13219, 16410, 16693, 17177

USING VISUAL STUDIO 2012 LAB MANAGER TO CREATE A BUILD LAB IN THE SKY

JULY 20, 2012 | MATHIAS OLAUSSON | 1 COMMENT

Since TFS first came out it has been easy to create an automated build solution based on the TFS services. With Visual Studio 2010 it also became easier to create an on-premises continuous deployment solution using the new Lab Management component in TFS.

With Visual Studio 2012 there are several improvments which further improves the build-deploy-test scenarios together with the introduction of hosted TFS services for those interested in a cloud based ALM solution.

Recently Microsoft announced a very sleek solution to enable continuous integration to Azure using the hosted Team Foundation Service. Mike Fourie has written about Using Continuous Deployment to Azure with Team Foundation Service that is a great start if you are interested in Azure deployment.

But what about projects not using Azure? It sure sounds nice to have the option to host the entire development infrastructure in the cloud. So in this post I want to show how we can use the new features in Visual Studio 2012 to build a complete continuous deployment solution using Visual Studio 2012 and cloud services.

The Design

The image below illustrates the environment I would like to create to support a continuous deployment scenario in the sky for a generic application:

- The Team Foundation Service (tfspreview.com) is used to manage code and build assets.
- Windows Azure is used to host virtual machines that we compose to a test lab.
- Microsoft Test Manager (MTM) is used to create lab environments and to run tests on the lab environment in Azure.
- The Hosted Build Service runs the builds, including the lab management builds.
- As part of the lab management build process the hosted build controller will deploy a build to the test lab and run automated tests on the lab.

So the questions now is how much work is needed to get this up and running? Let's go and implement the core components of this vision

and see for ourselves!

Configure Team Foundation Service

The Team Foundation Service is TFS in the cloud. It's currently available for free in a preview form but the service has been running for a while now and is very stable. Go to www.tfspreview.com to read more about the service and to create an account.

Since the Team Foundation Service is a hosted service the configuration we need to make is minimal for our scenario, just create an account, create a project and add the code.

Configuring a Build Service

A build service is currenty an integrated part of every Team Foundation Service account. This may of course change once the service is commercially available but for now it's there for free.

Brian Harry has a great post that describes how to setup and use the hosted build service here: http://blogs.msdn.com/b/bharry/archive/2012/03/27/announcing-a-build-service-for-team-foundation-service.aspx.

This means we don't have to do anything to get builds working with the Team Foundation Service.

Create Lab Machines in Windows Azure

So with the TFS infrastructure solved we now need to create a lab machine (or more) to run our tests on. For this example I created a simple Windows Azure worker role and enabled remote desktop in order to be able to log on to the machine to configure it and to install additional software as needed.

For information on how to create a Windows Azure machine and to configure it for RDP the following resources can be useful:

- Creating a Hosted Service for Windows Azure
- Using Remote Desktop with Windows Azure Roles

Configuring a Test Controller

The Team Foundation Service comes with an integrated build service but unfortunately not a test controller. We need a test controller to be able to setup a Visual Studio 2012 Lab Environment and also to control the test runs in the remote environment.

We can install the test controller wherever we want as long as it can access TFS and the lab machines. A simple solution for this scenario is to install the Test Controller on the lab machine itself. I used RDP to connect to the VM and then downloaded and installed the Test Agent directly on the Azure machine.

Note: It's generally not recommended to install the Test Controller inside a Lab Management environment so you should setup a

dedicated machine for this purpose.

The setup is simple but in order to use it with the Team Foundation Service we of course need to connect it the the TFS collection. But we also need to authenticate using an account that has been setup to access the TFS service, therefore I've used my local account from the Azure VM to both run the test controller service and as the lab service account:

Create a Visual Studio 2012 Standard Lab Environment

Visual Studio 2012 has a great enhancement to the way we create Lab Management envioronments; it is not required to install the System Center Virtual Machine Manager to create a lab instead we can build one from any physical or virtual machine. So in this case we will create a lab environment from the Azure VM created earlier.

