App Modernization Labs

Azure App Services with Azure DevOps

Author: Divi Mishra

Cloud Solution Architect, Azure App Innovation, Microsoft Qatar

divimishra@microsoft.com | +974 50219105

Introduction to the Lab Document

Objective

This workshop lab is intended to materialize on theoretical Azure App Services learnings to have hands-on experience with end-to-end Azure App Services tools across the Developer Cloud. Through this workshop lab, you will have a basic yet broad understanding of how to realize value from the different offerings within and beyond Azure App Services.

Motive

When it comes to modernizing your web apps, Azure App Service is the best destination. A recent GigaOM study found out that Azure App Service offers potential total cost of ownership (TCO) savings of up to 54% over running on-premises while offering tangible benefits around streamlined operations, increased developer productivity, DevOps readiness and reduced friction.

Intended Audience

The intended audience for this workshop lab includes, but is not limited to: development team, application managers, enterprise architects, and technical managers. The difficulty level of this workshop is beginners.

Duration

This workshop lab followed step-by-step will take approximately 3 hours to complete.

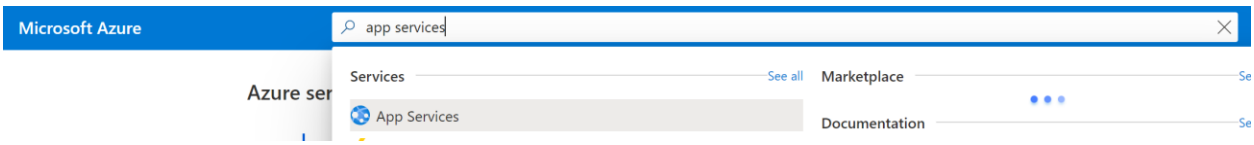
Pre-requisites

1. VS A GitHub account
2. An Azure DevOps setup
3. An active Azure subscription
4. .NET 5.0 SDK
5. Git

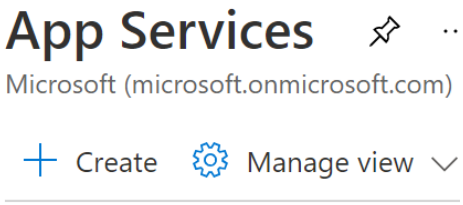
Lab 01: Azure App Service

Create an Azure App Service

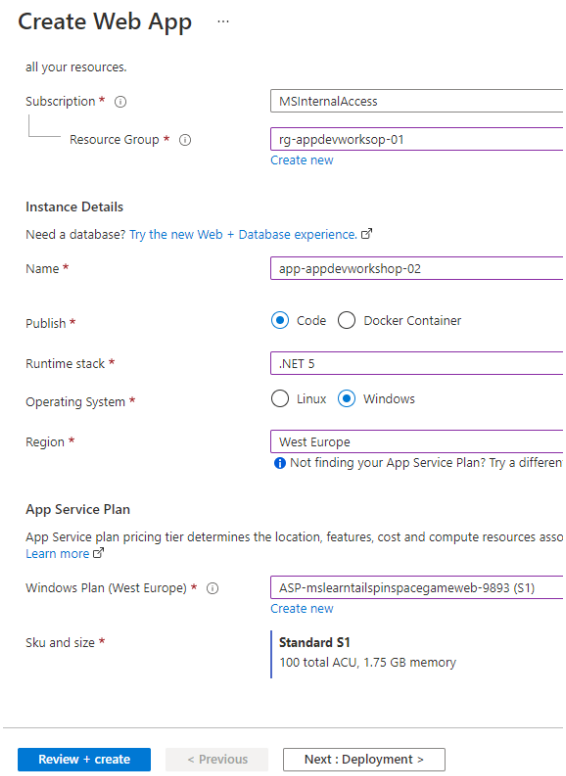
1. Go to the [Azure Portal]
2. Search for **App Services** at the search bar on top



3. Click on **+ Create**

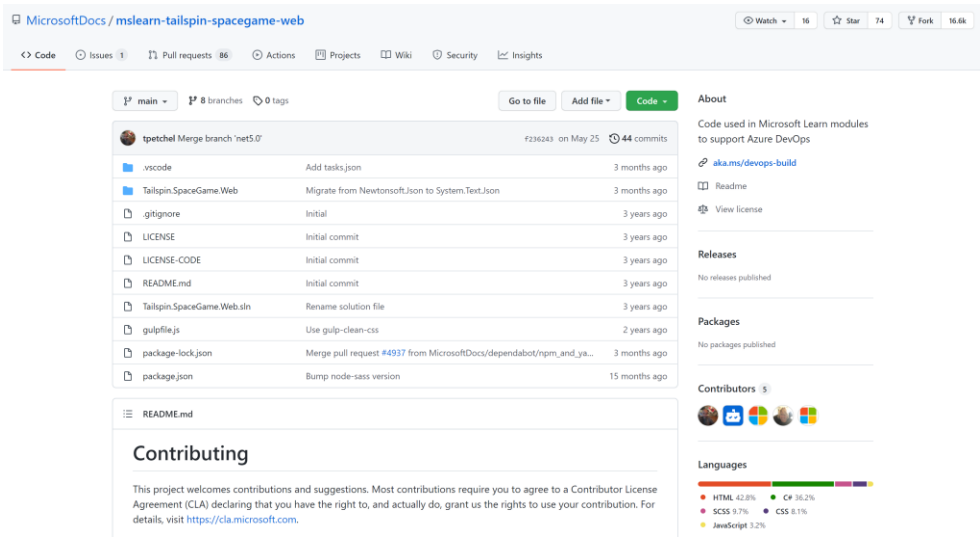


4. Under **Basic**, enter the following details:
   - **a.** Select the relevant RG or create a new one
   - **b.** Type an app service name. Example: app-appdevworkshop-01
   - **c.** Publish: **Code**
   - **d.** Runtime Stack: **.NET 5**
   - **e.** Operating System: **Windows**
   - **f.** Region: **West Europe** (or any other region close to you)
   - **g.** App Service Plan:
     - **i.** Create new > Enter a name (example: asp-appdevworkshop)
     - **ii.** Select **Standard S1** under Production as your app service plan size
5. Click on **Review + Create**
6. Click on **Create**

