

Hands-on Lab 4.1:

Monitoring the Continuous Delivery Pipeline



Table of Contents

Objectives	1
Prerequisites	2
Time	2
Exercise 1: Monitoring Instances of the Pipeline.....	2
Task 1: Trigger and Monitor Pipeline Instances.....	2
Exercise 2: Retriggering Failed Stages.....	14
Task 1: Retrigger a Failed Stage Other Than the Commit Stage	14
Task 2: Retriggering the Commit Stage.....	20
Exercise 3: Dealing with Bugs.....	22
Stop the Pipeline and Identify the Cause of the Failure	22
Use the Debugging Symbols	23
Fix the Bug and Run the Pipeline	23
Summary	23
Copyright.....	23

Objectives

This HOL demonstrates how to monitor instances of the continuous delivery release pipeline, and how to analyze the information you receive. You learn to interpret feedback from a pipeline that is running

normally. You also learn what to do when the pipeline stops, either because there are problems with the environments or with the code.

Prerequisites

You need to have completed all the previous labs.

Time

You should be able to complete this lab in about 40 minutes.

Exercise 1: Monitoring Instances of the Pipeline

In this exercise you learn how to monitor the pipeline, identify running instances and get information about them.

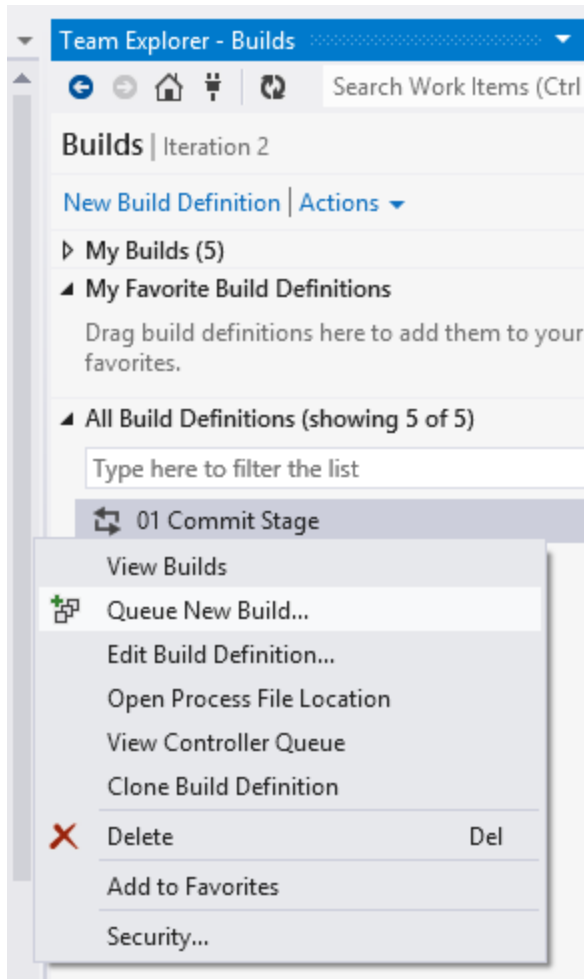
Task 1: Trigger and Monitor Pipeline Instances

In this task you trigger and monitor instances of the pipeline. You can trigger an instance of the pipeline in two ways.

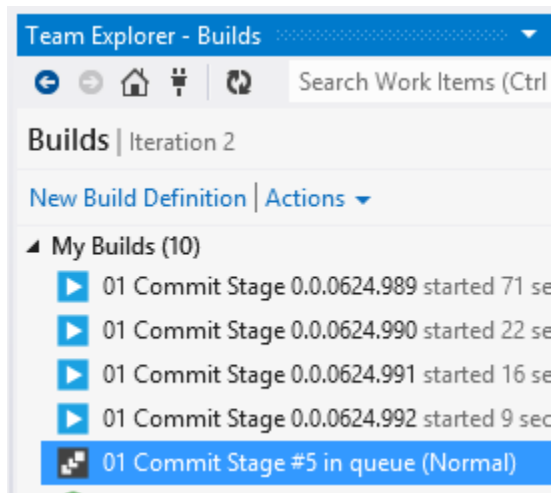
- Manually queue the 01 Commit Stage build definition.
- Check in changes to the code.

Both methods have the same outcome. This task uses the first method because it is easier.

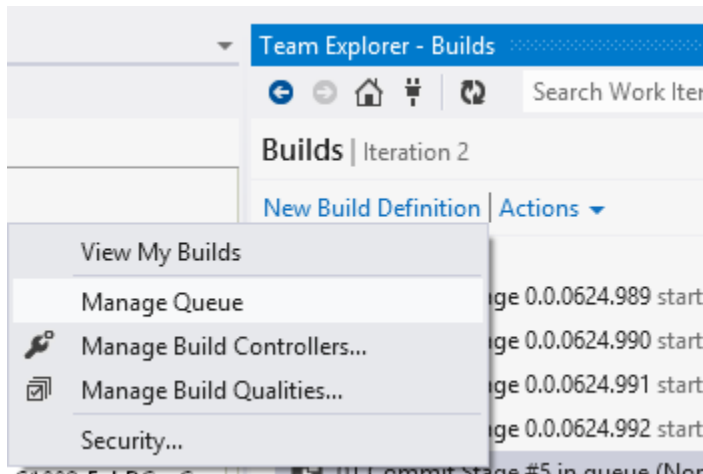
1. Trigger a new instance of the pipeline by queuing the **01 Commit Stage** build definition. In Team Explorer, click **Builds**. Click **All Build Definitions**. Click **01 Commit Stage**.



2. Repeat the operation several times to create several pipeline instances. In the following screenshot there are five instances. This is one more than the number of build agents so the last instance is queued rather than running. This is an example of how the pipeline works when there are concurrent instances. (Remember that you will have different instance numbers.)

