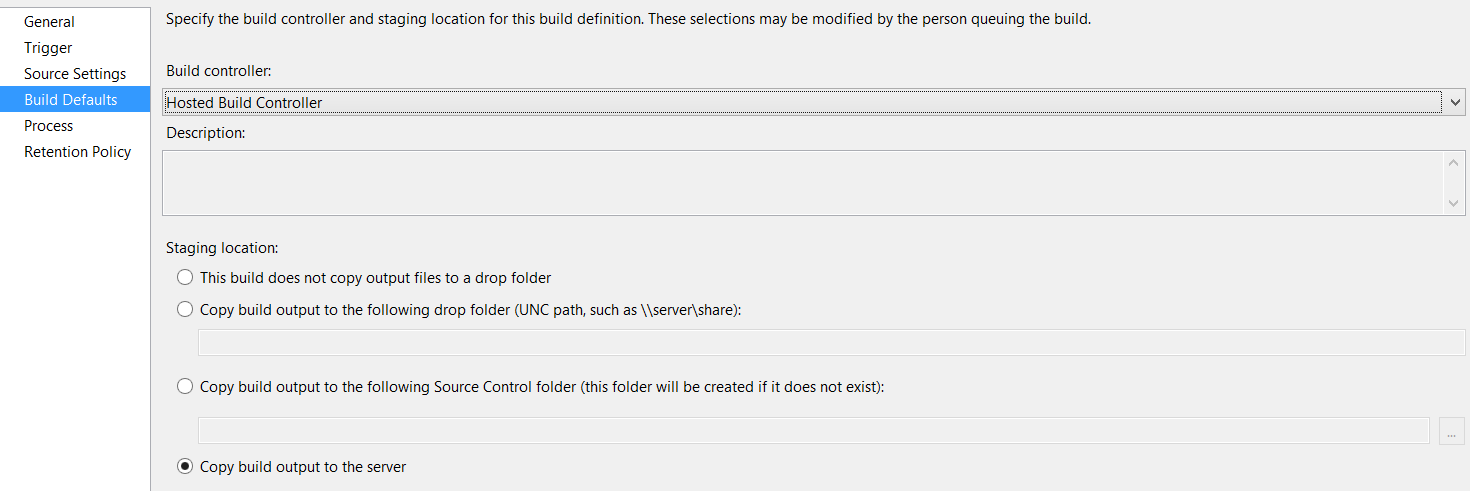
A la hora de la creación tendremos una serie de pestañas laterales en la que agregaremos la configuración de la Build.

**Trigger**; Esto parámetro establece cada cuento tiempo queremos ejecutar la BUILD. Depedendiendo de lo que realice la definición optaremos una solución u otra. Quizás subir a un entorno de Integración solo lo queremos realizar una vez al día, pero por el contrario si que queremos que cada Check In que se realice se analice el código subido al gestos del código fuentes. Todas estas opciones las configuramos en la siguiente pantalla:



