**Exercise - Promote to the Test stage**

Your release pipeline still has two stages, but they're now different than before. The stages are *Build* and *Dev*. Every change you push to GitHub triggers the *Build* stage to run. The *Dev* stage runs only when the change is in the *release* branch. Here you add the *Test* stage to the pipeline.

Recall that the team decided to use a scheduled trigger to promote the build from the *Dev* stage to the *Test* stage at 3 A.M. each morning. To set up the scheduled trigger:

* Define the schedule in your build configuration.
* Define the *Test* stage, which includes a condition that runs the stage only if the build reason is marked as Schedule.

For learning purposes, here you define the schedule but allow the build to go directly from *Dev* to *Test*. This setup avoids the need to wait for the schedule to be triggered. After you complete this module, experiment with different cron expressions to run the *Test* stage only at the scheduled time.

**Promote changes to the Test stage**

Here you modify your pipeline configuration to deploy the build to the *Test* stage.

1. In Visual Studio Code, modify *azure-pipelines.yml* like this:

**yml**

trigger:

- '\*'

variables:

buildConfiguration: 'Release'

releaseBranchName: 'release'

schedules:

- cron: '0 3 \* \* \*'

displayName: Deploy every day at 3 A.M.

branches:

include:

- release

always: false

stages:

- stage: 'Build'

displayName: 'Build the web application'

jobs:

- job: 'Build'

displayName: 'Build job'

pool:

vmImage: 'ubuntu-18.04'

demands:

- npm

variables:

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

- publish: '$(Build.ArtifactStagingDirectory)'

artifact: drop

- stage: 'Dev'

displayName: 'Deploy to the dev environment'

dependsOn: Build

condition: |

and

(

succeeded(),

eq(variables['Build.SourceBranchName'], variables['releaseBranchName'])

)

jobs:

- deployment: Deploy

pool:

vmImage: 'ubuntu-18.04'

environment: dev

variables:

- group: Release

strategy:

runOnce:

deploy:

steps:

- download: current

artifact: drop

- task: AzureWebApp@1

displayName: 'Azure App Service Deploy: website'

inputs:

azureSubscription: 'Resource Manager - Tailspin - Space Game'

appName: '$(WebAppNameDev)'

package: '$(Pipeline.Workspace)/drop/$(buildConfiguration)/\*.zip'

- stage: 'Test'

displayName: 'Deploy to the test environment'

dependsOn: Dev

#condition: eq(variables['Build.Reason'], 'Schedule')

jobs:

- deployment: Deploy

pool:

vmImage: 'ubuntu-18.04'

environment: test

variables:

- group: 'Release'

strategy:

runOnce:

deploy:

steps:

- download: current

artifact: drop

- task: AzureWebApp@1

displayName: 'Azure App Service Deploy: website'

inputs:

azureSubscription: 'Resource Manager - Tailspin - Space Game'

appName: '$(WebAppNameTest)'

package: '$(Pipeline.Workspace)/drop/$(buildConfiguration)/\*.zip'

The schedules section defines one cron expression. You can define more than one expression in your configuration. The expression triggers the pipeline to run against the release branch at 3 A.M. each day. The always flag is set to false so that the pipeline runs only when the release branch contains changes from the prior run.

The Test stage defines a condition that runs the stage only when the build reason equals Schedule. (The built-in variable Build.Reason defines the build reason.) If this condition is false, the stage is skipped, but the prior stages continue to run.

**Note**

This condition is shown for learning purposes. It's commented to enable the change to go from *Dev* to *Test* without waiting for the schedule to be triggered.

1. From the integrated terminal, add *azure-pipelines.yml* to the index. Then commit the change and push it up to GitHub.

**Tip**

Save *azure-pipelines.yml* before you run these Git commands.

