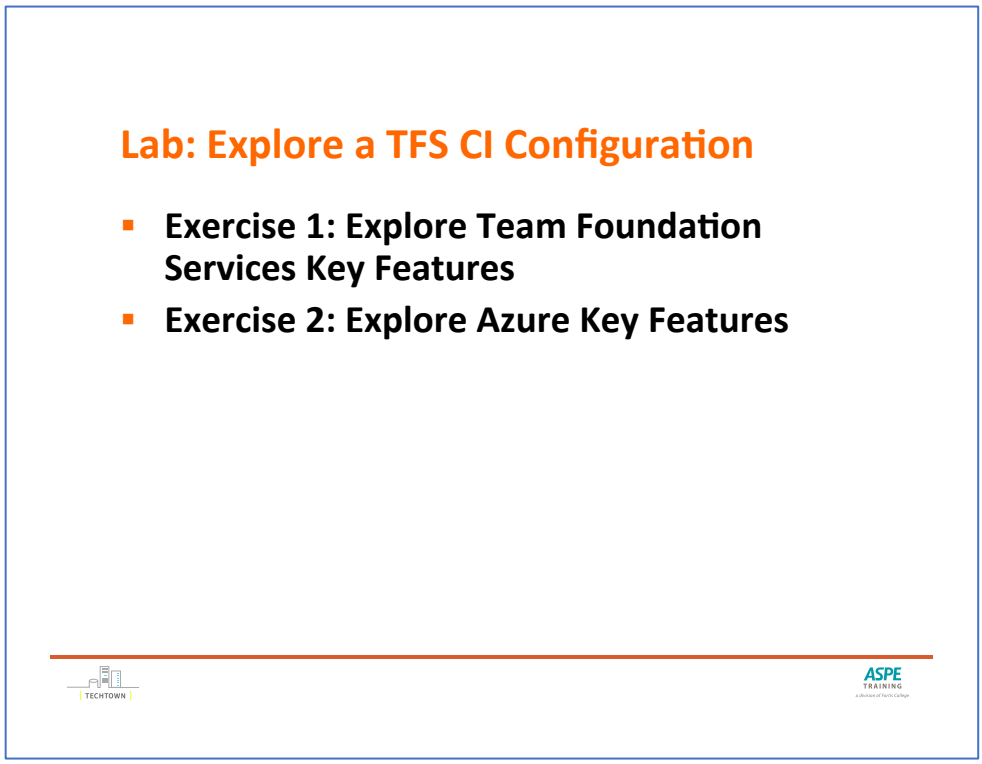
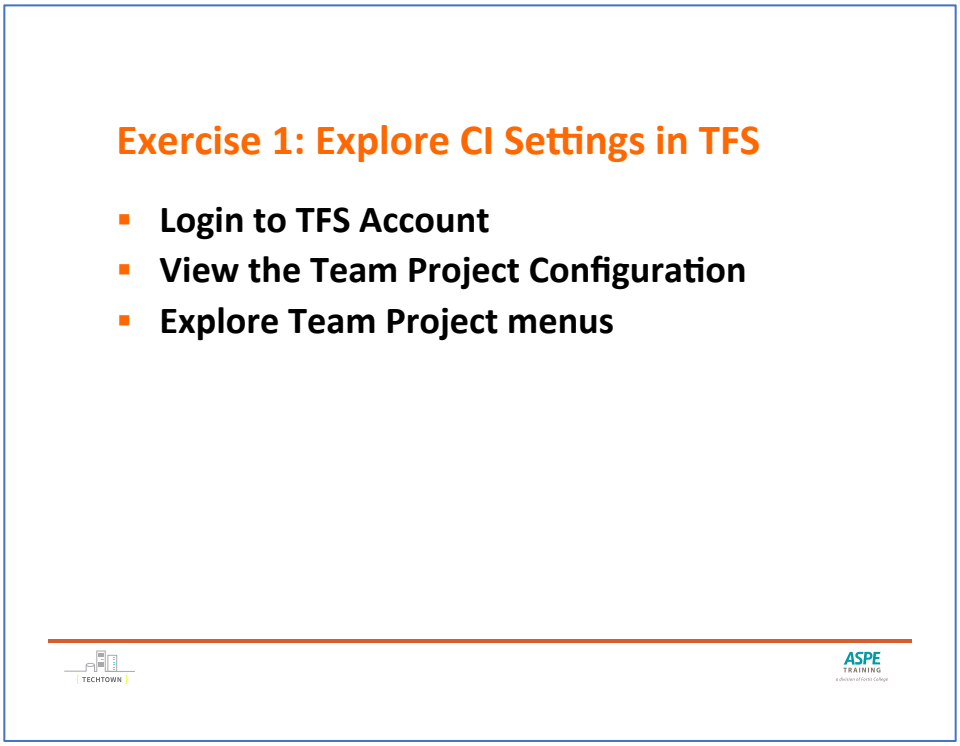
Continuous Delivery Lab Guide

This Guide contains step by step instructions to complete the labs associated with the discussions in class. In most cases the concepts illustrated in the following labs will have been demonstrated by the instructor during the chapter in which the lab appears.

Lab 1 Explore TFS CI Configuration



In Lab 1 we will explore key features of Team Foundation Services including Work Item tracking, Version Control and Build / Release Automation.

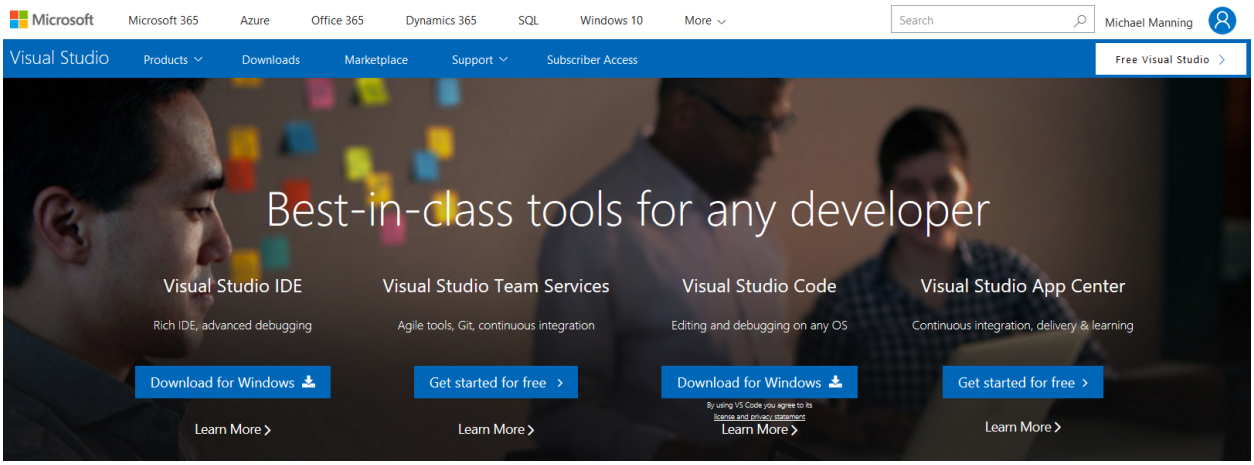


Exercise 1: Explore CI settings in TFS

In this exercise, we will set up an account in **Team Foundation Server** (**TFS**), choose our template, and take quick tour of the various menu items that host different facets of **TFS** functionality.

Login to TFS Account

1. Navigate to **https://devopsclassroom.visualstudio.com/_projects**

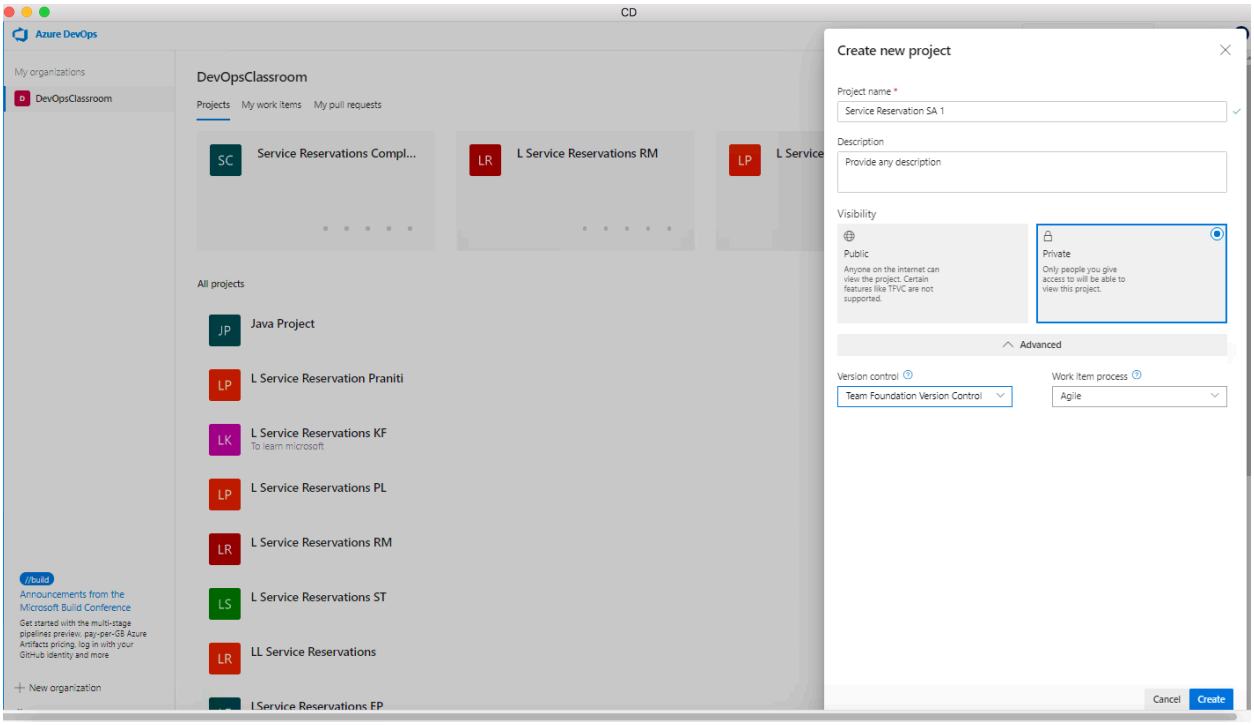


2. Sign in as DevOpsStudent@Outlook.com with the password **JustM300**

Create a new project

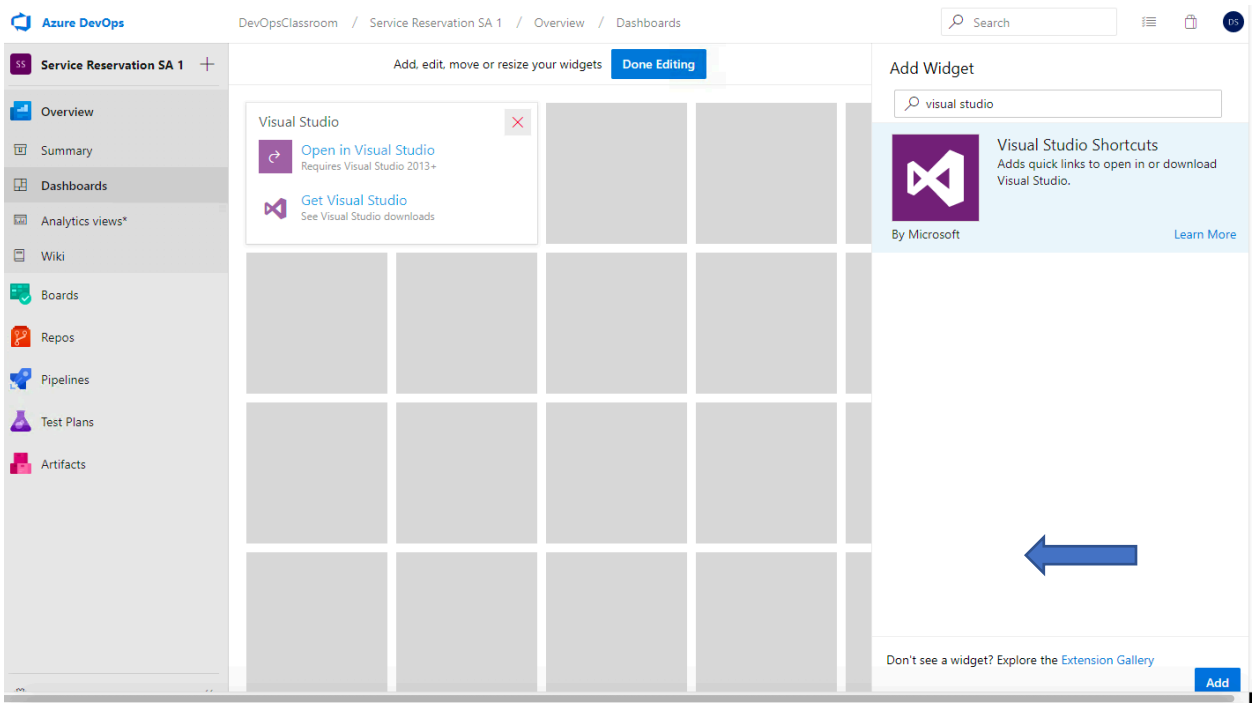
1. Create a new project by clicking the **New Project** button and click **Apply**.
   Note: Be sure to include your full name in the project name to maintain uniqueness as you are sharing project space with the rest of the class.

2. Name your new project **Service Reservations {First Name Last Name}**, choose **Team Foundation Version control** for the Version control, leave the **Work item** process set to **Agile**, and click **Create**

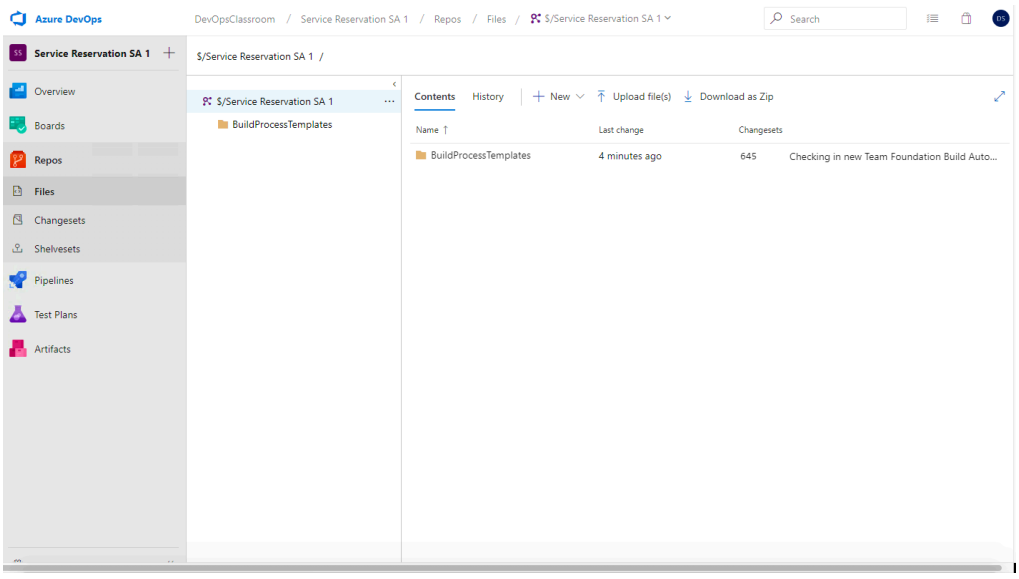


View the Team Project Configuration

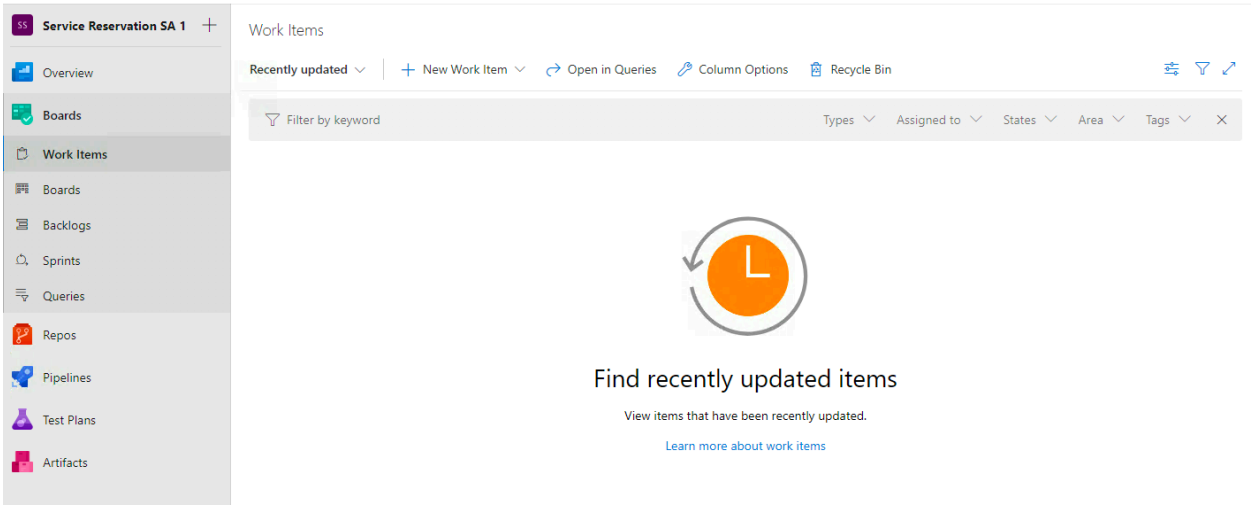
1. Click on the **Dashboards** tab, **Add Widget**, search for **Visual Studio Shortcuts** and add it to the dashboard. You may also add couple more widgets in the dashboard.
2. After adding all widgets, do **done editing**



