Hands-On Lab

# Introduction to Team Foundation Build 2015 (vNext), Test Run Analysis and Machines

Lab version: 14.0.25123.0

Last updated: 5/12/2016

* + 1. 

**TABLE OF CONTENT**

[Introduction to Team Foundation Build 2015 (vNext), Test Run Analysis and Machines 1](#_Toc450736183)

[Overview 3](#_Toc450736184)

[Prerequisites 3](#_Toc450736185)

[Exercises 3](#_Toc450736186)

[Exercise 1: Build Agent Pools and Queues 4](#_Toc450736187)

[Task 1: Build Hub in TFS Web Access 4](#_Toc450736188)

[Task 2: Creating an Agent Pool 5](#_Toc450736189)

[Task 3: Creating a Build Queue 7](#_Toc450736190)

[Task 4: Installing and Configuring an Agent 10](#_Toc450736191)

[Exercise 2: Build Definitions 12](#_Toc450736192)

[Task 1: Creating a Basic Build Definition from Template 13](#_Toc450736193)

[Task 2: Queuing and Executing a Build 25](#_Toc450736194)

[Exercise 3: Build and Test Result Analysis 26](#_Toc450736195)

[Task 1: Analyzing Build and Test Results 26](#_Toc450736196)

[Exercise 4: Continuous Integration and Deployment 29](#_Toc450736197)

[Task 1: Cloning a Build Definition 29](#_Toc450736198)

[Task 2: Adding a Deployment Step and Defining Machine Group 30](#_Toc450736199)

[Task 3: Configuring Continuous Integration 32](#_Toc450736200)

[Task 4: Triggering a Continuous Integration Build 34](#_Toc450736201)

[Exercise 4: Scale out Testing using Machine Groups 36](#_Toc450736202)

[Task 1: Modifying Build to use External Test Agents 36](#_Toc450736203)

[Task 2: Testing and Saving Modified Build Draft 41](#_Toc450736204)

## Overview

* 1. In this lab, you will learn how to use the new Team Foundation Build in order to build, test, and deploy your applications. This new scriptable build system is both web-based and cross-platform, and Microsoft recommends using it instead of the XAML build system whenever feasible. Although we won’t demonstrate all of the cross-platform possibilities in this lab, it is important to point out that you can also build for iOS, Android, Java (using Ant, Maven, or Gradle), and Linux.

### Prerequisites

* 1. In order to complete this lab, you will need the Visual Studio 2015 virtual machine provided by Microsoft. For more information on acquiring and using this virtual machine, please see [this blog post](http://aka.ms/almvm).

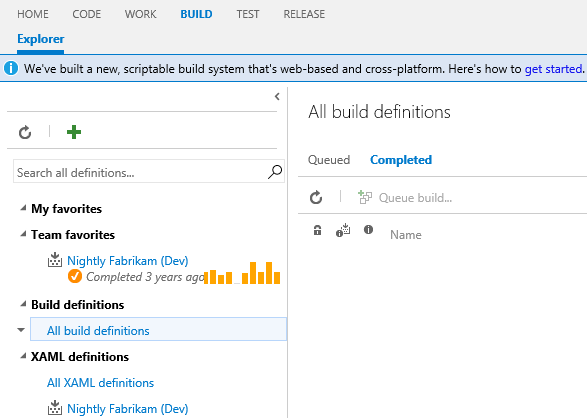
### Exercises

* 1. This Hands-On Lab comprises the following exercises:
  2. Build Agent Pools and Queues
  3. Build Definitions
  4. Build and Test Result Analysis
  5. Continuous Integration and Deployment
  6. Scale out Testing Using Machine Groups
  7. Estimated time to complete this lab: **60 minutes**.

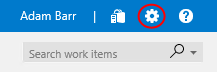
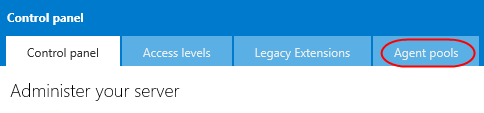
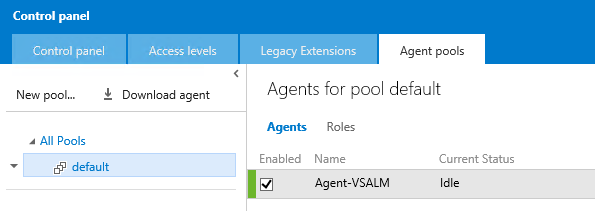
## Exercise 1: Build Agent Pools and Queues

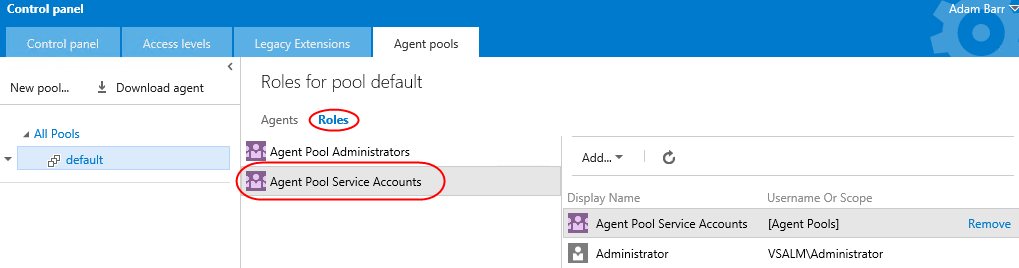
* 1. In this exercise, you will learn how to create and configure build agent pools and queues in order to support the new agents in Team Foundation Build 2015 (formerly referred to as Build vNext). This new scriptable build system is web-based and cross-platform, and is recommended for all new and existing builds going forward.

#### Task 1: Build Hub in TFS Web Access

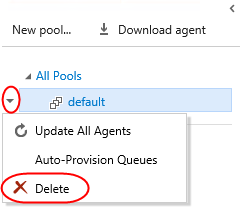
* 1. Log in as **Adam Barr** (VSALM\Adam). All user passwords are **P2ssw0rd**.
  2. Let’s say that the Fabrikam Fiber team has been building their web applications using XAML build system, but that they are ready to move to the new scriptable TFS build system that is available in 2015. Since they have been relying on a standard build template, the transition to the new build system should be relatively straightforward.
  3. Let’s get started by touring the **Build** hub in the web portal. Launch **Internet Explorer** from the taskbar and click **TFS FF Portal** from the favorites bar at the top.
     1. 
  4. Select the **Build** tab.
     1. 
  5. In the **Explorer** view, the definitions for the new build system are shown under the Build Definitions heading, while the XAML-based definitions are now shown under XAML Definitions heading. Here you can see the existing Nightly Fabrikam (Dev) XAML build definition that is currently used by the team to validate the development branch.
     1. 
  6. We’ll return to this view shortly once we are ready to translate the existing XAML build to the new system, but first we need to configure the infrastructure necessary for the new build system.

#### Task 2: Creating an Agent Pool

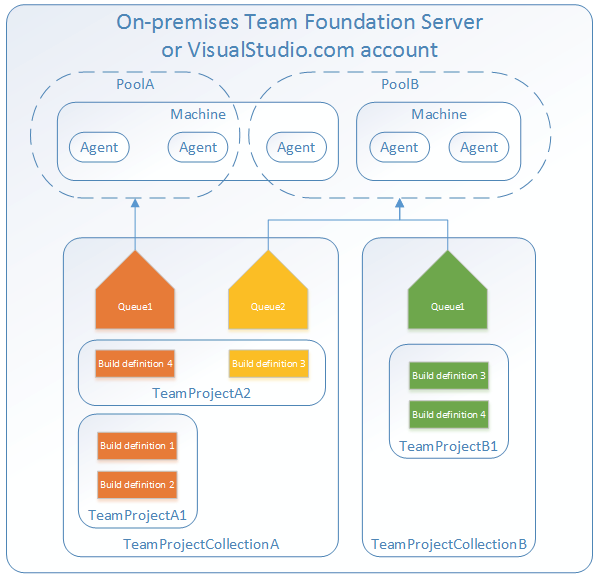
* 1. The first thing that we need to do is to setup an agent pool at the TFS account level. Agent pools are setup at this level so that they can be leveraged by multiple team project collections. This pool can contain both Windows and cross-platform agents.
  2. Click the **gear** icon in the top-right corner to open a tab to the administrative section of the portal.
     1. 
  3. Click the **Control Panel** root in the navigation breadcrumb at the top of the screen.
     1. 
  4. Click the **Agent Pools** tab.
     1. 
  5. Note that there is already an agent pool named “**default**” with a single agent shown here.
     1. 
  6. Select the **Roles** tab to view pool security. This view allows account-level **Agent Pool Administrators** to create new agent pools and even adjust the permissions on each agent pool individually.

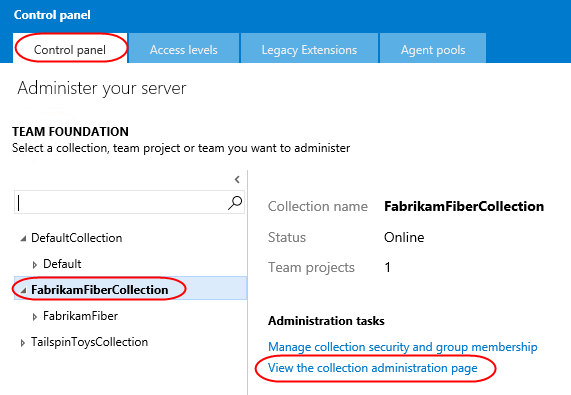


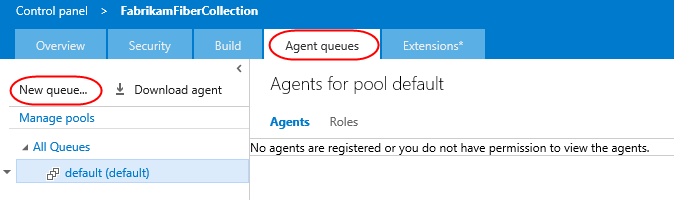
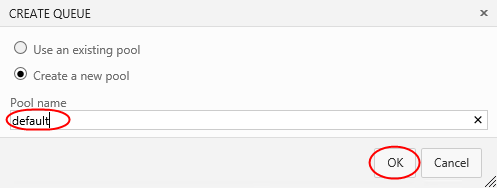
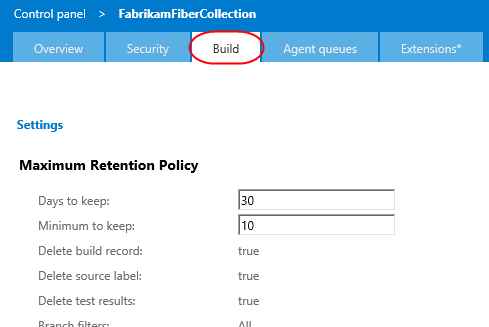
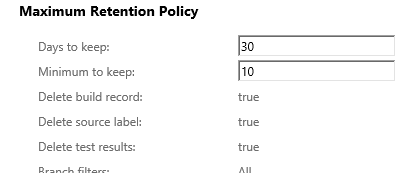
* 1. Select the **default** pool’s dropdown and select **Delete** to delete the pool and its agents. Confirm the delete when asked. You’ll go through the process of adding these back later on.