**Source Settings:** Agregaremos los proyectos que se van a compilar en cada Build.

**Build Defaults**: Seleccionamos el Build Controller y la ubicación donde va a estar el resultado de la build.

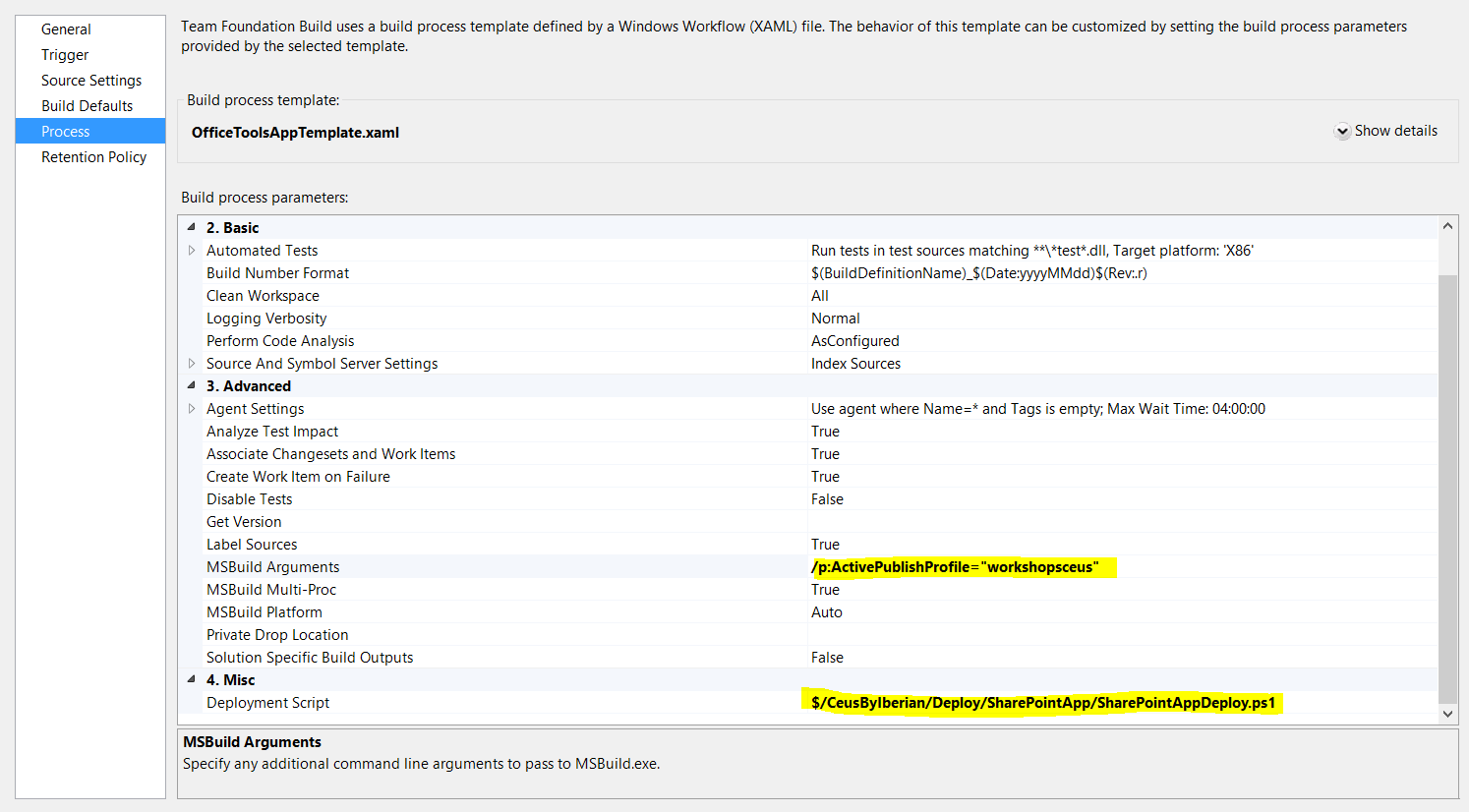


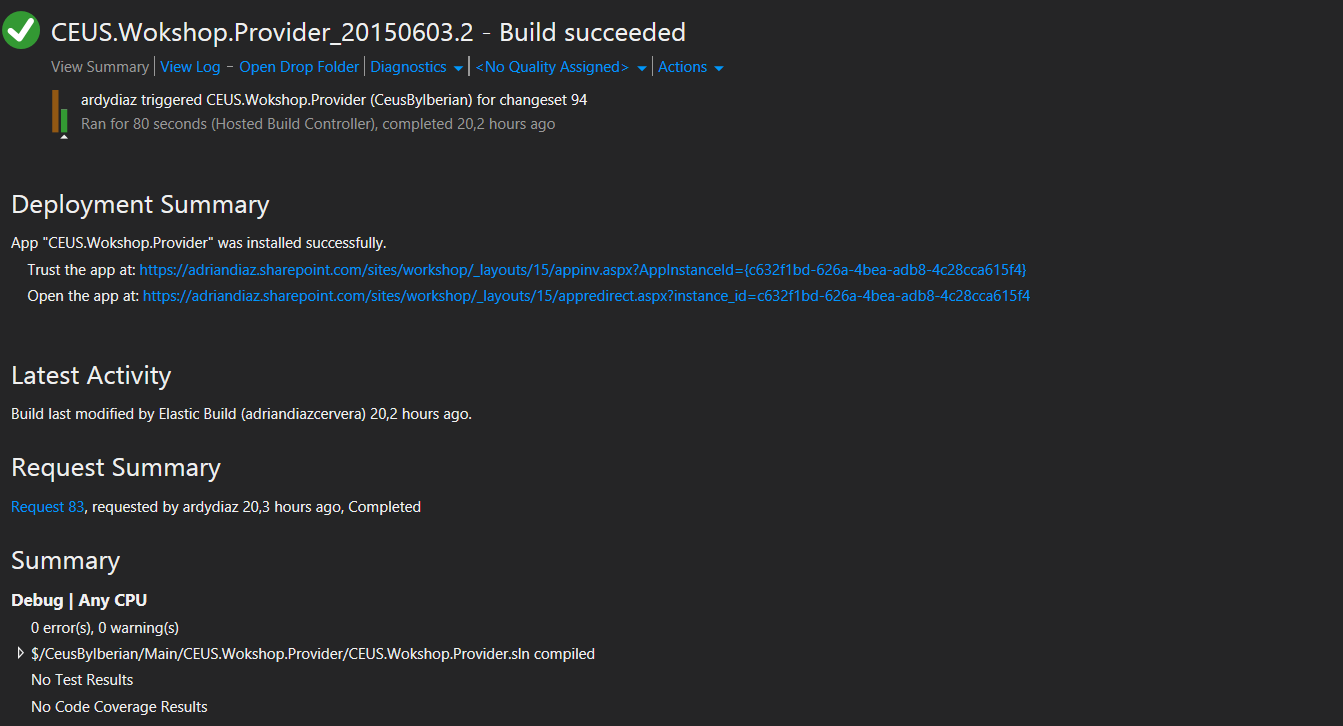
**Process:** En esta opción seleccionamos el Template que vamos a utilizar para esta Build, para realizar la integración continua con las APP seleccionaremos el template que nos hemos descargado de CodePlex (más adelante veremos cómo personalizar este template)