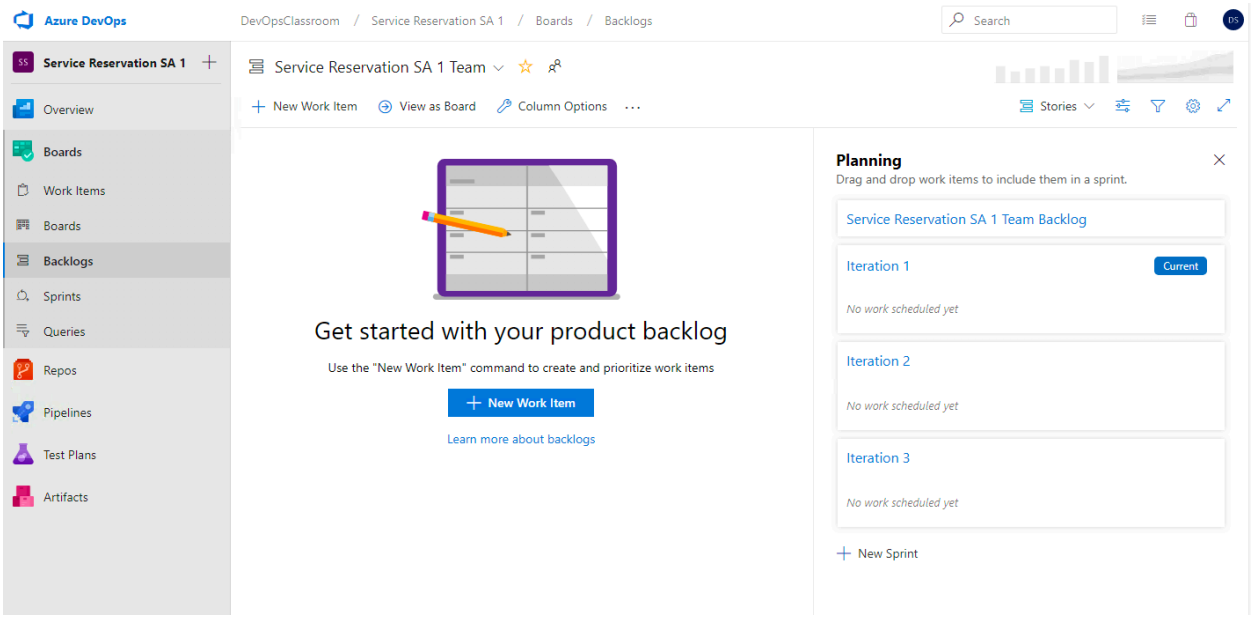
3. Click on the **Repo** tab. This is where the source code repository is, the **Team Foundation Server** version control where you upload code & check it into source control.



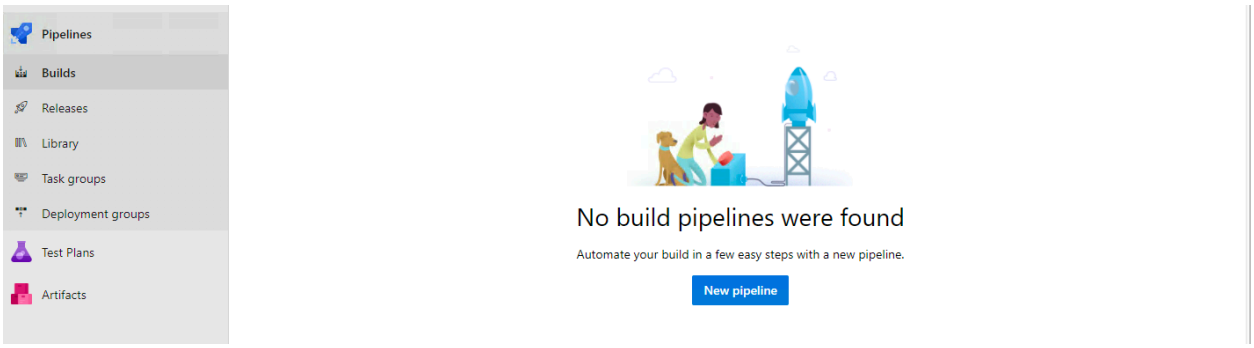
4. Click on the **Boards** tab. This is where all work items are maintained



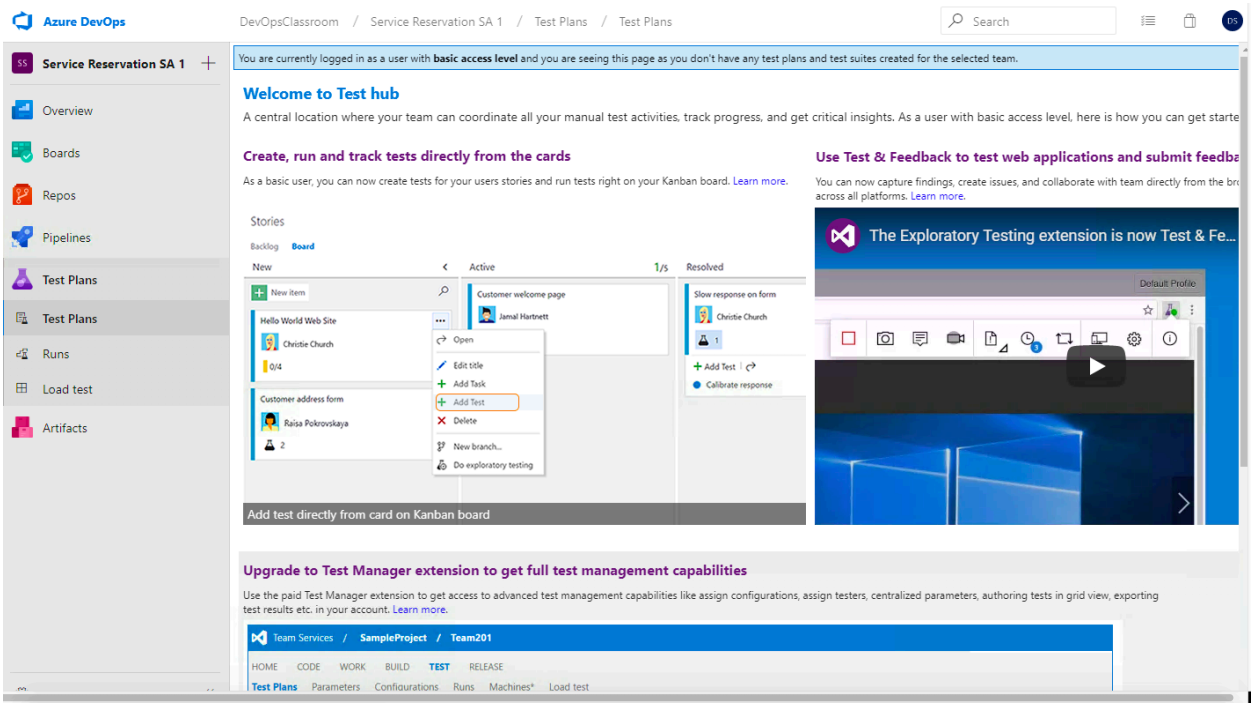
5. Click the **Backlogs** menu in the **Boards** tab. This is where we add new **User Stories** to our product backlog. This is also where we upload these **User stories** to the iterations of our sprints.



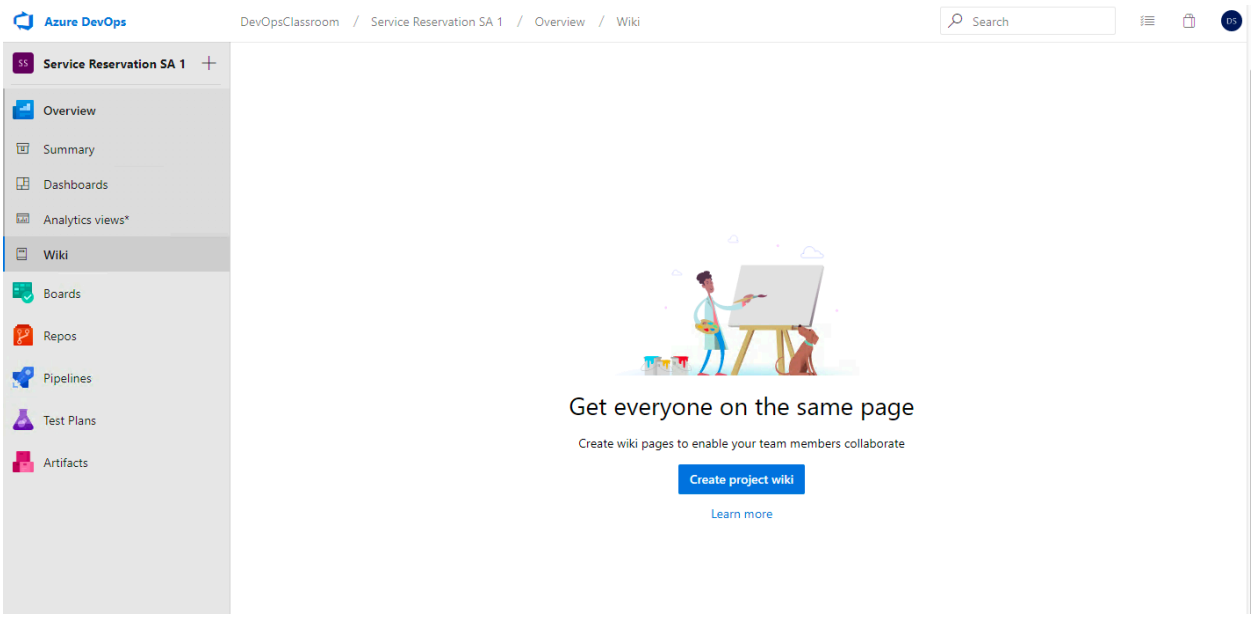
6. Click on the **Pipelines** tab. This is where we will create build definitions.

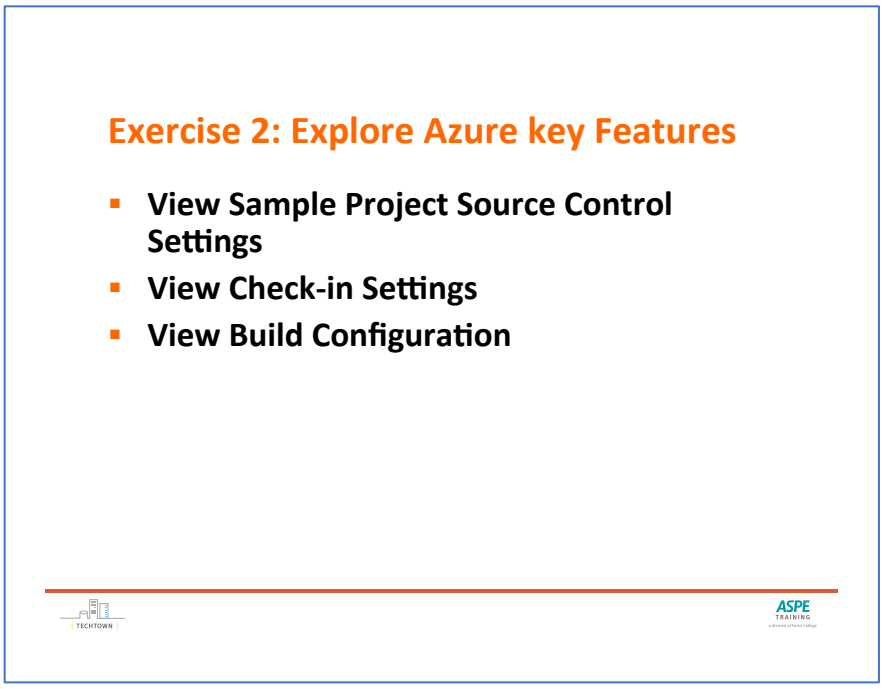


7. Click on the **Test Plans** tab. Here, you can create **Test plans**, do **Test runs**, and **Load tests**.



8. **Wiki**, In the **Overview** tab, click on the **Wiki** tab. If we have any issues or information that we want to share with our team, we can create a **Wiki** here. If we have a new way to automate something that will save everybody time, or we have discovered a more efficient process for product delivery, it can be documented here.


Azure DevOps
DevOpsClassroom / Service Reservation SA 1 / Overview / Wiki
Search
DS
Service Reservation SA 1
Overview
Summary
Dashboards
Analytics views*
Wiki
Boards
Repos
Pipelines
Test Plans
Artifacts
Get everyone on the same page
Create wiki pages to enable your team members collaborate
Create project wiki
Learn more



Exercise 2 – Explore Azure key Features

In this exercise, we take a brief look at our dashboard options, the code repository where we will be checking in our work, and the configuration areas which will allow us to define our builds and create tests.

View Sample Project Source Control Settings

1. Click on the **Dashboards** tab, and notice the tab in the lower right corner that has an option to open the project in **Visual Studio** in the lower right corner.

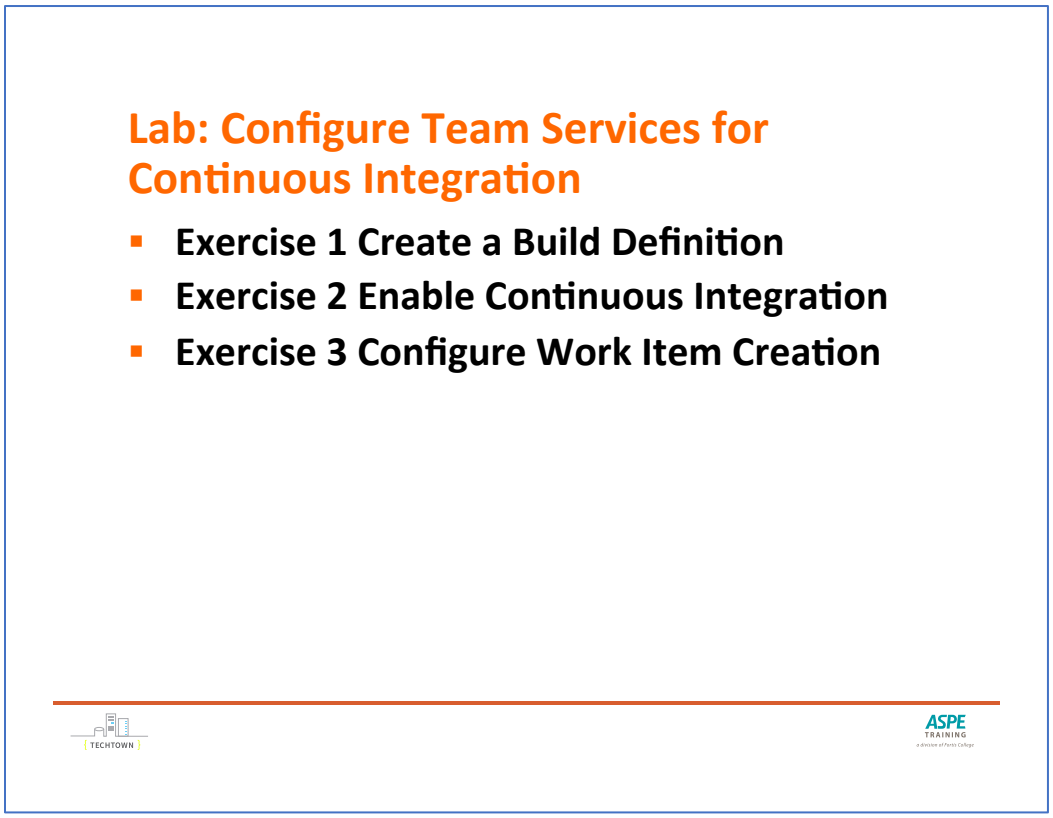
View Check-in Settings

1. Click on the **Repos** tab. This is where the source code repository control where you upload code & check it into source control.
2. Click on the **Boards** tab and click the **Backlogs** menu. This is where we add new **User Stories** to our product backlog. This is also where we these **User stories** to the iterations of our sprints.