With the test controller configured as described in the previous step we can now use MTM and create a new environment. The most interesting parts for our scenario is the Machines tab where we specify the Azure machines (getting the hostname for the VM is probably the most challenging part) and also an account on the VM that has permissions to configure the environment.

The next interesting configuration is the Advanced tab. Here we need to specify the test controller the environment should be connected to. We can also configure the environment to run UI tests and if so provide the account the test agent should be run under.

Completing the wizard will then start the environment creation process which can take some time. The most time consuming part is when the Visual Studio Test Agent gets installed but after a while the machine should be in a ready state:

Take a look at the following guide if you want more information on how to setup a Lab Management environment with Azure machines: How to get VS11 Lab Management to work with Azure VM Roles and Azure Connect.

Create a Build

Next we need to create a TFS build to compile a version of our project that we can deploy to our test lab and use for testing. Setting up a TFS build to create a deployment package can be very simple or quite a challenge depending on what kind of application we have.

We will just assume we have a TFS build called **ExpenseIT Dev** that

creates a deployment package for us. The build has only one different thing in the hosted build scenario and ther is where the build is dropped. With the hosted build service we can choose to drop the build in TFS, something that makes sense in a cloud scenario where we don't have access to a file share.

lab wf build default

If you want to learn more about creating automated builds with TFS 2012 then The Visual Studio ALM Rangers TFS Build Customization Guide is a great resource.

Create a Test Suite for Automated Testing

Continuous Deployment requires more than just an automated build and deployment process to work. We also need a way to verify that the deployment actually works and what better than to do so by using automated tests built into the process?!

We can implement these build verification tests (BVT) using any automated tests. We can then add the tests to a test suite in Microsoft Test Manager and use that test suite later in the automated deployment workflow, the image below shows how we have created a test suite called **ExpenseIT**:

image

If we open the test case we can see that it is an automated test that is associated with the **CreateStandardExpenseReport** method in the **ExpenseIT.UITests.dll** assembly:

image

Create a Build-Deploy-Test Build Workflow

Now we are ready to do what we were aiming to do; create a continuous deployment solution using cloud services. Visual Studio 2012 Lab Management has a build template to setup a build-deploy-test workflow in a simple way using the

First we create a new build definition using the **LabDefaultTemplate11.xaml** process template.

image

On the Process tab we just select the **LabDefaultTemplace.11.xaml** build process template:

image

Next we edit the Lab Process Settings using the Lab Workflow wizard. The first step is to select the **Azure Lab** environment we created for this scenario:

SNAGHTML3407e85c

Next we specify which build to deploy. We use the **ExpenseIT Dev** build and use the **<Latest>** available build.

Note: we cannot Queue a new build option together with the hosted build service because currently we are only assigned one build agent per account. The logic in the LabDefaultTemplate11.xaml build template is to queue a new build and wait for it to complete before running the deploy and test stages. With only one agent this workflow will be blocked and eventually timeout.

SNAGHTML3407cd7c

In the Deploy tab we get to add scripts to run locally on each machine in the environment:

SNAGHTML3407b0b9

In our case the deployment is very simple. just an xcopy of the build result and we're done. So the deploy.cmd script contains the logic to clean the existing deployment folder and then xcopy over the build result:

```
SET DeploymentPath=%2
RD %DeploymentPath% /S /Q
XCOPY %1 %DeploymentPath% /S /Y /I
```

Finally we select a test suite containing the automated tests to run on the environment, in our case the **ExpenseIT** suite in the **ExpenseIT** test plan:

SNAGHTML3407928f

That's it! We have now set up a complete build-deploy-test workflow using only hosted services!

To test the BDT workflow we just make sure we have a build to deploy (the **ExpenseIT Dev** build) and then queue a new build.

If this had been setup on-premises with a build drop on a file share the BDT process could access all would have worked just fine. But having the drop folder in TFS unfortunately causes the deploy and test steps to fail:

image

As we can see in the build report the application sees the drop folder from the build to deploy in TFS but the activities in the lab workflow doesn't know how to handle that and instead treats it as a local folder.

So to fix this we need to customize the BDT workflow to download the build result so the workflow can find it and use it as expected.