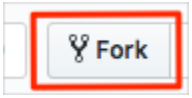


Fork the web project to your GitHub account

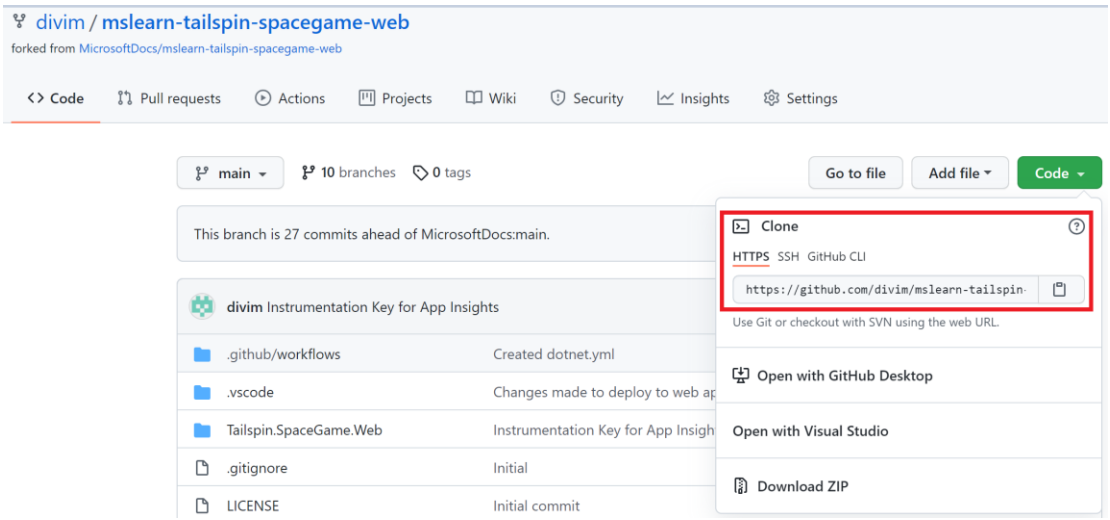
7. Go to GitHub and sign in
8. Visit the Space Game web project (ASP.Net Core application)



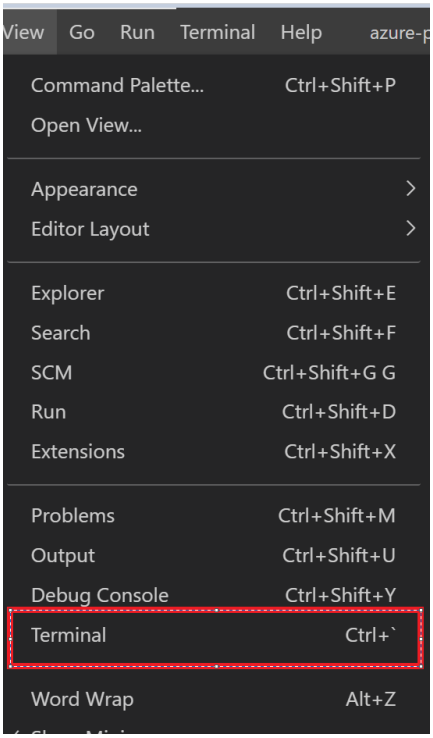
9. From the top right corner, fork your own copy of the repo to your account.



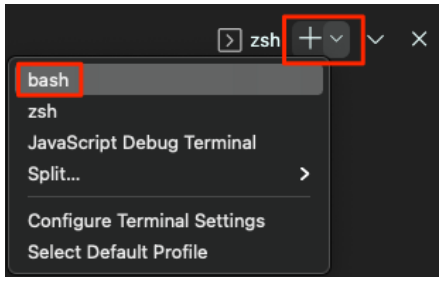
10. Go to your fork of the Space Game project. The forked repository will be saved as *"<your-account-name>/mslearn-tailspin-spacegame-web"*.
11. Select "**Code**". Under the "**HTTPS**" tab, copy the URL.



12. Open Visual Studio code and open the terminal window. You can do so by going to View > Terminal. Alternatively, the keyboard shortcut is "Ctrl + `". Ensure that your terminal is set to Git Bash.