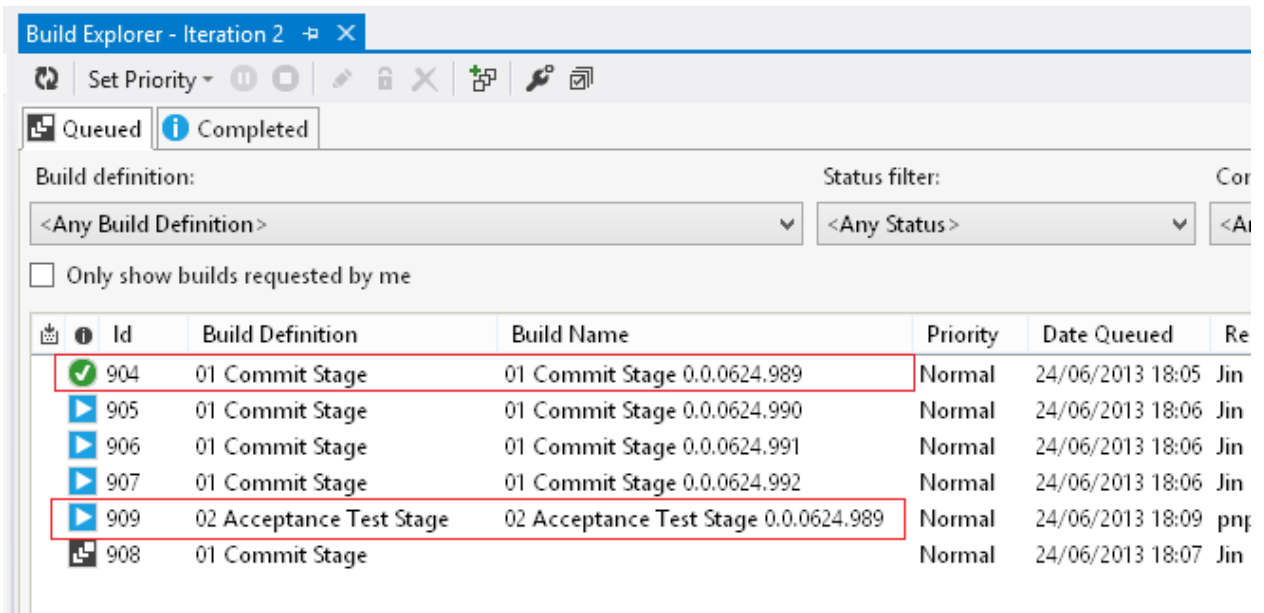


3. In **Team Explorer – Builds**, click **Actions**. Select **Manage Queue** from the drop-down list. **Build Explorer** opens. This is the main monitoring tool for the pipeline.

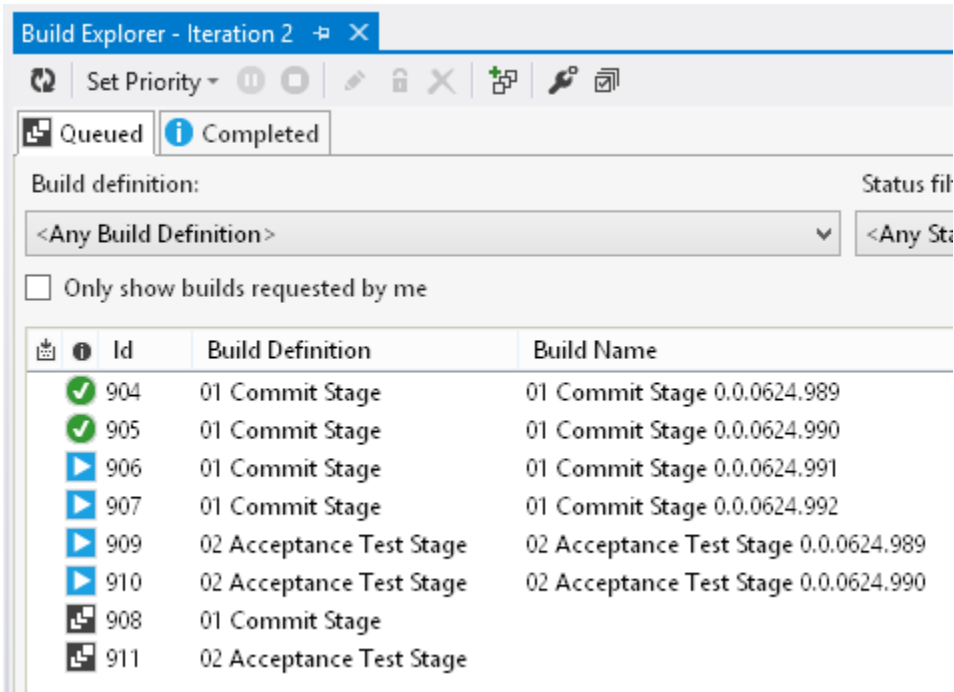


TIP: Most of the activities that you can perform with Build Explorer can also be done by selecting the **BUILD** tab from the Web Access portal associated with the team project.

4. In Build Explorer, clear the **Only show builds requested by me** option if necessary. Automatically triggered stages such as the acceptance test stage are run by the build service account, so you won't be able to see them if you have this option checked.
5. Each pipeline instance is identified by the version number assigned to it during the commit stage. Use the version number to relate running stages to each other when you monitor a pipeline instance. In the following screenshot, the pipeline instance **0.0.0624.989** has successfully finished the commit stage and has automatically triggered the acceptance test stage. You know this because both stages have the same version number. The acceptance test stage takes precedence over the queued commit stage at the bottom of the screen because of the automatic trigger.