Customizing the BDT Workflow

To customize the default BDT workflow we just make a copy of it and

open up the workflow in Visual Studio. If you want to learn how to customize a build template see the TFS Build Customization Guide.

First we will change the deploy and test activites to use a configurable location to find the build result. Then we can extend the deployment script to first download the build result to TFS before performing the deployment tasks.

We add a custom argument to the process to be able to specify the LabDropFolder from the build definition:

To show the configuration argument in a proper way in the build definition we add it as metadata in the process parameter metadata editor:

Next we find the "Run Deployment Task" activity and change the BuildLocation to the LabDropFolder:

And then the "Running Tests" activity and set the TestDirectory to the LabDropFolder:

Now we checkin the customized lab build template to TFS and update our deployment build definition to use the new template including specifying a Lab drop folder (on the Azure VM in this case):

With the build definition using the new build template all that remains is to update the deployment script to download the build result from TFS to a folder on the lab machine. The following script does just this. First we clean the target folders, then create a temporary TFS workspace and download the build result there. Finally the build result files are copied over to the deployment folder and the workspace is removed.

```
Echo off
REM —
SET DeploymentPath=%2
RD %DeploymentPath% /S /Q
RD c:\temp\drop /S /Q
CD c:\Temp
MD drop
tf workspace /new Temp
/collection:https://alm.tfspreview.com/defaultcollection /noprompt
tf get %1 /recursive
REM — Deploy App
XXCOPY drop %DeploymentPath% /S /Y /I
tf workspace /delete Temp /noprompt
```

With this we can now queue a new build and this time the entire build-deploy-test workflow will succeed since the deploy and test activities in the workflow are now using a local directory for the build result:

image

That's it! You have now seen how you can use Visual Studio 2012 with the hosted TFS and Build Service together with Windows Azure VMs to create a complete build lab in the sky.

PREVIOUS POST
Book Project "Pro Application Lifecycle Management with Visual Studio 2012" Completed!

NEXT POST
Final call for Dev Days 2012! (SWE)

ONE THOUGHT ON "USING VISUAL STUDIO 2012 LAB MANAGER TO CREATE A BUILD LAB IN THE SKY"

Danny Crone
JULY 31, 2012 AT 11:53 AM
Mathias, this is a truly awesome blog, great vision and well executed.

REPLY

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Comment

You may use these HTML tags and attributes: <a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <i> <q cite=""> <strike>

POST COMMENT

Search …

RECENT POSTS

- Brian Keller's Visual Studio 2013 ALM VM Updated!
- Automatically update test plan with build number
- Presentation till "Har du en DevOps i ditt team?" (SWE)
- Extending TFS Work Item using a web page and the WebpageControl Revisited
- TechDays 2014 (SWE) – Från kod till produktion på 60 minuter

RECENT COMMENTS

- Nicolas on WitCustomControls for Visual Studio 2013 released!
- Mathias Olausson on Extending TFS Work Item using a web page and the WebpageControl
- Rafael Oliveira on Extending TFS Work Item using a web page and the WebpageControl
- Mathias Olausson on Extending TFS Work Item using a web page and the WebpageControl
- Gene on Extending TFS Work Item using a web page and the WebpageControl

ARCHIVES

- September 2014
- August 2014
- July 2014
- June 2014
- May 2014
- February 2014
- November 2013

October 2013

September 2013

July 2013

June 2013

May 2013

April 2013

February 2013

December 2012

October 2012

September 2012

July 2012

June 2012

May 2012

April 2012

March 2012

January 2012

December 2011

November 2011

October 2011

September 2011

August 2011

July 2011

June 2011

May 2011

April 2011

March 2011

December 2010

October 2010

September 2010

August 2010

July 2010

March 2010

February 2010

January 2010

December 2009

November 2009

October 2009

September 2009

August 2009

CATEGORIES

11401

11402

11483

11560

11561

11925

12010

13219

16373

16409

16410

16692

16693

16891

16965

16978

17177

17316

17691

17841

17924

18112

18113

18132

Uncategorized

META

Log in

Entries RSS

Comments RSS

WordPress.org