Hands-on Lab 4.1:
Monitoring the Continuous Delivery Pipeline

Table of Contents

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.

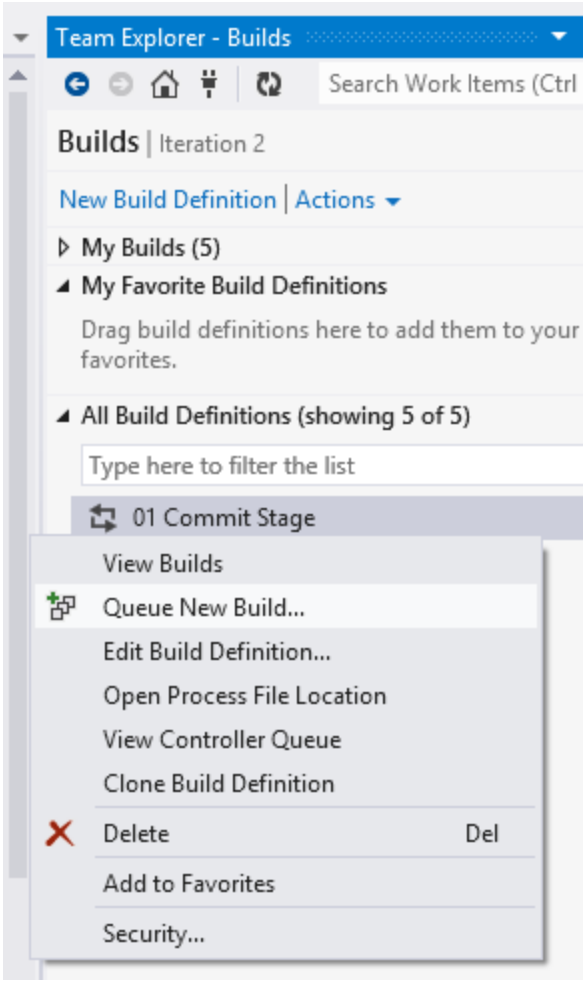
Task 1: Trigger and Monitor Pipeline Instances

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

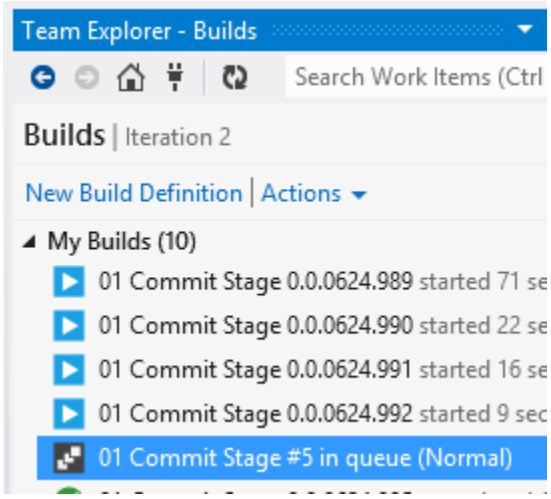
- Manually queue the 01 Commit Stage build definition.
- Check in changes to the code.

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

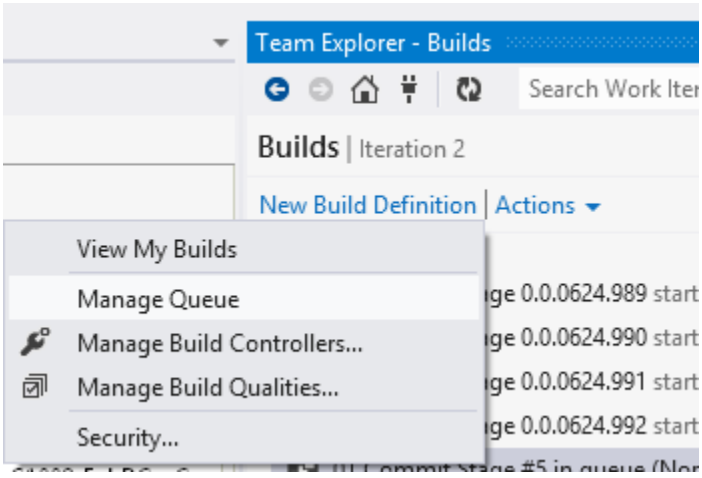
1. Trigger a new instance of the pipeline by queuing the **01 Commit Stage** build definition. In Team Explorer, click **Builds**. Click **All Build Definitions**. Click **01 Commit Stage**.



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.)

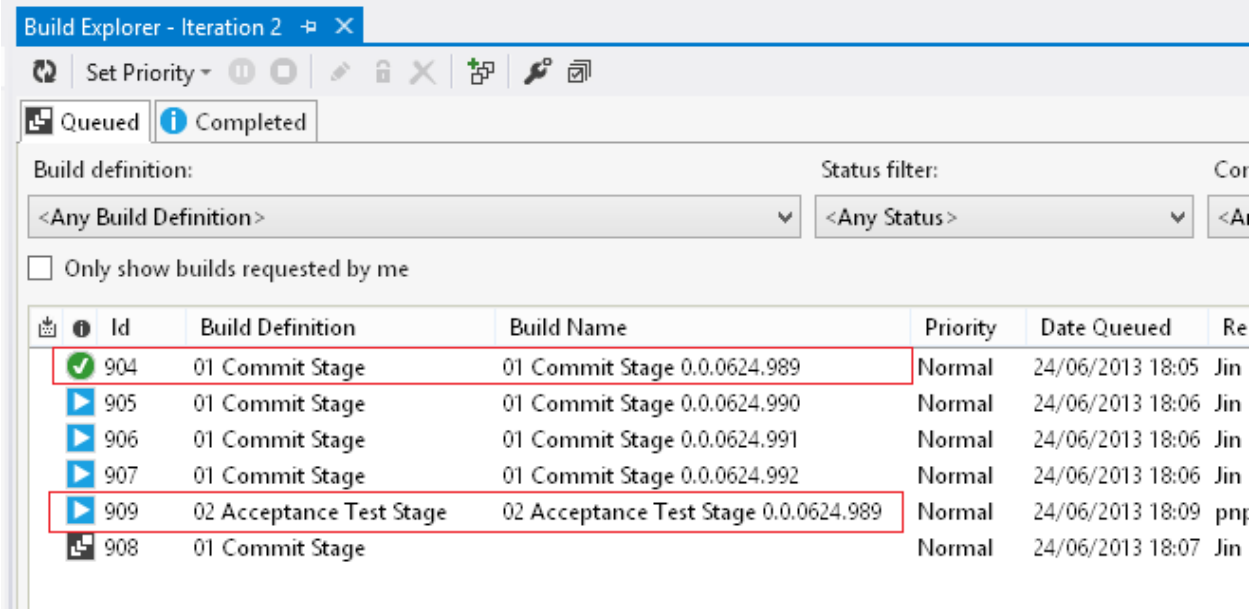


3. In **Team Explorer – Builds**, click **Actions**. Select **Manage Queue** from the drop-down list. **Build Explorer** opens. This is the main monitoring tool for the pipeline.

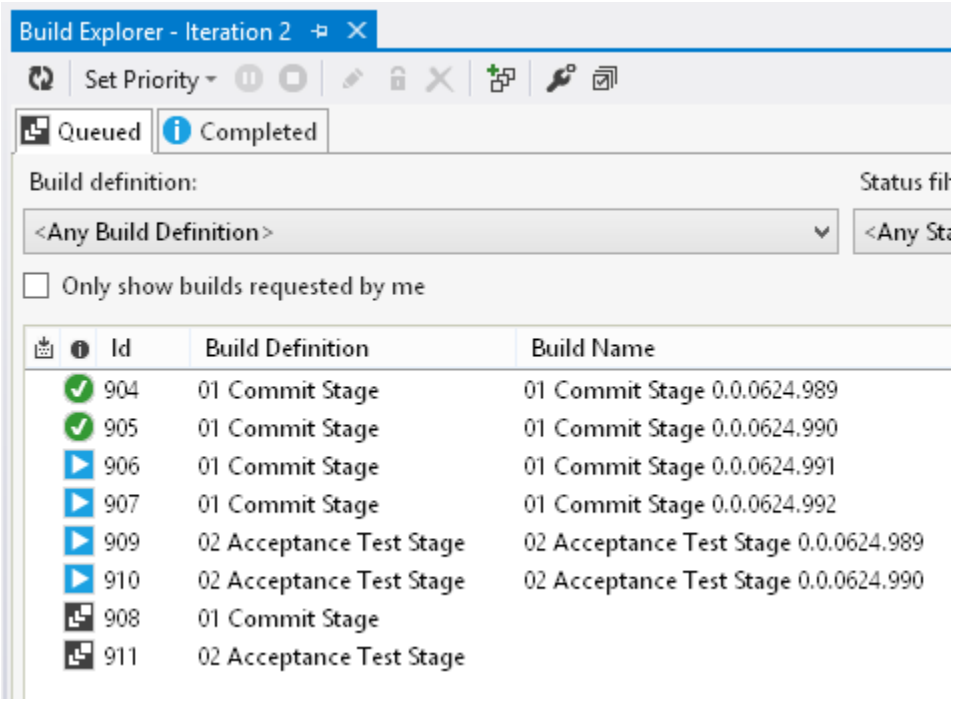


TIP: Most of the activities that you can perform with Build Explorer can also be done by selecting the **BUILD** tab from the Web Access portal associated with the team project.