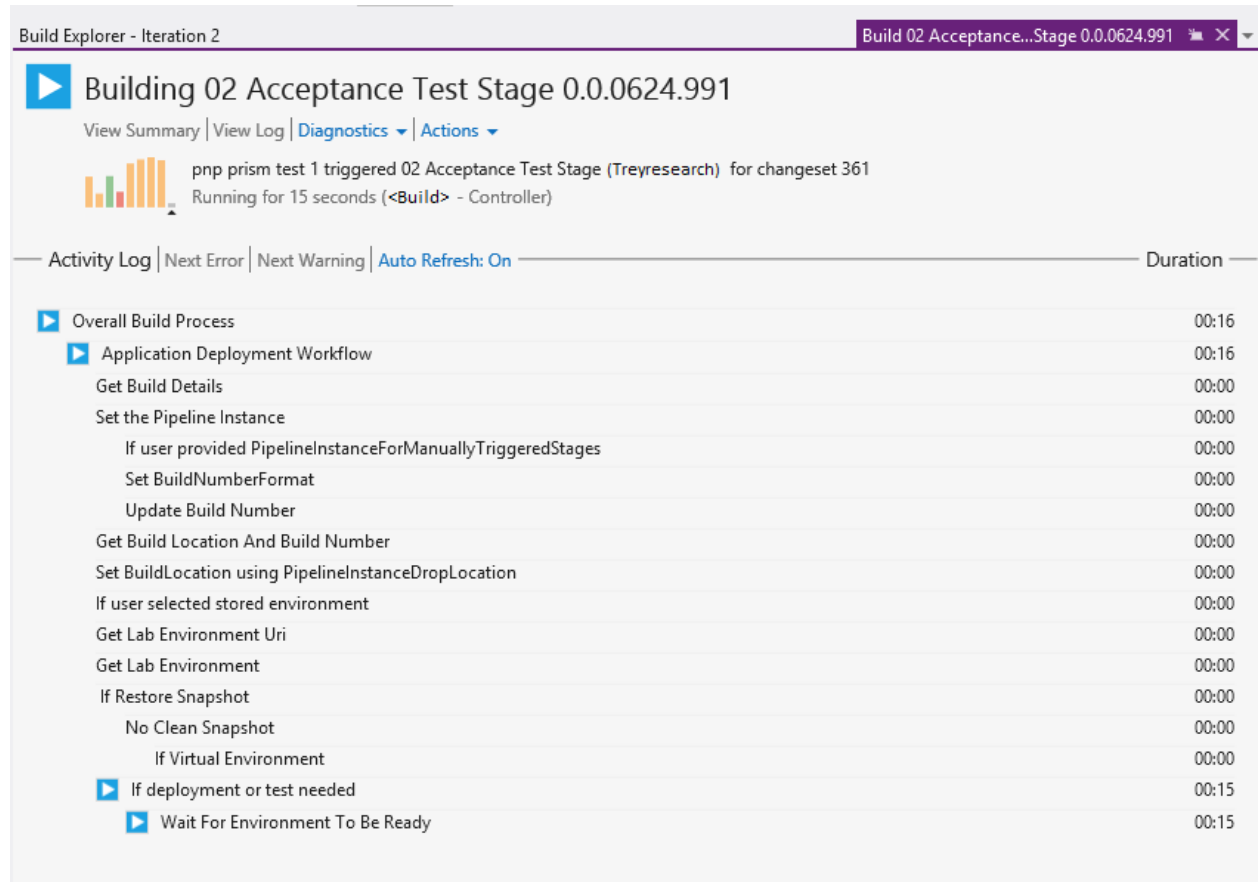


4. New acceptance test stages are queued as soon as the commit stages in the same pipeline instances have finished. They run as long as there are enough build agents available. In the following screenshot, the 0.0.0624.990 instance has advanced and is running its acceptance test stage. However, there is another acceptance test stage in the queue because all four build agents are busy.

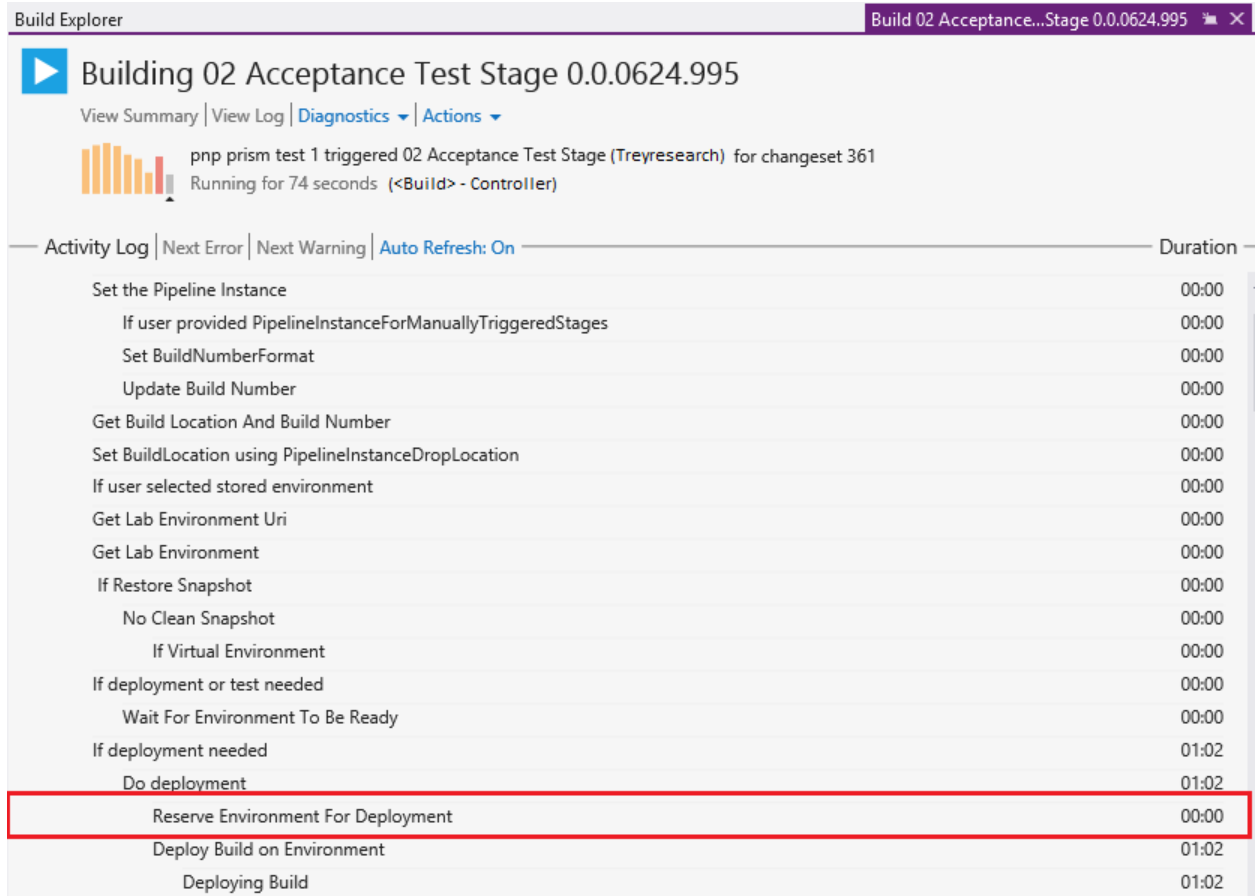


5. All the pipeline stages but the commit stage have dependencies on environments because they perform automated deployments and tests on specific machines. This means that the pipeline

orchestration must ensure that the target environment is not already being used by another pipeline instance. For example if a pipeline instance deploys to an environment where another instance is running tests, the deployment will fail. If you open the details of a running acceptance test stage (select it from Build Explorer), it's likely that you will see that, even though the stage is running from the point of view of the build agent, in fact it has been blocked by either the deployment agent or the test agent. The blocked acceptance test stage is waiting for the environment to be ready . The following screenshot shows an example of this situation.



- As soon as the environment is released, the next blocked stage that is waiting for it will be able to continue. In the following screenshot, the environment is reserved (look for the line **Reserve Environment for Deployment**). This means it is in use and other stages are blocked until it is released.



- After all the stages finish, you can use Build Explorer to see the overall results. You can get details about a particular pipeline instance by selecting any of its stages. The following screenshot shows an example of the overall results.

Build Explorer - Iteration 2

Set Priority

Queued Completed

Build definition: <Any Build Definition> Status filter: <Any Status>

☐ Only show builds requested by me


Id	Build Definition	Build Name	Price
905	01 Commit Stage	01 Commit Stage 0.0.0624.990	Non
906	01 Commit Stage	01 Commit Stage 0.0.0624.991	Non
907	01 Commit Stage	01 Commit Stage 0.0.0624.992	Non
909	02 Acceptance Test Stage	02 Acceptance Test Stage 0.0.0624.989	Non
910	02 Acceptance Test Stage	02 Acceptance Test Stage 0.0.0624.990	Non
908	01 Commit Stage	01 Commit Stage 0.0.0624.995	Non
912	02 Acceptance Test Stage	02 Acceptance Test Stage 0.0.0624.992	Non
911	02 Acceptance Test Stage	02 Acceptance Test Stage 0.0.0624.991	Non
913	02 Acceptance Test Stage	02 Acceptance Test Stage 0.0.0624.995	Non

- If you look at the **0.0.0625.990** instance, you see that its acceptance test stage has not completely succeeded. If you select the stage, you can see more details. These are shown in the following screenshot.