Dentro de los parámetros de este proceso de Build tendremos que añadir lo siguiente:

1. Advanced-> MSBuild Arguments: /p:ActivePublishProfile="perfil de Azure"
2. Misc-> Deployment Script Ruta donde hemos dejado el fichero SharePointAppDeloy.ps1 (descargado previamente de Codeplex)



A continuación ejecutamos la build de forma manual y si todo ha ido bien se muestra una pantalla como la siguiente: 

**NOTA:** En caso de utilizar TFS OnPremises es necesario instalar una serie de PreRequisitos en el servidor de la Build para poder empezar con ella.

# Resumen del laboratorio

En el anterior laboratorio fue posible ejercitarse en:

* Configurar la Build.
* Implementar Integración Continua en Visual Studio Online

# Laboratorio 2 Personalizar el Proceso de la BUILD

**Requisitos**

**Descargar SPCAF pre compilado desde la siguiente dirección** [**https://visualstudiogallery.msdn.microsoft.com/6273311f-d947-4179-be03-fa6a6f9ffa63**](https://visualstudiogallery.msdn.microsoft.com/6273311f-d947-4179-be03-fa6a6f9ffa63)

El personalizar el proceso de la Build es algo de lo más corriente y más necesarios una vez establecemos Integración Continua. Ejemplos hay muchos:

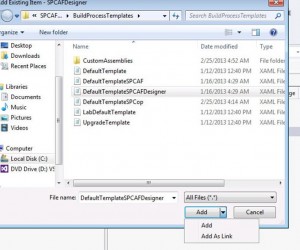
* Cuando compilo quiero pasarle una verificación del código de acuerdo a unos patrones establecidos.
* Unificar los ficheros JavaScript
* Compilar ficheros de diseño
* …

En este laboratorio vamos a modificar el template que utilizamos en el anterior laboratorio para poder incorporar SPCAF dentro del proceso de la BUILD.

#### Añadir una actividad de la Build con Designer es Visual Studio

Si no desea añadir la actividad de construcción para la plantilla de proceso de XML se puede utilizar un diseñador visual en Visual Studio.

1. Haz una copia del Prorcces Template e.g. DefaultTemplate.xaml in TFS como nombre para esta instancia DefaultTemplateSPCAF.xaml
2. Creamos un nuevo proyecto de Ttipo Librería de Clases en Visual Studio y eliminamos la clase creada por defecto
3. Añadimos el Templates ‘DefaultTemplateSPCAF.xaml’ to the project via “Add Existing Item” and abrimos la carpeta donde esta DefaultTemplateSPCAF.xaml. Important: Seleccionamos “Add As Link” en el dialogo yselecciaionamo “add only a link to the file.”

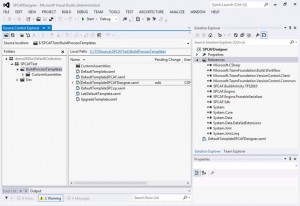
[](http://joost.haneveer.nl/wp-content/uploads/2013/12/120613_1318_HowtorunSPC3.jpg)

1. Añadimos las siguientes referencias al proyecto:

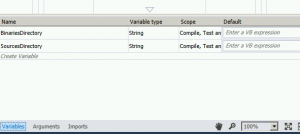
* Microsoft.TeamFoundation.Build.Workflow.dll
* Microsoft.TeamFoundation.VersionControl.Client.dll
* Microsoft.TeamFoundation.VersionControl.Common.dll

1. Añadimos la referencia SPCAF.BuildActivity.TFS2012

El proyecto de Visual Studio quedará de la siguiente forma

[](http://joost.haneveer.nl/wp-content/uploads/2013/12/120613_1318_HowtorunSPC4.jpg)

1. Abre el Proceso de Build (Xaml) en el diseñador.
2. Añade la Actividad SPCAF en la BUILD desde el ToolBoox Botón Derecho en el Toolbox y seleccionaar “Choose items…”. En la pestaña System.Activites Components select “Browse…” y seleccionala la librería SPCAF.BuildActivity.TFS2012. The Build Activity appear in the toolbox.
3. Debido a que en TFS 2013 la plantilla de proceso de construcción por defecto es un poco diferente tenemos que añadir una variable para almacenar la ruta de acceso al directorio de binarios. Abra la pestaña "Variables", añadir una variable llamada "BinariesDirectory" con el tipo "String", puede dejar el alcance en su configuración por defecto.