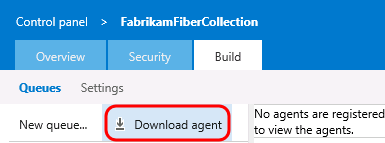
#### Task 3: Creating a Build Queue

* 1. Before we continue with the installation of an agent, let’s also ensure that we set up our team project collection with a build **queue** that points to the default agent pool. Since queues are scoped to your team project collection, you can share them across build definitions and team projects.
  2. This diagram from the MSDN documentation helps to illustrate the relationship between pools, queues, team project collections, and build definitions. Note that you can also install multiple agents on a single machine.
     1. 
  3. To navigate to the collection administration page for **FabrikamFiberCollection**, select the **Control panel** tab and locate the **FabrikamFiberCollection** node in the tree view. Click **View the collection administration page**.



* 1. Select the **Agent queues** tab and click **New queue**.
     1. 
  2. Enter a **pool name** of “**default**” and click **OK**.
     1. 
  3. It is also possible to configure collection-level build settings here. Click the **Build** tab to see them.
     1. 
  4. From here, you can specify the default and maximum settings for how long the system retains completed builds. The default retention policy is set at 10 days, with the maximum at 30 days. This means that regardless of what is set on the individual build definition all builds that have not been marked to "Retain indefinitely" will be deleted 30 days after they complete.
     1. 

#### Task 4: Installing and Configuring an Agent

* 1. Return to the **Agent queues** tab.
  2. Now we are ready to install a build agent, so click **Download Agent**.
     1. 
     2. **Note:** Prerequisites for the machine deploying the Windows build agent include Visual Studio 2013 (or newer) as well as PowerShell 3 (or newer).
  3. Notice that the agent installation process involves downloading a zip. Since we previously had this downloaded and unzipped to **c:\agent**, click **Cancel** to skip that part of the install.

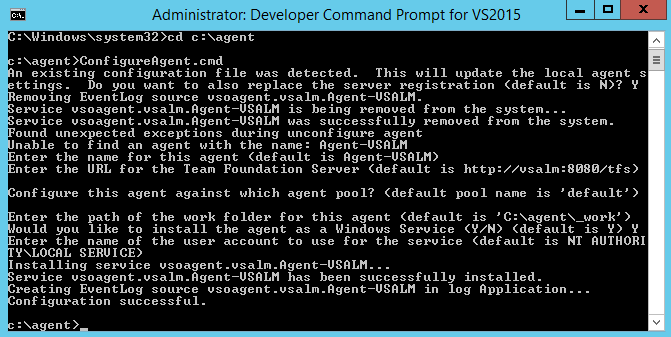
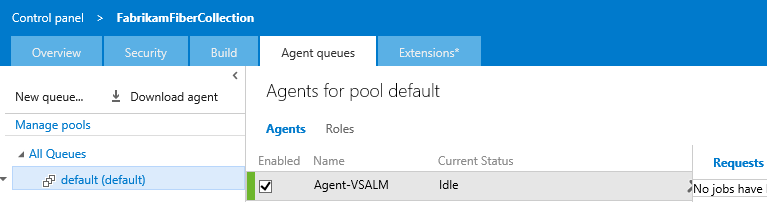
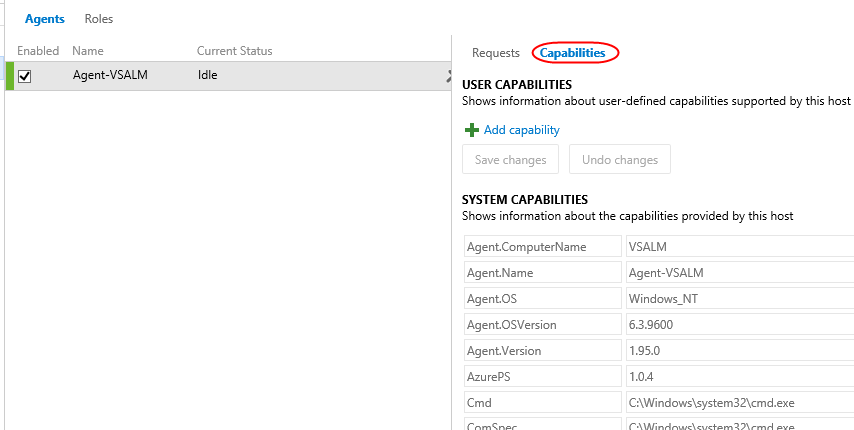


* 1. Launch an instance of **Command Prompt as Administrator**. One way to do this is to hold down **Ctrl+Shift** and then click the **Command Prompt** icon in the taskbar.
  2. Change to the unzipped agent directory.

cd c:\agent

* 1. Execute the agent configuration script.

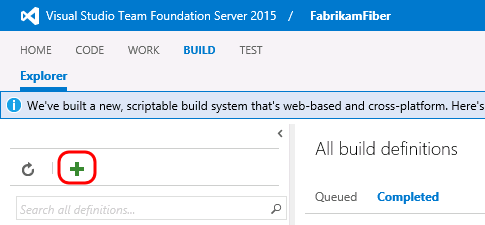
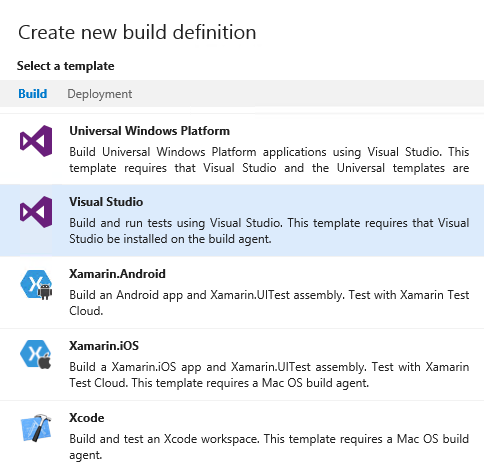
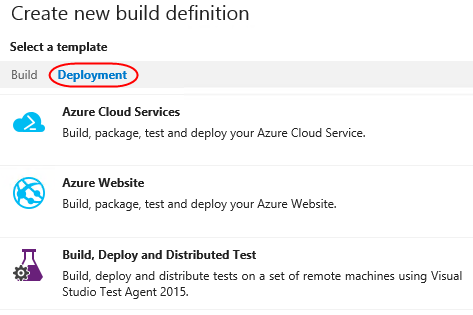
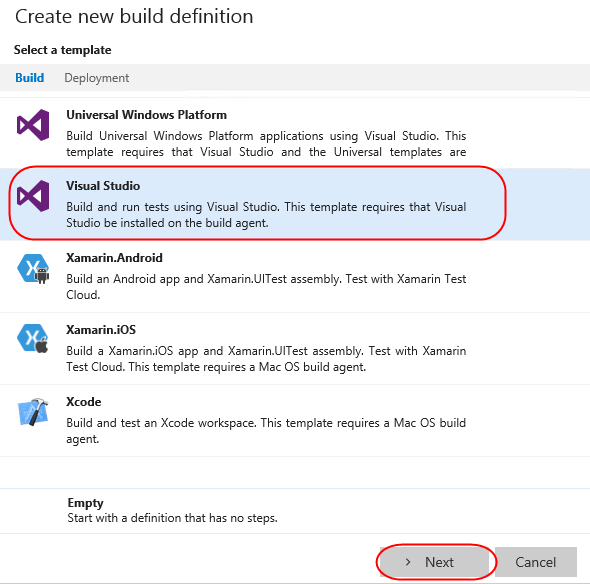
ConfigureAgent.cmd

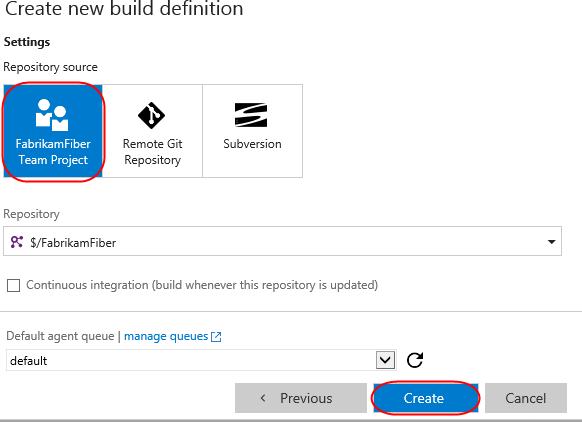
* 1. Since you are running this configuration over an existing configuration (the one deleted earlier), you will be asked whether you want to replace the registration. Type **“Y”** and press **Enter**.
  2. Press **Enter** to use the default agent name of “Agent-VSALM”.
  3. Press **Enter** to use the default URL for TFS, which in this case is <http://vsalm:8080/tfs>. Note that this is the “default” because it was the URL used in the previous configuration of the agent. When installing for the first time, the default will be different.
  4. Press **Enter** to use the default agent pool of “default”.
  5. Press **Enter** to use the default path proposed for the agent work folder “c:\agent\\_work”.
  6. When asked if you want to install as a Windows Service, type **“Y”** and then press **Enter**. Note that you could also configure the agent to run in interactive mode, which you may want to do if you were planning to execute coded UI tests.
  7. Press **Enter** to run as local service, rather than providing a specific user account.
  8. After a few moments, the script should complete with the successful installation and configuration of the new agent.
     1. 
     2. **Note:** You weren’t prompted for credentials in this case, but under normal circumstances when installing on a remote machine you would be asked to sign in as an agent pool administrator. These credentials are only used once during the configuration process.
  9. Return to Internet Explorer and refresh the page to verify that the agent is shown as being enabled and ready.
     1. 
  10. While we are here, select the **Capabilities** tab to take note of the System Capabilities list shown for the agent. System capabilities are name/value pairs that you can use to ensure that your build definition is only run by build agents that meet specified criteria. Environment variables automatically appear in the list. Some additional capabilities (such as .NET Frameworks) are also added automatically. You can also add your own capabilities to the list based on additional requirements for your builds. Later, when a build is queued, the system sends the job only to agents that have the capabilities demanded by the build definition.
      1. 