13. Ensure that your terminal is set to bash



14. Run the **git clone** command with the URL you copied. Your command will look like:

    $ `git clone https://github.com/<your-name>/mslearn-tailspin-spacegame-web.git`

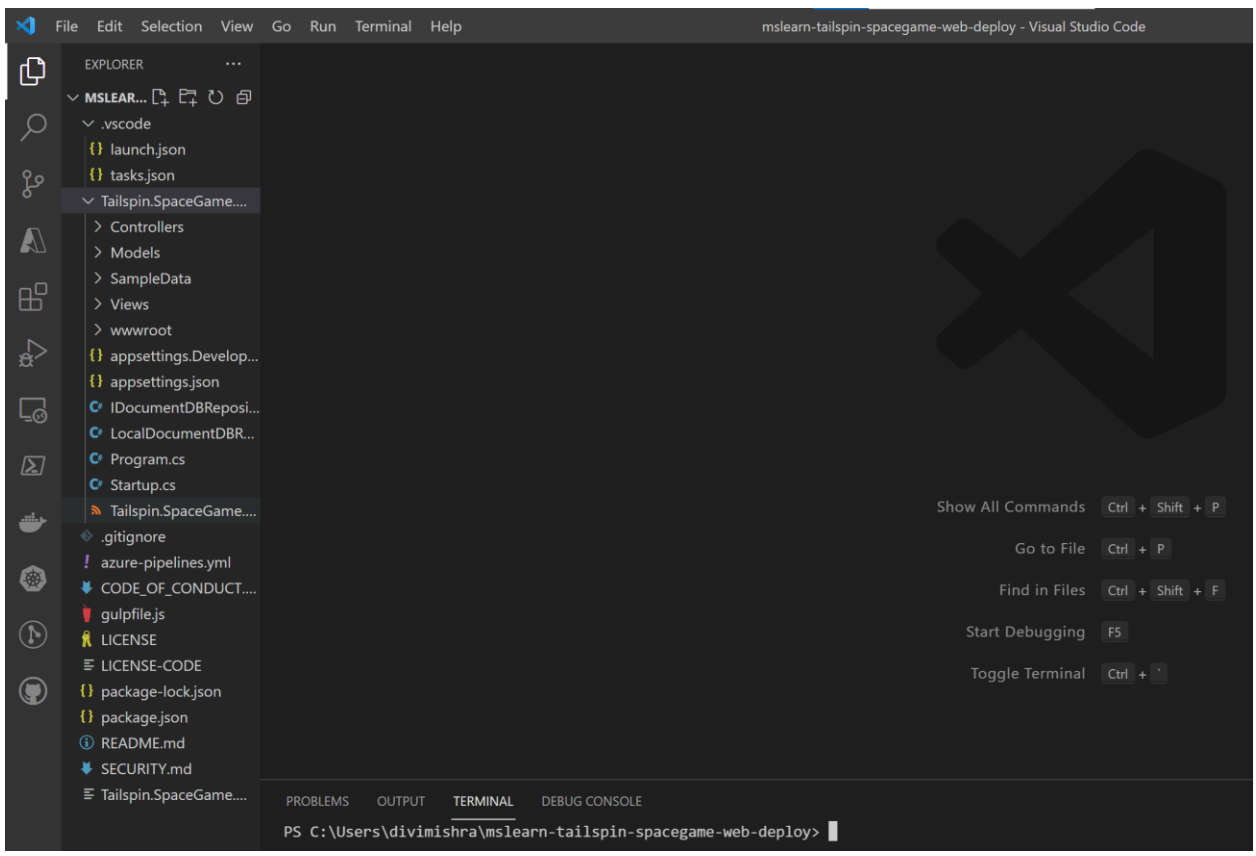
15. Move to the directory with the Space Game project by entering the following command:

    $ `cd mslearn-tailspin-spacegame-web`

16. Open the project with the following command:

    $ `code -r .`

17. You are now at the root of your web project. Reopen the terminal by going to "View > Terminal" or "Ctrl + `"



18. Enter the following commands with "Sample Name" and sampleemail@abc.com replaced with your name and your commit email address.

$ `git config –-global user.name “Sample Name”`

$ `git config –-global user.email “sampleemail@abc.com`

Note: The “--global” tag sets the entered username and email address for every repository on your computer. Is you want to set your username or email address for a single repository, use:

$ `git config user.name “Sample Name”`

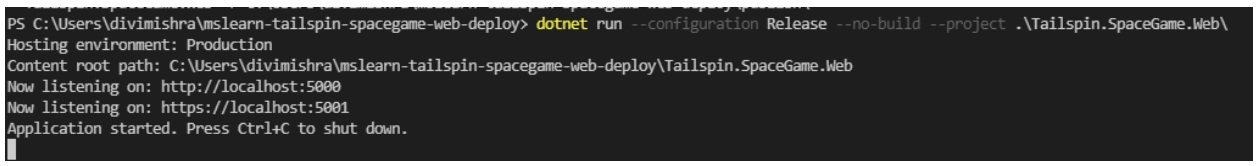
$ `git config user.email “sampleemail@abc.com`

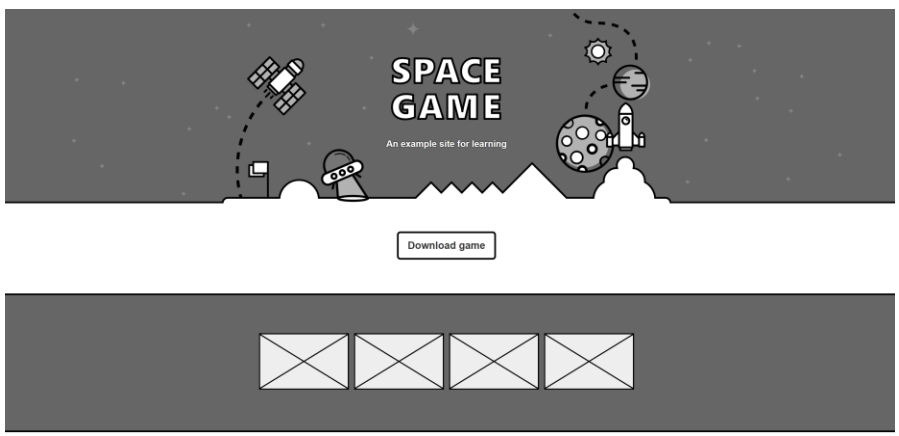
19. Enter the following command on the terminal to make sure the code runs on your local host:

$ dotnet build --configuration Release

$ dotnet run –-configuration Release –-no-build –-project Tailspin.SpaceGame.Web

Navigate to your http://localhost:5000



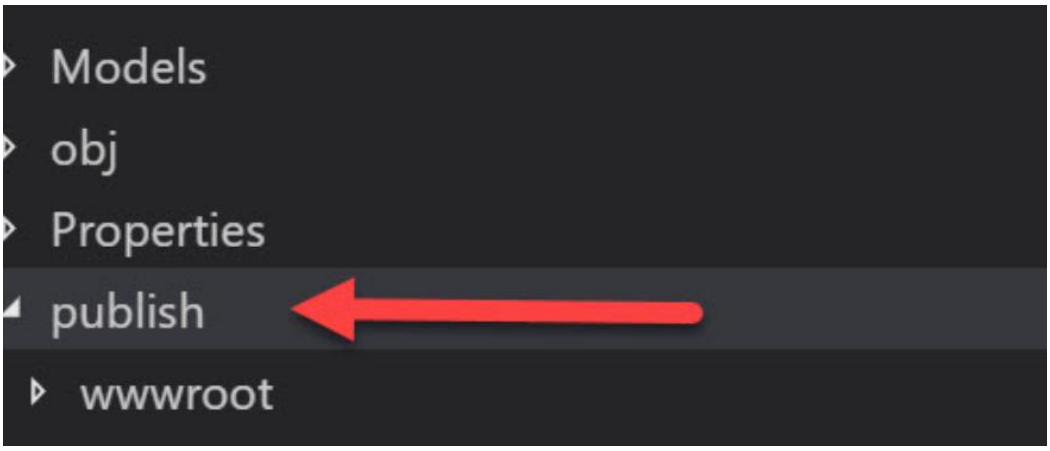


Ctrl+C when you’re done.