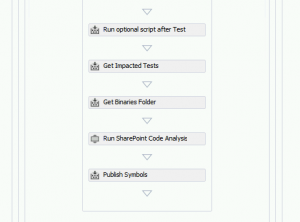
[](http://joost.haneveer.nl/wp-content/uploads/2013/12/120613_1318_HowtorunSPC5.png)

1. En el template ir aCompile, Test and Publish and open the try block. Arrastra la Actividad GetEnvironmentVariable<T> debajo de “Get Impacted Tests”-activity. Establce los argumentos de la actividad de acuerdo a la siguiente tabla :

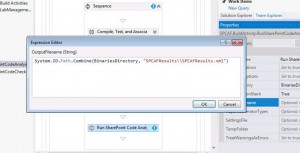
|  |  |
| --- | --- |
| **Argument** | **Value** |
| Name | Microsoft.TeamFoundation.Build.Activities.Extensions.WellKnownEnvironmentVariables.BinariesDirectory |
| DisplayName | Get Binaries Folder |
| Result | BinariesDirectory |

1. Arrastra la Actividad SPCAF justo debado de la Actividad Get Binaries Folder.

Deberias de ver lo siguiente:

[](http://joost.haneveer.nl/wp-content/uploads/2013/12/120613_1318_HowtorunSPC6.png)

1. Configura la Actividad de SPCAF.

[](http://joost.haneveer.nl/wp-content/uploads/2013/12/120613_1318_HowtorunSPC7.jpg)

1. Salvamos el proceso como Template

### Parameters of Build Activity

The following parameters can be configured for the build activity.

|  |  |  |
| --- | --- | --- |
| **Argument** | **Description** | **Sample Value** |
| DisplayName | Required String. Title of activity, used in logfiles. | “Run SharePoint Code Analysis SPCAF” |
| SettingsFile | Optional String. Name of settings file in folder “/RuleSets” or full path to a different ruleset file. |  |
| TempFolder | Optional String. Path to temp folder. If not set SPCAF uses a standard temp folder. |  |
| FailBuildOnError | Optional Boolean. Default FALSE. If TRUE the build fails if SPCAF detects errors. | FALSE |
| IgnoreExceptions | Optional Boolean. Default TRUE. TRUE to ignore exceptions during analysis. Settings this value helps to ensure that the build finishes successfully also in case of exceptions. | TRUE |
| InputDirectory | Required String. Path to WSP solutions. Can be a folder name or a list of files (separated by ‘;’). | BinariesDirectory |
| LogExceptionStack | Optional Boolean. Default FALSE. In case of problems with SPCAF setting this value to TRUE adds more information to the logfile which describes the error in detail. | FALSE |
| OutputFilename | Optional String. Defines the ouptput filename of the reports. Typically the output folder should be in the BinariesDirectory. After successful build the whole BinariesDirectory is copied to the DropLocation. Alternatively the DropLocation or any other folder can be used to save the results. | System.IO.Path.Combine(BinariesDirectory, “SPCAFResults\\SPCAFResults.xml”) |
| LogFile | Optional String. Defines the path to the log file. | System.IO.Path.Combine(BinariesDirectory, “SPCAFResults\\spcaf.log”) |
| ReportGeneratorTypes | Optional String. Defines which reports should be created | “HTML;XML;DGML;DOCX” (DOCX is supported since version 4.5.2) |
| TreatWarningsAsErrors | Optional Boolean. Default FALSE. If TRUE all warnings will be evaluated as errors. | FALSE |
| ThrowExceptionOnError | Optional Boolean. Default FALSE. If TRUE an exception will be throw in case of critical errors or errors. | FALSE (Available since version 4.5.2) |
| Verbosity | Optional String. Detail level of output (e.g. minimal, default: normal). | normal |

The build activity returns the following values after analysis. (Supported since version 4.5.2)

|  |  |
| --- | --- |
| **Argument** | **Description** |
| CriticalErrors | Integer. Contains the number of critical errors after analysis. |
| Errors | Integer. Contains the number of errors after analysis. |
| CriticalWarnings | Integer. Contains the number of critical warnings after analysis. |
| Warnings | Integer. Contains the number of warnings after analysis. |
| Information | Integer. Contains the number of information after analysis. |