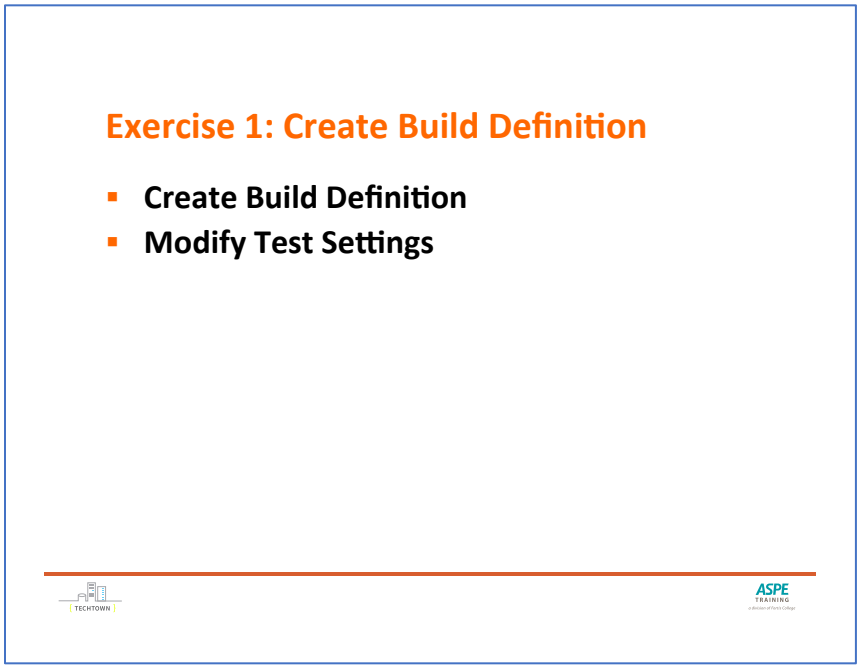
View Build Configuration

1. Click on the **Pipelines** tab. This is where we will create build definitions.
2. Click on the **Test Plans** tab. Here, you can create **Test plans**, do **Test runs**, and **Load tests**.

Lab 2 – Configure TFS for Continuous Integration



In this Lab we will configure **Team Foundation Services** for Continuous Integration. We will also configure **Gated Check-Ins** to restrict code that does not build or does not pass all tests from being allowed into source control. In addition, we will have the system automatically assign a new work item (**Bug**) to the user that checked bad code into Source / Version control.



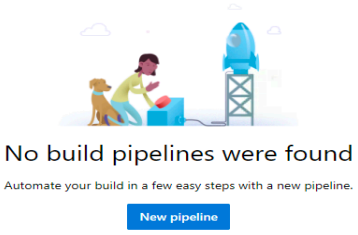
Exercise 1: Configure Build Trigger

In this exercise, we will configure a Continuous Integration Build Trigger in **Team Foundation Services**.

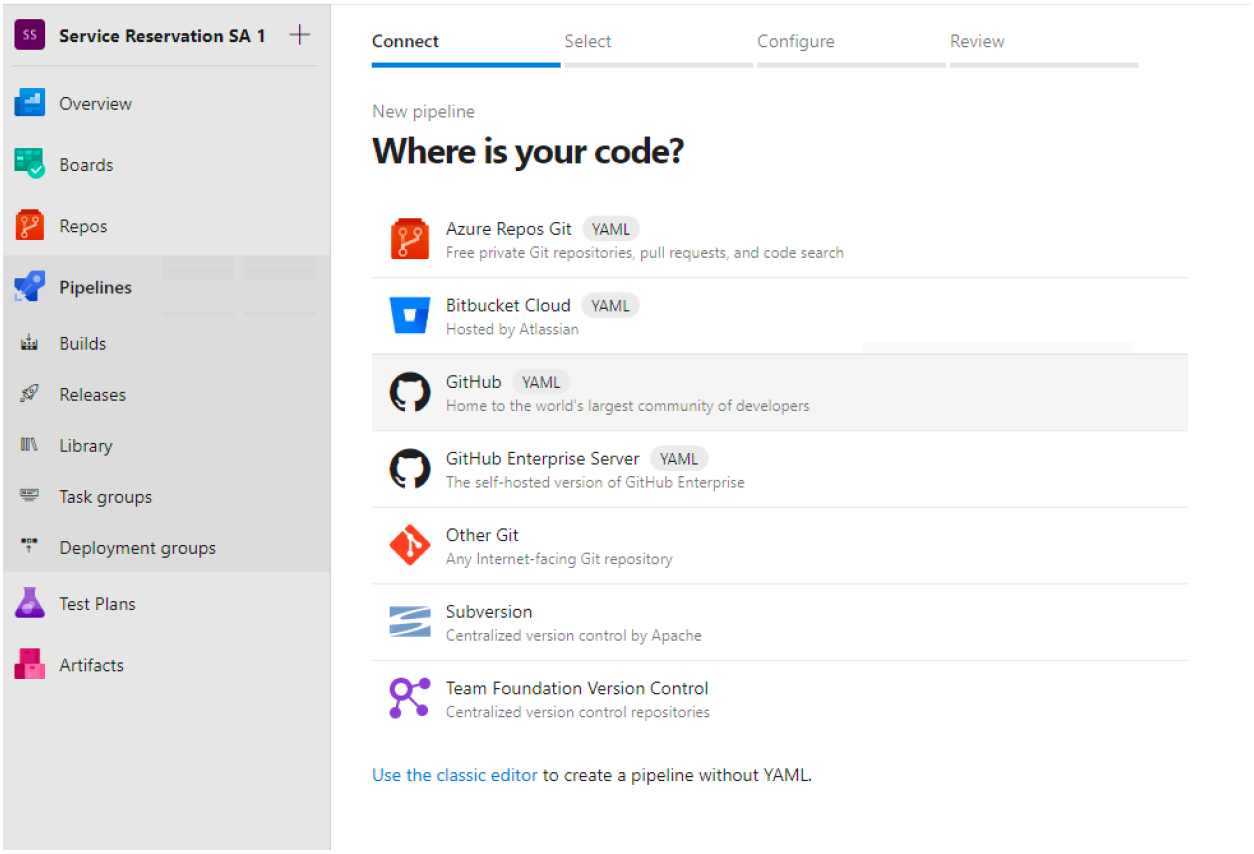
Note: You are continuing in the same Service Reservation Team Project that you created in the previous Lab in Team Foundation Services Online. (http://DevOpsClassroom.VisualStudio.com)

Create Build Definition

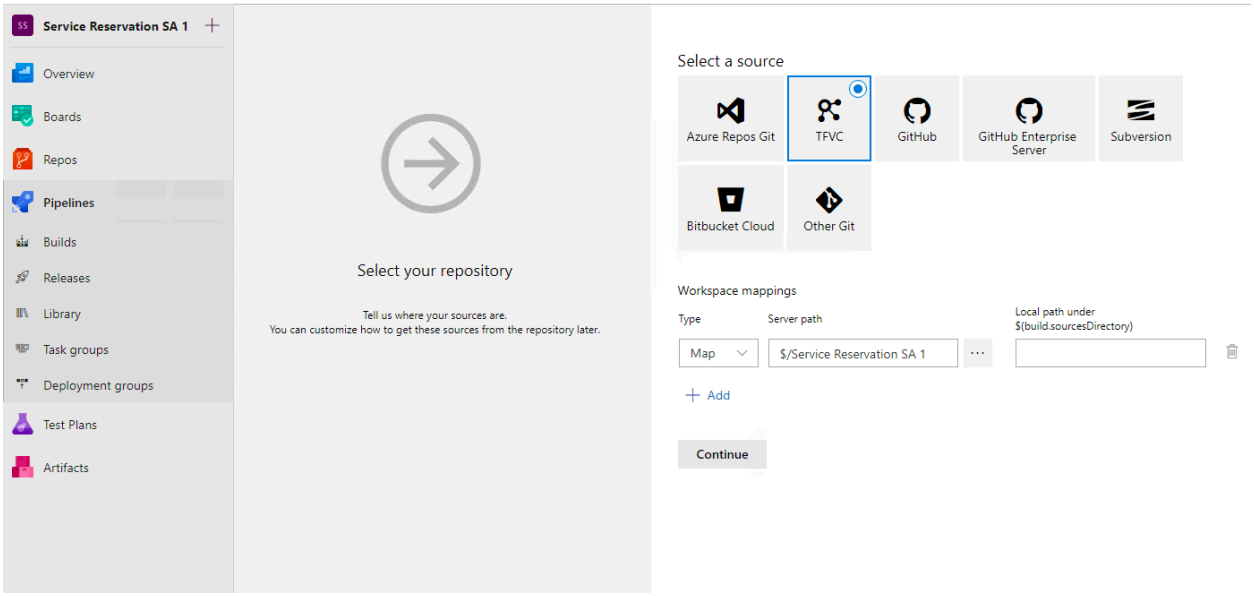
1. Open a browser & navigate to: **https://devopsclassroom.visualstudio.com/_projects**
2. Login to ***{Your Account}***
3. Select the **Service Reservations {First Name Last Name}** project.
4. Inside of the **Service Reservations** project, click on the **Pipelines** tab.
5. Click the **+ New Pipeline** button in the top right corner.



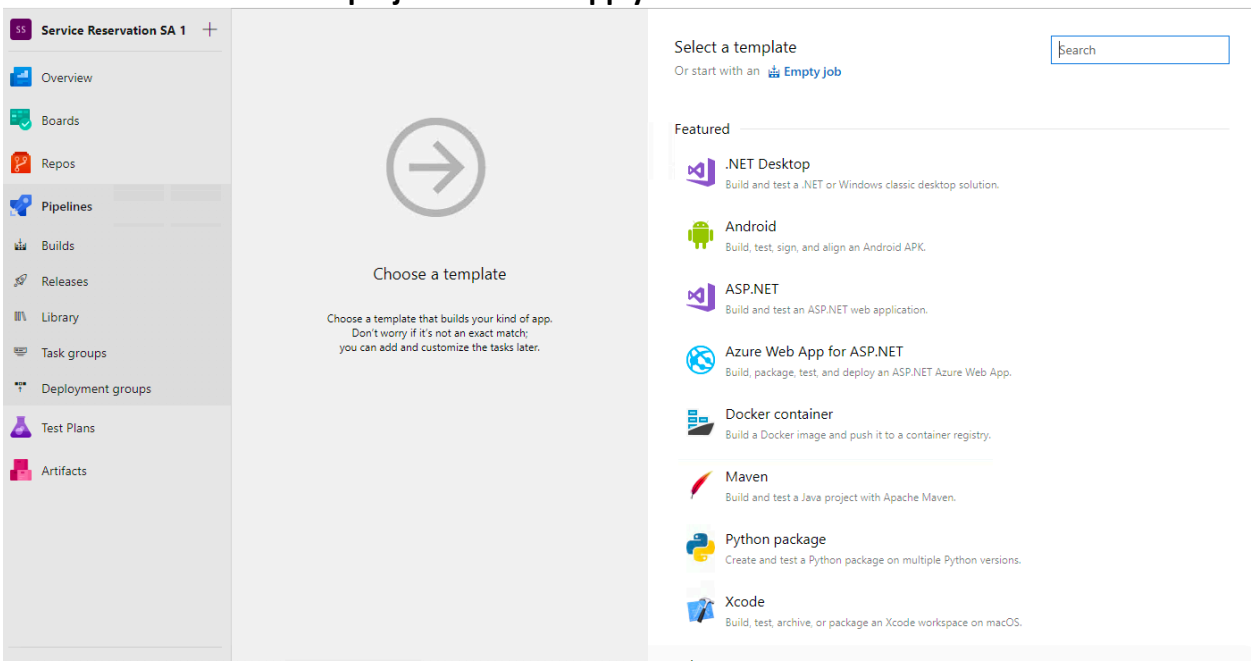
6. Choose **Team Foundation Version Control**

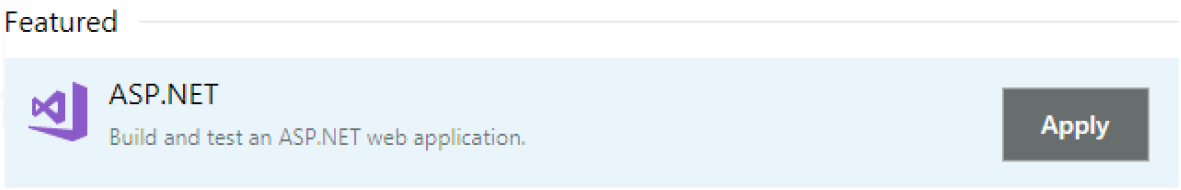


7. Select **TFVC** again and **continue**

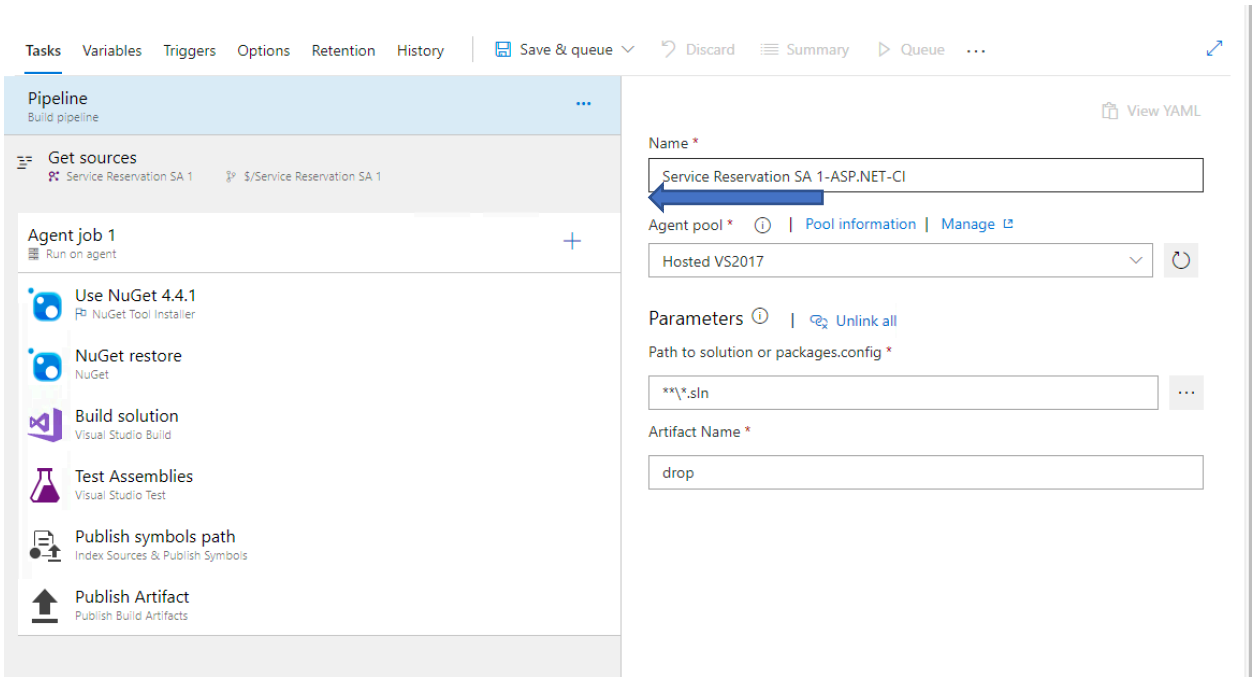


8. Search or choose **ASP.NET project** and click **Apply**





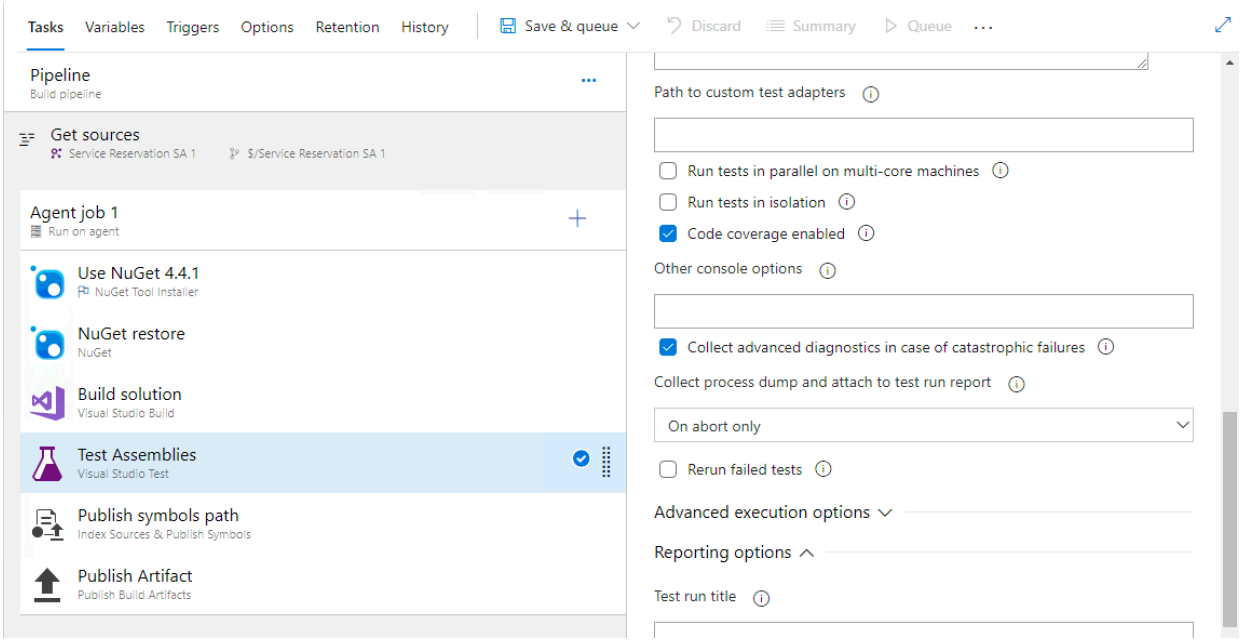
9. The *Build Definition* Name property will be set to **Service Reservations {First Name Last Name}-ASP.NET-CI** by default