Build Explorer Build 02 Acceptance...Stage 0.0.0624.990

02 Acceptance Test Stage 0.0.0624.990 - Build partially succeeded

[View Summary](#) | [View Log](#) | [Open Drop Folder](#) | [Diagnostics](#) | [<No Quality Assigned>](#) | [Actions](#)

 pnp prism test 1 triggered 02 Acceptance Test Stage (Iteration 2) for changeset 361
Ran for 2,2 minutes (<Build>-Controller), completed 20,2 hours ago

Latest Activity

Build last modified by pnp prism test 1 20,2 hours ago.

Request Summary

[Request 910](#), requested by pnp prism test 1 20,2 hours ago, Completed


Deployment Information

Deployment

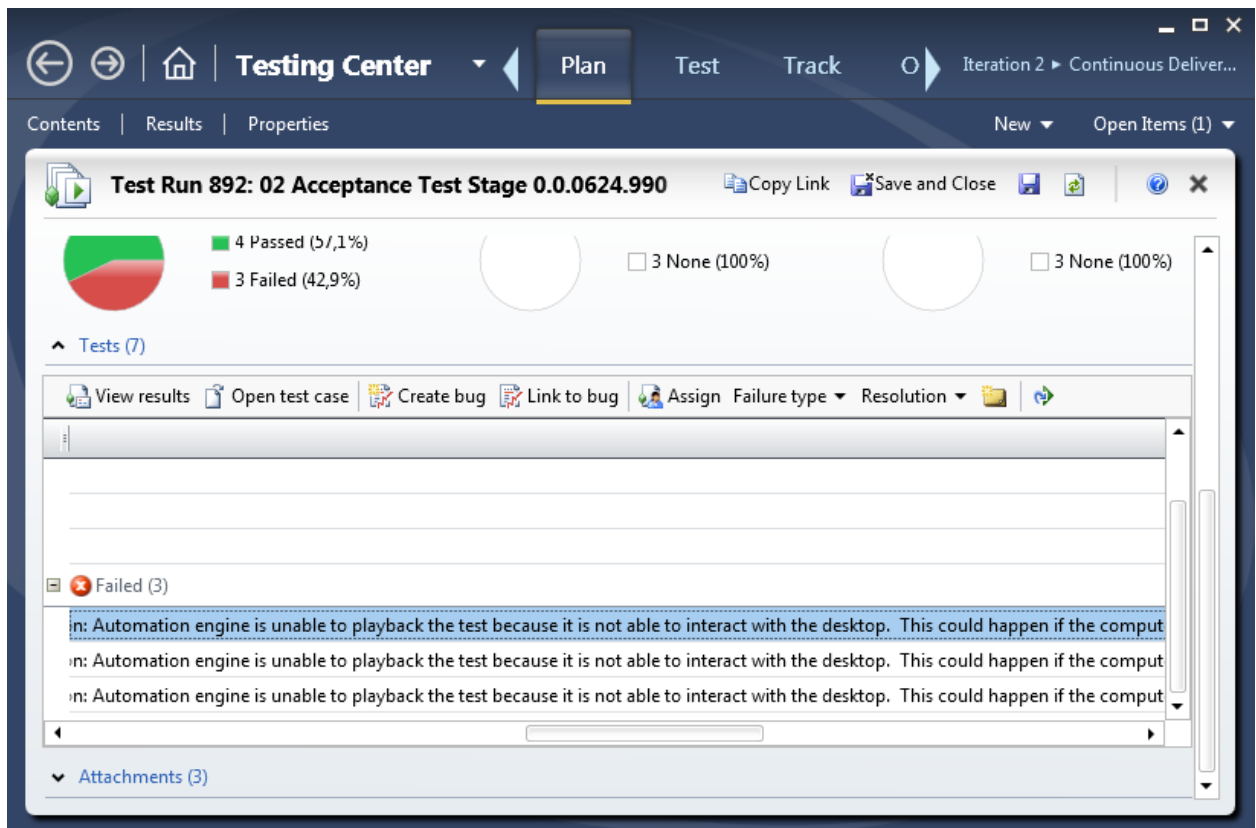
Lab environment: Test+Staging+Production
The application was deployed successfully from the following build location: [\\<Drop> \builds\Iteration 2\01 Commit Stage\01 Commit Stage 0.0.0624.990](#)

Test Results

Test run (Id) : 02 Acceptance Test Stage 0.0.0624.990 (892)
Test run needs investigation


[Test run details](#)
 4 of 7 test(s) passed, 3 failed, 0 inconclusive, [View Test Results](#)

9. Notice that three tests have failed. No code changed since the last time the tests passed, so you know it must be a problem related to the environments.
10. Click **View Test Results**. Microsoft Test Manager opens. Here is an example screenshot.




11. The error message indicates that there is no active session (either by using the remote desktop or locally) on the computer that runs the UI tests. The UI tests require an active session. After you create the session, rerun the pipeline.
12. Next, examine the reasons why the **0.0.0624.991** acceptance test stage completely failed. The following screenshot shows the details.

Build Explorer Build 02 Acceptance...Stage 0.0.0624.991



02 Acceptance Test Stage 0.0.0624.991 - Build failed

[View Summary](#) |
 [View Log](#) -
 [Open Drop Folder](#) |
 [Diagnostics](#) ▾ |
 [<No Quality Assigned>](#) ▾ |
 [Actions](#) ▾



pnp prism test 1 triggered 02 Acceptance Test Stage (Treyresearch) for changeset 361
 Ran for 2,4 minutes (<Build> - Controller), completed 20,4 hours ago

Latest Activity

Build last modified by pnp prism test 1 20,4 hours ago.