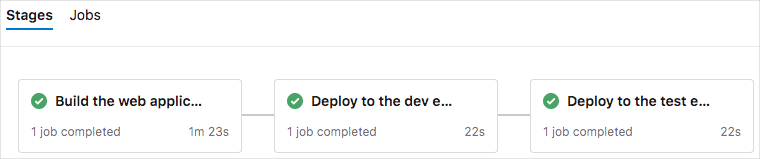
**Bash**

**git add azure-pipelines.yml**

**git commit -m "Deploy to the Test stage"**

**git push origin release**

1. In Azure Pipelines, go to the build. Trace the build as it runs.
2. After the build finishes, select the back button to return to the summary page.

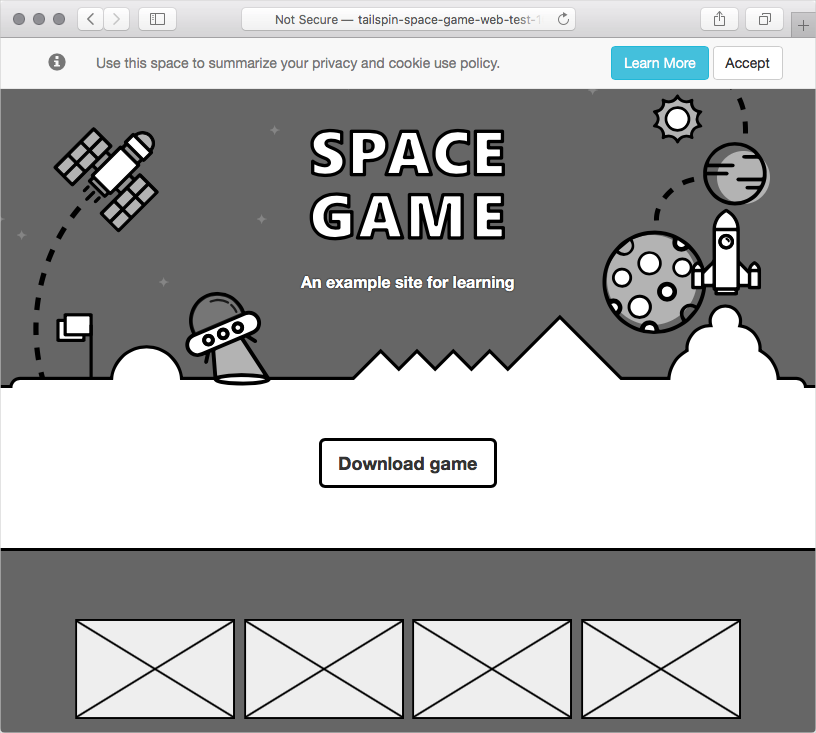


You see that the deployment finished successfully.

1. From a web browser, navigate to the URL that's associated with the App Service instance for your *Test* environment.

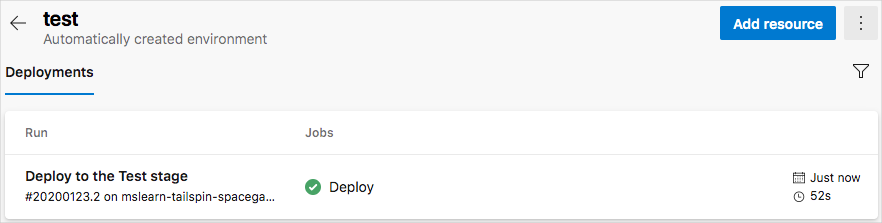
If you still have the browser tab open, refresh the page. If you don't remember the URL, find it in the Azure portal, on the App Service details page.

You see that the *Space Game* website is deployed to App Service, and it's running.



1. As an optional step, in Azure Pipelines, select **Environments**. Then select the **test** environment.

Azure Pipelines records your deployment history. In the history, you can trace changes in the environment back to code commits and work items.



Andy and Mara add the *Test* stage to the pipeline. They show the results to Amita.

**Amita:** I like that changes are built and deployed so that I can test them each morning. But I don't see how I can control when changes arrive at *Staging*.

**Mara:** Yes, deploying through automation is a huge time saver. But remember that we included only the scheduled trigger. We'll add a release approval for you when we set up the *Staging* environment for Tim. That way, changes will move to *Staging* only when you're ready.