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.

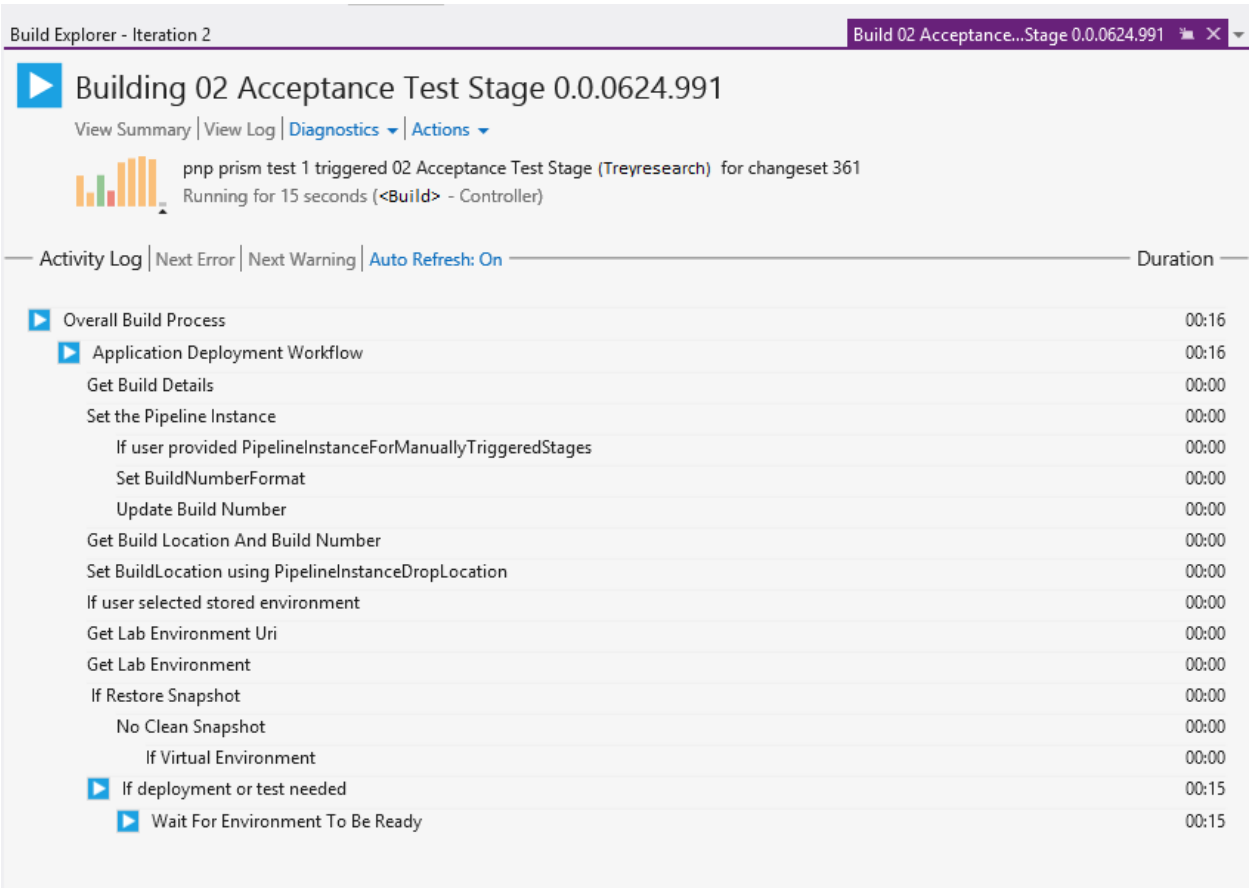


4. New acceptance test stages are queued as soon as the commit stages in the same pipeline instances have finished. They run as long as there are enough build agents available. In the following screenshot, the 0.0.0624.990 instance has advanced and is running its acceptance test stage. However, there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

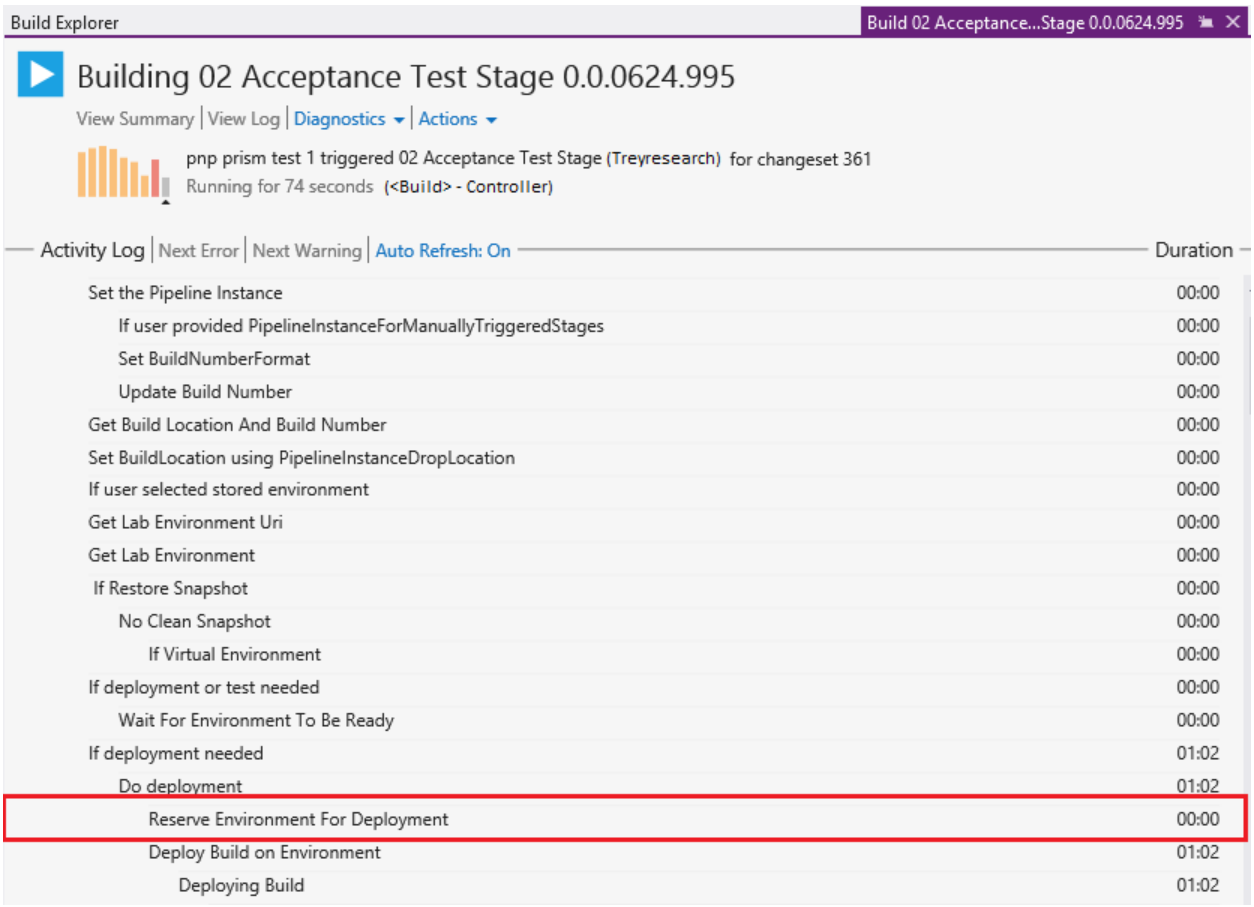


5. All the pipeline stages but the commit stage have dependencies on environments because they perform automated deployments and tests on specific machines. This means that the pipeline