Request Summary

[Request 911](#), requested by pnp prism test 1 20,4 hours ago, Completed

Summary

Other Errors and Warnings

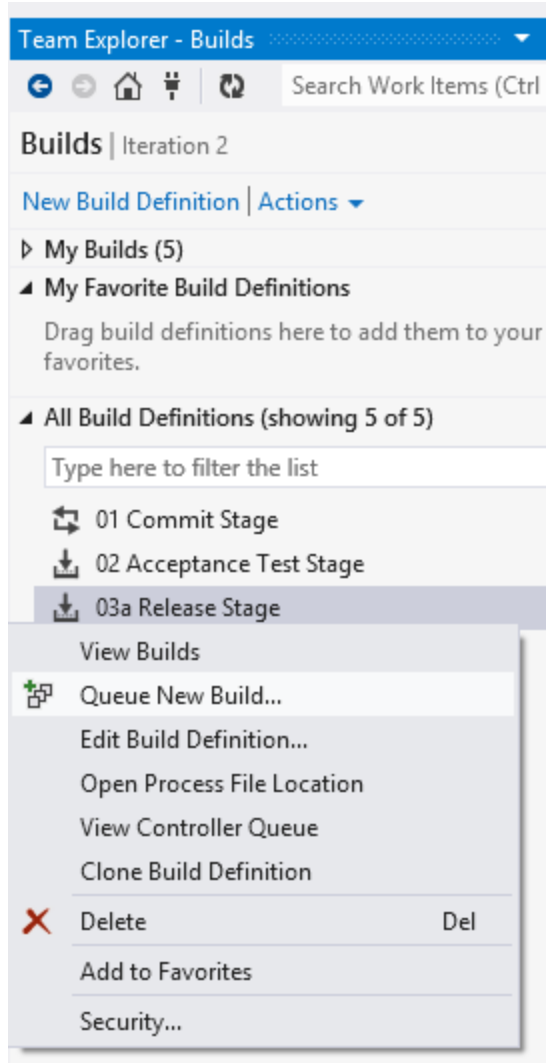
▲ 1 error(s), 0 warning(s)



Exception Message: TF259617: Team Foundation Server could not complete the request because one or more test machines are not ready. Wait a few minutes and then try the operation again. (type LabDeploymentProcessException)
 Exception Stack Trace: at Microsoft.TeamFoundation.Lab.Workflow.Activities.ReserveEnvironmentForDeployment.Execute (CodeActivityContext context)
 at System.Activities.CodeActivity.InternalExecute(ActivityInstance instance, ActivityExecutor executor, BookmarkManager bookmarkManager)
 at System.Activities.Runtime.ActivityExecutor.ExecuteActivityWorkItem.ExecuteBody(ActivityExecutor executor, BookmarkManager bookmarkManager, Location resultLocation)

The issue is that the single test controller could not handle the number of pipeline instances that were triggered simultaneously. The instance failed because the step where the acceptance test stage waits for the environment to be ready took too long and the operation timed out.

13. The only instance that has successfully passed the acceptance test stage is **0.0.0625.995**. The reason is that, when the UI tests were failing because there was no active session on the target machine, a session was opened before the acceptance test stage was triggered. This instance can advance to the release stage and the UAT stage, which are manually triggered. To trigger the stage you need the pipeline instance number (**0.0.0625.995**), and the full path to the commit stage drop location, which in the example, is [\\G1008-FabBC\builds\Iteration 2\01 Commit Stage\01 Commit Stage 0.0.0624.995](#). The following two screenshots show how to manually trigger the release stage.



Queue Build "Iteration 2" ? X

General Parameters

Build process parameters:

1. Required	
Lab Process Settings	To see or edit the details, click ...
2. Basic	
Logging Verbosity	Normal
3. Release pipeline	
Next stages in pipeline	
Pipeline Instance - ONLY FOR MAI	0.0.0624.995
Pipeline Instance Drop Location -	it Stage\01 Commit Stage 0.0.0624.995
4. Misc	
Timeout For Each Deployment Scr	30

Pipeline Instance Drop Location - ONLY FOR MANUALLY TRIGGERED ST
Drop Location of the Pipeline Instance, where items to be deployed/tested were left by the commit stage. USE IT ONLY FOR MANUALLY TRIGGERED S...

Queue Cancel

After you queue a manually triggered stage, you can monitor and manage it from Build Explorer, just as you would with a manually triggered stage. Note that the naming convention for the build definitions helps to identify the sequence followed while running the stages (1 – 2 – 3a and 3b), where 1 signifies a commit stage build definition, and 2 signifies an acceptance test build definition. These two stages run one after the other. A 3a signifies a release stage and 3b signifies a UAT stage. These stages run in parallel (3a and 3b). The following screenshot shows these two stages and who triggered them.

The screenshot shows the TFS Build Explorer interface. At the top, there are tabs for 'Start Page', 'Source Control Explorer', and 'Build Explorer - Iteration 2'. The 'Build Explorer' tab is active, showing a list of build stages. Below the tabs, there are filters for 'Build definition:', 'Status filter:', and 'Controller filter:'. The 'Build definition:' dropdown is set to '<Any Build Definition>'. The 'Status filter:' dropdown is set to '<Any Status>'. The 'Controller filter:' dropdown is set to '<Any Build Co...'. There is a checkbox labeled 'Only show builds requested by me' which is currently unchecked. Below the filters, there is a table with the following columns: 'Id', 'Build Definition', 'Build Name', 'Priority', 'Date Queued', and 'Requested By'. The table contains two rows of data:

Id	Build Definition	Build Name	Priority	Date Queued	Requested By
916	03a Release Stage	03a Release Stage 0.0.0624.995	Normal	25/06/2013 14:58	Jin Lehnert
917	03b UAT Stage	03b UAT Stage 0.0.0624.995	Normal	25/06/2013 14:59	Jin Lehnert

Exercise 2: Retriggering Failed Stages

In this exercise you learn how to retrigger a particular stage of the pipeline when there is a failure. If a pipeline instance fails it stops so that the issues aren't propagated to further stages. This happens by default. You then must determine the reason for the failure. There are two possibilities.

- Failures that are caused by the code that was checked in (the change that triggered the pipeline instance). These failures are because of bugs in the code. See Exercise 3 for more information.
- Failures that are caused by the environment, including both the target environment where deployment, testing, and other activities are done, and also the pipeline environment itself, which is composed of the build and test controllers and agents, the binaries repository and any other related entities. After you fix the environment, there are two ways to proceed.
 - Create a new instance of the pipeline by triggering the commit stage. This is generally the preferred way but it doesn't suit every situation.
 - If you want to retry specific check-ins but there are more recent changes checked in, you can't simply trigger the pipeline because it always uses the latest version of the code when it is instantiated. In this case, you must retrigger the stage that failed.

Task 1: Retrigger a Failed Stage Other Than the Commit Stage

If you want to retrigger a stage that is not the commit stage, you can always do it manually, no matter whether the stage is a manually triggered one or an automatically triggered one. The problem is that if you run it using the same pipeline instance number, you will get an error because the name of any build has to be unique in TFS. The following screenshot shows an example.

Note: The steps shown in this task should be used to help debug the pipeline and should not be performed when using the pipeline in a production environment.

Build Explorer Build 1003

1003 - Build failed

View Summary | View Log | Open Drop Folder | Diagnostics | <No Quality Assigned> | Actions

Jin Lehnert triggered 02 Acceptance Test Stage (Treyresearch) for changeset 362
Ran for 4 seconds (<Build> - Controller), completed 6 seconds ago

Latest Activity

Build last modified by pnp prism test 1 6 seconds ago.