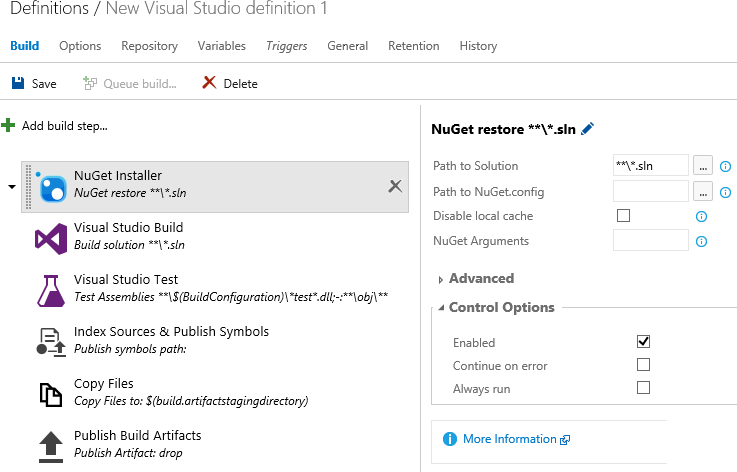
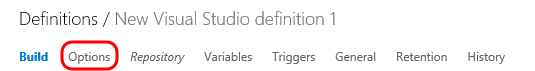
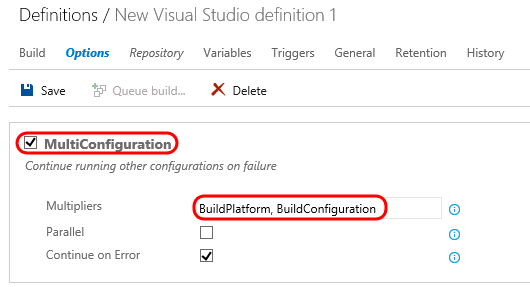
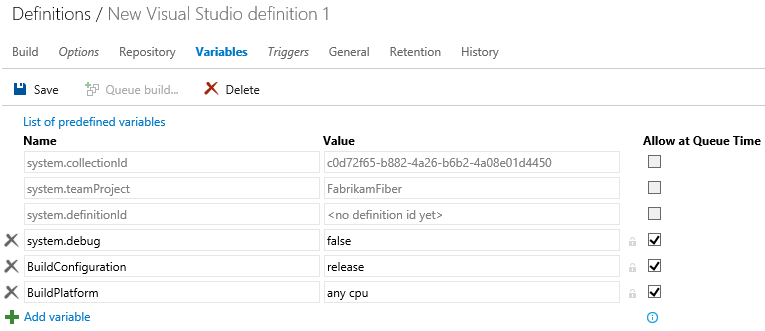
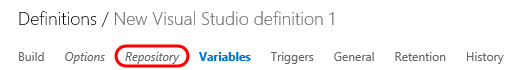
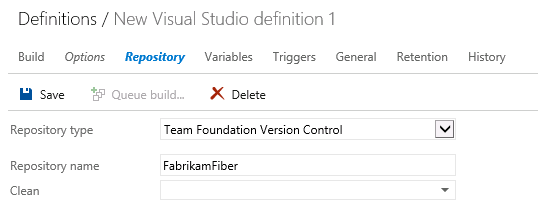
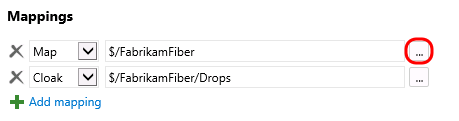
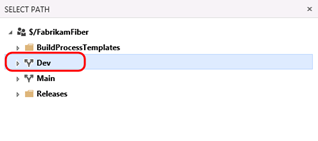
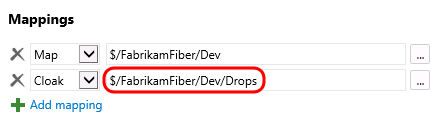
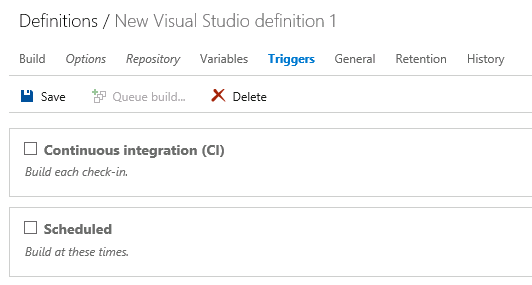
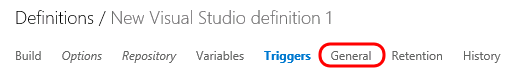
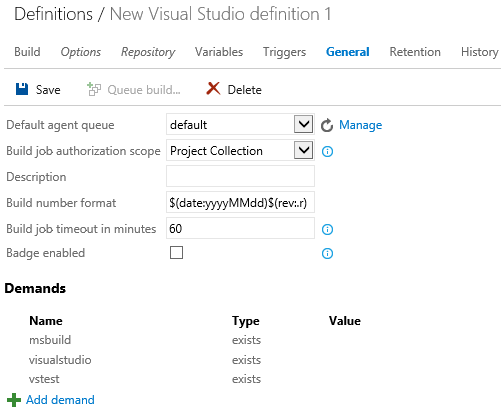
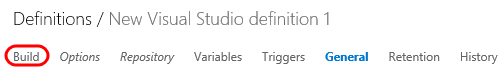
## Exercise 2: Build Definitions

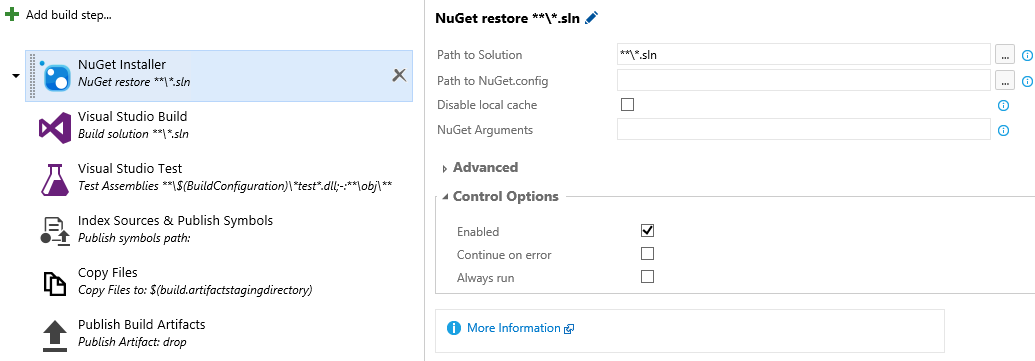
* 1. In this exercise, you will learn how to create a basic build definition from one of the provided templates and then queue the build for execution.

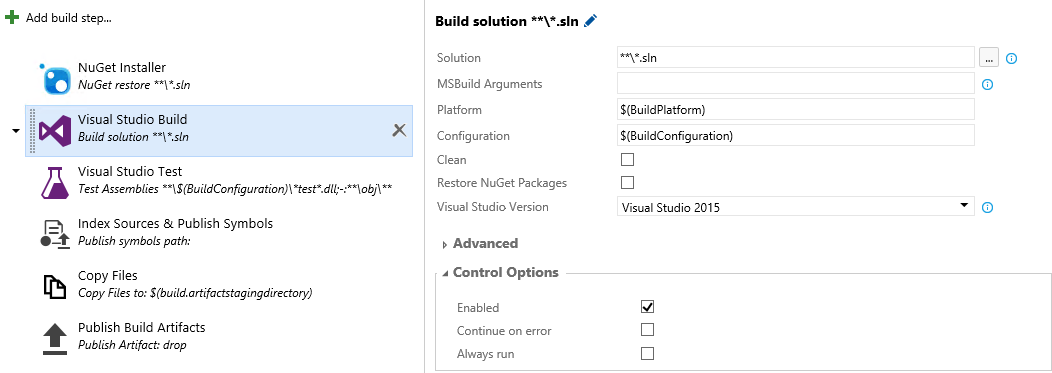
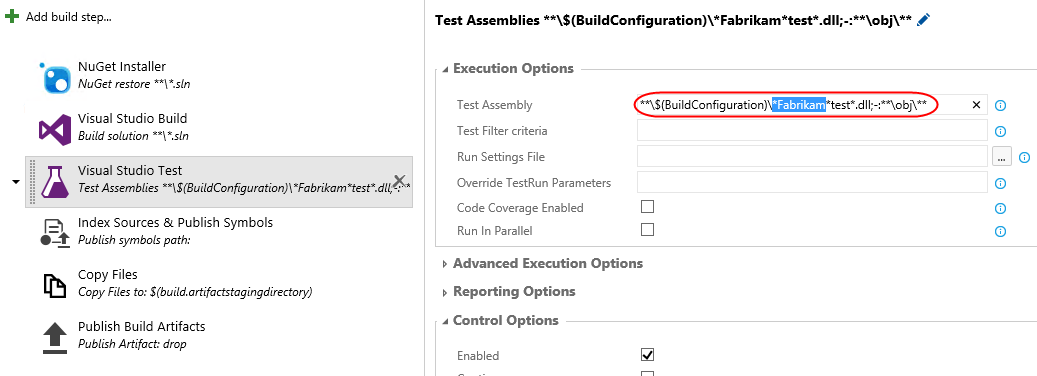
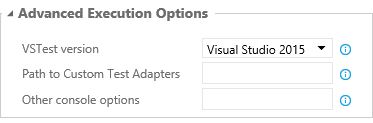
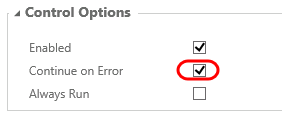
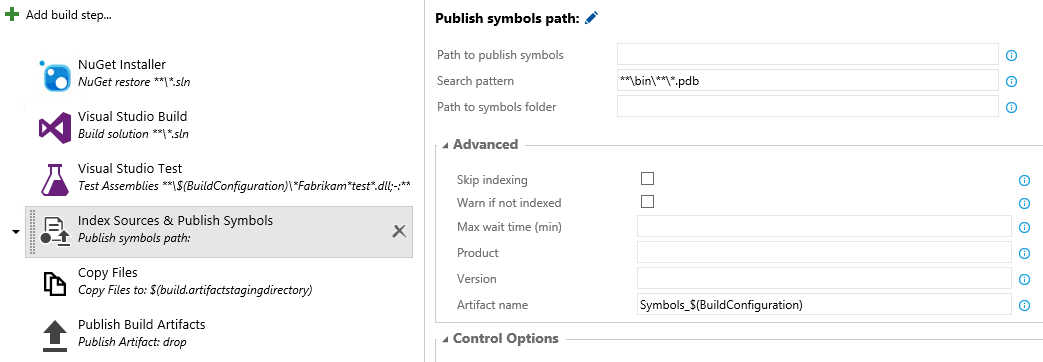
#### Task 1: Creating a Basic Build Definition from Template

