**Exercise**

**Creating a pull request**

**Introduction**

In this unit, you practice the process of submitting a pull request and merging your changes into the master branch so that everyone can benefit from your work.

In [Create a build pipeline with Azure Pipelines](https://docs.microsoft.com/en-us/learn/modules/create-a-build-pipeline), you created a Git branch named build-pipeline, where you defined a basic build pipeline for the *Space Game* website. Recall that your build definition goes in a file named *azure-pipelines.yml*.

Although your branch produces a build artifact, that work exists only on the build-pipeline branch. You need to merge your branch into master.

Recall that a *pull request* tells the other developers that you have code ready to review, if necessary, and you want your changes merged into another branch, such as master.

Before we start, let's check in with Mara and Andy.

**Andy:** Hi, Mara. I know you've got a build pipeline running on Azure. I'm adding a feature to the website and I want to see the build process for myself. Are we ready to do that?

**Mara:** Absolutely. I created the pipeline on a branch. Why don't we set up a pull request and get it merged into master so you can use the pipeline, too?

**Andy:** Sounds great. Let's take a look.

**Create a branch and add starter code**

Although you could use the build pipeline you built in the previous module, let's create a new branch, named code-workflow, that's based off of master so that you can practice the process from the beginning. If you completed the previous module, your branch includes a basic but complete build configuration in your *azure-piplines.yml* file. If you haven't completed the previous module, you will create the file.

1. In Visual Studio Code, open the integrated terminal.
2. Switch to the master branch.

**Bash**

**git checkout master**

1. Ensure that you have the latest version of the code from GitHub.

**Bash**

**git pull origin master**

1. Create a branch named code-workflow.

**Bash**

**git checkout -b code-workflow**

The -b argument specifies to create a new branch if it doesn't exist. Omit the -b argument when you want to switch to an existing branch.

By default, your new branch builds on the previous branch from where you ran the git checkout command. Here, the parent branch is master. But the parent branch can be another one, such as a feature branch someone else started that you want to build on or experiment with.

It's now safe to make whatever changes you need, because you're on your own local branch. If you want to see which branch you're on, run git branch -v.

1. From the file explorer, open *azure-pipelines.yml* and replace its contents with this:

**yml**

trigger:

- '\*'

pool:

vmImage: 'ubuntu-18.04'

demands:

- npm

variables:

buildConfiguration: 'Release'

wwwrootDir: 'Tailspin.SpaceGame.Web/wwwroot'

dotnetSdkVersion: '3.1.300'

steps:

- task: UseDotNet@2

displayName: 'Use .NET Core SDK $(dotnetSdkVersion)'

inputs:

version: '$(dotnetSdkVersion)'

- task: Npm@1

displayName: 'Run npm install'

inputs:

verbose: false

- script: './node\_modules/.bin/node-sass $(wwwrootDir) --output $(wwwrootDir)'

displayName: 'Compile Sass assets'

- task: gulp@1

displayName: 'Run gulp tasks'

- script: 'echo "$(Build.DefinitionName), $(Build.BuildId), $(Build.BuildNumber)" > buildinfo.txt'

displayName: 'Write build info'

workingDirectory: $(wwwrootDir)

- task: DotNetCoreCLI@2

displayName: 'Restore project dependencies'

inputs:

command: 'restore'

projects: '\*\*/\*.csproj'

- task: DotNetCoreCLI@2

displayName: 'Build the project - $(buildConfiguration)'

inputs:

command: 'build'

arguments: '--no-restore --configuration $(buildConfiguration)'

projects: '\*\*/\*.csproj'

- task: DotNetCoreCLI@2

displayName: 'Publish the project - $(buildConfiguration)'

inputs:

command: 'publish'

projects: '\*\*/\*.csproj'

publishWebProjects: false

arguments: '--no-build --configuration $(buildConfiguration) --output $(Build.ArtifactStagingDirectory)/$(buildConfiguration)'

zipAfterPublish: true

- task: PublishBuildArtifacts@1

displayName: 'Publish Artifact: drop'

condition: succeeded()

This configuration resembles the basic one you created previously. For brevity, it builds only your project's Release configuration.

**Note**

You have this file if you did the previous module. If you didn't do the previous module, create the file in the root of your repository.

**Push your branch to GitHub**

Here, you push your code-workflow branch to GitHub and watch Azure Pipelines build the application.