Request Summary

Request 918, requested by Jin Lehnert 15 seconds ago, Completed

Summary

Other Errors and Warnings

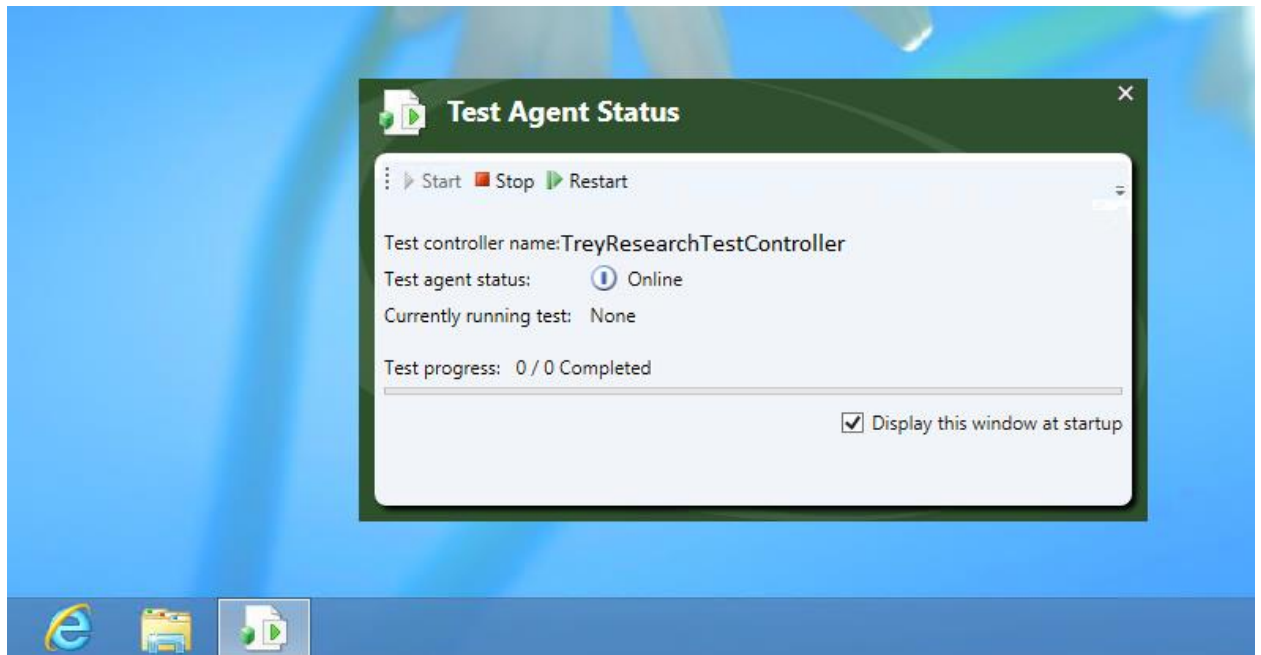
1 error(s), 0 warning(s)

- Exception Message: TF42064: The build number '02 Acceptance Test Stage 0.0.0624.990' already exists for build definition '\Iteration 2\02 Acceptance Test Stage'. (type BuildNumberAlreadyExistsException)
Exception Stack Trace: at Microsoft.TeamFoundation.Client.Channels.TfsHttpClientBase.HandleReply(TfsClientOperation operation, TfsMessage message, Object[]& outputs)
at Microsoft.TeamFoundation.Build.Client.BuildWebService4.UpdateBuilds(BuildUpdateOptions[] updateOptions)
at Microsoft.TeamFoundation.Build.Client.BuildDetail.Save()
at Microsoft.TeamFoundation.Build.Workflow.Activities.UpdateBuildNumber.Execute(CodeActivityContext context)
at System.Activities.CodeActivity`1.InternalExecute(ActivityInstance instance, ActivityExecutor executor, BookmarkManager bookmarkManager)
at System.Activities.Runtime.ActivityExecutor.ExecuteActivityWorkItem.ExecuteBody(ActivityExecutor executor, BookmarkManager bookmarkManager, Location resultLocation)

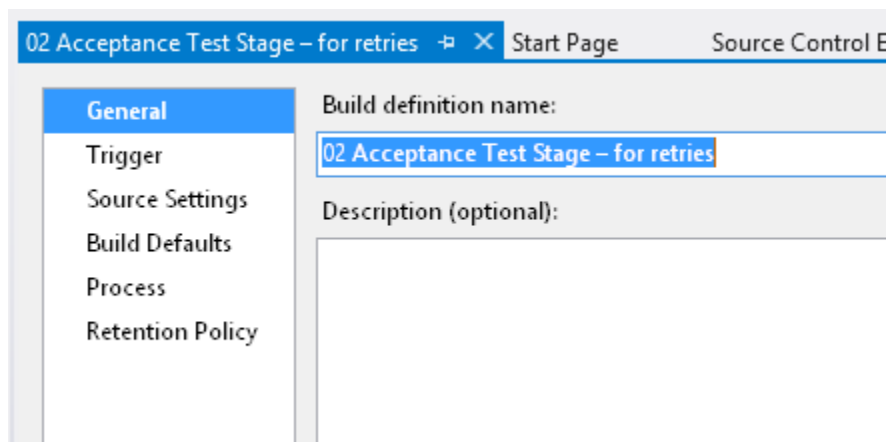
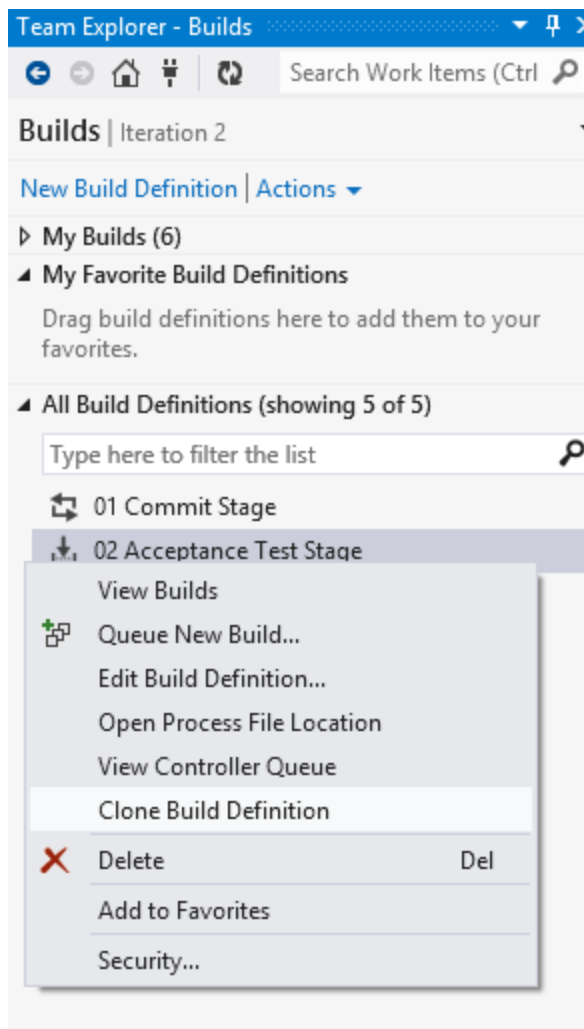
Inner Exception Details:

The easiest workaround is to prepare a temporary build definition that mirrors the stage you want to retry, but with a different name, and use it instead of the original one. As a concrete example, here is what to do if you want to retry an acceptance test stage that only partially succeeded for the **0.0.0624.990** instance.