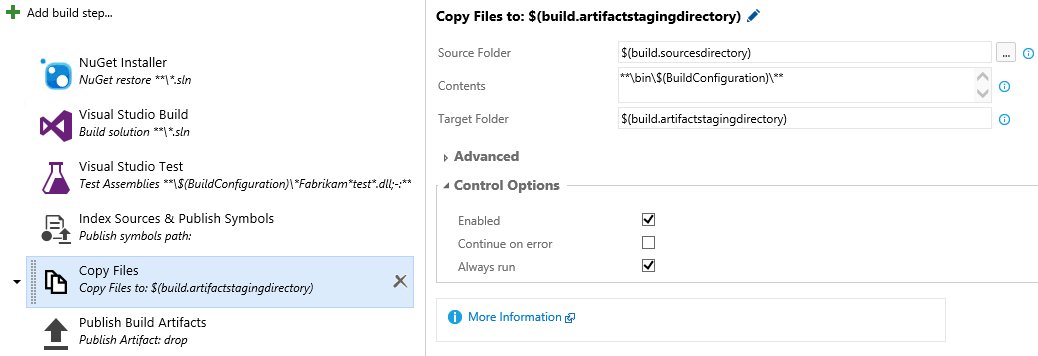
* 1. Now it is time to create a new build definition for the Fabrikam Fiber team. Close the administration browser tab to return to the browser tab open to the **Build** section of the FabrikamFiber project in the web portal.
  2. Click the **Actions** button (green plus).
     1. 
  3. The **Definition Templates** dialog lists a few different build templates that you can start with in order to build and test using Visual Studio, Xamarin, and Xcode. Alternatively, you can also simply start with an empty definition and add in the tasks that you need.
     1. 
  4. Click the **Deployment** tab and note that templates are also provided that help accelerate deployment to Azure.
     1. 
  5. Return to the **Build** tab, select the **Visual Studio** template, and then click **Next**.
     1. 
  6. Select **FabrikamFiber Team project** as the **Repository source** and click **Create** to create the build definition.

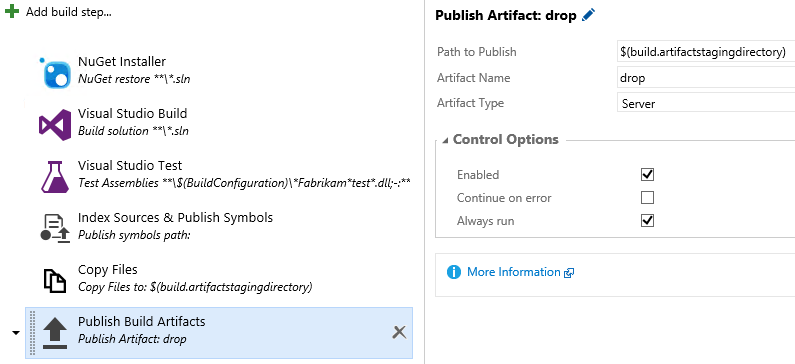
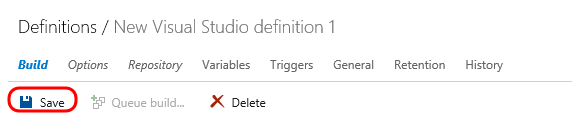
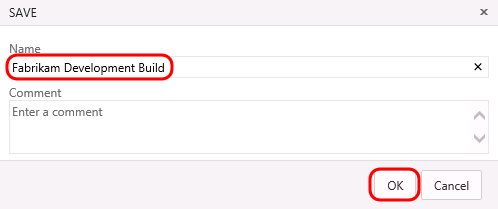


* 1. The new build definition is initially set up with build steps that initiate a NuGet restore, a Visual Studio build, a test pass, publication of symbols for archival, a copy of the files to a staging location, and finally a publish of the build output to a drop location. These steps are all defined on the **Build** tab.
     1. **Note:** The build engine and tasks are both extensible, and are designed to be cross platform. In the event that you need a task that isn’t offered out of the box, you can create your own using the open source activities found on GitHub [here](https://github.com/Microsoft/vso-agent-tasks). This is also a good location to dig into if you would like to learn more about how the existing tasks work.
     2. 
  2. Before we take a closer look at the configuration of the individual build steps, let’s take a look at a few of the key configuration options for the build definition itself, which are shown as tabs across the top. Click the **Options** tab.
     1. 
  3. The Options tab provides the MultiConfiguration option than can be enabled to build multiple configurations for multiple platforms. Select the **MultiConfiguration** option to enable it.
  4. For **Multipliers**, enter “**BuildPlatform, BuildConfiguration**” in order to take into account these variables during the build.
  5. Enabling the **Parallel** option would allow you to distribute the jobs (one from each combination of values from Multipliers) to multiple agents in parallel (if available). Since we only have one agent installed, however, we won’t be able to parallelize the build.
     1. 
  6. Click the **Variables** tab.
     1. 
  7. This list shows variables that will be available to all build steps (tasks). There are a number of predefined variables that can be used by tasks during the build, all listed [here](http://go.microsoft.com/fwlink/?LinkId=550988), as well as any additional variables that can be added in this view. By default, both **BuildConfiguration** and **BuildPlatform** are already defined here (we just referenced these in our configuration of Multipliers prior).
     1. 
  8. Although we will not do so for the purposes of this demo, you could modify BuildConfiguration to be “debug, release” in order to build both versions.
     1. **Note:** Using variables is a great way to specify secrets as well. If you were to add in a variable to contain a password, for example, you could click the Secret button just to the right of the Value column (lock icon) to prevent it from being displayed here.
  9. Click the **Repository** tab.
     1. 
  10. This is where you specify the source repository for build. In this case, **Team Foundation Version Control** is already selected for Repository Type and **FabrikamFiber** is selected as Repository Name. Of course, it is certainly possible to build from any Git repo as well.
      1. 
  11. You can also configure mappings in order to pull down just the subset of code necessary in order to build. Click the **ellipses** button to the right of the first default mapping currently configured as “$/FabrikamFiber”.
      1. 
  12. In this case, we would just like to build the Dev branch of the code, so select the **Dev** branch node and then click **OK**.
      1. 
  13. Modify the second default mapping manually (of type Cloak) to be **“$/FabrikamFiber/Dev/Drops**”.
      1. 
  14. Select the **Triggers** tab.
      1. 
  15. Although we will not use them in this definition, this is where you can configure continuous integration or schedule the build.
      1. 
  16. Select the **General** tab.
      1. 
  17. This is where you configure the default queue to use, build number format, build timeout, and other general settings. This is also where you can specify demands for specific agent capabilities. By default, we have demands already in place to ensure that MSBuild, Visual Studio, and VSTest capabilities are defined for the agent.
      1. 
  18. Select the **Build** tab.
      1. 
  19. The first build step restores any NuGet packages that might be out of date. There are a variety of options for configuring this, such as the ability to disable the local cache. Leave these options as defaults.