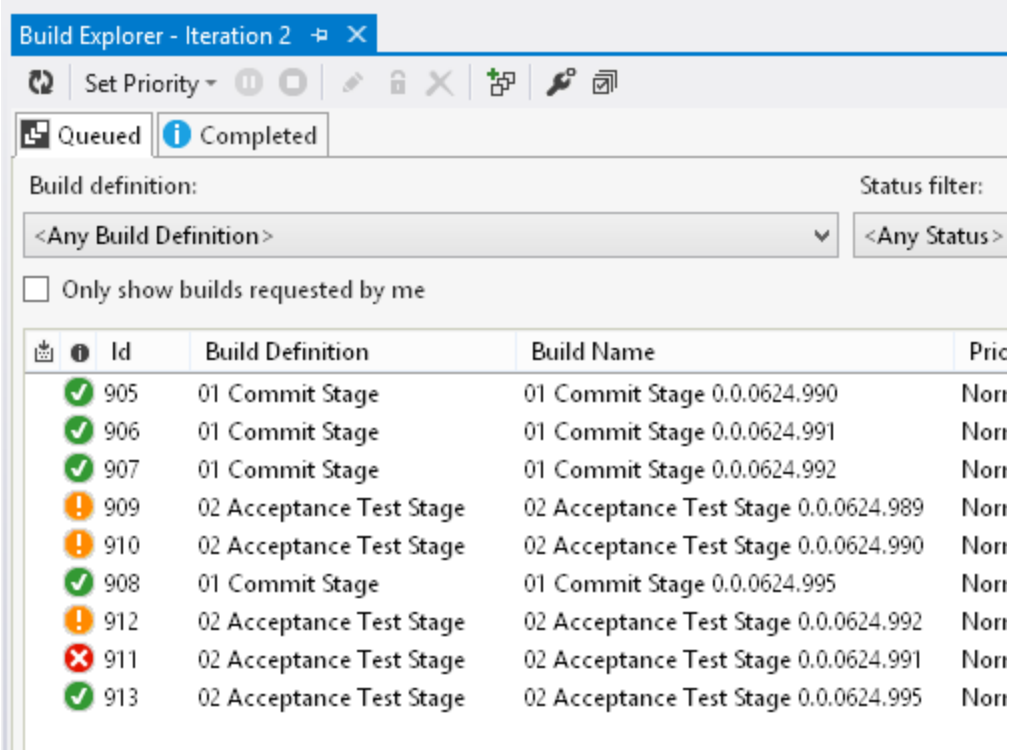
orchestration must ensure that the target environment is not already being used by another pipeline instance. For example if a pipeline instance deploys to an environment where another instance is running tests, the deployment will fail. If you open the details of a running acceptance test stage (select 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.



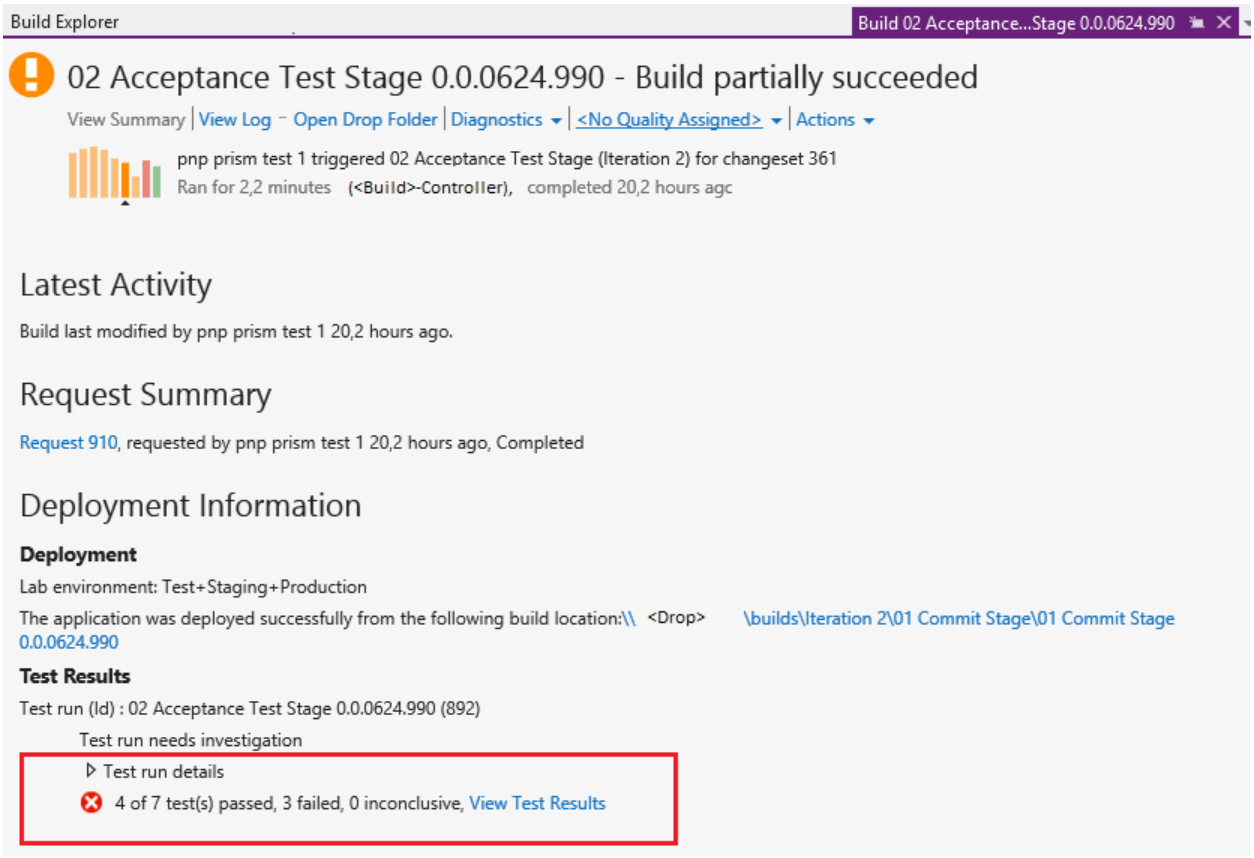
6. As soon as the environment is released, the next blocked stage that is waiting for it will be able to continue. In the following screenshot, the environment is reserved (look for the line **Reserve Environment for Deployment**). This means it is in use and other stages are blocked until it is released.