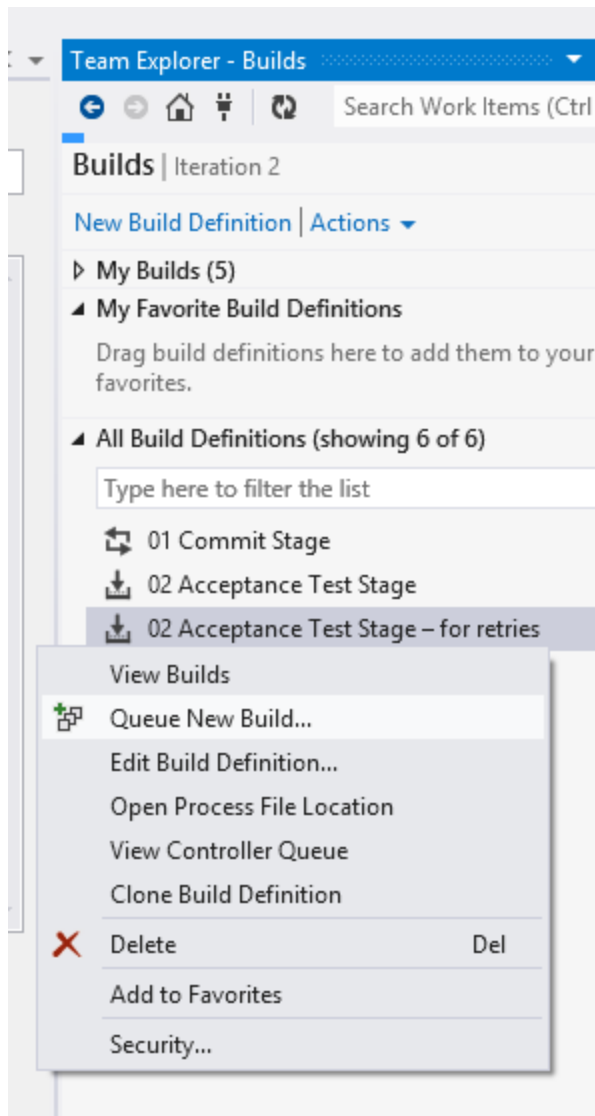
1. Fix the problem with the environment. In this example, open a remote desktop session to the computer that runs the automated UI tests, using the service account used by the test agent. Leave it active.



2. From Team Explorer, clone the build definition for the acceptance test stage and give it a different name. In this example, the stage is renamed to **02 Acceptance Test Stage – for retries**. You can enter the name in the build definition editor and then save the build definition. This is shown in the following two screenshots.



3. Manually run the stage.



4. Provide the expected parameters, which are the pipeline instance and its drop location.

Queue Build "Iteration 2"

General Parameters

Build process parameters:

- 1. Required

Lab Process Settings	To see or edit the details, click ...
----------------------	---------------------------------------
- 2. Basic

Logging Verbosity	Normal
-------------------	--------
- 3. Release pipeline

Next stages in pipeline	
Pipeline Instance - ONLY FOR MAI	0.0.0624.990
Pipeline Instance Drop Location -	it Stage\01 Commit Stage 0.0.0624.990
- 4. Misc

Timeout For Each Deployment Scr	30
---------------------------------	----

Pipeline Instance Drop Location - ONLY FOR MANUALLY TRIGGERED ST
Drop Location of the Pipeline Instance, where items to be deployed/tested were left by the commit stage. USE IT ONLY FOR MANUALLY TRIGGERED S...

Queue Cancel

5. Use Build Explorer to monitor the stage. The stage should pass if the environment is now fixed. The following two screenshots show that the stage runs and that it passes.

Acceptance Test Stage – for retries Start Page Source Control Explorer Build Explorer - Iteration 2

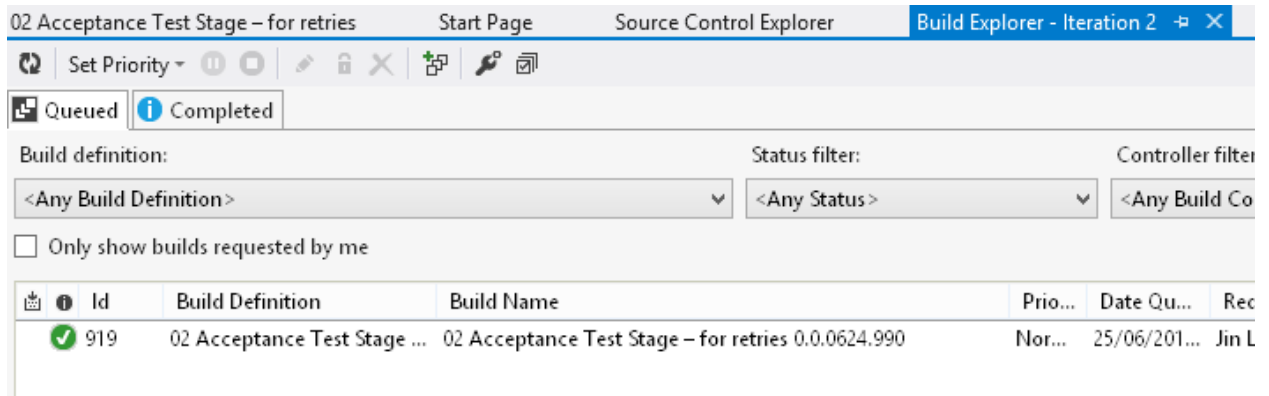
Set Priority [Icons]

Queued [i] Completed

Build definition: Any Build Definition > Status filter: <Any Status> Controller filter: <Any Build C

Only show builds requested by me

Id	Build Definition	Build Name	Prio...	Date Qu...	Re
919	02 Acceptance Test Stage ...	02 Acceptance Test Stage – for retries 0.0.0624.990	Nor...	25/06/201...	Jin



- After the stage has finished, you can either delete the cloned build definition or leave it just in case you need it again.

Follow the same procedure for manually triggered stages that fail.

Task 2: Retriggering the Commit Stage

In this task you learn how to retrigger the commit stage. The commit stage differs from the other stages because it always uses the latest changes in version control. If you want to retry a specific change, but not the latest change, you need to tell the pipeline which check-in to use.