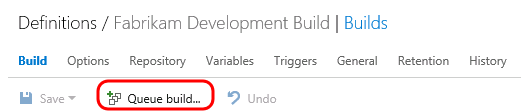
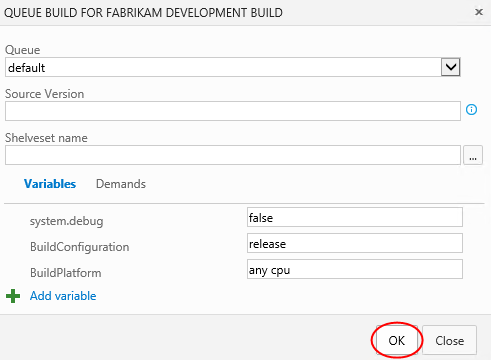
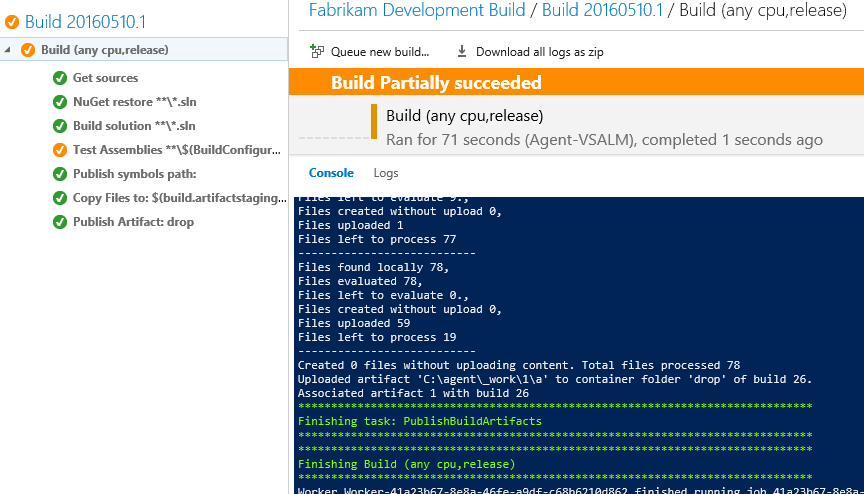


* 1. The second build step defined by this template is for **Visual Studio Build**. By default, it is configured to build all solution files in all subfolders, use the platform specified by the BuildPlatform variable, and use the configuration specified by the BuildConfiguration variable.
     1. 
  2. The third build step is **Visual Studio Test**. We will mostly use default options here to test assemblies with “\*Fabrikam\*test” in the name. Update the **Test Assembly** field to **“\*\*\$(BuildConfiguration)\\*Fabrikam\*test\*.dll;-:\*\*\obj\\*\*”**. Note that this is really just inserting “\*Fabrikam” before “\*test”.
     1. 
  3. Expand the **Advanced Execution Options** section and note the additional options, such as the ability to specify a specific version of VSTest or paths to custom test adapters.
     1. 
  4. Before we move on to the next build step, select the **Continue on Error** option (from the Control Options section) so that subsequent steps will be executed even if some tests fail. The reason that we are doing this is because this virtual machine has some tests that are setup to fail for demonstration purposes.
     1. 
  5. Select the next build step named **Index Sources & Publish Symbols**. This is where you can specify a path to a symbol store share, although we will not do so at this time.
     1. 
  6. The fifth build step is **Copy Files** and places a copy of the build into a staging directory.



* 1. Select the final build step named **Publish Build Artifacts**. This step will take the build output from the bin folder, zip it up into a build artifact named “drop”, and then upload it to the TFS server.
     1. 
  2. Click **Save**.
     1. 
  3. Name the new build definition “**Fabrikam Development Build**” and click **OK**.
     1. 

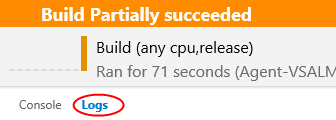
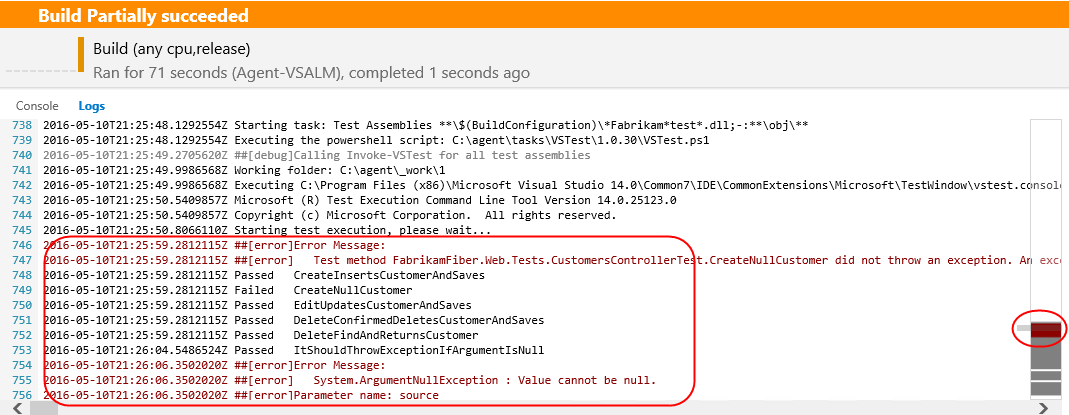
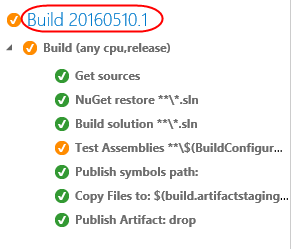
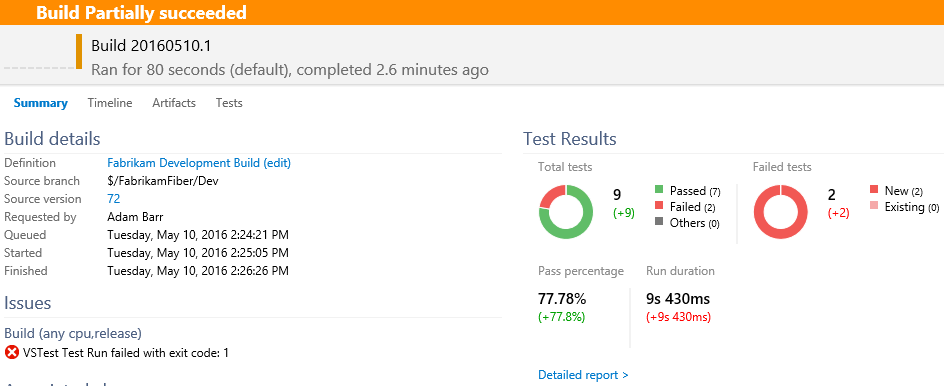
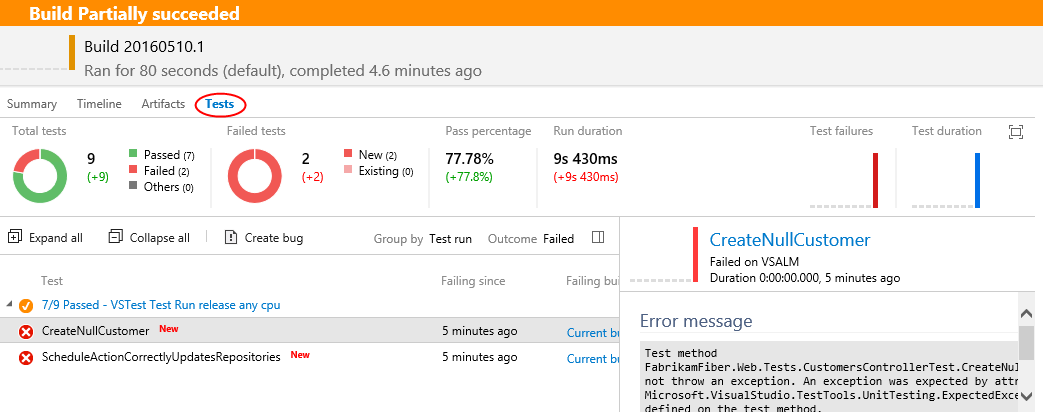
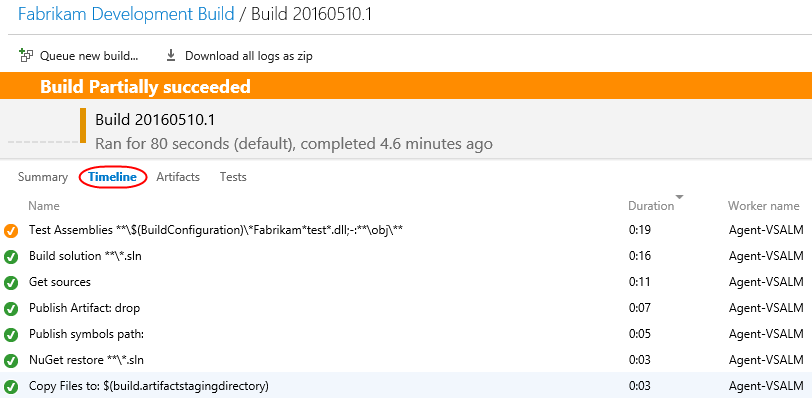
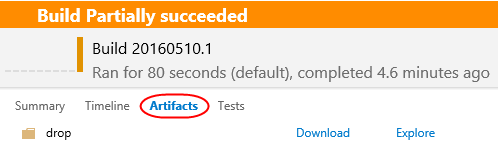
#### Task 2: Queuing and Executing a Build

* 1. Click the **Queue Build** button.
     1. 
  2. Note that the **Queue Build** dialog allows you to configure the queue, optionally select a shelveset, variable values, and demands. Use the defaults presented by clicking **OK**.
     1. 
  3. Note that once the build starts, you can monitor the real-time build status in the live console view. As the Test step proceeds, you will see some red error text as there are some tests in this project that are designed to fail for demonstration purposes. At the end of the build you should see the message in orange displaying “Build Partially succeeded”.
     1. 

## Exercise 3: Build and Test Result Analysis

* 1. In this exercise, you will examine the build results and also see how to view the results from the tests that were executed as part of the build.

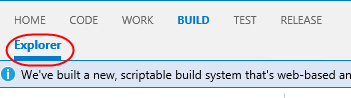
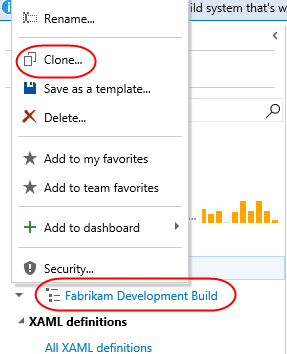
#### Task 1: Analyzing Build and Test Results

* 1. Select the **Logs** tab from the build results.
     1. 
  2. Note that the scrollbar has a section that is highlighted red towards the bottom. If you scroll to this region of the log, you will see messages related to the failed tests.
     1. 
  3. In the left-hand pane showing the build number (and build node with build steps/status), click the root node to view the overall build summary view.
     1. 
  4. Here you can see top-level information about the build including associated definition, source branch, who requested it, and timestamps. It also shows that the VSTest test run failed.
     1. 
  5. To further investigate and analyze the test run results, select the **Tests** tab. Here you can see the details about the test run, build platform and flavor, test environment used, attachments, and some charts showing outcome.
     1. 
  6. Select the **Timeline** tab, which shows the overall steps from the build and how long they took to complete. It also shows the agent used, so you may be able to use this to see if there is good reason to scale out the agent pool in order to parallelize the build.
     1. 
  7. Select the **Artifacts** tab. This is where you can download or explore the artifact containing the build output.
     1. 