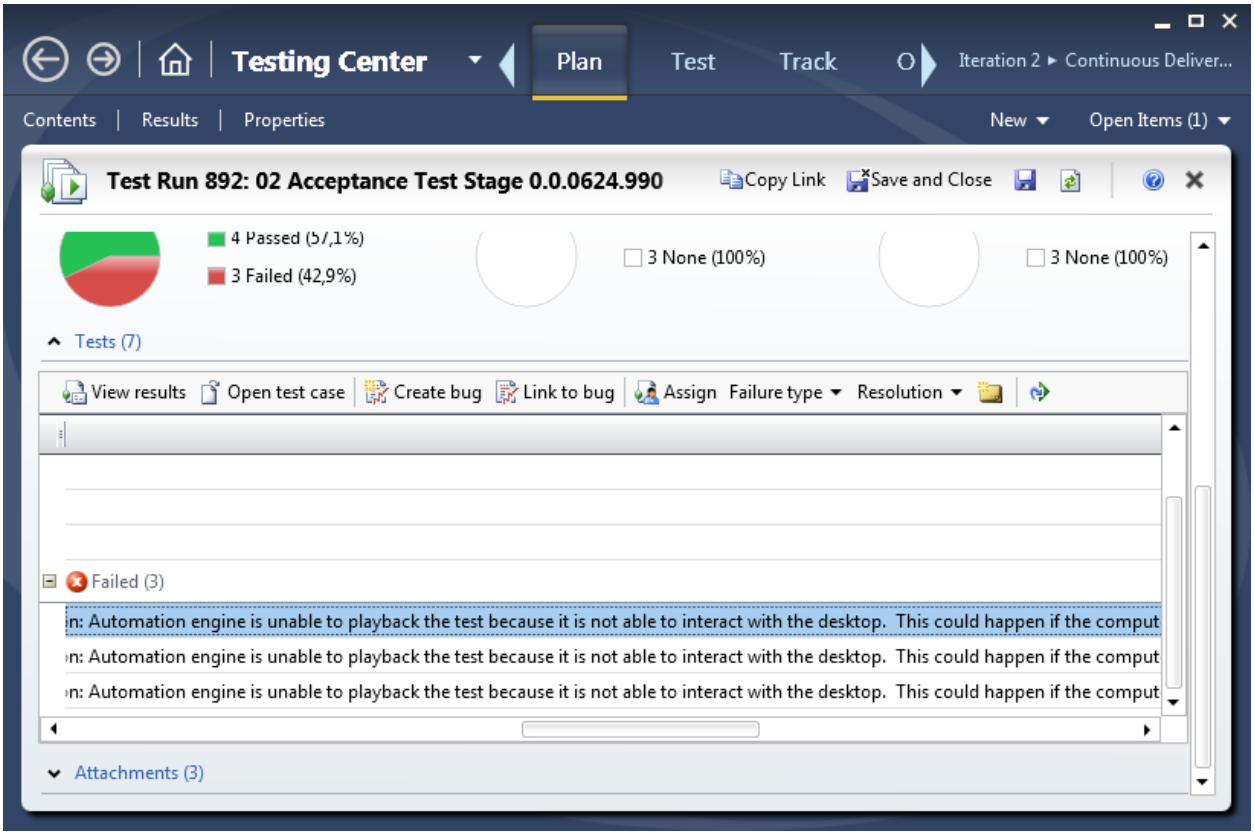
7. After all the stages finish, you can use Build Explorer to see the overall results. You can get details about a particular pipeline instance by selecting any of its stages. The following screenshot shows an example of the overall results.



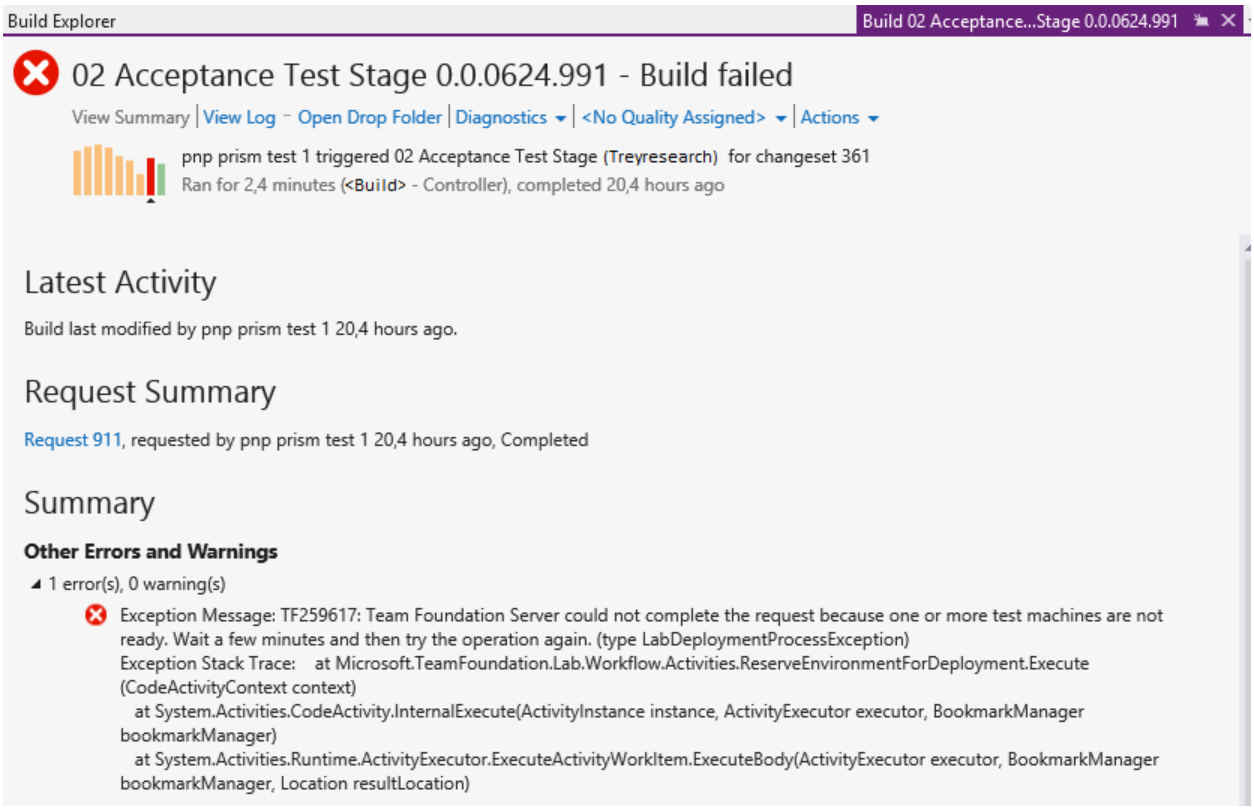
8. If you look at the **0.0.0625.990** instance, you see that its acceptance test stage has not completely succeeded. If you select the stage, you can see more details. These are shown in the following screenshot.



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.