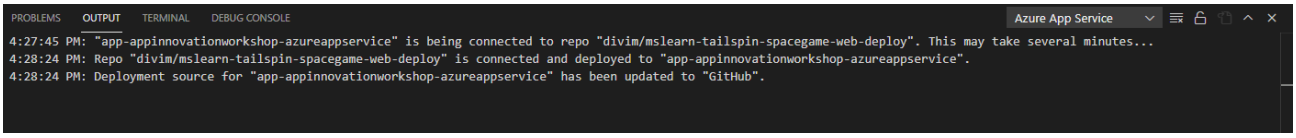
20. Go back to your terminal and enter the following command:

$ dotnet publish -c Release -o ./publish

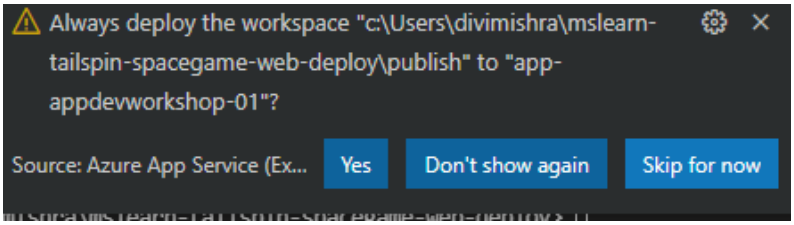
You will notice that a new **publish** folder has been created.



21. Right click on the **publish** folder.
22. Select **Deploy to web app**… and select the right subscription.
23. Select the Azure App service you had created.
24. Select **Deploy** to confirm.
25. To track the progress, open the terminal window. You can do so by going to View > Terminal. Alternatively, the keyboard shortcut is "Ctrl + `". Navigate to **Output** and ensure that your selection is "**Azure App Service**" from the dropdown.



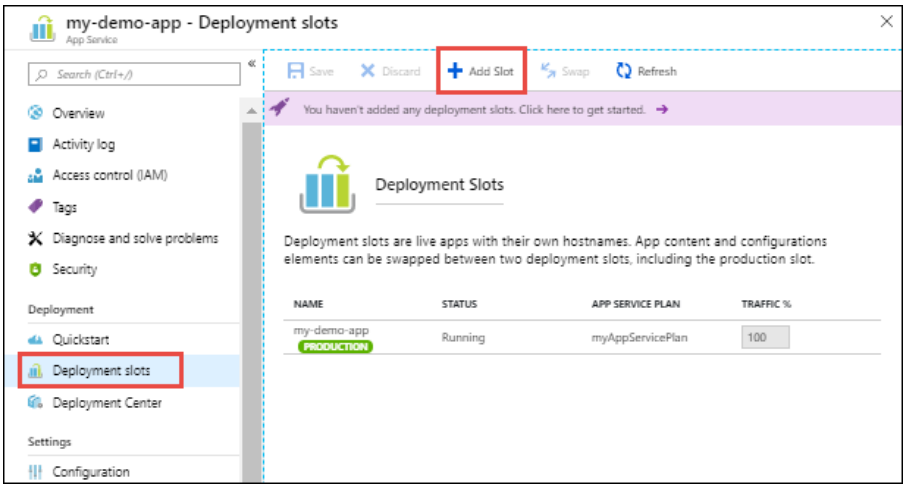
26. If you get the following prompt, click on **Skip for now**. We will be deploying Azure Pipelines for CI/CD in Lab 4.



27. Once the deployment is done, click on Browse Website to validate the deployment.

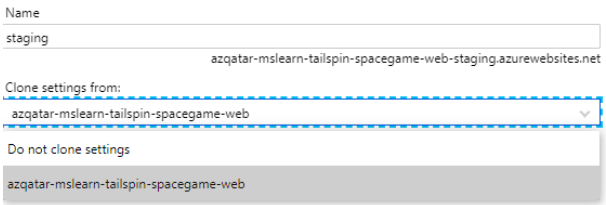
Create deployment slots

1. Navigate to your newly created app service from Azure Portal.
2. Under deployment, click on Deployment Slots. Then, add a slot.



Note: The app service tier must be Standard, Premium, or Isolated to enable staged publishing.

3. In the "Add Slot" dialog box, give the slot a name "**staging"** and select whether you want to clone an app configuration from another deployment slot.



4. Once the slot has been created, close the dialog box. You will notice that for the staging environment:
    a. The default traffic is set to 0.
    b. The slot's URL will be of the format http://sitename-slotname.azurewebsites.net
    c. The web URL is empty even if we clone it from the production app. Hence, you can use a separate branch or a separate repository altogether to test the application.

Note: If you would like to add an access rule restriction for the staging environment, please follow the steps from this document: [Azure App Service access restrictions - Azure App Service | Microsoft Docs]

Lab 02: Monitor your app with Azure Application Insights

Enabling live telemetry through instrumentation key using VS Code

1. Navigate to your Azure App Service through your Azure Portal.
2. Under settings, go to Application Insights.
3. Make sure that the collection is **Enabled.**
4. Click on Apply.
5. Go to Cloud Shell.
6. Enter the following command to get the instrumentation key:

```
az resource show \
  --resource-group <resource_group_name> \
  --name <resource_name> \
  --resource-type "Microsoft.Insights/components" \
  --query properties.InstrumentationKey
```

   Copy the output value.