1. In the terminal, run git status to see what uncommitted work exists on your branch:

**Bash**

**git status**

You see that *azure-piplines.yml* has been modified. You'll commit that to your branch shortly, but you first need to make sure that Git is tracking this file. This is called *staging* the file.

Only staged changes are committed when you run git commit. Next, you run the git add command to add *azure-pipelines.yml* to the staging area, or index.

1. Run the following git add command to add *azure-piplines.yml* to the staging area:

**Bash**

**git add azure-pipelines.yml**

1. Run the following git commit command to commit your staged file to the code-workflow branch.

**Bash**

**git commit -m** **"Add the build configuration"**

The -m argument specifies the commit message. The commit message becomes part of a changed file's history. It helps reviewers understand the change as well as help future maintainers understand how the file changed over time.

**Tip**

The best commit messages complete the sentence, "If you apply this commit, you will ..."

If you omit the -m argument, Git brings up a text editor where you can detail the change. This option is useful when you want to specify a commit message that spans multiple lines. The text up to the first blank line specifies the commit title.

1. Run this git push command to push, or upload, the code-workflow branch to your repository on GitHub.

**Bash**

**git push origin code-workflow**

1. As an optional step, go to your project in Azure Pipelines and trace the build as it runs.

This build is called a *CI build*. Your pipeline configuration uses what's called a *trigger* to control which branches participate in the build process. Here, "\*" specifies all branches.

**yml**

trigger:

- '\*'

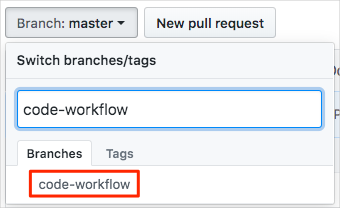
Later, you'll see how to control your pipeline configuration to build from only the branches that you need.

You see that the build completes successfully and produces an artifact that contains the built web application.

**Create a pull request**

Here, you create a pull request for your branch.

1. In a browser, sign in to [GitHub](https://www.github.com/" \t "az-portal).
2. Go to your **mslearn-tailspin-spacegame-web** repository.
3. In the **Branch** drop-down list, select your code-workflow branch.

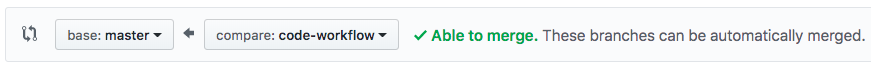


1. To start your pull request, select **New pull request**.

The "New pull request" button

1. Ensure that the **base** specifies your forked repository and not the Microsoft repository.

Your selection looks like this:



**Important**

This step is important because you can't merge your changes into the Microsoft repository. Make sure that the base repository points to your GitHub account and not MicrosoftDocs.

If you end up with a pull request against MicrosoftDocs, simply close the pull request and repeat these steps.

This process involves an extra step because you're working from a forked repository. When you work directly with your own repository, and not a fork, your master branch is selected by default.

1. Enter a title and description for your pull request.
   * Title:

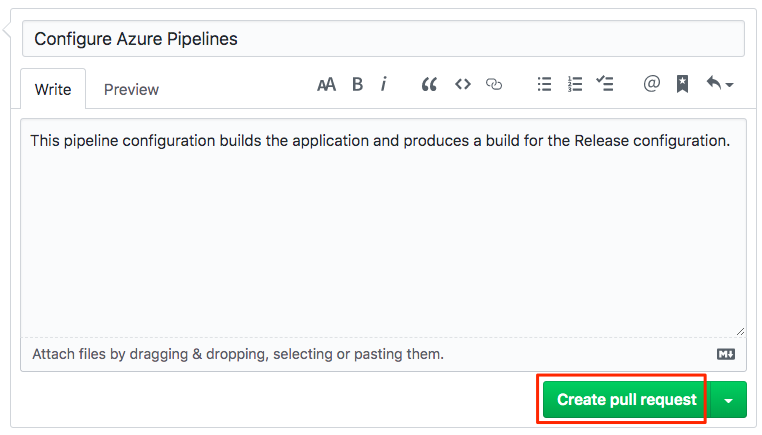
***Configure Azure Pipelines***

* + Description:

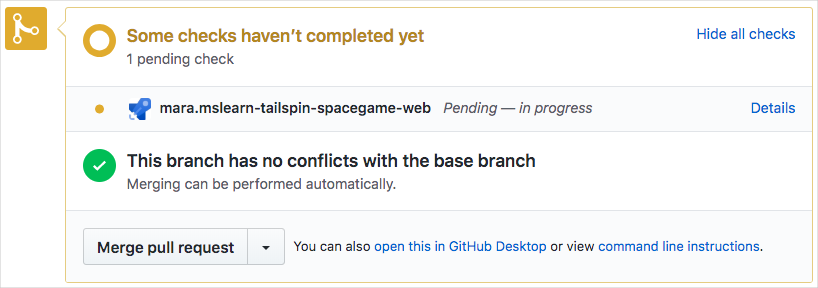
*This pipeline configuration builds the application and produces a build for the Release configuration.*

1. To complete your pull request, select **Create pull request**.

This step does not merge any code. It tells others that you have changes that you are proposing be merged into master.



The pull request window is displayed. You can see that the build status in Azure Pipelines is configured to appear as part of the pull request. That way, you and others can view the status of the build as it's running.



Just like when you push a branch to GitHub, a pull request, by default, triggers Microsoft Azure Pipelines to build your application.

**Tip**

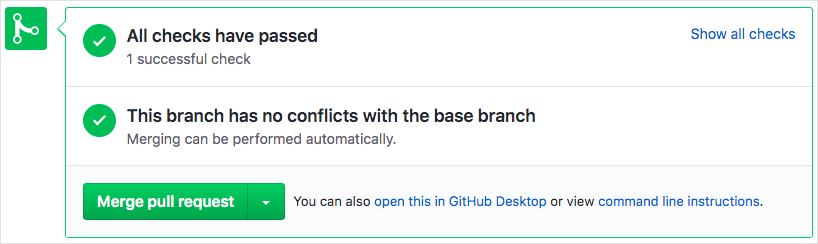
If you don't see the build status appear right away, wait a few moments or refresh the page.

1. Optionally, select the **Details** link, and then trace the build as it moves through the pipeline.

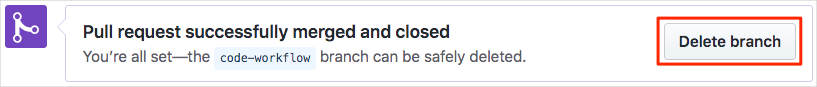
You can hand off your build to the next step in the process, such as QA. Later, you can configure the pipeline to push your change all the way out to your QA lab or production.

1. Go back to your pull request on GitHub.

Wait for the build to complete. You're now ready to merge your pull request.



1. Select **Merge pull request**, and then select **Confirm merge**.
2. To delete the code-workflow branch from GitHub, select **Delete branch**.



It's completely safe to delete a branch from GitHub after you've merged your pull request. In fact, it's a common practice, because the branch is no longer needed. The changes are merged and you can still find the details about the changes on GitHub or from the command line. Deleting a merged branch also helps others see only the work that's currently active.

Git branches are meant to be short-lived. After you merge a branch, you don't push additional commits onto it or merge it a second time. In most cases, every time you start on a new feature or bug fix, you start with a clean branch that's based on the master branch.

Deleting a branch on GitHub doesn't delete that branch from your local system. To do that, you would pass the -d switch to the git branch command.

**How many times is a change built?**

The answer depends on how your build configuration is defined. Azure Pipelines enables you to define *triggers* that specify which events cause builds to happen. You can control which branches get built or even which files trigger the build.

As an example, let's say that you want a build to happen when a change is pushed to GitHub on any Git branch. But you don't want the build to happen when the only changes are to files in your project's *docs* folder. You might include this trigger section in your build configuration:

ymlCopy

trigger:

branches:

include:

- '\*' # build all branches

paths:

exclude:

- docs/\* # exclude the docs folder

By default, a build is triggered when a change is pushed to any file on any branch.

A *continuous integration* (CI) build is a build that runs when you push a change to a branch.

A *pull request* (PR) build is a build that runs when you open a pull request or when you push additional changes to an existing pull request.

The changes you make through the code-workflow branch are built under three conditions:

* A CI build happens when you push changes to the code-workflow branch.
* A PR build happens when you open a pull request on the code-workflow branch against the master branch.
* A final CI build happens after the pull request is merged to master.

PR builds help you verify that your proposed changes *will* work correctly after they're merged to master or another target branch.

The final CI build verifies that the changes are still good after the PR was merged.

As an optional step, go to Azure Pipelines and watch the final CI build happen on the master branch.