- Manually queue the commit stage build definition.
- Provide the change to use. You can either specify the label (by default, TFS labels the code being built) or the changeset number in the **Queue Build** dialog box. Enter it in **Build process parameters > 2. Advanced > Get Version**. Use the format **C<Changeset number>** or **L<Label name>** or. The following two screenshots show an example of each method.

Queue Build "Iteration 2"
?
X

General
Parameters

Build process parameters:

1. Basic

Clean Workspace	All
Logging Verbosity	Normal
Perform Code Analysis	Always
Source And Symbol Server Setting	Index Sources

2. Advanced

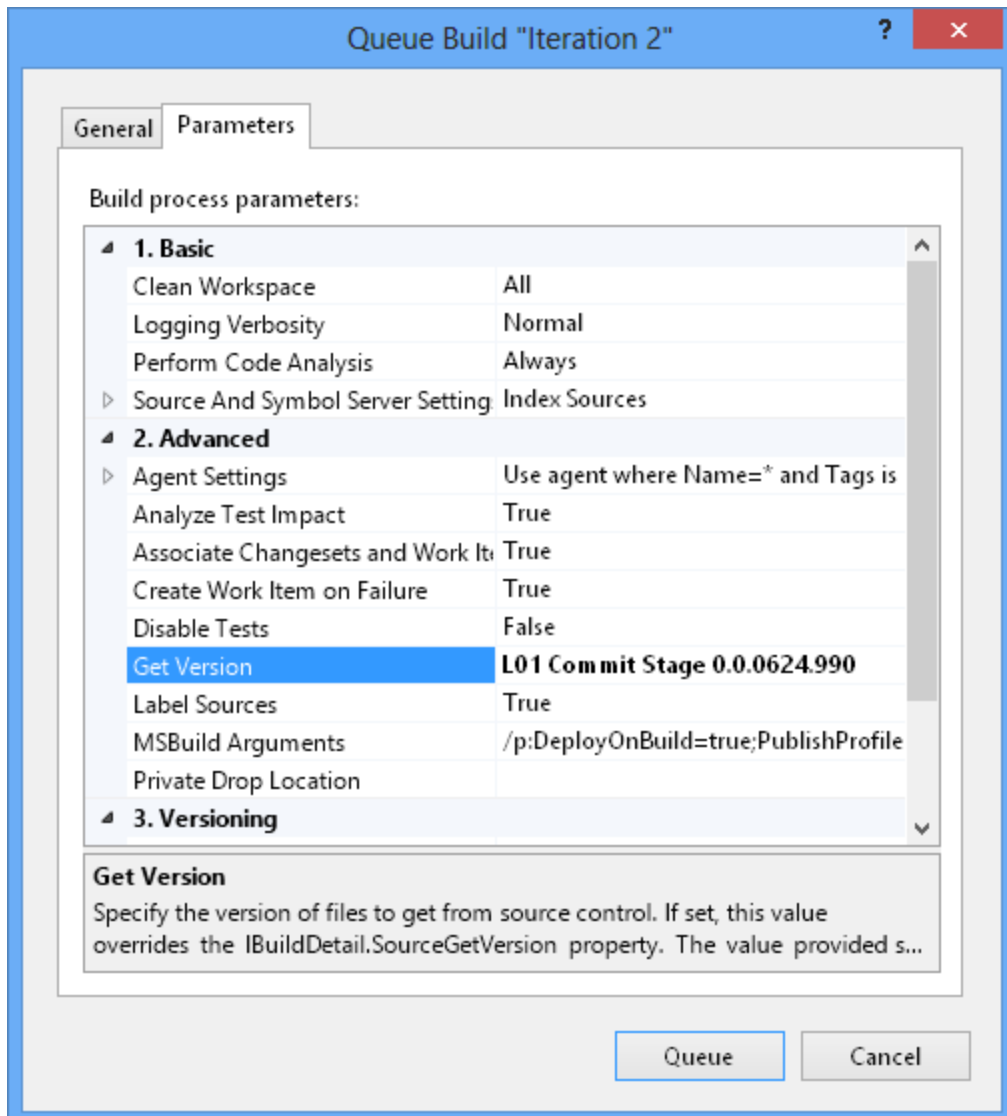
Agent Settings	Use agent where Name=* and Tags is
Analyze Test Impact	True
Associate Changesets and Work Items	True
Create Work Item on Failure	True
Disable Tests	False
Get Version	C121
Label Sources	True
MSBuild Arguments	/p:DeployOnBuild=true;PublishProfile
Private Drop Location	

3. Versioning

Get Version

Specify the version of files to get from source control. If set, this value overrides the IBUILDDETAIL.SourceGetVersion property. The value provided s...

Queue
Cancel



Exercise 3: Dealing with Bugs

In this exercise you learn some general approaches to use when a stage of the pipeline fails because of a bug in the code. This exercise is not a step-by step procedure but a set of guidelines about what you can do.

Stop the Pipeline and Identify the Cause of the Failure

If the pipeline stops, investigate the build summary, build details and logs to find information about the failure. If the failure is caused by the code (either the application code or the automated testing code), it should be straightforward to trace the problem. Because different pipeline instances run for different changes, the failure must be caused by the change checked in to version control.

Use the Debugging Symbols

For problems that are difficult to debug, you might find the debuggings symbols that contain all the debugging information to be useful. The commit stage generates a debug version as well as a release version. It is located in the same drop folder. Select the binary and program database file (PDB) and deploy it to the appropriate environment.

A potential improvement for the pipeline would be to tokenize the deployment scripts so that they receive the appropriate binary (debug or release) as a parameter. This would mean that deployments are also automated for debugging as well as for standard tests and validations.

Fix the Bug and Run the Pipeline

After you fix the code and check it in, a new instance of the pipeline is created. A continuous delivery pipeline should be treated as a one way system for code (as opposed to environments). You shouldn't try to rerun a single stage. Instead, the pipeline should complete its entire run, from beginning to end. If you don't do this, you can't be sure that your fix has passed all the validations that the pipeline performs.

Summary

In this lab you learned how to monitor instances of the pipeline and what to do when there are problems, either with the environment or the code. You first learned how to use Build Explorer to track the progress of a pipeline instance, associate stages that belong to the same pipeline instance, and look at the build details for a particular stage.

You then learned how to retrigger a failed stage that was not a commit stage when there were problems with an environment. You then adapted that technique so that you could retrigger a commit stage.

Finally, you learned a general approach to addressing bugs in the code.

Copyright

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet website references, may change without notice. You bear the risk of using it. Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

© 2013 Microsoft. All rights reserved.

Microsoft, Windows, Windows Server, Windows Vista, Windows Azure, Windows PowerShell, Silverlight, Expression, Expression Blend, MSDN, IntelliSense, IntelliTrace, Internet Explorer, SQL Azure, SQL Server, Visual C#, Visual C++, Visual Basic, and Visual Studio are trademarks of the Microsoft group of companies.

All other trademarks are the property of their respective owners.