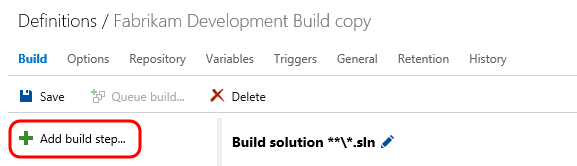
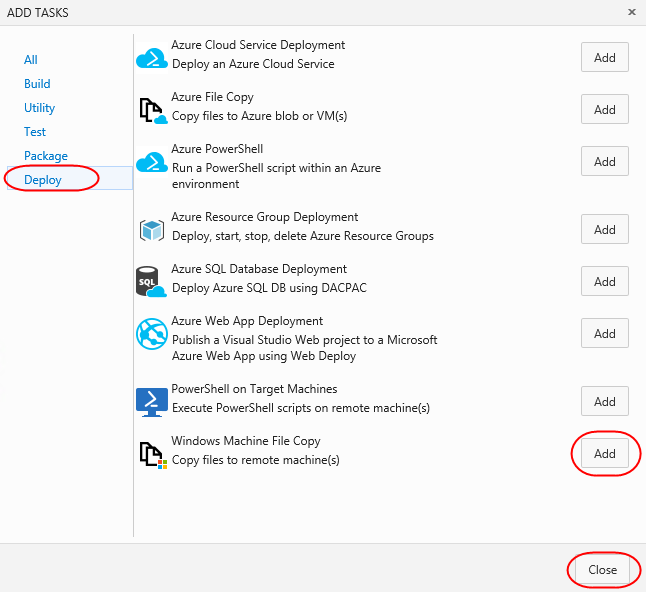
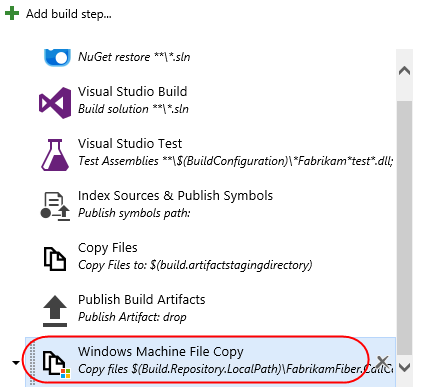
## Exercise 4: Continuous Integration and Deployment

* 1. In this exercise, you will learn how to modify a build to support continuous integration. In addition, you will also learn about some of the deployment options available.

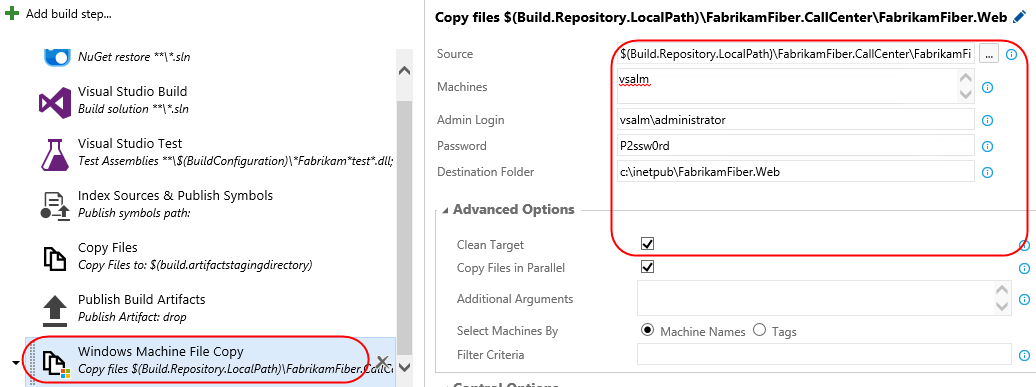
#### Task 1: Cloning a Build Definition

* 1. Now let’s create a similar build definition, but this time include a step to deploy the Fabrikam Fiber website. Imagine that this build definition will be part of a continuous integration scenario.
  2. Return to the main Build view by clicking on **Build | Explorer**.
     1. 
  3. Select the drop-down just to the left of the **Fabrikam Development Build** definition to load the associated context menu. Note the options that allow us to create a new build definition using the current one as a starting point: **Clone** and **Save as a Template**. If we wanted to share this build definition as a template with the rest of our team, we could do so here and it would show up as a Custom template when creating a new definition. In this case, click **Clone** to create a copy of the current definition.
     1. 

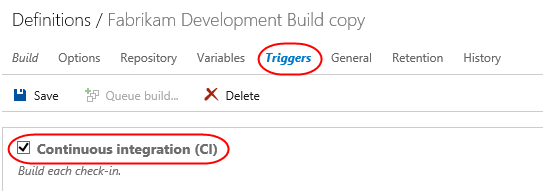
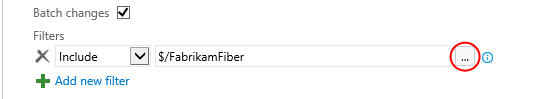
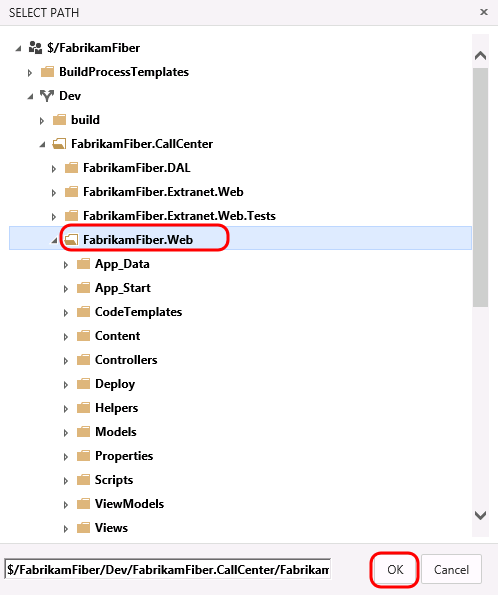
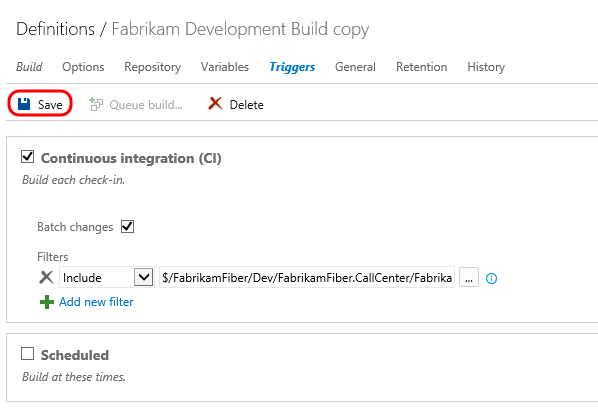
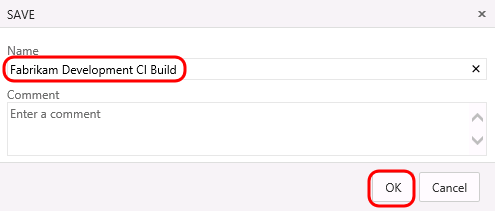
#### Task 2: Adding a Deployment Step and Defining Machine Group

* 1. Click **Add build step**.
     1. 
  2. Select the **Deploy** tab and check out the available options that range from Azure Web Site deployment to PowerShell execution and file copy. For simplicity, let’s deploy to the local machine using the **Windows Machine File Copy** step, so click **Add**. Then **Close** the dialog.
     1. 
  3. Select the new **Windows Machine File Copy** step.
     1. 
  4. Set the **Source** property to be the following:

$(Build.Repository.LocalPath)\FabrikamFiber.CallCenter\FabrikamFiber.Web

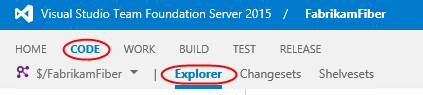
* 1. Set **Machines** to **“vsalm”**.
  2. For the purposes of this demo, we will continue to work within this virtual machine, so enter credentials for **vsalm\administrator** (password is **P2ssw0rd**).
  3. Now we need to specify the target copy location on the destination machine. Enter “**c:\inetpub\FabrikamFiber.Web**” to the right of the **Destination Folder** field.
  4. Select the **Clean Target** option under **Advanced Options** so that the copy location will be cleaned before each deployment.
     1. 

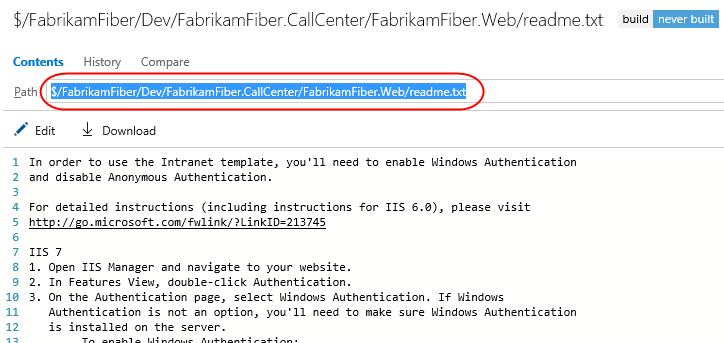
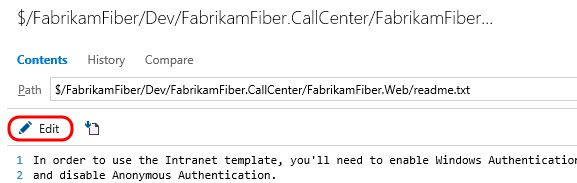
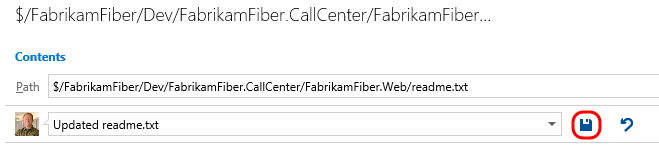
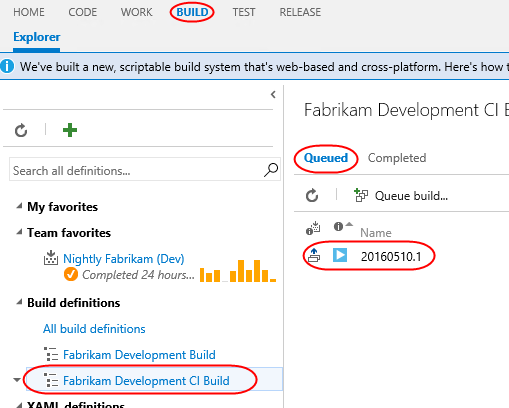
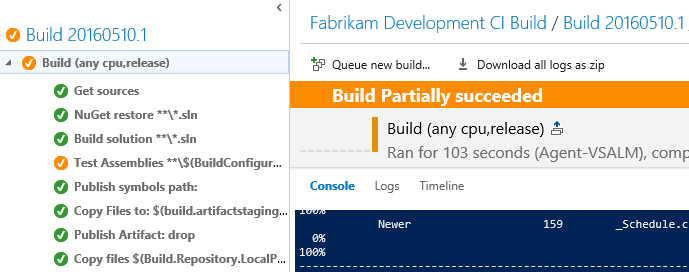
#### Task 3: Configuring Continuous Integration