7. On the VS Code terminal, navigate to ~/mslearn-tailspin-spacegame-web/Tailspin.SpaceGame.Web using the cd command.
8. Run the following command:

   dotnet add package Microsoft.ApplicationInsights.AspNetCore --version 2.18.0

9. Navigate to Tailspin.SpaceGame.Web**.csproj** file. Add/ensure the following code sample:

```
<ItemGroup>
    <PackageReference Include="Microsoft.ApplicationInsights.AspNetCore" Version="2.16.0" />
</ItemGroup>
```

   Your code will look something like this:

Tailspin.SpaceGame.Web.csproj

Tailspin.SpaceGame.Web > Tailspin.SpaceGame.Web.csproj

You, 2 days ago | 2 authors (divim and others)

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
    <ProjectGuid>{A0C4E31E-AC75-4F39-9F59-0AA19D9B8F46}</ProjectGuid>
  </PropertyGroup>

  <ItemGroup>
    <Folder Include="wwwroot\images\avatars\" />
  </ItemGroup>

  <ItemGroup>
    <Content Remove="SampleData\profiles.json" />
    <Content Remove="SampleData\scores.json" />
    <Content Remove="SampleData\profiles.json" />
    <Content Remove="SampleData\scores.json" />
  </ItemGroup>
  <ItemGroup>
    <Content Update="SampleData\profiles.json">
      <CopyToOutputDirectory>Always</CopyToOutputDirectory>
    </Content>
    <Content Update="SampleData\scores.json">
      <CopyToOutputDirectory>Always</CopyToOutputDirectory>
    </Content>
  </ItemGroup>
  <ItemGroup>
    <EmbeddedResource Include="SampleData\profiles.json">
      <CopyToPublishDirectory>PreserveNewest</CopyToPublishDirectory>
    </EmbeddedResource>
    <EmbeddedResource Include="SampleData\scores.json">
      <CopyToPublishDirectory>PreserveNewest</CopyToPublishDirectory>
    </EmbeddedResource>
  </ItemGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.ApplicationInsights.AspNetCore" Version="2.18.0" />
  </ItemGroup>
</Project>
```

10. Navigate to the Startup class under **Startup.cs**. Add the following command under the ConfigureServices() method:

    services.AddApplicationInsightsTelemetry();

    services.AddMvc();

    The method will look something like this:

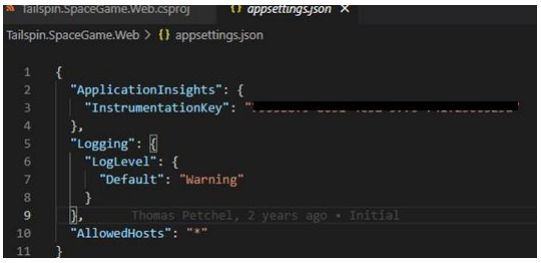
    Screenshot:

```
namespace TailSpin.SpaceGame.Web
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllersWithViews();
            services.Configure<CookiePolicyOptions>(options =>
            {
                // This lambda determines whether user consent for non-essential cookies is needed for a given request.
                options.CheckConsentNeeded = context => true;
                options.MinimumSameSitePolicy = SameSiteMode.None;
            });

            // Add document stores. These are passed to the HomeController constructor.
            services.AddSingleton<IDocumentDBRepository<Score>>(new LocalDocumentDBRepository<Score>(@"SampleData/scores.json"));
            services.AddSingleton<IDocumentDBRepository<Profile>>(new LocalDocumentDBRepository<Profile>(@"SampleData/profiles.json"));

            // The following line enables Application Insights telemetry collection.
            services.AddApplicationInsightsTelemetry();

            // This code adds other services for your application.
            services.AddMvc();
        }
```

11. Although you can add the instrumentation key as an argument to AddApplicationInsightsTelemetry, we recommend that you specify it in configuration. Under **appsettings.json** file, edit the file to the following:

```
{
    "ApplicationInsights": {
     "InstrumentationKey": "putinstrumentationkeyhere"
    },
    "Logging": {
     "LogLevel": {
      "Default": "Warning"
     }
    }
  }
```

Screenshot:



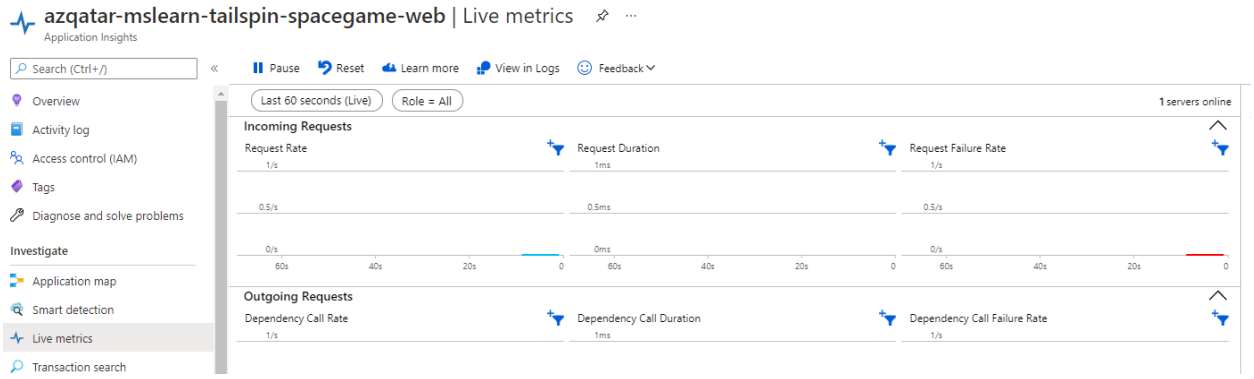
Save all the changes.

12. As done in the previous lab, Go back to your terminal and enter the following command:

```
$ dotnet publish -c Release -o ./publish
```

13. Right-click on your app service and click on **Deploy to web app** for the changes to be reflected. (Soon, you'll deploy a CI/CD pipeline using Azure Pipelines to automate this process and other processes for you).

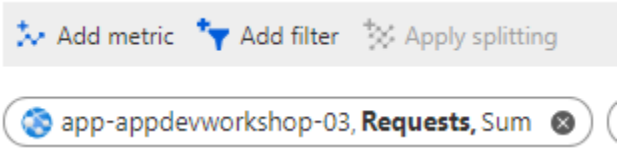
14. To verify that the application insights was configured accurately, navigate to **Live Metrics.** Please note that it may take a few minutes for the telemetry to appear.



This verifies that your application insights with instrumentation key is enabled accurately!

Monitor apps

1. Under **Monitoring,** click on **Metrics**
2. Select **Requests** as your metric, and **Sum** as your aggregation.
3. Select **Add metric** from the top left.



4. Select metrics of your choice and display them into meaningful bar representations.

5. Explore the **Pin to Dashboard** option at the top right for developing shared dashboards to share with team members.

Select Pin to Dashboard to be able to easily monitor your desired queries and metrics.

Configure alerts for Azure App Service

1. Navigate to your App Service from the Azure Portal.
2. Under Monitoring, select Alerts.
3. Select “Add Alert Rule”.
4. Select the resource as the App Service.
5. Select the Condition as Http 4xx
    a. Set the Alert logic for “greater than 10”
    b. Leave the granularity and evaluation as their default values.

    Note: You can add up to 5 conditions with a static threshold. The alerts will be evaluated with a logical AND.
6. Select the Action Group: You can leave it as Application Insights Smart Detection or “Create Action Group” for your team.