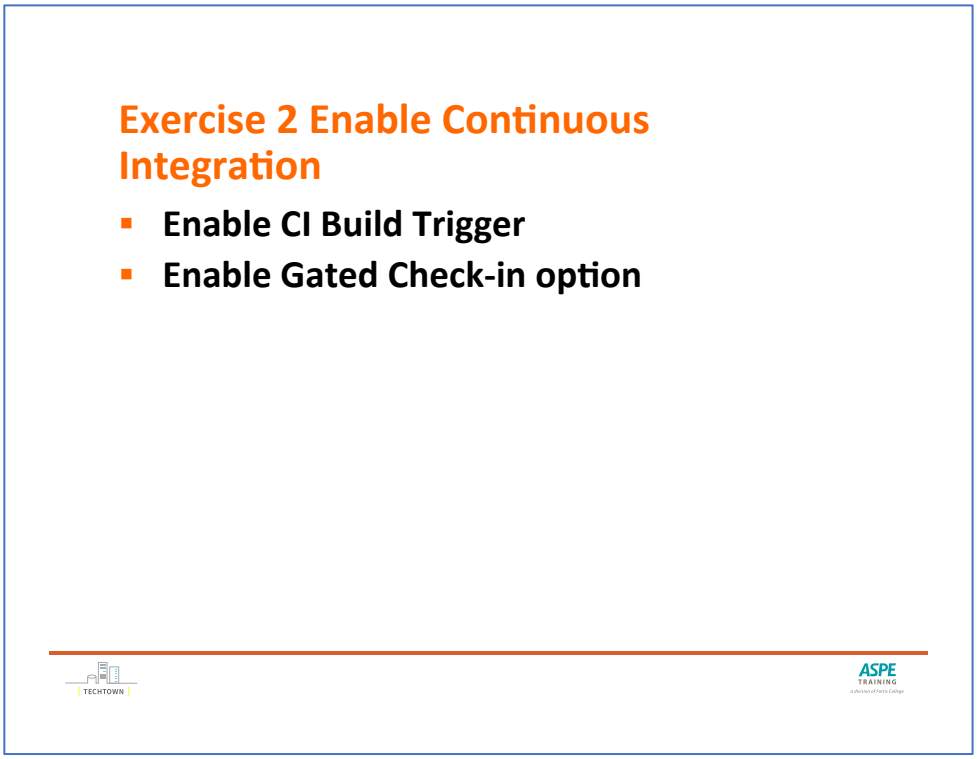


*(Notice that we have an option to use the "**Hosted VS2017**" build)*

Modify Test Settings

1. On the **Task** tab, click on **Test Assemblies.** Scroll down until you see the "**Code coverage enabled**" option.
2. Check the corresponding checkbox. Click the dropdown next to **Save and Queue** and Select **Save** to save the **Build Definition.**
   *** Do not **Save and Queue**, the build will fail because you do not have any code yet

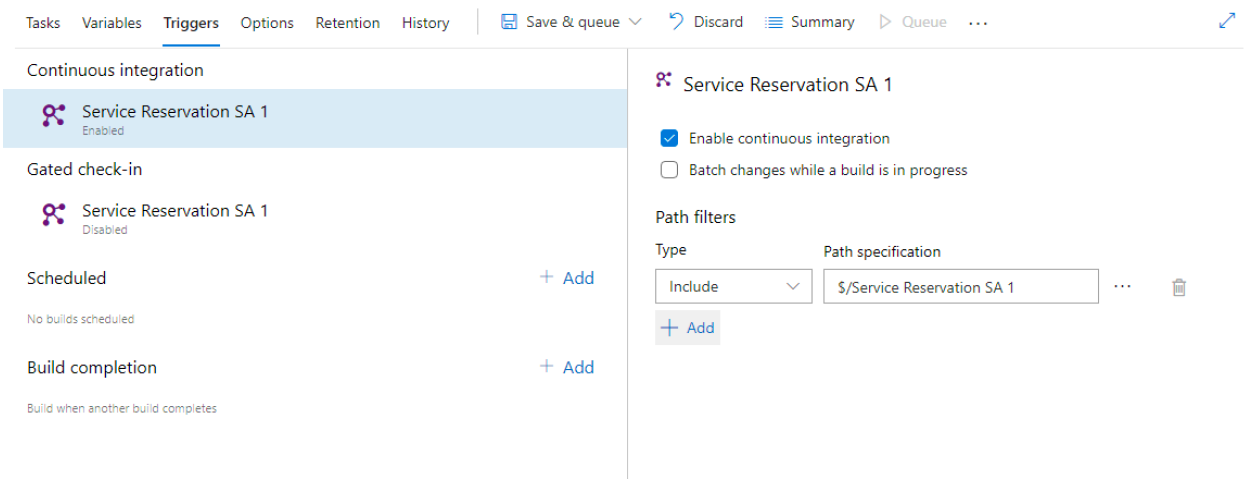




In this Exercise, we will enable a build trigger for Continuous Integration, and ensure that bad code cannot be checked in by enabling **Gated Check-in.** You will continue where you left off with the in the last exercise in your Service Reservations build definition. Make sure you are in edit mode for your build definition.

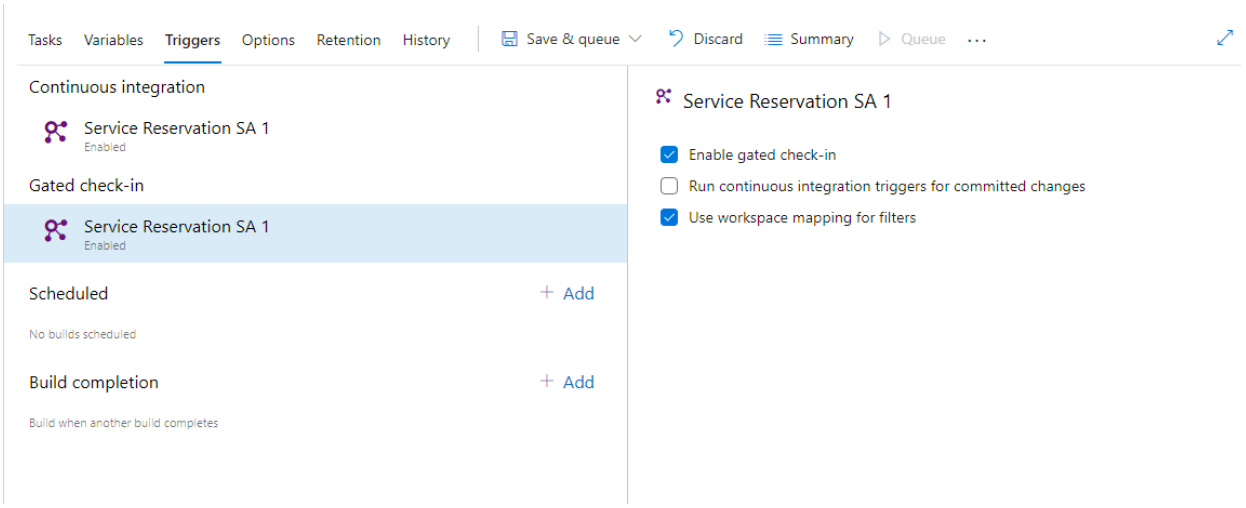
Enable CI Build Trigger

1. Navigate to the **Triggers** tab
2. Enable the Continuous Integration Trigger by flipping the switch under **Enable this Trigger** that says **"Disabled"** to the **"Enabled"** position.

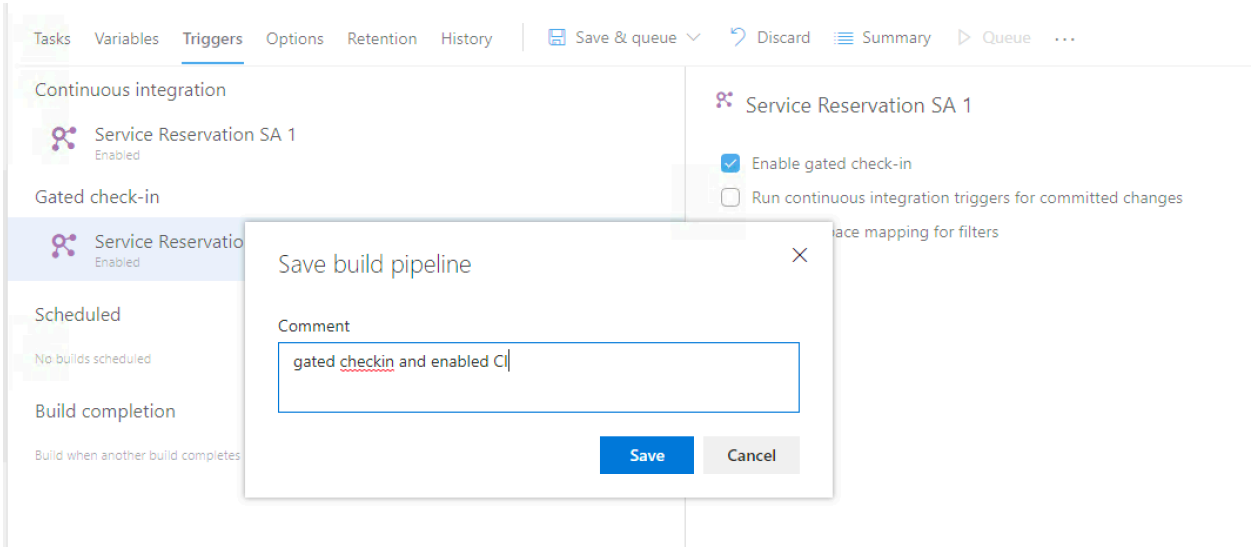


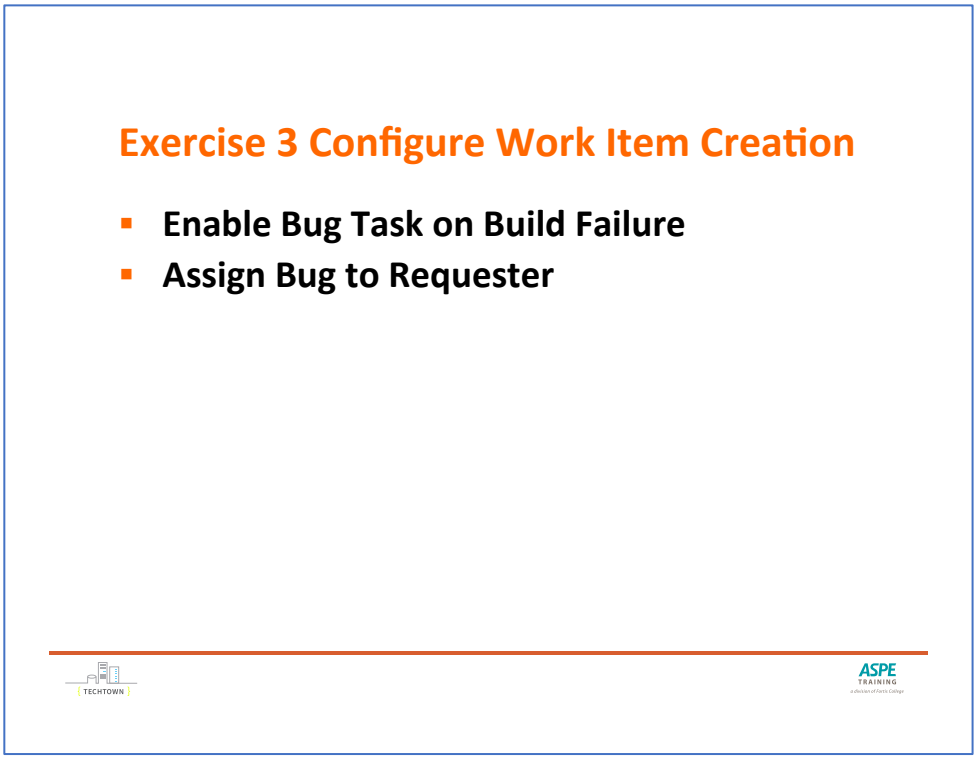
Enable Gated Check-in option

1. Enable Gated Check-ins by flipping the switch under **Enable Gated Check-ins** that says **“Disabled”** to the **“Enabled”** position.
   (Notice the difference between the way it looks when it’s switched on, as opposed to when it’s switched off.)



2. Accept the rest of the defaults and save (Make sure you choose **"Save" and not "Save and Queue"** Building will fail as we have no code yet).


Tasks
Variables
Triggers
Options
Retention
History
Save & queue
Discard
Summary
Queue
Continuous integration
Service Reservation SA 1
Enabled
Gated check-in
Service Reservatio
Enabled
Scheduled
No builds scheduled
Build completion
Build when another build completes
Service Reservation SA 1
Enable gated check-in
Run continuous integration triggers for committed changes
ace mapping for filters
Save build pipeline
Comment
gated checkin and enabled CI
Save
Cancel

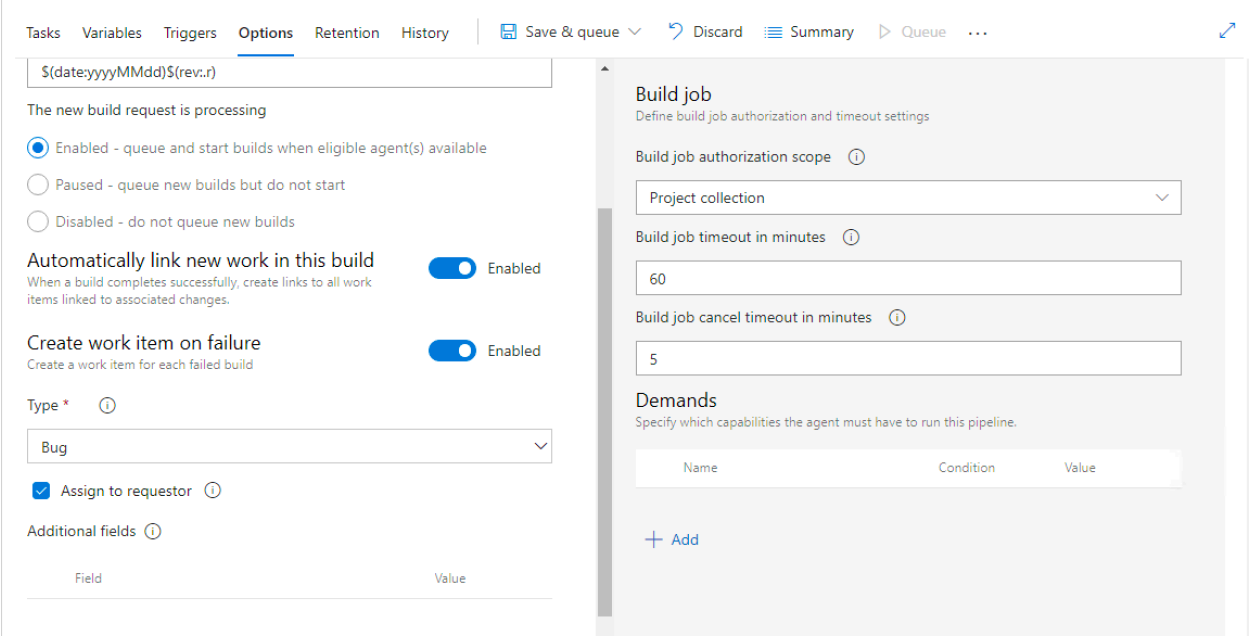


In this Exercise, we will set up a bug task which will be introduced upon build failure, and ensure that the bug is sent to the individual who checked in the code that caused the build to fail.
Note: You are continuing in the same Service Reservation Team Project that you created in a previous Lab in Team Foundation Services Online. (http://DevOpsClassroom.VisualStudio.com)

Enable Bug Task on Build Failure

Continue these steps in the build definition you created in your **Service Reservations {First Name Last Name}** TFS Project.

1. Click on the **Options** tab.
2. Under options, select **Create work item on failure** by flipping the corresponding switch to "**Enabled**".
   *(Notice that the type of work item that will be created upon failure is **Bug**.)*

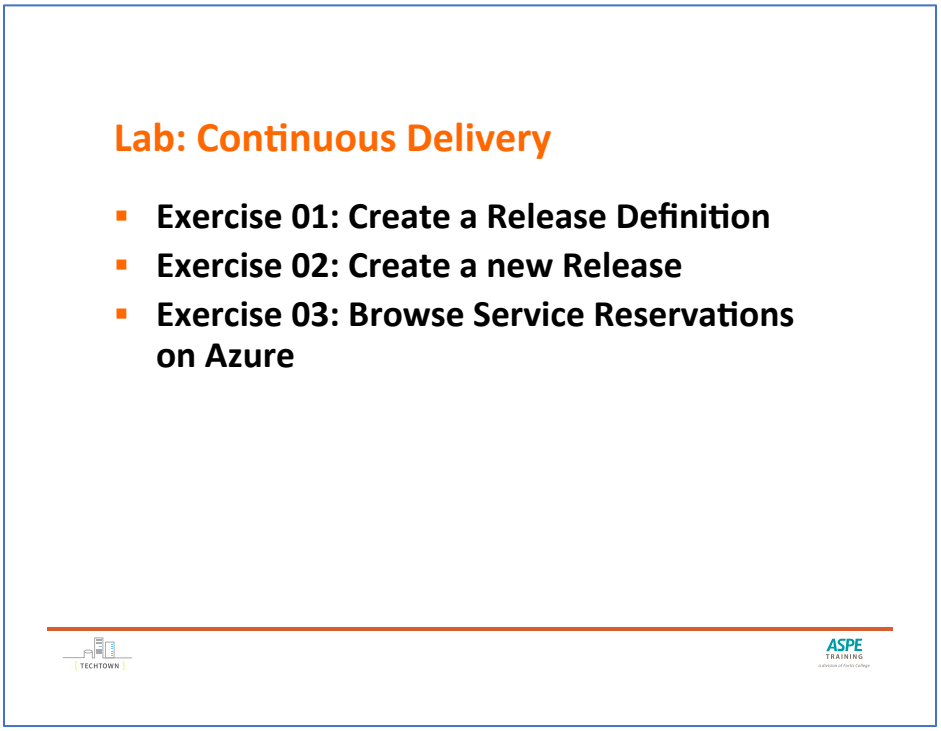


Assign Bug to Requester

Notice the **Assign to requester** checkbox is checked by default, so whoever committed the changes will be notified if any bugs appear.