* 1. Now let’s get the continuous integration functionality hooked up. Select the **Triggers** tab and check the **Continuous integration** option.
     1. 
  2. You can add filters to include or exclude certain source paths. Click the **ellipses** button to the right of the default include path that is currently set to “$/FabrikamFiber”.
     1. 
  3. In the **Select Path** window, expand and select $/FabrikamFiber/Dev/FabrikamFiber.CallCenter/FabrikamFiber.Web”. Click **OK**.
     1. 
  4. There is also an option here to trigger the build on a set schedule. For example, we could set this build up to run every evening if desired.
  5. Click **Save**.
     1. 
  6. Name the new build “**Fabrikam Development CI Build**” and then click **OK**.
     1. 

#### Task 4: Triggering a Continuous Integration Build

* 1. In a new browser window or tab, navigate to the **Code | Explorer** view.



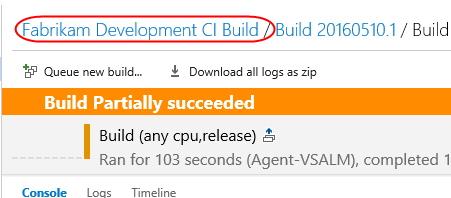
* 1. Open **$/FabrikamFiber/Dev/FabrikamFiber.CallCenter/FabrikamFiber.Web/readme.txt** by entering the path into the **Path** field and pressing **Enter**. Alternatively, you can use the tree navigation on the left.
     1. 
  2. Click the **Edit** button and make a small change to the file, such as adding a newline at the top of the file.
     1. 
  3. Click **Save**.
     1. 
  4. Return to the **Build | Explorer** view, select the CI build and note that a build was queued as expected. If the build completes before you return here, you may need to select the Completed tab to locate it.
     1. 
  5. **Double-click** the CI build to monitor its progress (or view the results).
     1. 

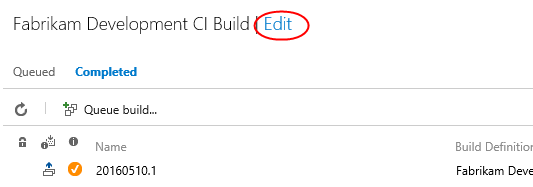
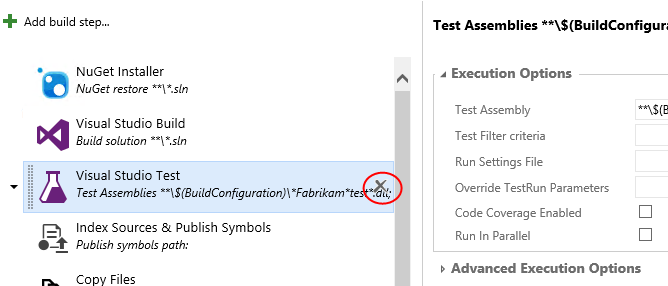
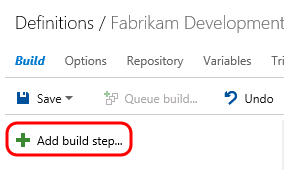
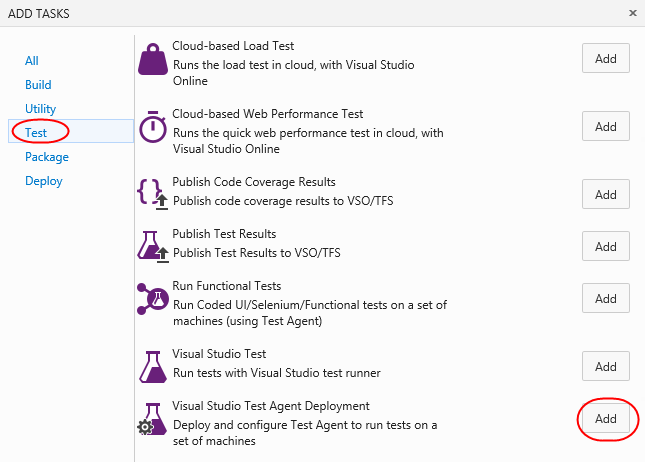
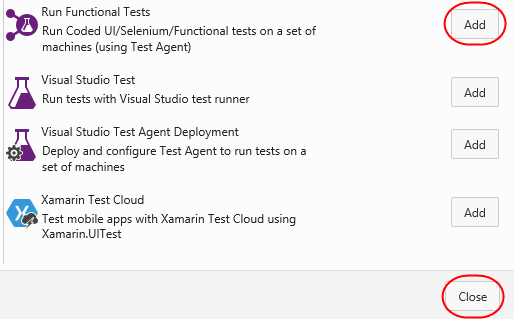
## Exercise 4: Scale out Testing using Machine Groups

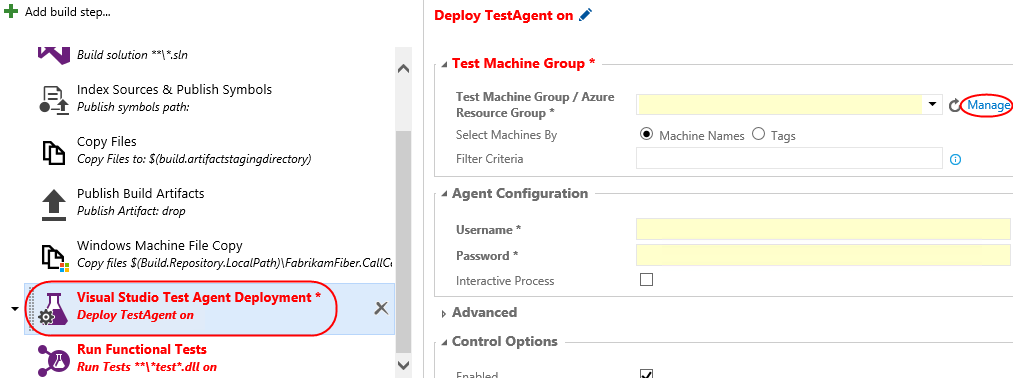
* 1. In this exercise, you will learn how to scale out your testing needs by making use of machine groups and test agents running on other machines.
  2. **Note:** This step is optional because it needs to download and install a test agent installer (approx. 130 MB) over the Internet, and therefore it will take a few minutes to complete.

#### Task 1: Modifying Build to use External Test Agents

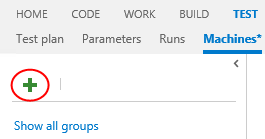
* 1. Let’s say that we would like to modify our CI build to perform testing on dedicated testing hardware using a test agent. Click **Fabrikam Development CI Build** to open the build definition.



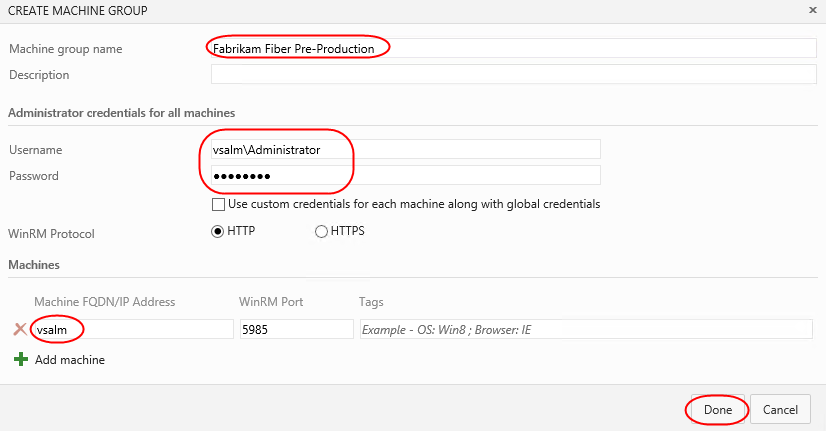
* 1. Click **Edit** to edit the build definition.
     1. 
  2. **Delete** the existing **Visual Studio Test** step, as we will replace this shortly.
     1. s
  3. Click the **Add Build Step** button.
     1. 
  4. First, we need to ensure that a test agent will be installed on our test machine(s). From the **Test** tab, click the **Add** button for the **Visual Studio Test Agent Deployment** task.
     1. 
  5. Next, we need to specify the tests to run. Click the **Add** button for the **Run Functional Tests** task and click **Close**.
     1. 
  6. Select the **Visual Studio Test Agent Deployment** step and click **Manage** to manage the test machine group to be used. This will open in a new browser tab.



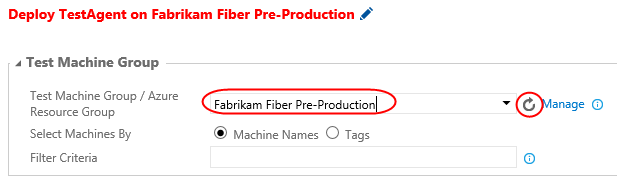
* 1. Click the **Add machine group** button to add a new machine group.