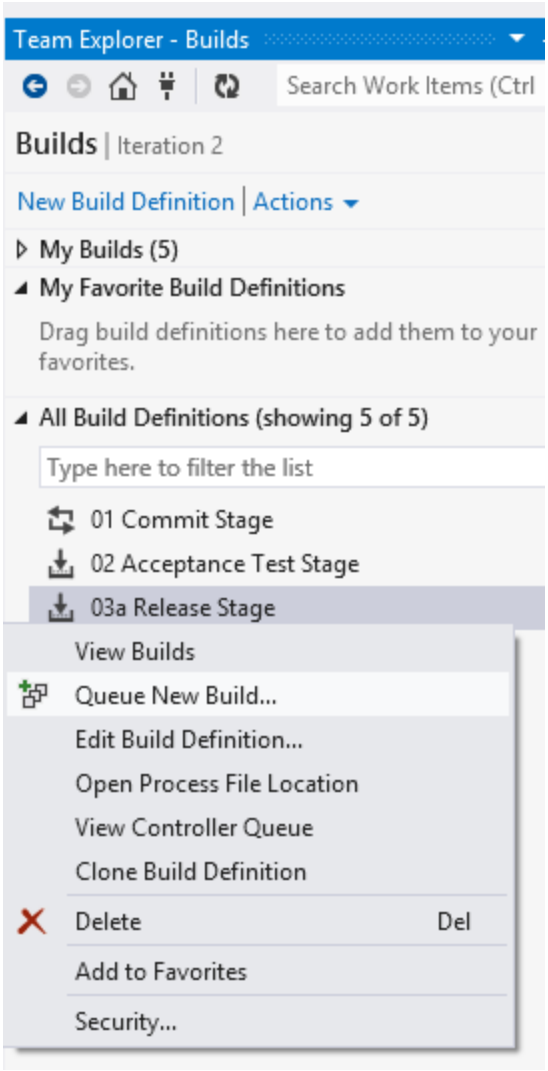


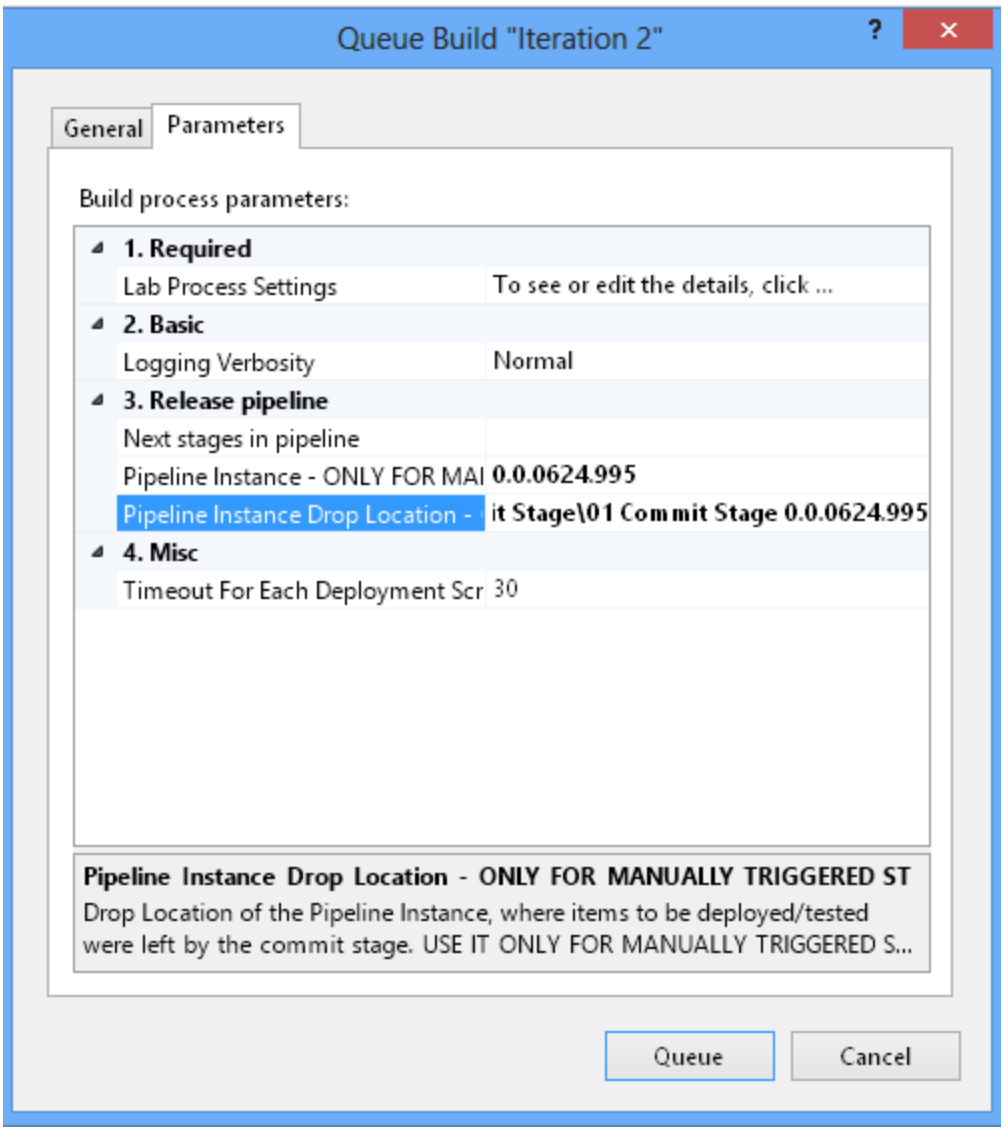
11. The error message indications that there is no active session (either by using the remote desktop or locally) on the computer that runs the UI tests. The UI tests require an active session. After you create the session, rerun the pipeline.
12. Next, examine the reasons why the **0.0.0624.991** acceptance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



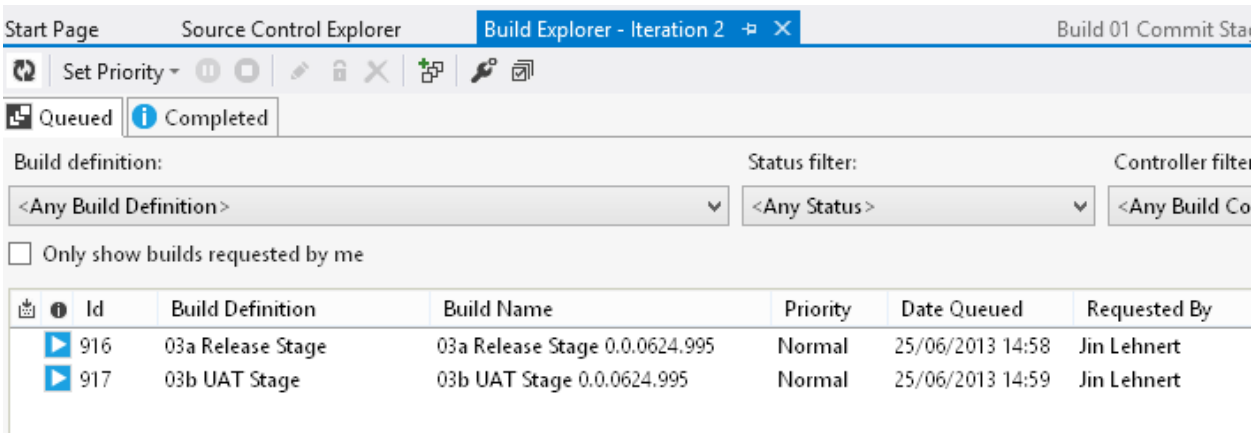
The issue is that the single test controller could not handle the number of pipeline instances that were triggered simultaneously. The instance failed because the step where the acceptance test stage waits for the environment to be ready took too long and the operation timed out.

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.


Team Explorer - Builds
Search Work Items (Ctrl
Builds | Iteration 2
New Build Definition | Actions
My Builds (5)
My Favorite Build Definitions
Drag build definitions here to add them to your favorites.
All Build Definitions (showing 5 of 5)
Type here to filter the list
01 Commit Stage
02 Acceptance Test Stage
03a Release Stage
View Builds
Queue New Build...
Edit Build Definition...
Open Process File Location
View Controller Queue
Clone Build Definition
Delete Del
Add to Favorites
Security...



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.



Exercise 2: Retriggering Failed Stages

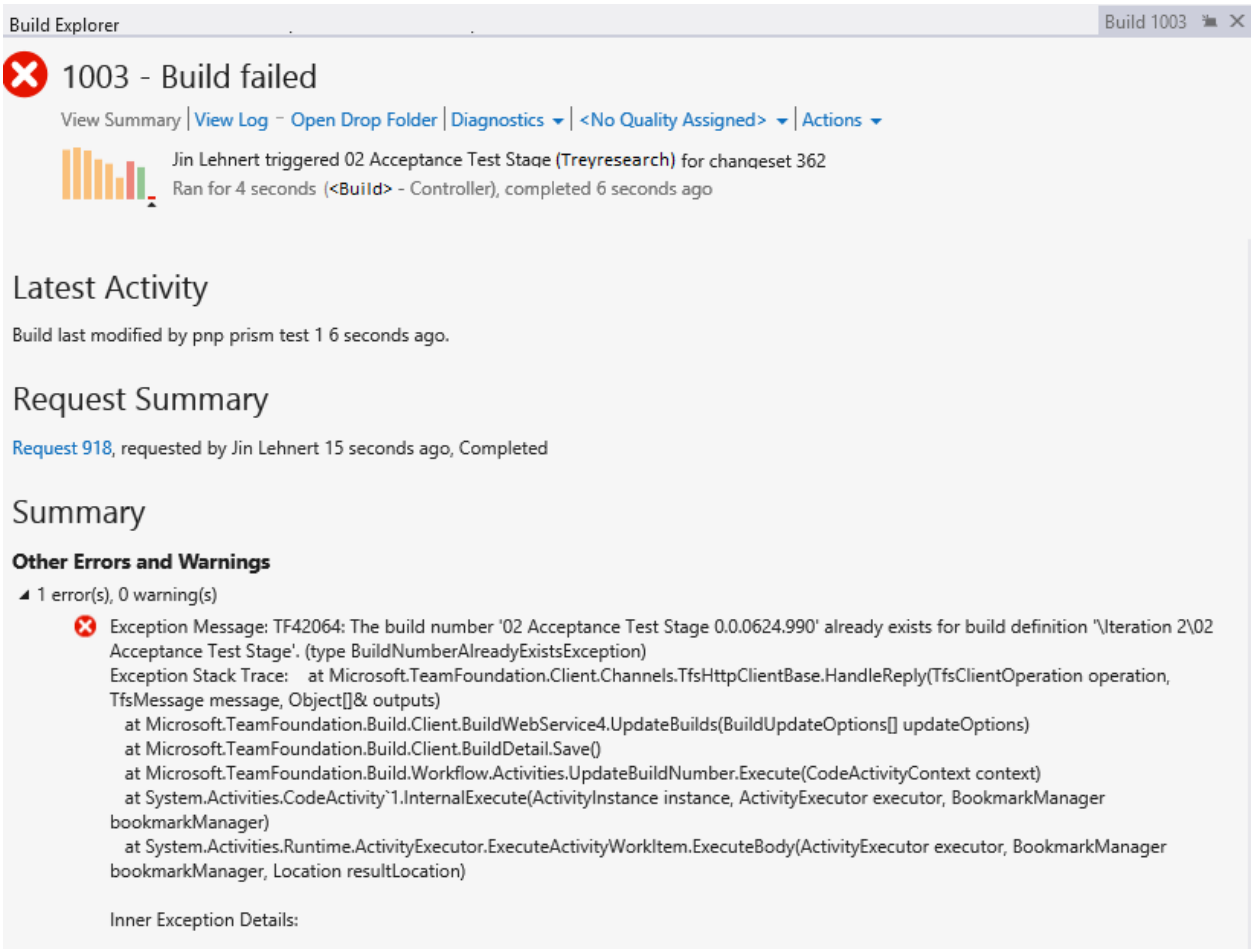
In this exercise you learn how to retrigger a particular stage of the pipeline when there is a failure. If a pipeline instance fails it stops so that the issues aren't propagated to further stages. This happens by default. You then must determine the reason for the failure. There are two possibilities.