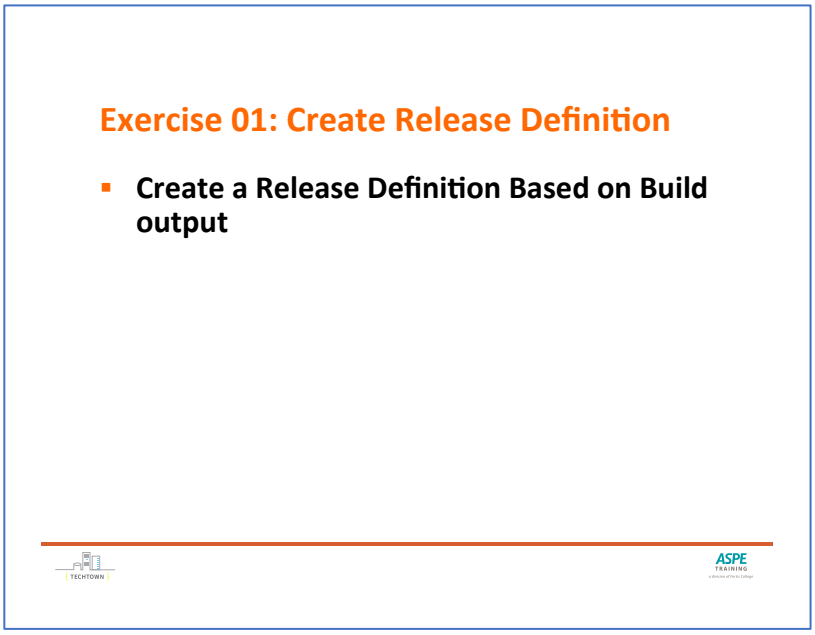
1. Click the dropdown next to Save and Que and select **Save** to save the **Build Definition.** (Make sure you choose "Save" and not "Save and Queue" Building will fail as we have no code yet)

2. In the **Save build definition** dialog enter “Created new build definition” into the **comments** field.



In this lab we will create a Release definition and deploy our solution to Azure then use a browser to review our deployed solution.

Slide 33



Exercise 1: Create Release Definition

In this exercise, we will create a release definition in our project based on the build output that is generated in our Service Reservations Team Project in TFS Online.

1. In **Team Foundation Server,** navigate to **Build and Release > Releases**
2. Click the **+Create Release Definition** button
3. In the **Select a Template** window choose **Azure App Service Deployment,** and click the **Apply** button.
4. Enter "**Staging"** into the **Environment name** text box.
5. Click the text **New Release Definition** to rename the definition **Release to Staging**
6. Under **Artifact** click **Add Artifact**
7. Click **Source (Build definition) * > Service Reservation-ASP.NET-CI** > **Add**
8. From the **Tasks** menu select **Staging**.
9. Click on the **Manage** link next to **Azure subscription.**
10. Click +**New Service Endpoint** and select **Azure Resource Manager**.
11. Enter "**Service Reservations Azure App Service"** as the **Connection name.**
12. Select your **Azure** subscription from the dropdown and click **OK**.
13. If prompted enter you **Azure** credentials
14. Back in your **Release Definition** Select your **Azure** subscription from the dropdown and click **Authorize**
15. In the **App type** field, choose **Web App**
16. In the **App service name** dropdown, select **ServiceReservationsXX** (Where xx is your initials)

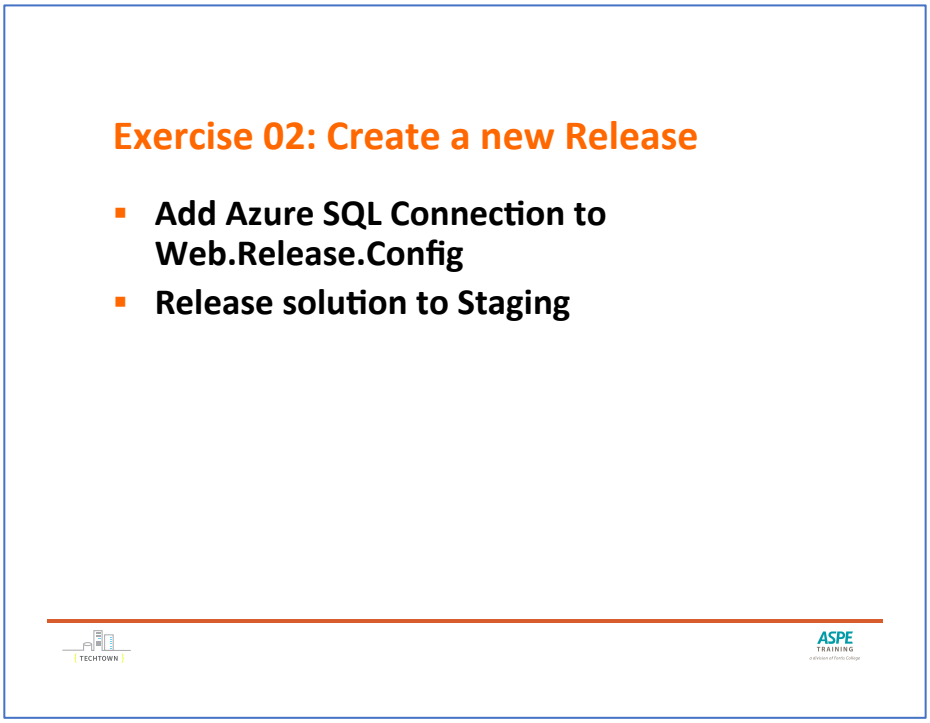
Add a Task to the Phase

1. Click the plus sign to the right of Run on agent to Add a new task to the phase
2. In the **Add task** window select the **Deploy** tab
3. Scroll down and select **Azure SQL Database Deployment** and click **Add**.
4. Drag **Execute Azure SQL: DacpacTask** above the **Deploy Azure App Service** task.
5. In the **Azure Connection Type** dropdown select Azure Resource Manager
6. In the **Azure Subscription** dropdown select your subscription.

Get the Azure Database Server Name

1. Navigate back to your **Azure** subscription dashboard
2. Under **SQL Databases** select **ServiceReservationDB**
3. Copy the **Server Name**
4. Switch back to the **Release Definition**, and paste the server name that you just copied into the **Azure SQL Server Name** field
5. Enter **ServiceReservationsDB** as the Database name.
6. Enter your username in the **Server Admin Login** field, and enter your password into the **Password** field
7. In the type field, choose the **SQL DACPAC** File option
8. In the **DACPAC** File text box enter ****/*.dacpac**
9. Accept the remaining defaults, click the **Save** button., and click **OK** to save your release definition.

Slide 34



Exercise 2: Create a new Release

Now we are almost ready to attempt to perform a release, but first we need to update the Web Application configuration so that it knows where our new Azure database server is.

<u>Add Azure SQL Connection to Web.Release.Config</u>

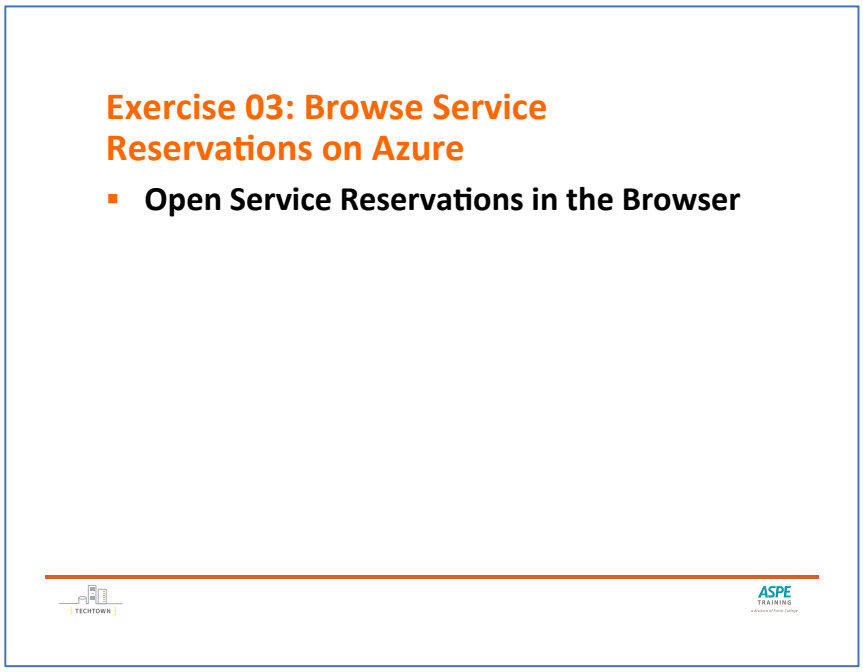
1. **Navigate back to Visual Studio, go to the solution explorer, and click on the Web.Config file**
2. **In the Code Window, line 12 where you see "DefaultConnection"**
3. **In Web.Release.config code window, we need to uncomment the connectionString in line 11.**
4. **Change the name of it from "MyDB" to "DefaultConnection",**
5. **Change the name of the connectionString to what we have in Azure, which is:**
6. **To find it, navigate back to Azure, click SQL databases from the menu items, and choose ServiceReservationsDB**
7. **Click the Show database connection strings option and copy the connection string by clicking on the icon next to the field where it resides.**
8. **Paste the connection string into the connectionString property of the DefaultConnection in Web.Release.Config**

NOTE – this takes place in the Web Release.config tab in Visual Studio

Release solution to Staging

1. In the **Team Foundation Server** navigate to **Release >** > **Create Release**
2. In the Create new release dialogue, accept the defaults, and click the **Create** button.
3. Monitor your release by clicking the Release link in the Release XXXXX has been created field that is highlighted in green.
4. Here, we can take a look to ensure that everything is going according to plan.
5. Notice, beneath **Deployment Status**, it reads **IN PROGRESS** highlighted in blue.
6. Click on the **Logs** tab, and we can watch the release run. Notice that the blue icon next to **Initialize Agent** turns green.

Slide 35



Exercise 3: Browse Service Reservations on Azure

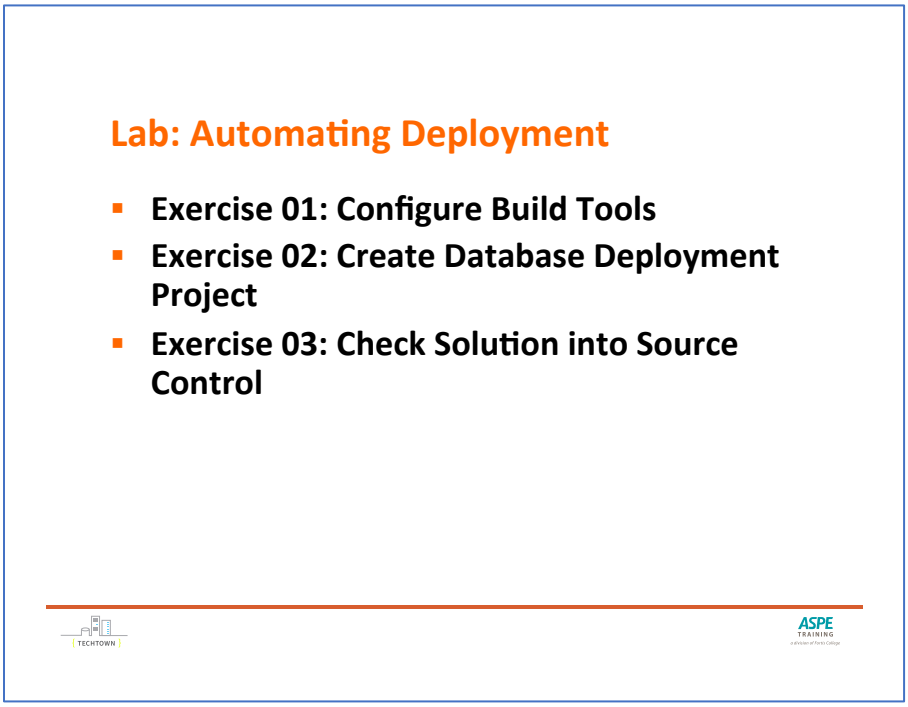
In this exercise, we will open the Service Reservation project, and register with the necessary information

Open Service Reservations in the Browser

1. Open the browser and navigate to **ServiceReservationsXX.AzureWebsites.net**
2. Click Register
3. Enter email
4. Enter password
5. Click register

Now you should be logged in

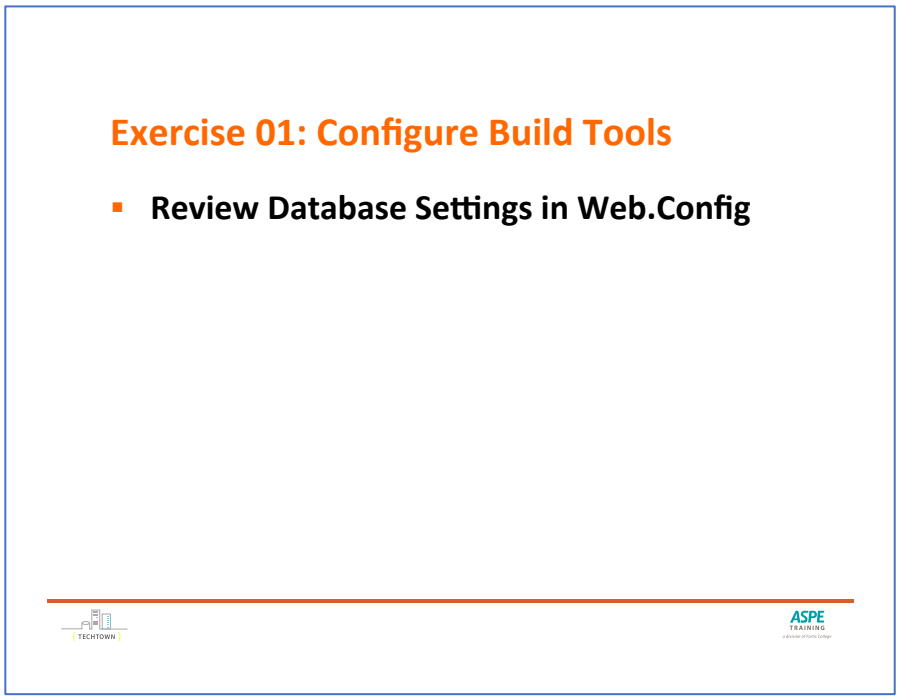
Slide 104



LAB: Automating Deployment

In this lab, we will automate deployment for our project by configuring the build tools, creating a Database Deployment Project, and check our solution into Source Control.

Slide 105



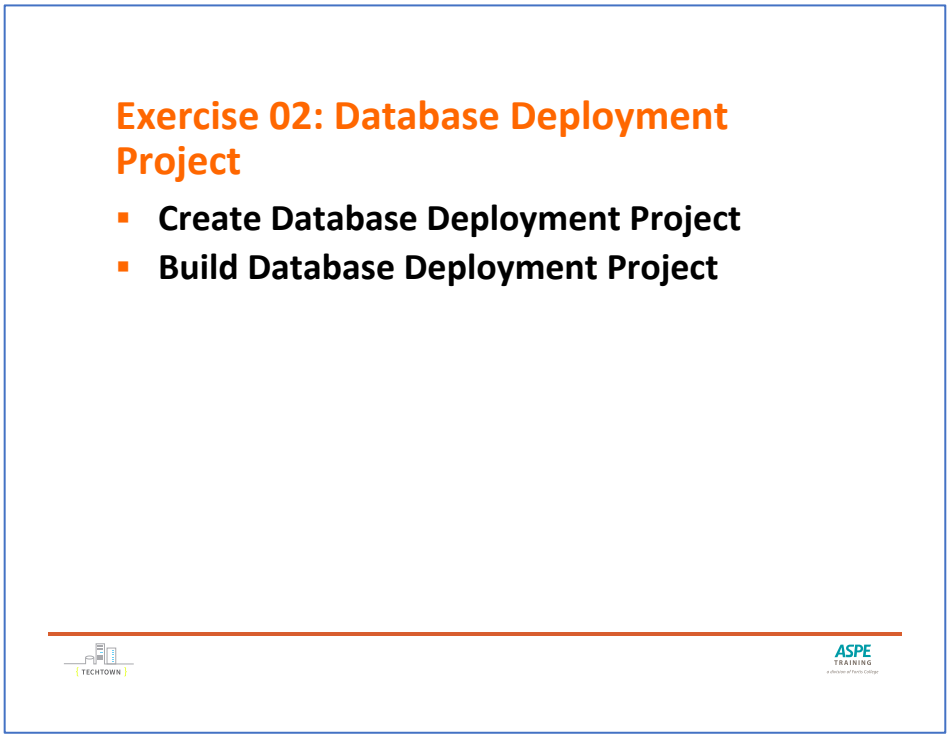
Exercise 1: Configure Build Tools
In this exercise, we will configure our build tools, and do a brief review of some of the database settings.