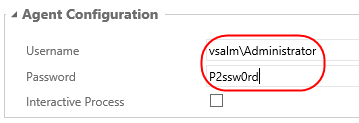
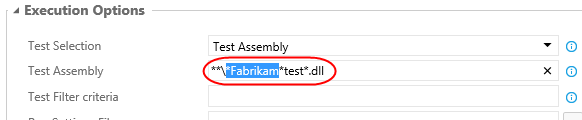
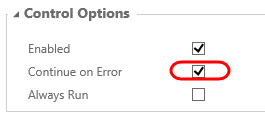
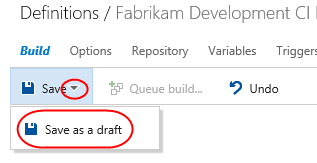


* 1. Set the **Machine group name** to **“Fabrikam Fiber Pre-Production”**.
  2. Set the credentials to **“vsalm\Administrator”** with password **“P2ssw0rd”**.
  3. Set the first machine name to **“vsalm”** and click **Done.**

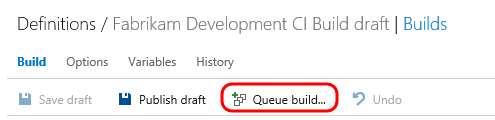
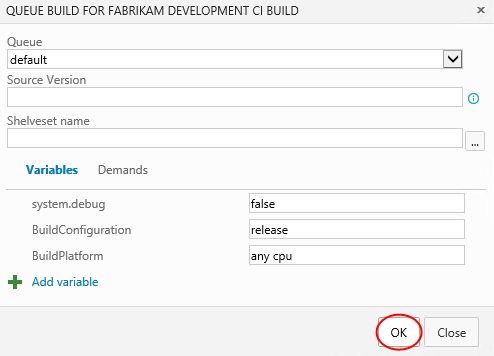
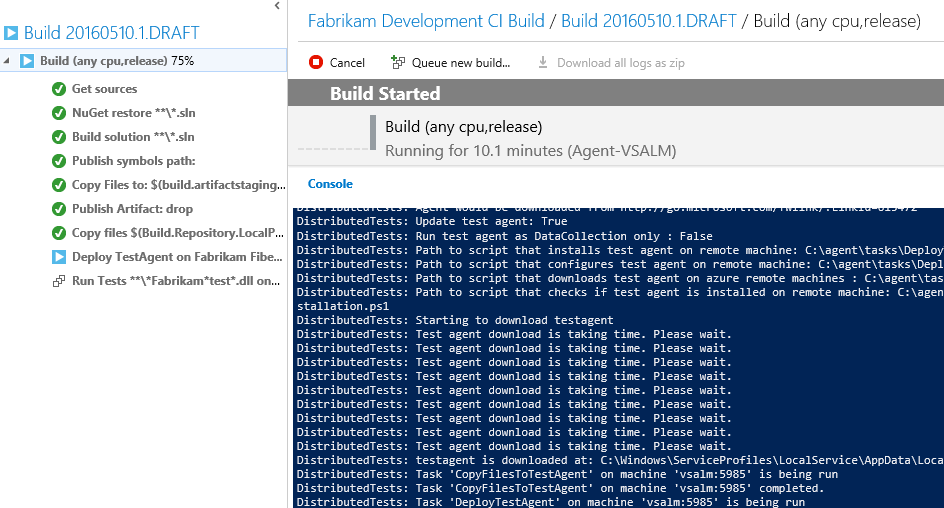


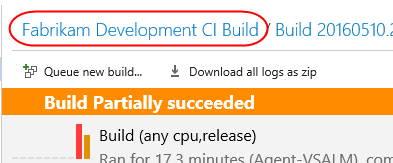
* 1. After completion, close the current browser tab to return to the **Build Definition** tab you were working on previously.
  2. In **Test Machine Group**, click the **Refresh** button to refresh the list of groups available. Then select the machine group that you just created.



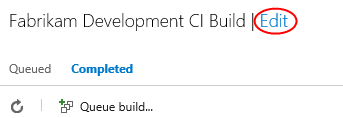
* 1. For **Agent Configuration**, set the **Username** to **“vsalm\Administrator”** with a **Password** of **“P2ssw0rd”**.
     1. 
  2. Select the **Run Functional Tests** step.
  3. For **Test Machine Group**, select the same **Test Machine Group** (you may need to use the provided **Refresh** button again) as before and set the **Test Drop Location** to **“$(Build.Repository.LocalPath)”**.
  4. Under **Execution Options**, set **Test Assembly** to **“\*\*\\*Fabrikam\*test\*.dll”**. Note that this is really just inserting “\*Fabrikam” before “\*test”.
  5. Although we will not do so in this demo, note that there are a number of other test settings that can be specified here such as platform, configuration, run settings file, test configurations, and so on.
  6. Select the **Continue on Error** option from the **Control Options** section.
     1. 
  7. Since we are making some major changes to an existing build definition, perhaps it would be best to save it as a draft until we can validate that it is functioning as expected. Click the **Save** drop-down and then select the **Save as Draft** option.
     1. 

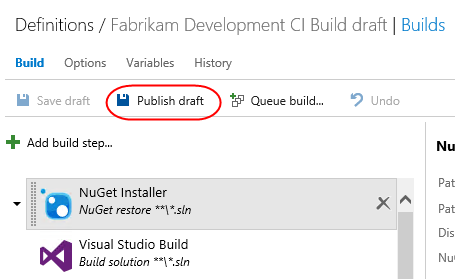
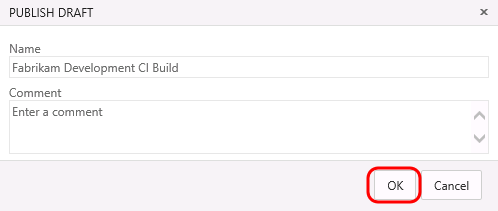
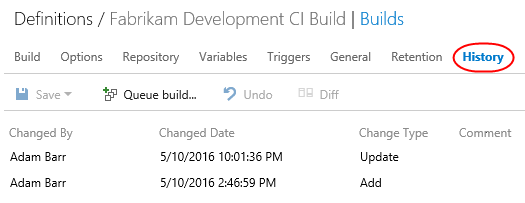
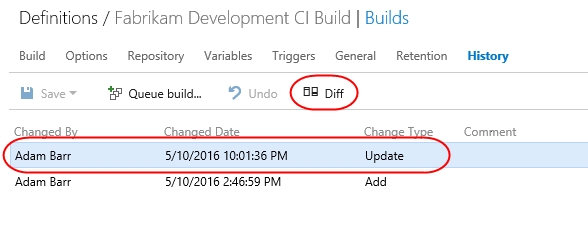
#### Task 2: Testing and Saving Modified Build Draft

* 1. Click **Queue Build**.
     1. 
  2. Click **OK** to use the default options.
     1. 
  3. Grab a coffee and watch the build proceed. Depending upon the speed of your Internet connection and hardware setup, it will take a few minutes to download and install the test agent. Note that this initial setup only happens once per machine from the machine group, and that subsequent build runs will check for an existing test agent install and use it if possible.
     1. 
  4. The build should partially succeed in the end, again due to some failing tests.
     1. 
  5. Let’s say that we are satisfied with this draft build definition. Return to the build definition page by clicking **Fabrikam Development CI Build**.



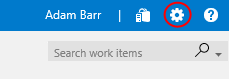
* 1. Click **Edit**.



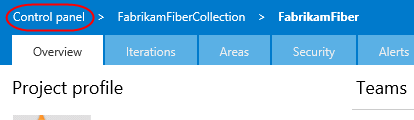
* 1. Click **Publish Draft**.
     1. 
  2. In the **Publish Draft** dialog, click **OK**.
     1. 
  3. Changes to the build definition can be viewed on the **History** tab.
     1. 
  4. Note that the changes that we just applied show up at the top of the list. Select the first change in the list and then click the **Diff** button to see changes.
     1. 
  5. The diff view should look something like the following screenshot.
     1. 

#### Task 3: Reviewing build load

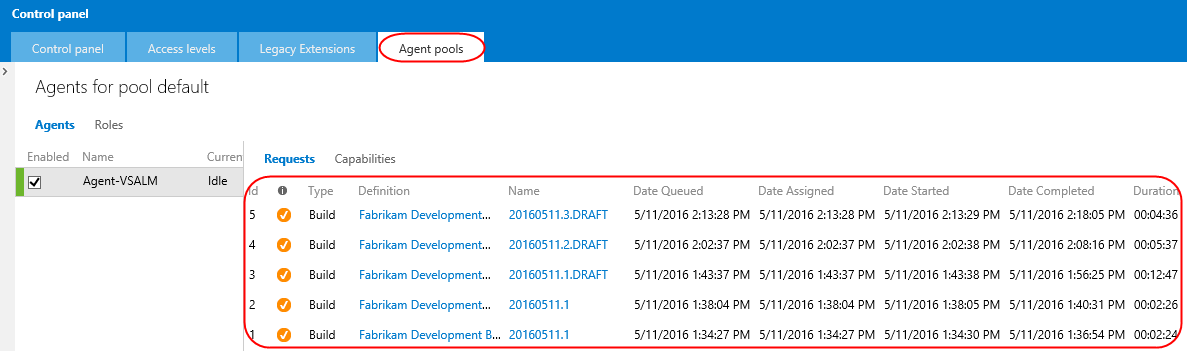
* 1. Click the **Manage Project** gear icon in the top right corner to view the administrative pages.



* 1. Click the root **Control panel** breadcrumb link.



* 1. Select the **Agent pools** tab. Here you can get a unified view of all the builds processed per pool. In the event that you need to perform capacity planning, these details can help you understand how each pool is handling their current load across all serviced queues and build definitions.



To give feedback please write to [VSKitFdbk@Microsoft.com](mailto:VSKitFdbk@Microsoft.com)

Copyright © 2016 by Microsoft Corporation. All rights reserved.