7. Select the alert rule name, description, severity. The following is a table that describes severity:
    - Sev 0 = Critical
    - Sev 1 = Error
    - Sev 2 = Warning
    - Sev 3 = Informational
    - Sev 4 = Verbose
8. Select “Create Alert Rule”. It may take a few minutes to be configured.

Lab 03: Scale applications with Azure App Service Plan

Setting up a dashboard to review changes

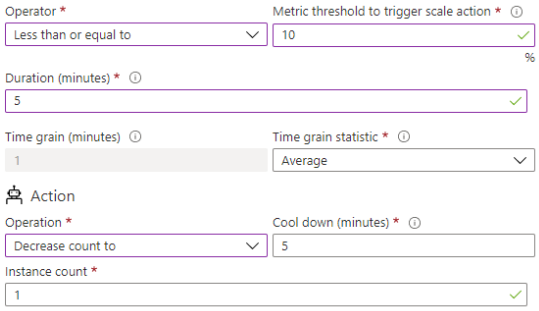
1. Navigate to your app service through Azure Portal.
2. Under Monitoring, select Metrics.
3. From the top left, change “Local time: Last 24 hours (Automatic)” to “Last 30 minutes”. Then, select Apply.
4. In the pane, under Chart Title, add the following metrics to the chart:
    a. Select Add metric, and under the Metric dropdown list, select CPU Time. For Aggregation, select Sum.
    b. Select Add metric, and under the Metric dropdown list, select Http Server Errors. For Aggregation, select Sum.
    c. Select Add metric, and under the Metric dropdown list, select Http 4xx. For Aggregation, select Sum.
    d. Select Add metric, and under the Metric dropdown list, select Response Time. For Aggregation, select Avg.
5. Select “Pin to Dashboard”. Click on the notification for viewing your dashboard. d

Scale up your app service

1. Navigate to “Scale up (App Service plan)”.
2. Choose your tier and select **Apply.**

Scale out your app service

1. Navigate to “Scale out (App Service plan)”.
2. **Manual Scale:**
    a. Configure your instance count to your desired instance count.
    b. Select Save.
    c. Review your dashboard to monitor performance.