Review Database Settings in Web.Config

1. In our **Service Reservation Web UI** project, notice the default connection in the SQL Server Object Explorer. This default connection came from our projects **Web.config** file, which you can see in the **solution explorer**

2. Open the **Web.config** file. In the code window
   Notice we have connection strings, and a connection called **Defaultconnection**.
   Also notice the **Defaultconnection** settings:
   It has its data source set to **LocalDB MSSQLocalDB**;
   The Initial Catalog (aka the Dabasebase) is set to the project we created.
   Creating the file from our template automatically included this connection string.
   Debugging, and registering the user forced the creation of the database.

Slide 106



Exercise 2: Database Deployment Project

In this exercise, we will Create a Deployment Project, and then create a build solution to make sure that what we've done has not introduced any defects to our production environment.

Create Database Deployment Project

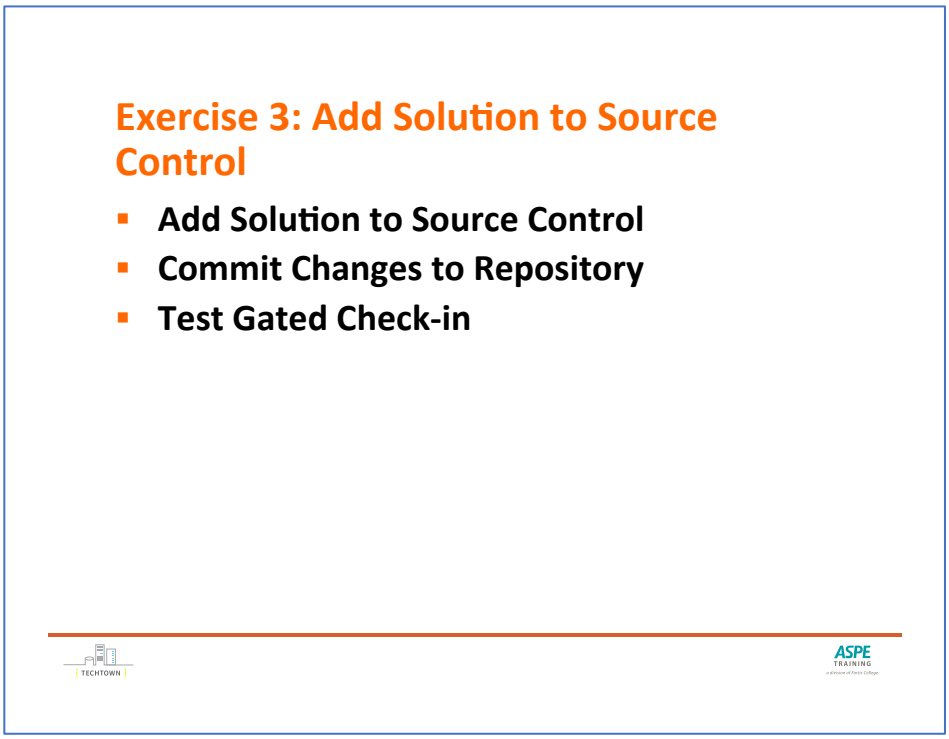
1. In the **Server Explorer**, right click on the default connection, and select **Browse in SQL Server Object Explorer**
2. In the **Service Reservation WebUI Properties** window, check the path to your **WebUI** project, and follow it so we can verify the structure.
3. Follow the path all the way to the end, and you will see the **Service Reservation WebUI folder**, and the **Service Reservation WebUI.Tests** folder.
4. In the **SQL Server Object Explorer** locate your database, right click on it, and **create a new project.**
5. In the Import Database dialogue box, name the project **Service Reservations DB.**

Build Database Deployment Project

1. Click the **Browse** button, and save your project to the area containing the **Service Reservation WebUI** folder.
2. Accept the rest of the defaults, and click **Start**.

3. When your database project finishes, you will see it in the **Solution Explorer** along with the **WebUI** and the **WebUI tests**.
4. In the database project, expand the folder, open the **dbo** folder, and expand the table folder.
5. You should see a **SQL Server** deployment script for each one of the **SQL Server** tables that was created when we registered the user earlier.
6. Click on the **Team Explorer** Tab, and notice that you already have the **Map and Get.**
7. Accept the defaults, and click the **Map and Get** button.

Slide 108



Exercise 3: Add a Solution to Source Control

In this exercise, we will add a solution to source control, test gated check in, and ensure changes to the repository can be seen by the entire team.

Add Solution to Source Control

1. Navigate back to the **Solution Explorer**, right click on the **Solution Service Reservations** projects, and choose **Add Solution to Source Control**.
2. When the **Add Solution Service Reservations to Source Control** dialogue comes up, locate Service Reservation in the menu, accept the defaults and click OK. *(You should now see plus signs next to the three projects in your solution).*
3. Go back to your browser, and click on the **Work** tab to view the work items in our backlog.
4. In the Solution Explorer, right click on the **Solution**, and choose check in.
5. Your check in should get rejected, because we purposefully introduced a defect.
6. Click on the Home icon in Team Explorer, and choose Work Items.

7. Inside of Visual Studio, you will be able to see all the tasks that are assigned. Expand Shared Queries, and double click on My Tasks. These tasks will appear in the Query Results window:
   Test the membership database implementation
   Implement the ASP.Net Membership database

Commit Changes to Repository

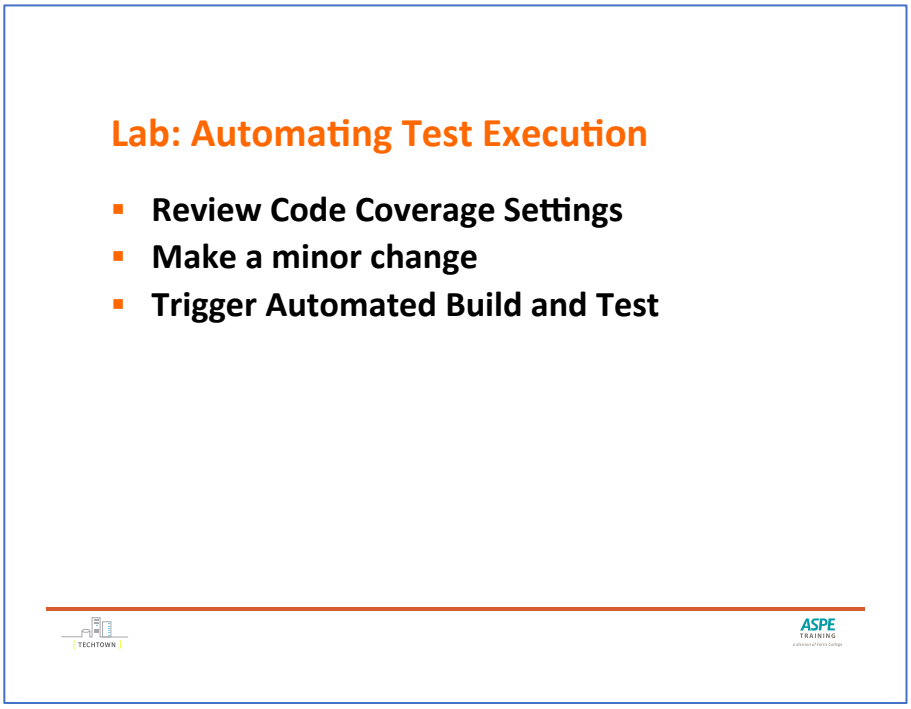
1. Create a work item by clicking **New Work Item** > **Create a new task** In the **New Task,** name the task **Create Initial Project Structure**
2. In the **Effort (Hours)** field, type **.5** for the half hour of setup that we did, and click the **Save** button.
3. Go back to **Visual Studio**, and in the **Team Explorer – Work Items** window, double click on **My Tasks.**
4. When the **My Tasks (Results)** window pops up, you should see the **Create Initial Project Structure** task that we just created.
5. In the **Team Explorer – Work Items** window, click on the home icon, and go to **Pending Changes**, where you will see the check-in of all of your files in progress.
6. Drag the **Create Initial Project Structure** task into the **Related Work Items** area beneath **Queries**. else's tasks in the project timeline.
7. Click on the **Resolve** option. Notice that you have the option to **Resolve**, or **associate your items**.
8. If. In this scenario, so you will want to **Associate** task, not **Resolve** it. *(you are not finished with an item, you probably don't want it committed until you have finished working on it)*
9. Click on the **Check In** button to test all of the necessary items in the **Included Changes** window.
10. When the **Check-in Confirmation** dialogue pops up, click **Yes.**
11. When the **Gated Check in** dialogue box pops up, click the **Build Changes** button to validate your changes.

Test Gated Check-in

1. Once your changes are done compiling, click on the build definition link located beneath **Build Summary.**
2. In the build monitor, you will see that it has initialized the agent, initialized the job, copied the sources from the repository into the build…, and finally compiled the build.
3. Eventually, you should see red text, because our test has failed.
4. Right click on the call to the controllers **About** methods, and choose **Go To definition** from the menu.
5. Navigate back to your **Test Explorer**, and click on **About**.
6. In the **Assert.AreEqual** method, change "**Your**" back to **My,** *(which is what it our test was expecting).*

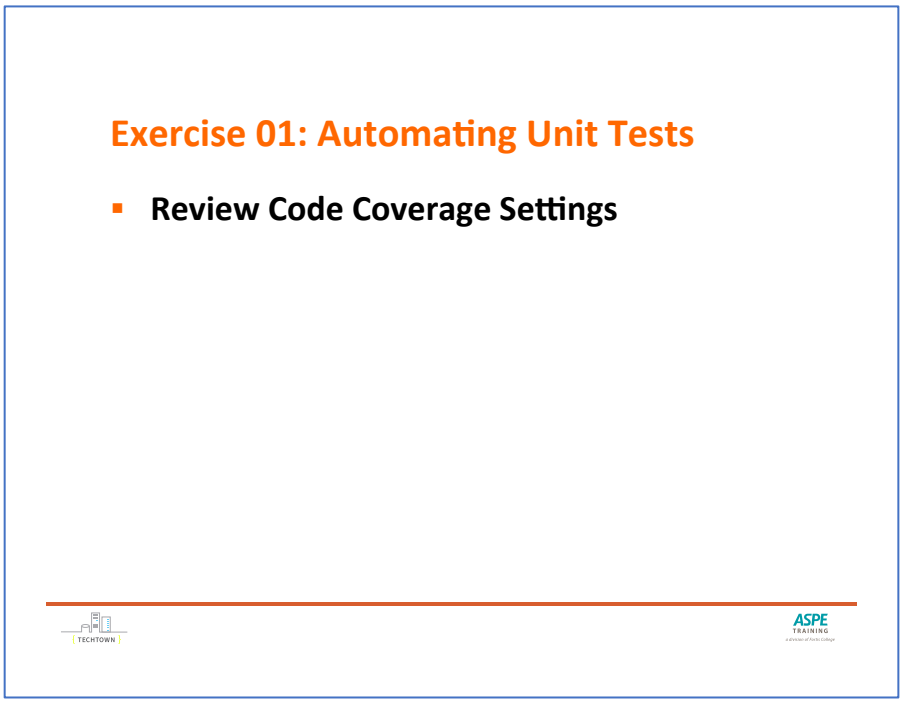
7. Run the test again by selecting **Test** > **Run** > **All Tests** from the main menu. Notice that all of our tests have passed.
8. Right click on the solution and choose **Check-in.** Notice, the **Gated Check-in** rejected our project.
9. Click the **Check In** button, and click **Yes** when the **Check-in Confirmation** dialogue box pops up.
10. When the **Gated Check-in validation** dialogue pops up, click on the **Build Changes** button.
11. Under **Pending Changes**, we can see that our check in has been queued for validation.
12. Click on the **here** link to see the status of the validation build.
13. Beneath the **Build Summary**, click on the **build ID** link so we can view the details.
14. Once it finishes compiling, you will see the green bar, so we know that our build has succeeded.

Slide 153



In this lab we will review code coverage results after making a minor change and triggering an automated ***Continuous Integration*** build and test.

Slide 154

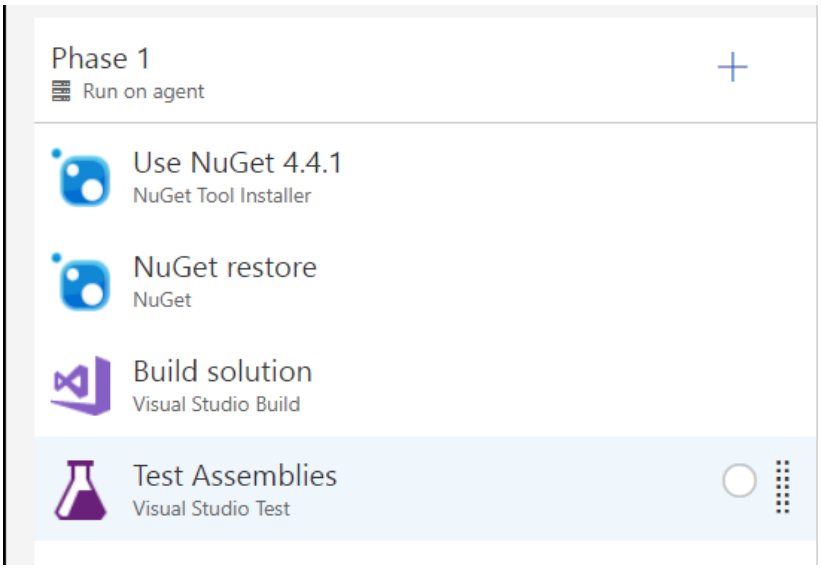


Exercise 1: Automating Unit Tests

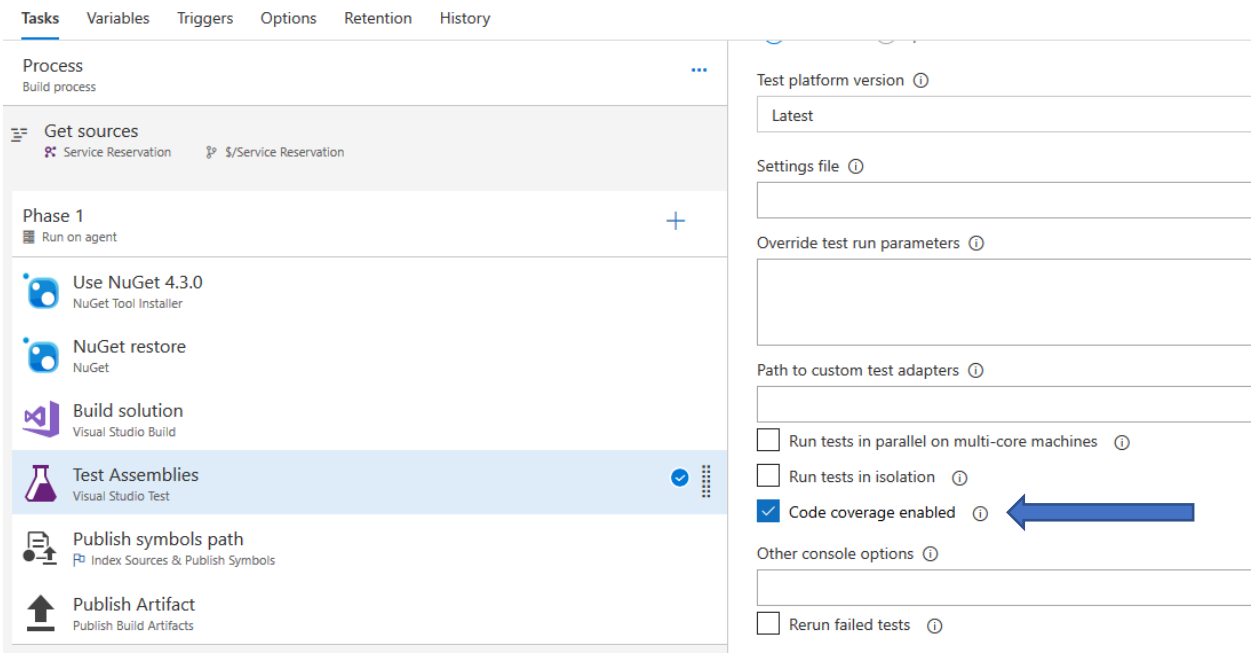
In this exercise, we will enable code coverage, so we can estimate the value of our current testing criteria.