- 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

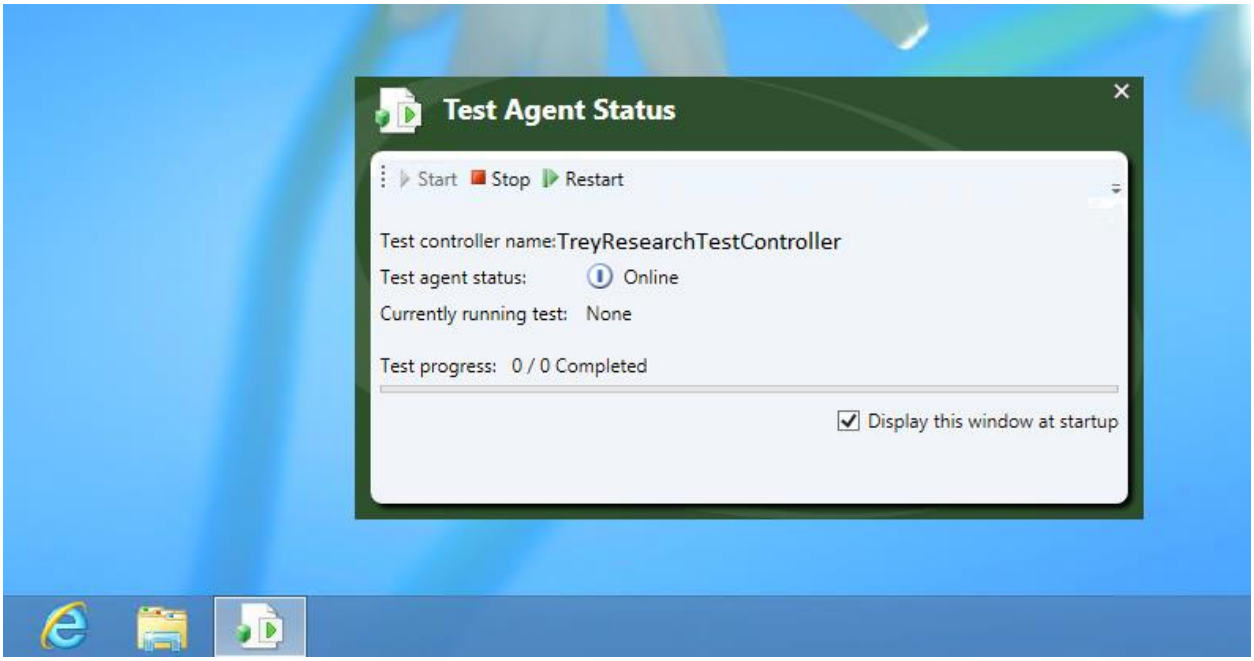
If you want to retrigger a stage that is not the commit stage, you can aways do it manually, no matter whether the stage is a manually triggered one or an automatically triggered one. The problem is that if you run it using the same pipeline instance number, you will get an error because the name of 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.

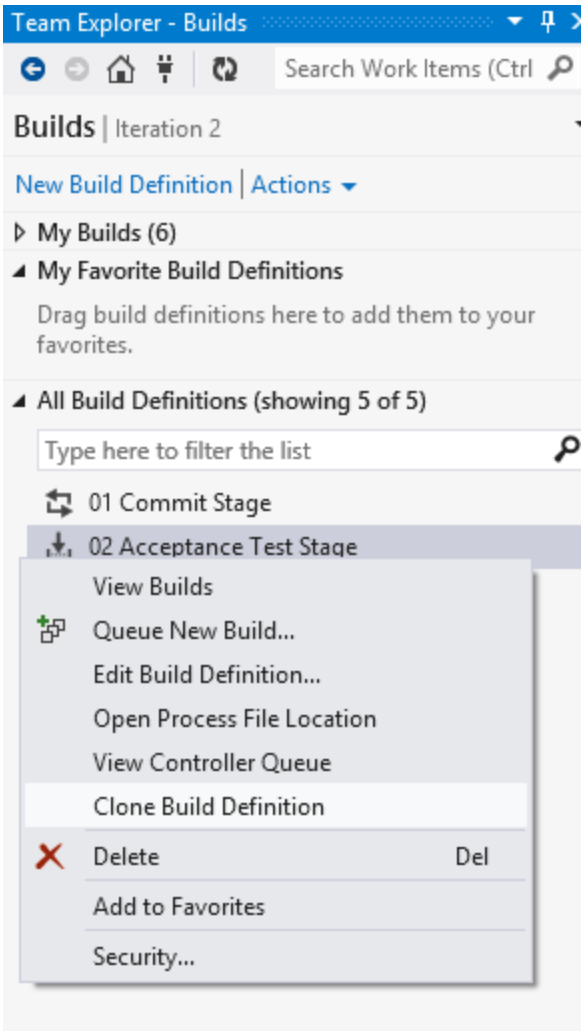


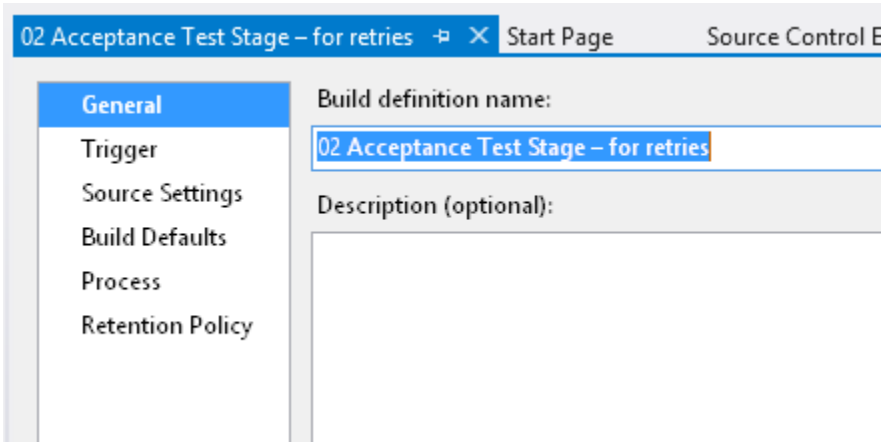
The easiest workaround is to prepare a temporary build definition that mirrors the stage you want to retry, but with a different name, and use it instead of the original one. As a concrete example, here is what to do if you you want to retry an acceptance test stage that only partially succeeded for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