Review Code Coverage Settings

1. Navigate to the **Team Foundation Services** dashboard.
2. Click on the **Build and Release** tab, and select **Build.**
3. In the **Service Reservation ASP.NET-CI** project, click on the ellipses link under the Status menu option and choose **Edit**
4. Click on **Test Assemblies**. (Beneath Phase 1 on the left)

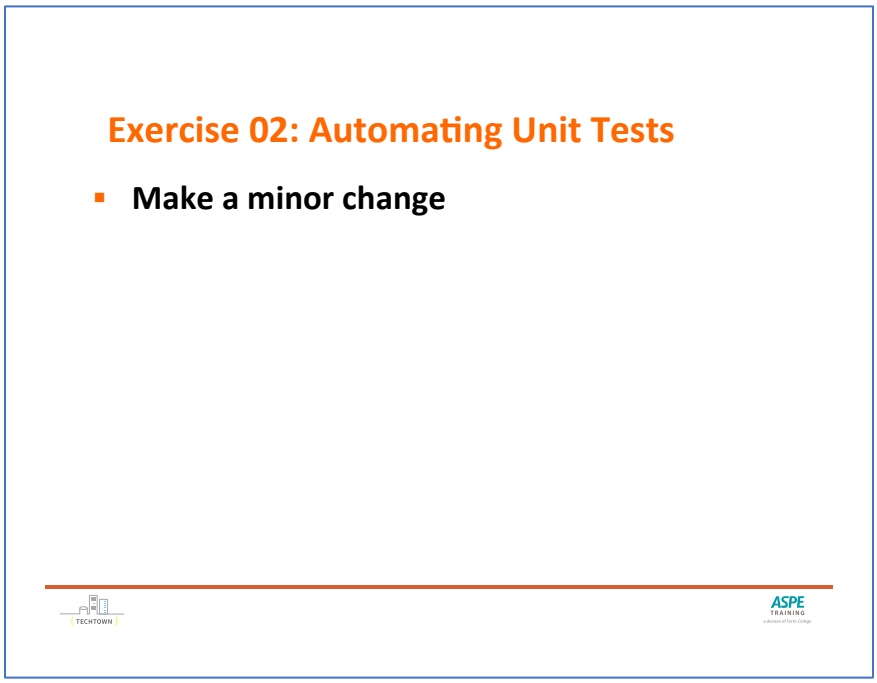


5. Scroll down until you see the "**Code coverage enabled**" option.
6. Check the corresponding checkbox.



7. Expand the **Save & Queue** menu, and choose **Save** *(do not save and queue).*
8. Accept the defaults, and click the **Save** button

Slide 155



Exercise 2: Automating Unit tests

In this exercise, we will make some changes to our code, and automate code coverage.

Make a minor change

1. In **Microsoft Visual Studio,** navigate back to the **Service Reservations WebUI** project.
2. In the **Solution Explorer**, click on the **HomeController cs.**
3. In the corresponding code window, scroll down until you see:

```
public ActionResult About()
{
ViewBag.Message = "Your application description"
return View();
}
public ActionResult Contact()
{
ViewBag.Message = "Your contact page";
```

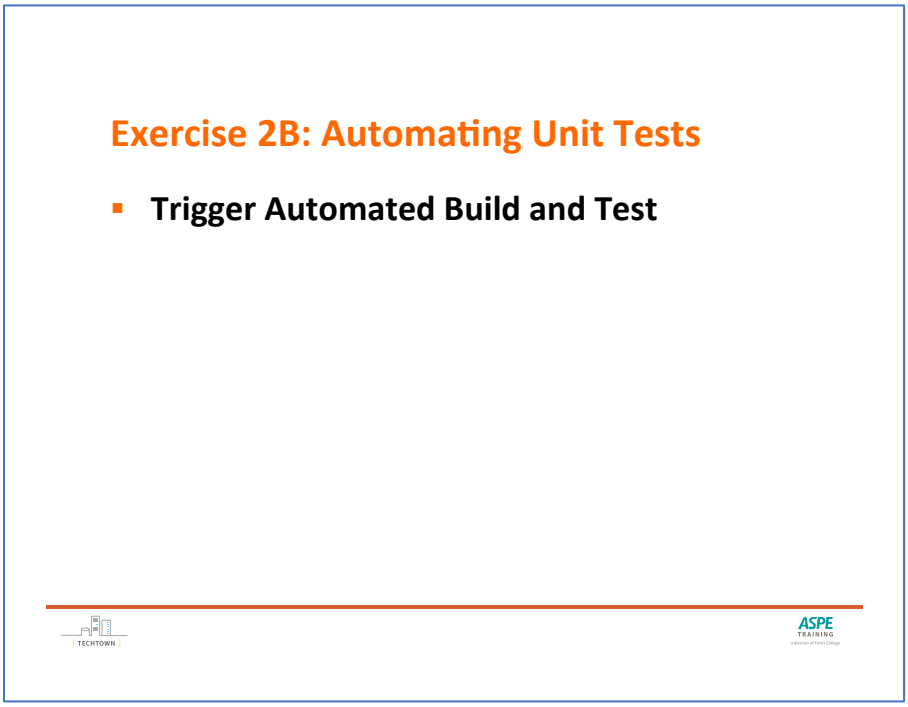
4. Review the ViewBag.Message property value. (This is where the heading of the page is set)
5. Navigate back to the Solution Explorer, and scroll down to Service **Reservation WebUI.Tests.**

6. Expand the **Controllers** menu, and click on **HomeControllerTest.cs**
7. In the code window, scroll down to:
   // Assert
   Assert.IsNotNull(result);
8. Replace **Assert.IsNotNull(result);** with **Assert.AreEqual("Service Reservation Contact page.", result.ViewBag.Message);**

*Note: It may be easiest to copy **Assert.AreEqual("Your Application Description page.", result.ViewBag.Message);** and change the text to Service Reservation Contact Page instead of retyping the whole line*

9. Scroll back up to **"Your Application description page."**, and change it to: **"Service Reservations description page"** *(We are modifying the code to cause the tests to fail)*

Slide 157



Exercise 2B: Automating Unit Tests

In this exercise, we will continue the automation of our Unit tests by creating a trigger that will fire off when changes are applied to our source control repository.

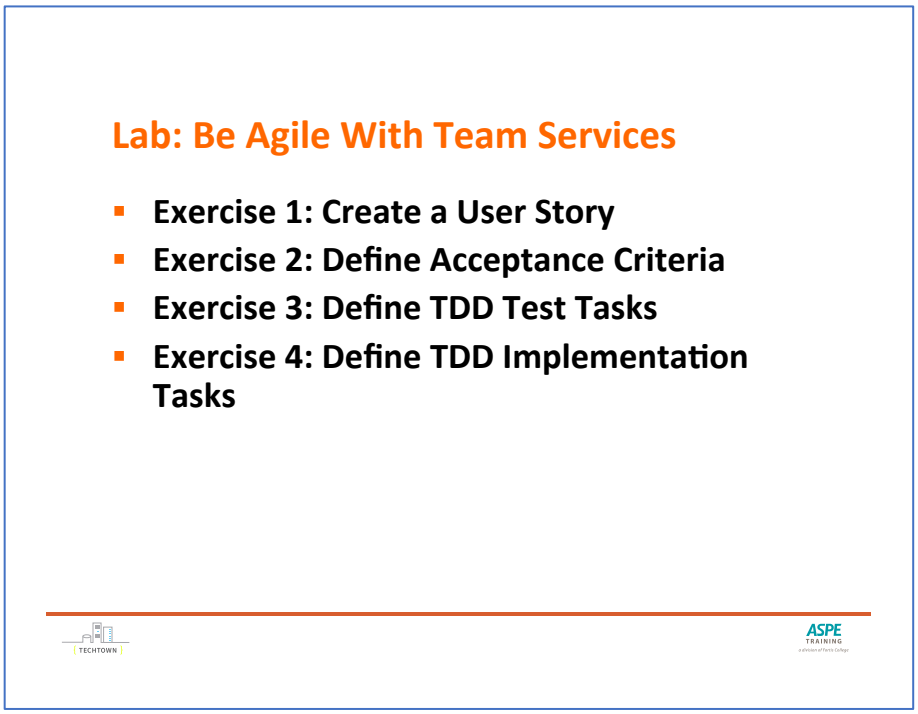
Trigger Automated Build

1. Navigate to the Team Foundation Services Dashboard, and chose the **Build and Release** tab
2. In the **Solution Explorer**, right click on "**Service Reservations WebUI**" **(2 projects),** select the **Check In** menu item, and click the **Yes** button when the dialogue pops up.
3. When the **Gated Check** in dialogue pops up, accept the defaults and click the **Build Changes** button.
4. In the **Solution Explorer**, click on the link "**Your check-in has been queued for validation. Click here to see the status of the validation build."**
5. Click on the **Build Summary** link to view the results.
6. Notice our Tests have failed and our **Gated check-in** has been rejected.
7. Review the Test Results in the Build summary
8. Back in **Visual Studio** in the **Solution Explorer**, click on the **HomeController cs.**
9. In the corresponding code window, scroll down until you see:
   public ActionResult About()

```
{
ViewBag.Message = “Your application description”
return View();
}
public ActionResult Contact()
{
ViewBag.Message = “Your contact page”;
```

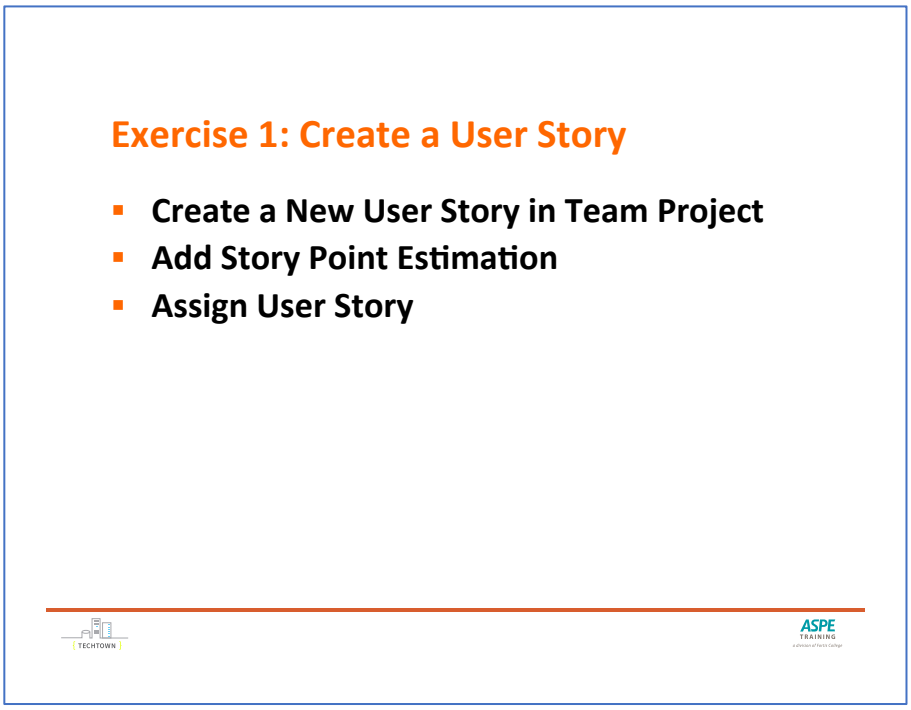
10. Change **“Your application description”** to “**Service reservation description**”
11. Change “**Your contact page**” to “**Service reservation contact page**”
12. In the **Solution Explorer**, right click on Solution **“Service Reservations WebUI”**
13. When select the **Check In** menu item, and click the **Yes** button when the dialogue pops up
14. The **Gated Check** in dialogue pops up, accept the defaults and click the Build Changes button
15. In the **Solution Explorer**, click on the link “**Your check-in has been queued for validation. Click here to see the status of the validation build.”**
16. Click on the **Build Summary** link to view the results. Notice our Tests have failed and our **Gated check-in** has been rejected.
17. Review the Test Results in the **Build summary**
18. The build completes, notice that our Code coverage summary shows a very low percentage**.**

Slide 286



Lab 2 – Input User Stories
In this lab, we will change Project Settings and add new User Stories to the Backlog

Slide 287

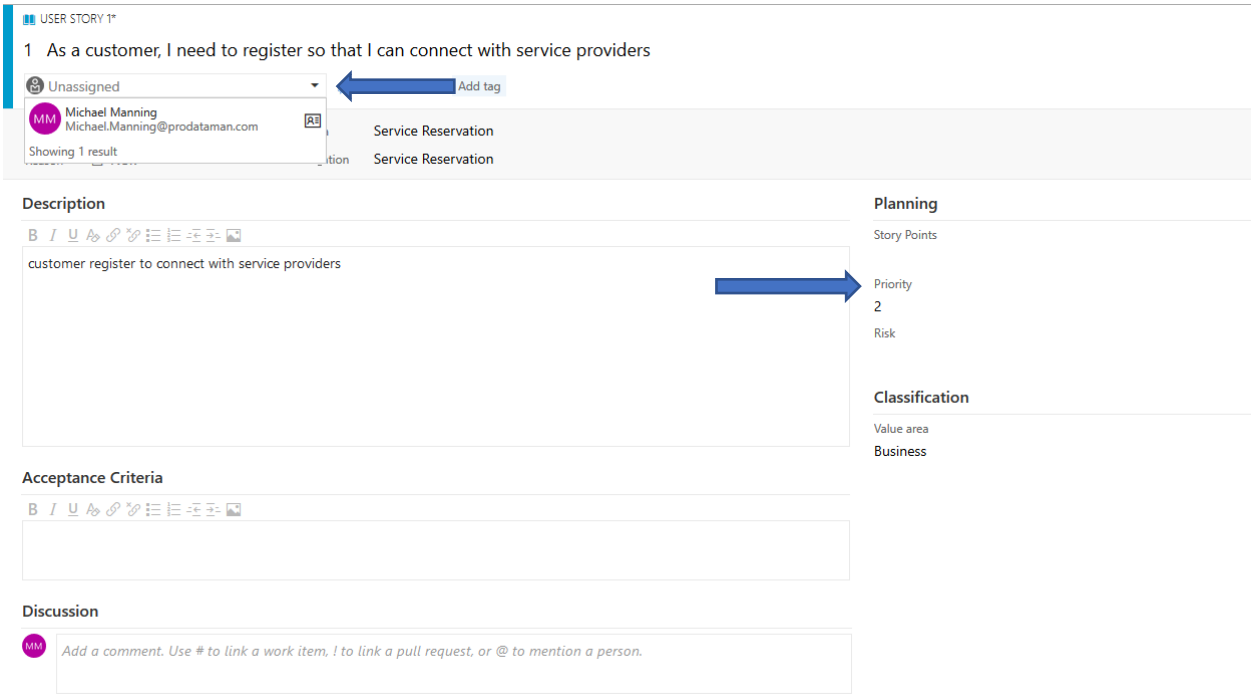


Exercise 1 Add User Stories to Product Backlog
In this exercise we will add 2 customer stories and 2 service provider stories with their associated acceptance criteria.
Note: You are continuing in the same Service Reservation Team Project that you created in the previous exercise in Team Foundation Services Online.
(http://DevOpsClassroom.VisualStudio.com)

1. Select **Work** from the **VisualStudio.com** menu
2. On the board that appears click the **New** item button
3. Enter the following **User Story** in the Title box and press enter

Customer Story 1
As a customer, I need to register so that I can connect with service providers

4. Click on the link text of the user story
5. When the user story editor appears **Assign** the user story to yourself
6. Add a 2 to the Story points field
   Repeat the steps to add the User Stories below



Customer Story 2
As a customer, I need login capabilities so that I can create reservations

Customer Story 3
As a customer I need to enter the service I wish to purchase so that I can be connected with many service providers

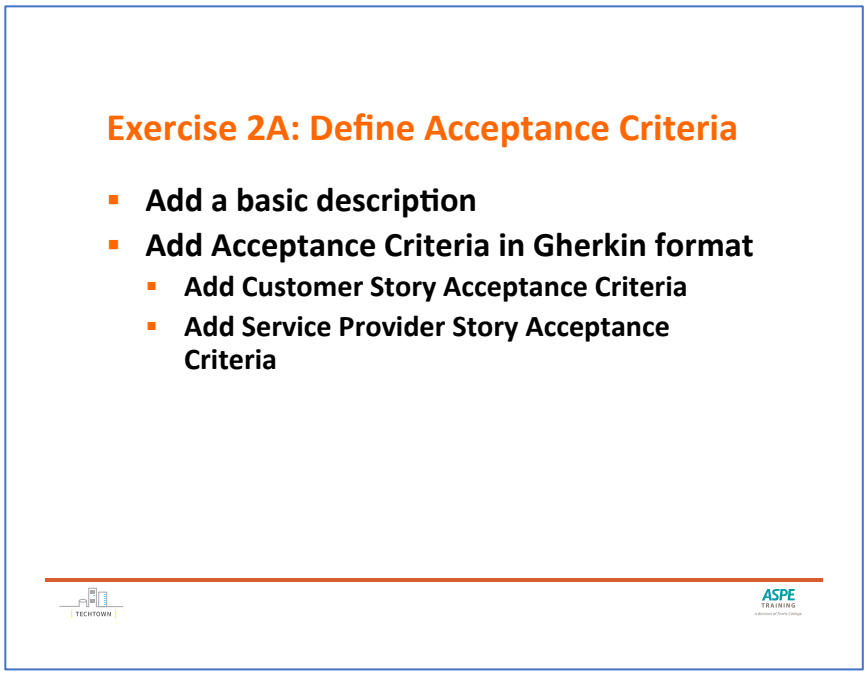
Customer Story 4
As a customer, I need enter the price I wish to pay so that I can be connected with many service providers in that price range

Service Provider Story 1
As a Service Provider, I need to register my company so customers can connect with me

Service Provider Story 2
As a service provider, I need to specify each service I provide with a range of prices so only customers in my price range are connected with me

Note: Can you think of other User Stories for this scenario?

Slide 289



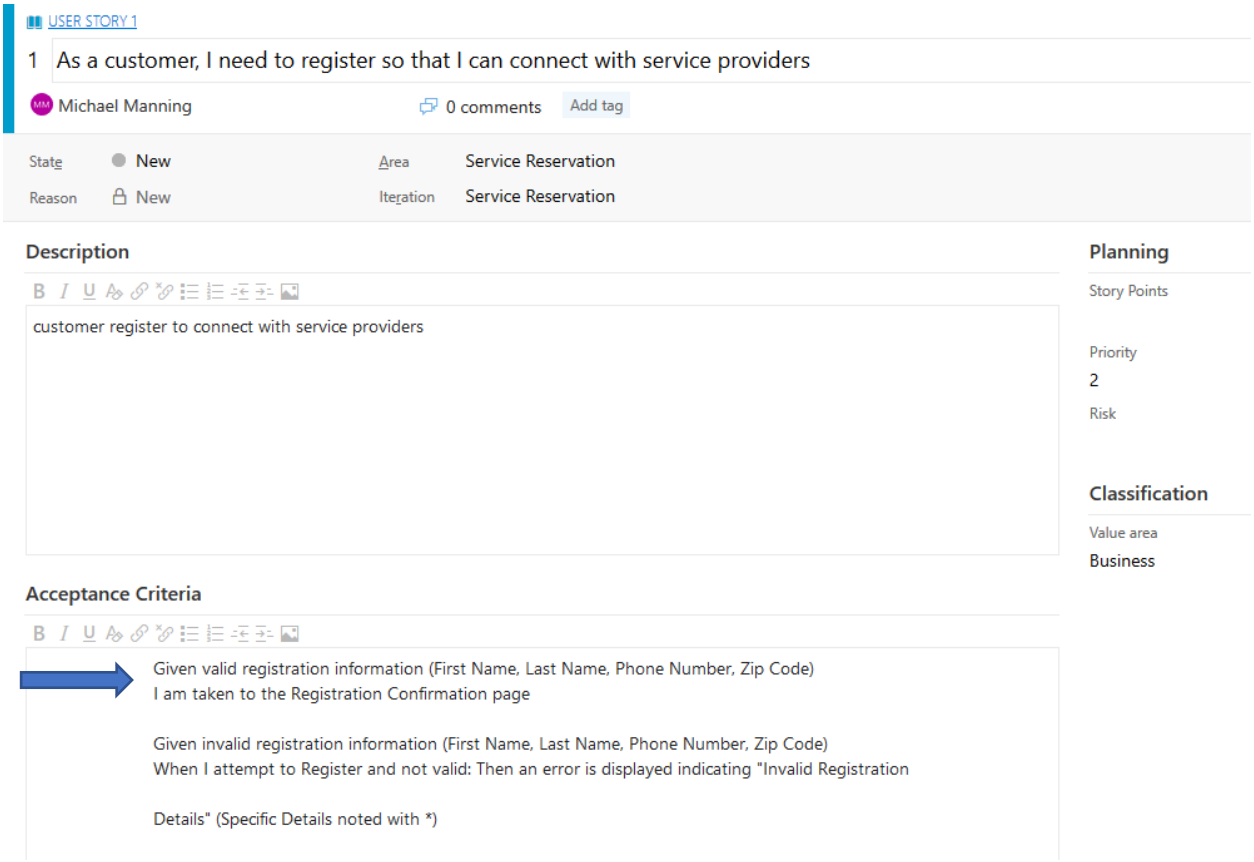
Exercise 2 Add Acceptance Criteria to User Stories
In this exercise we will add **Acceptance Criteria** to the 2 customer stories and 2 service provider stories added in the previous exercise.
Note: You are continuing in the same Service Reservation Team Project that you created in the previous exercise in Team Foundation Services Online.
(http://DevOpsClassroom.VisualStudio.com)

1. Select Work from the **VisualStudio.com** menu
2. Once the Board appears click the link to the hyperlink to each User Story
3. Enter the following User Story in the Text box that appears and press enter:

Customer Story 1
As a customer I need to register so that I can connect with service providers

4. Click on the link text of the user story
5. When the user story editor appears enter the following **Acceptance Criteria:**

    Given a valid registration information (First Name, Last Name, Phone Number, Zip Code)
    When I attempt to Register
    Then I am taken to the Registration Confirmation page
    Given invalid registration information (First Name, Last Name, Phone Number, Zip Code)
    When I attempt to Register
    Then an error is displayed indicating "Invalid Registration Details" (Specific Details noted with *)



Customer Story 2

As a Customer I need login capabilities so that I can create reservations

Acceptance Criteria

Given a valid username and password
When I attempt to log in
Then I am taken to the profile page

Given an invalid username or password
When I attempt to log in
Then an error is displayed indicating "invalid username or password"

Customer Story 3

As a Customer, I need enter the service I wish to purchase, So I can be connected with many service providers

Acceptance Criteria

Given a valid Service Name and details
When I submit my service request
Then I am taken to the price and confirmation page

Customer Story 4

As a Customer

I need enter the price I wish to pay

So that I can be connected with many service providers in that price range

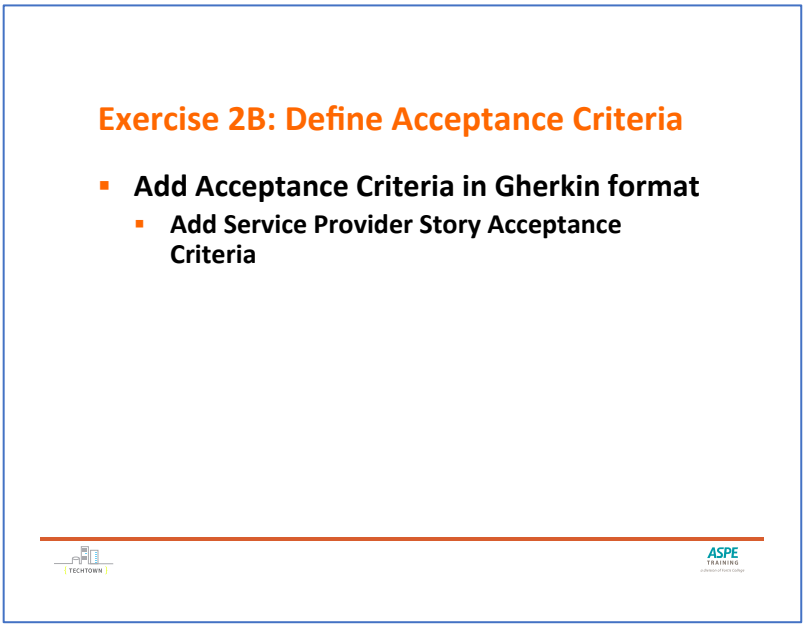
Customer Story 4 Acceptance Criteria

Given a Price within service providers defined range of prices this service

When I submit my request

Then I am taken to the confirmation page

Slide 292



Exercise 02B Convert Acceptance Criteria to Gherkin format
In this exercise we will convert Acceptance Criteria to a Standard Format Gherkin Format to ensure consistency regardless of the project or the author.
Note: You are continuing in the same Service Reservation Team Project that you created in the previous exercise in Team Foundation Services Online.
(http://DevOpsClassroom.VisualStudio.com)

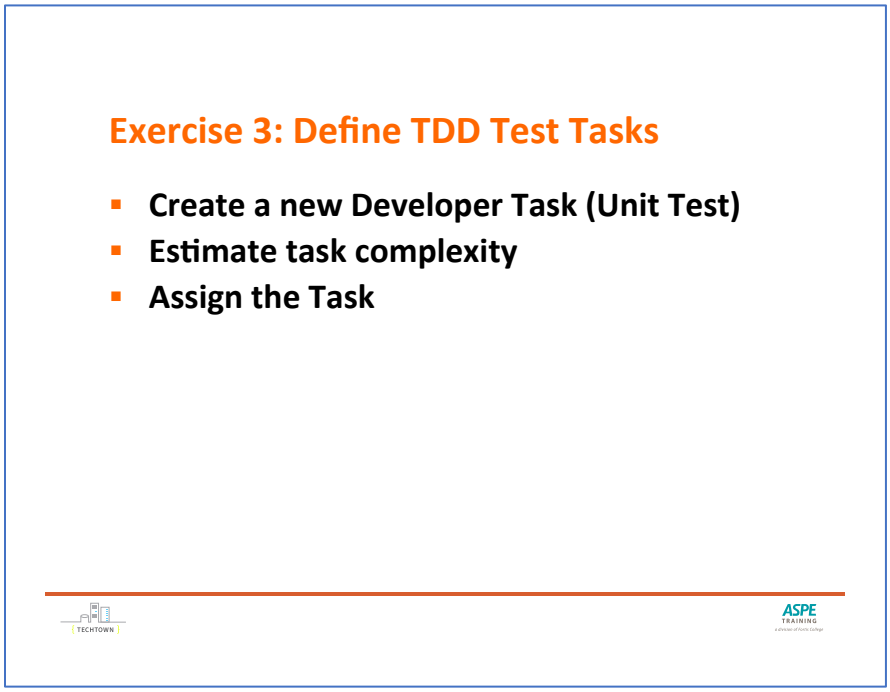
Service Provider Story 1
As a Service Provider, I need to register my company so that customers can connect with me

Acceptance Criteria
Given a valid username and password
When I attempt to log in
Then I am taken to the profile page
Given an invalid username or password
When I attempt to log in
Then I error is displayed indicating "invalid username or password “

Service Provider Story 2
As a service provider, I need to specify each service that I provide with a range of prices so that only customers in my price range are connected with me

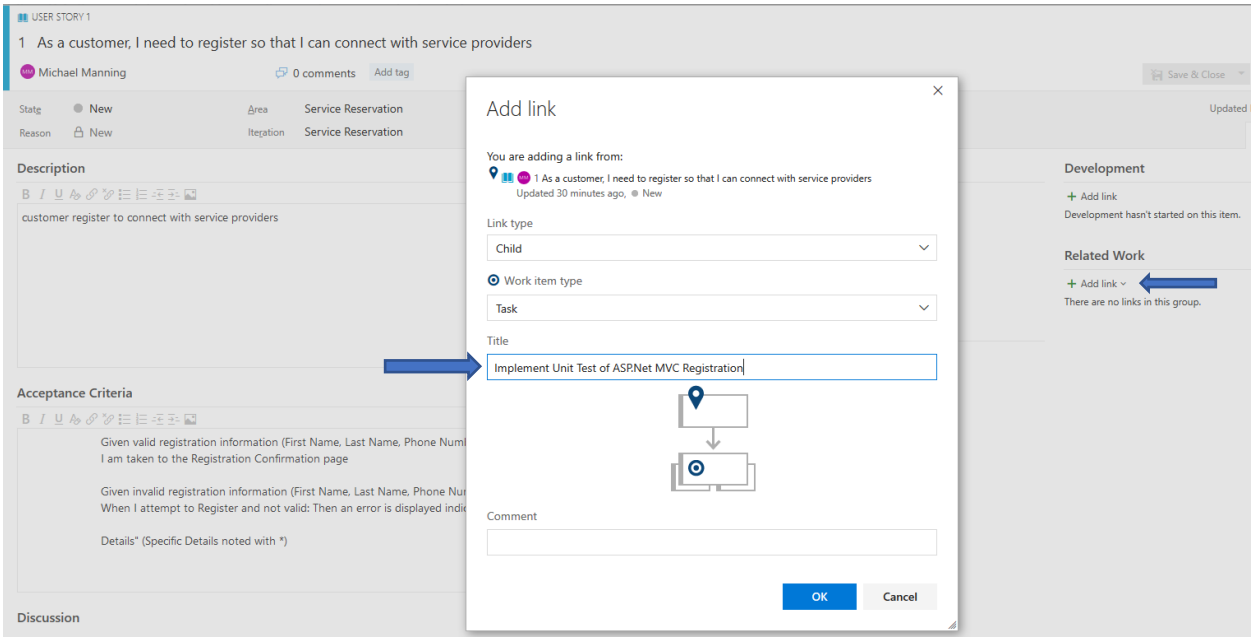
Acceptance Criteria
Given a valid service name and price range
When I submit my service offering
Then I am taken to the confirmation page

Slide 294



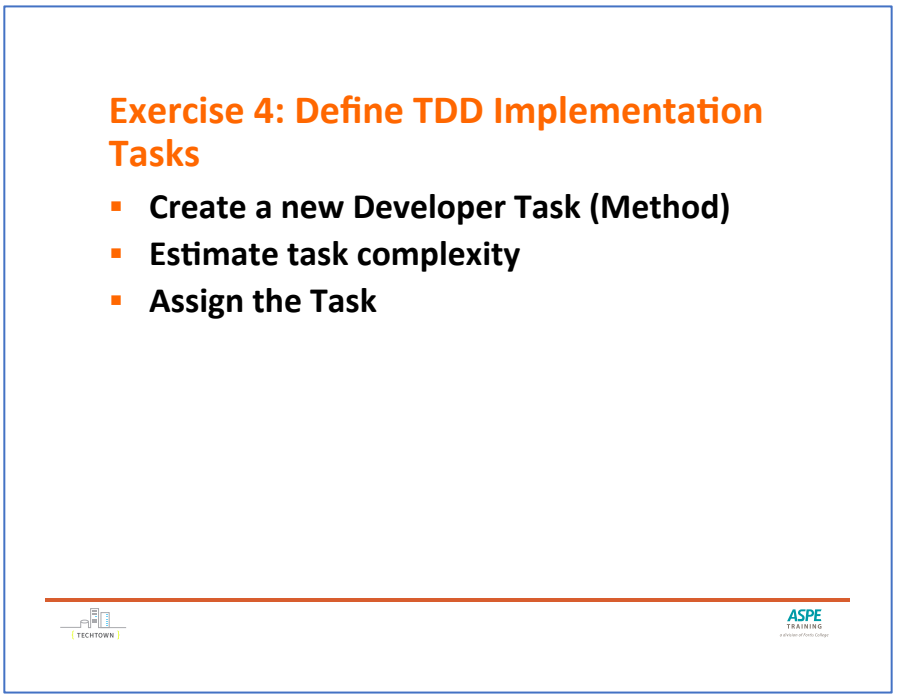
Exercise 3 Add Delivery Team Tasks to User Stories
In this exercise we will add Delivery Team Tasks (Test details) to our user stories and surface any currently undocumented dependencies and risks that may impede the teams progress during the sprint.
Note: You are continuing in the same Service Reservation Team Project that you created in the previous exercise in Team Foundation Services Online.
(http://DevOpsClassroom.VisualStudio.com)

1. Select Work from the VisualStudio.com menu
2. Once the Board appears click on the link text of **Customer Story 1**
3. When the user story editor appears click Add Link in the Related Work section to the right.
4. In the Add Link menu choose New Item, make sure to select **Task** under the *Work item type* drop down menu
5. In the Add Link dialog enter “Implement Unit Test of ASP.Net MVC Registration” as the Title for the task and click OK.



6. Enter 1 in the Original Estimate field under **Effort (Hours)**
7. Ensure that the task is assigned to you then click **Save & Close**
8. Click on the link text of **Customer Story 2**
9. When the user story editor appears click Add Link in the Related Work section to the right.
10. In the Add Link menu choose New Item
11. In the Add Link dialog enter “Implement Unit Test for ASP.Net MVC Login” as the Title for the task and click OK.
12. Enter 1 in the Original Estimate field under **Effort (Hours)**
13. Ensure that the task is assigned to you then click **Save & Close**

Slide 195



Exercise 4 Add Delivery Team Implementation Tasks to User Stories
In this exercise we will add Delivery Team Tasks (Implementation details) to our user stories and surface any currently undocumented dependencies and risks that may impede the teams progress during the sprint.
Note: You are continuing in the same Service Reservation Team Project that you created in the previous exercise in Team Foundation Services Online.
(http://DevOpsClassroom.VisualStudio.com)

1. Select Work from the VisualStudio.com menu
2. Once the Board appears click on the link text of **Customer Story 1**
3. When the user story editor appears click Add Link in the Related Work section to the right.
4. In the Add Link menu choose New Item
5. In the Add Link dialog enter “Implement ASP.Net Membership” as the Title for the task and click OK.
6. Enter 1 in the Original Estimate field under **Effort (Hours)**
7. Ensure that the task is assigned to you then click **Save & Close**
8. Click on the link text of **Customer Story 2**
9. When the user story editor appears click Add Link in the Related Work section to the right.
10. In the Add Link menu choose New Item

11. In the Add Link dialog enter “Create a new ASP.Net MVC Project with Individual User Accounts template” as the Title for the task and click OK.
12. Enter 1 in the Original Estimate field under **Effort (Hours)**
13. Ensure that the task is assigned to you then click **Save & Close**