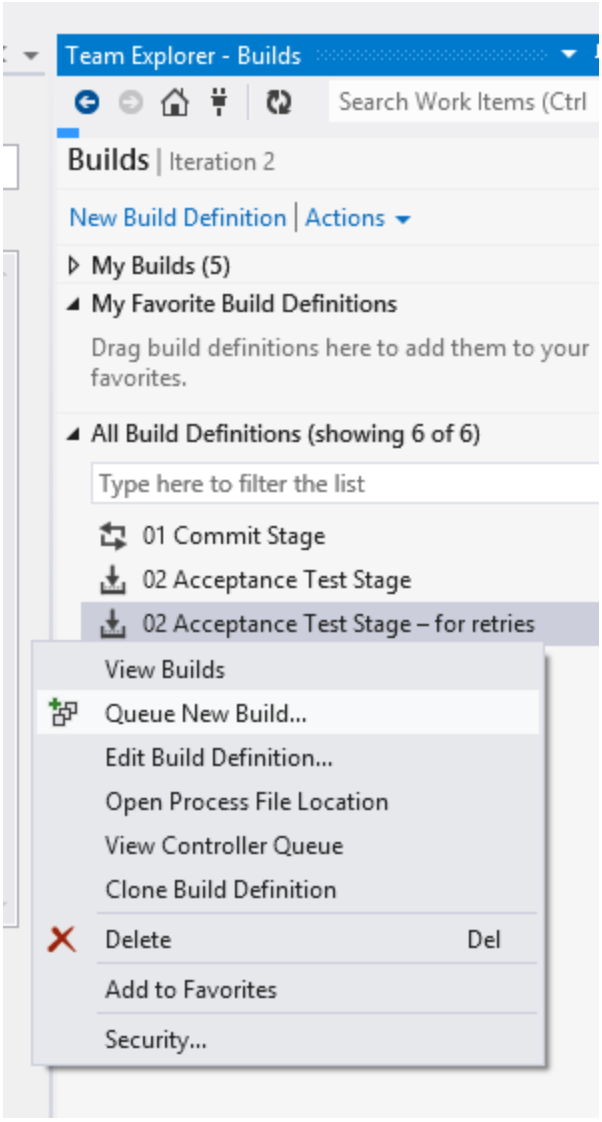


2. From Team Explorer, clone the build definition for the acceptance test stage and give it a different name. In this example, the stage is renamed to **02 Acceptance Test Stage – for retries**. You can enter the name in the build definition editor and then save the build definition. This is shown in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

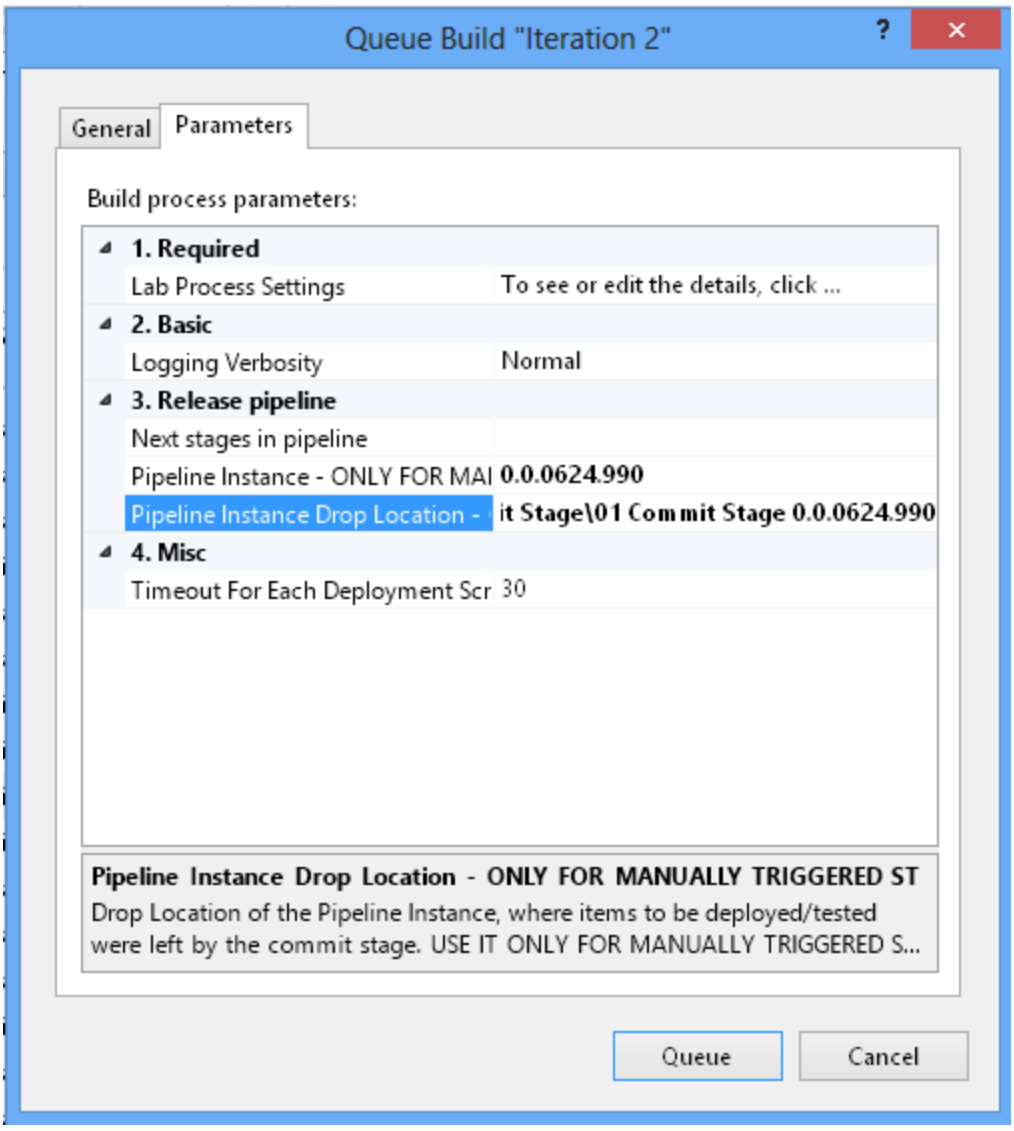




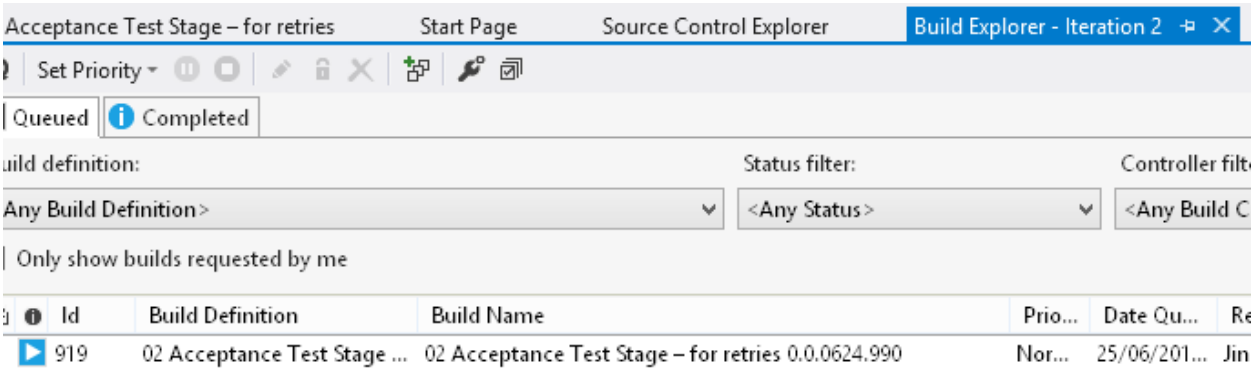
3. Manually run the stage.

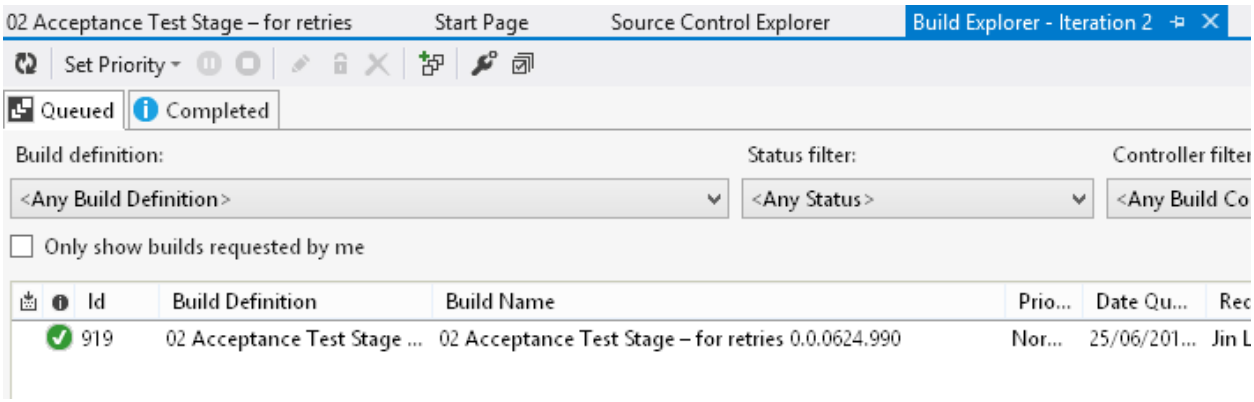


4. Provide the expected parameters, which are the pipeline instance and its drop location.



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.





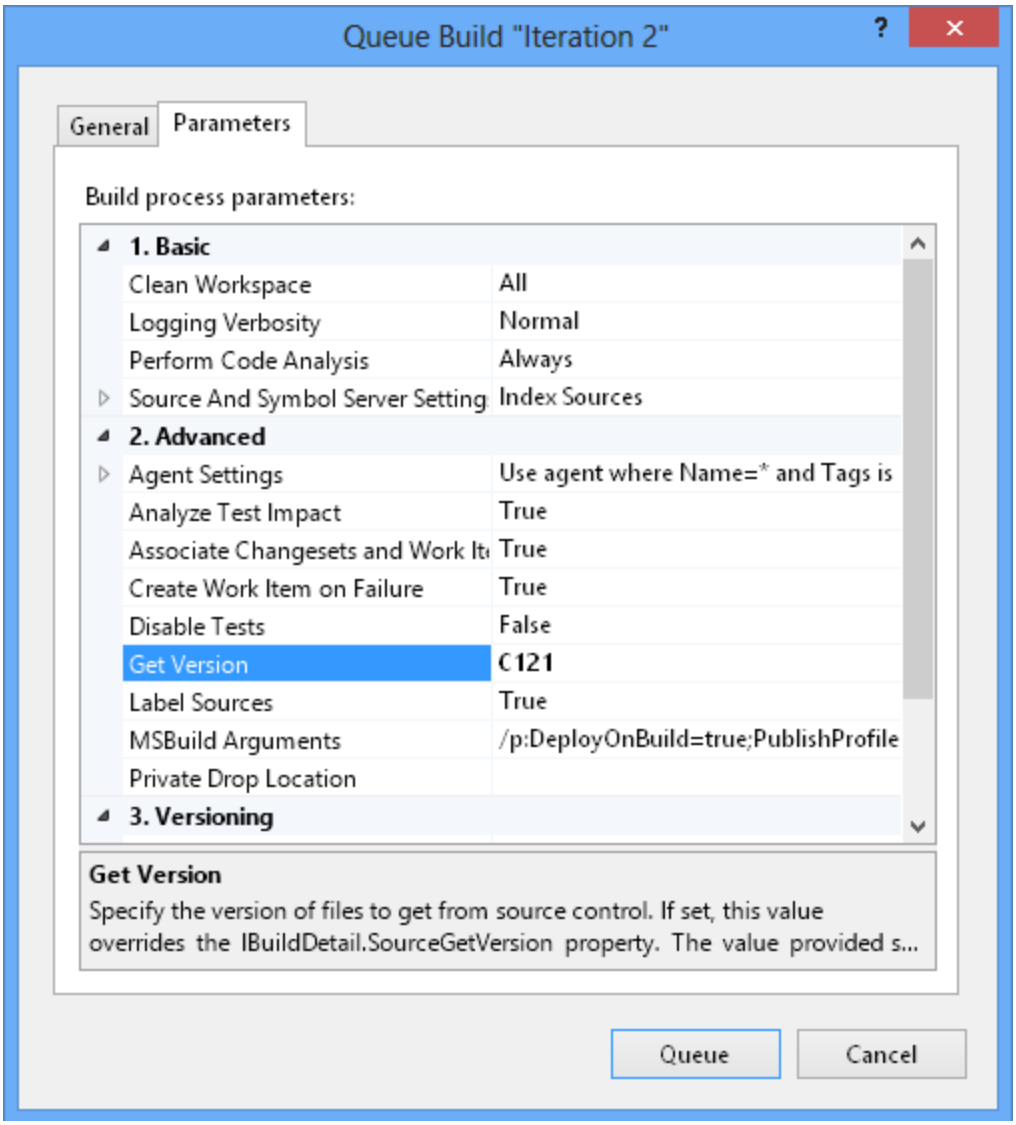
6. After the stage has finished, you can either delete the cloned build definition or leave it just in case you need it again.

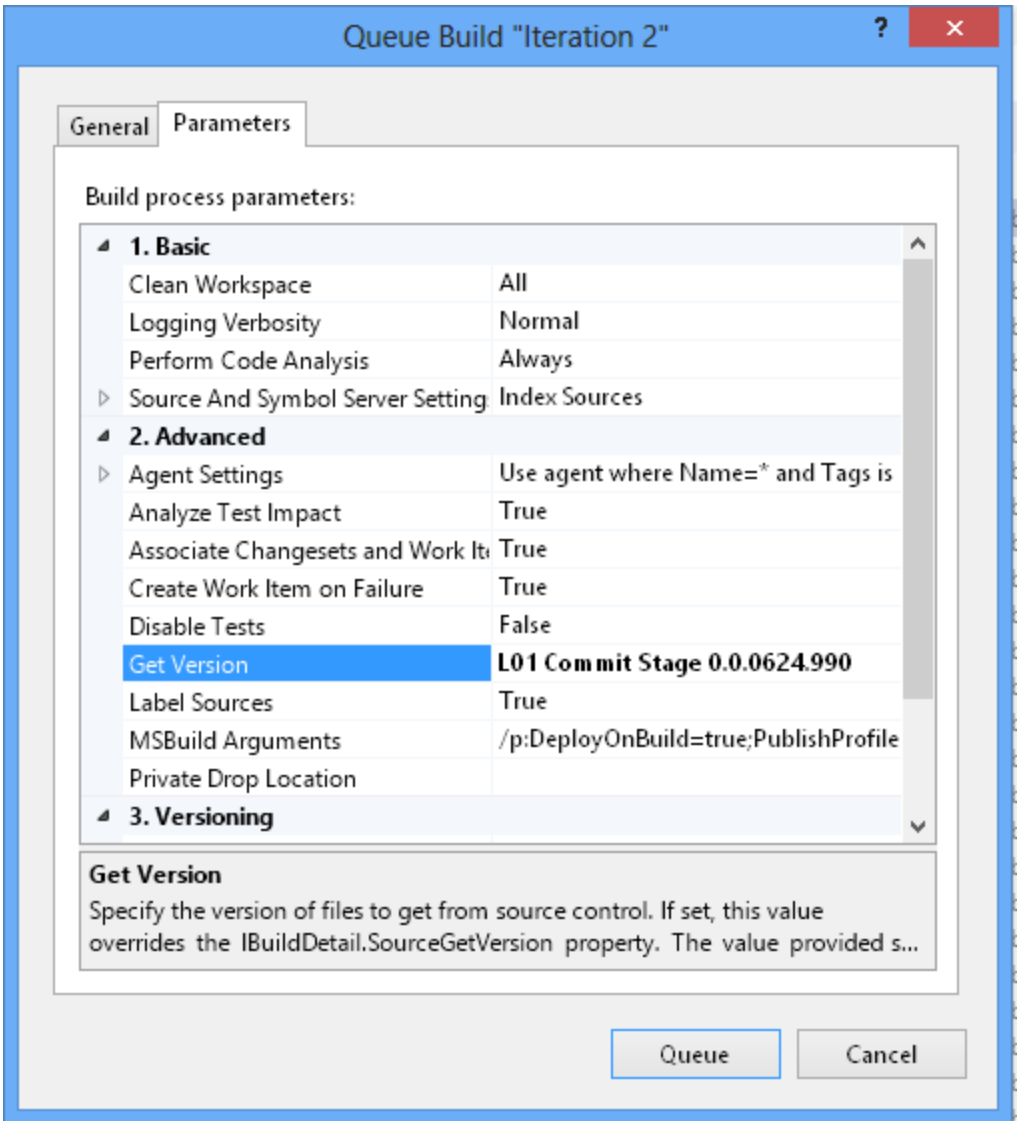
Follow the same procedure for manually triggered stages that fail.

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.

1. Manually queue the commit stage build definition.
2. 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"
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 It
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 thay you could retrigger a commit stage.

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

Copyright