3. **Automatic Scale (Custom Autoscale):**
    a. Add a rule to set instance count to 1 if the CPU percentate is less than 10%.



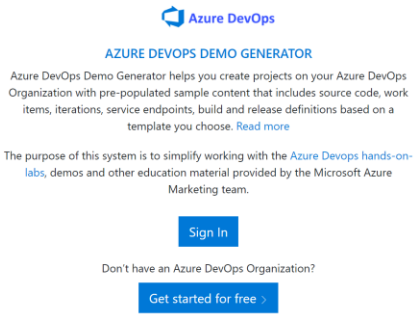
b. Click on Save.

c. Review your dashboard to monitor performance.

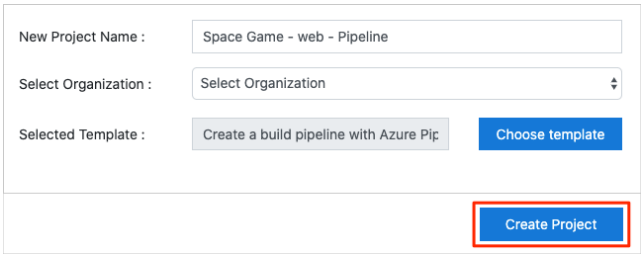
Lab 04: Continuous Integration with Azure DevOps

Configure Azure DevOps project

1. We will be using a template that sets everything up in your Azure DevOps organization. Run the template using this link: [Azure DevOps Demo Generator]



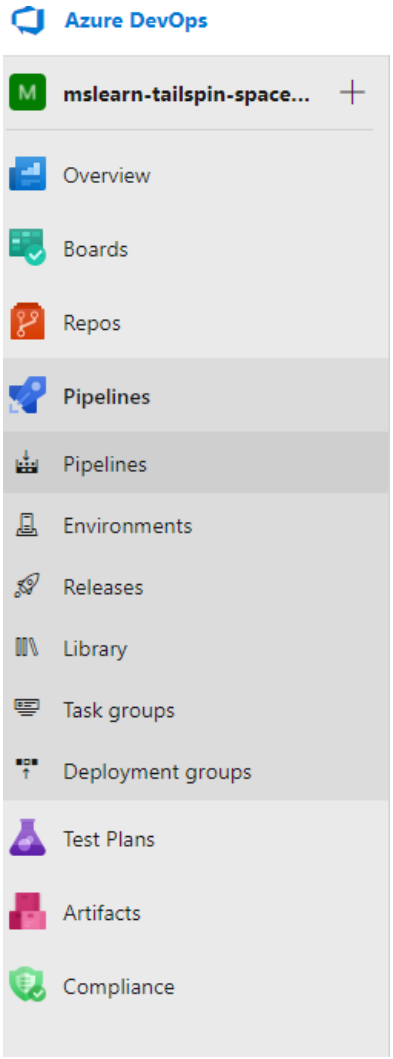
2. Sign in and accept the usage terms.
3. Create a new project with your Azure DevOps (ADO) organization and a project name. For Selected template: under MS learn, chose "Create a build pipeline with Azure Pipelines". Finally, create project.



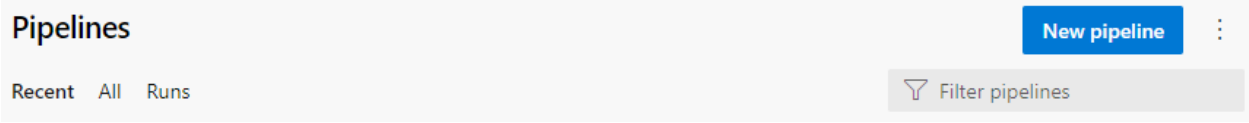
4. Once the deployment is done, select "Navigate to the project"

Create a build pipeline with Azure Pipelines

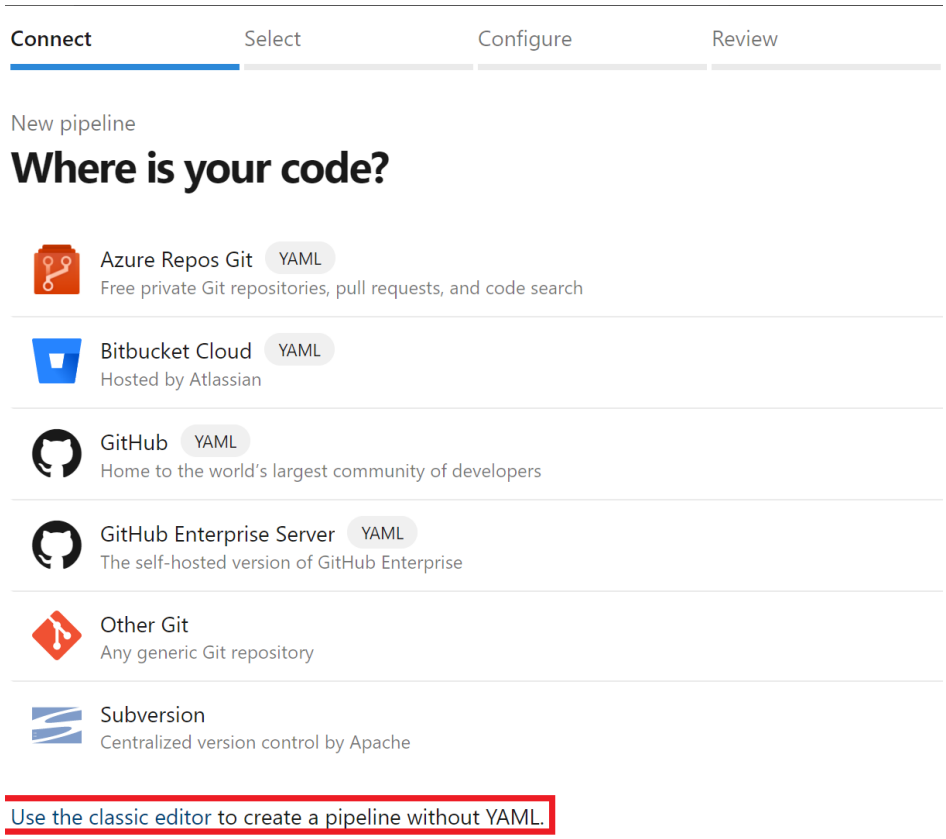
1. Once your project is deployed, navigate to **Pipelines**.



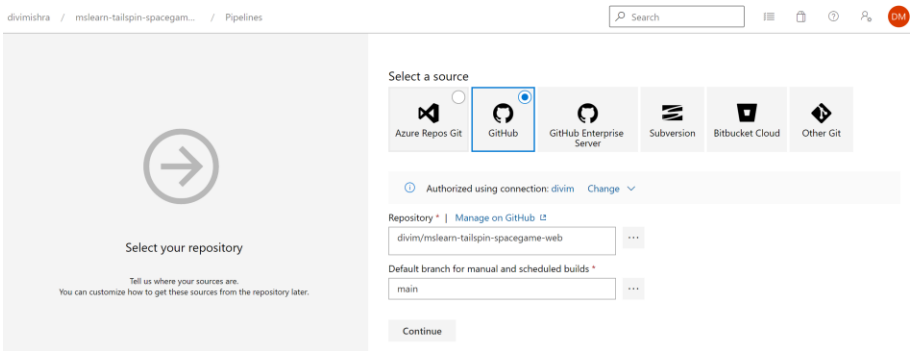
2. Create a new pipeline.



3. Use the classic editor option at the bottom.

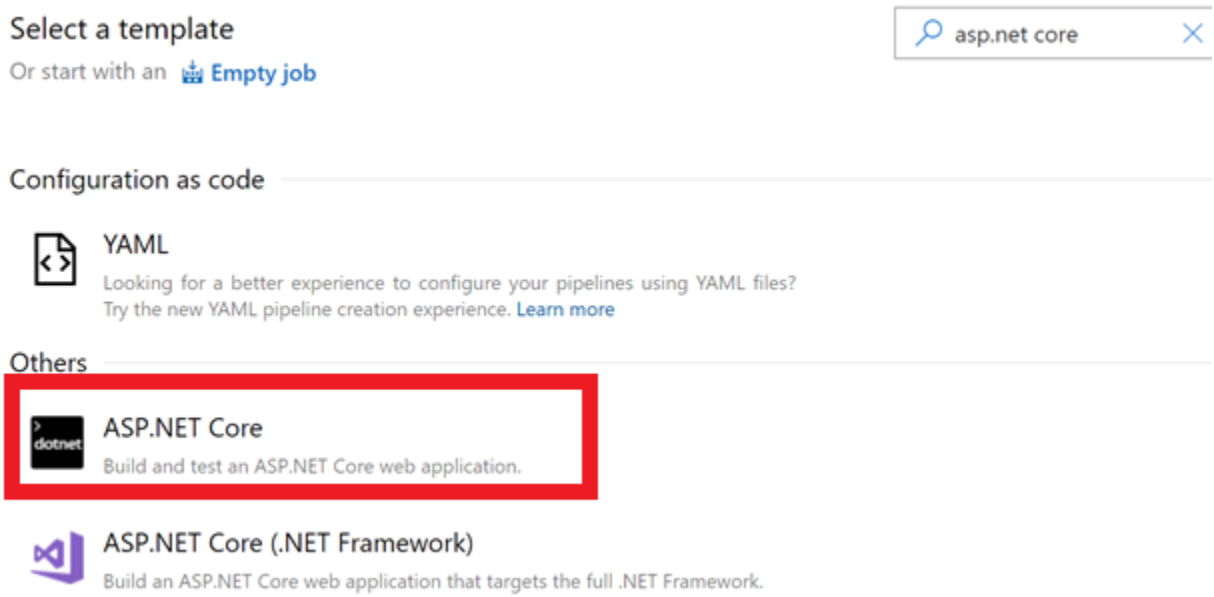


4. Select your source as GitHub. Select your forked repository and the default branch as **main**.

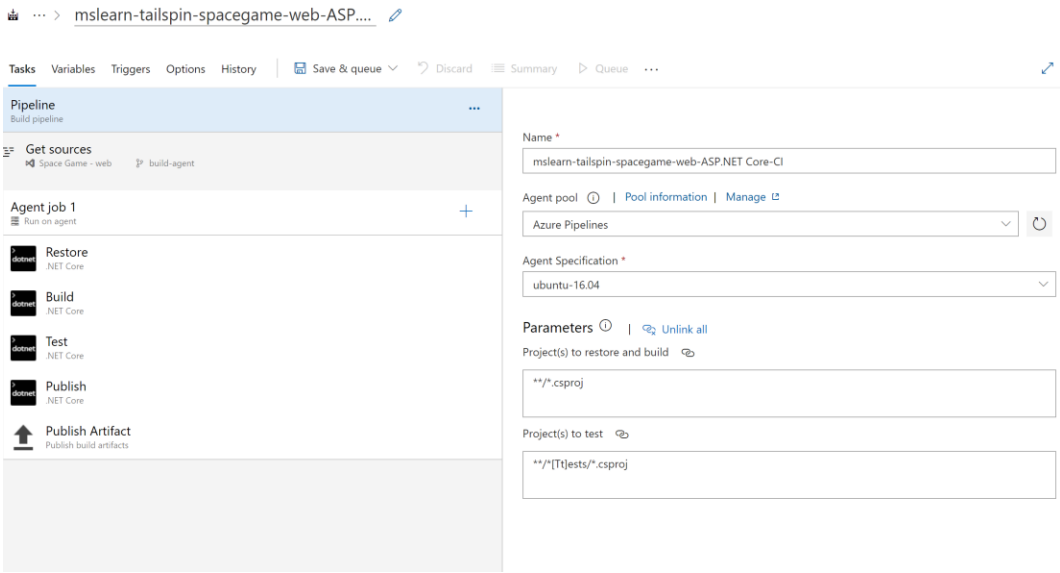


Select "Continue".

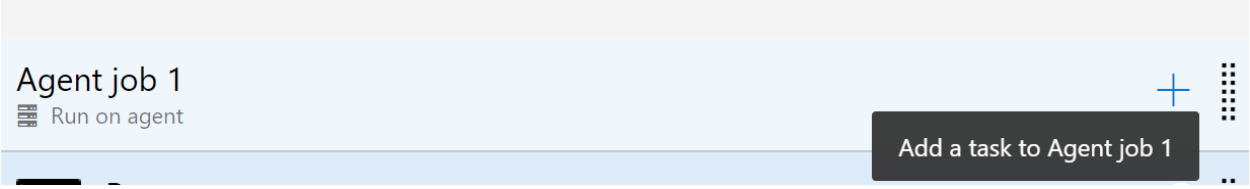
5. Search for **ASP.NET Core.** Apply that as your template.



6. Your pipeline currently looks like this:

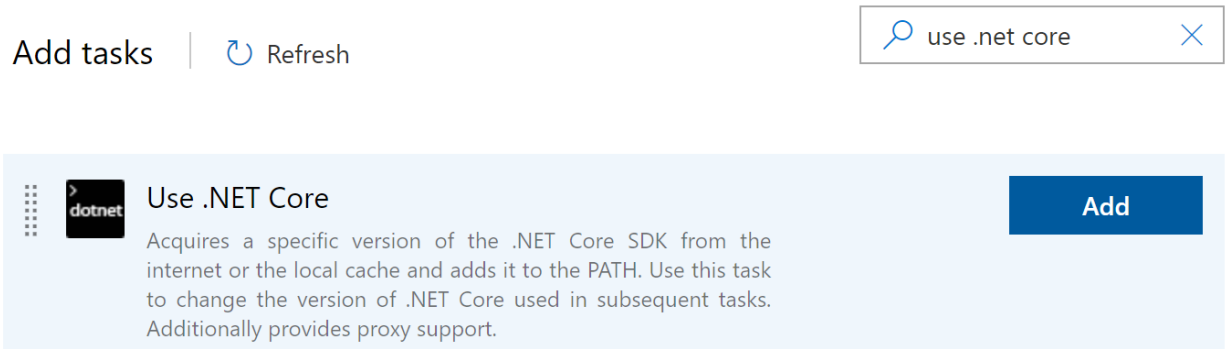


7. Click on the "+" to add an agent job.

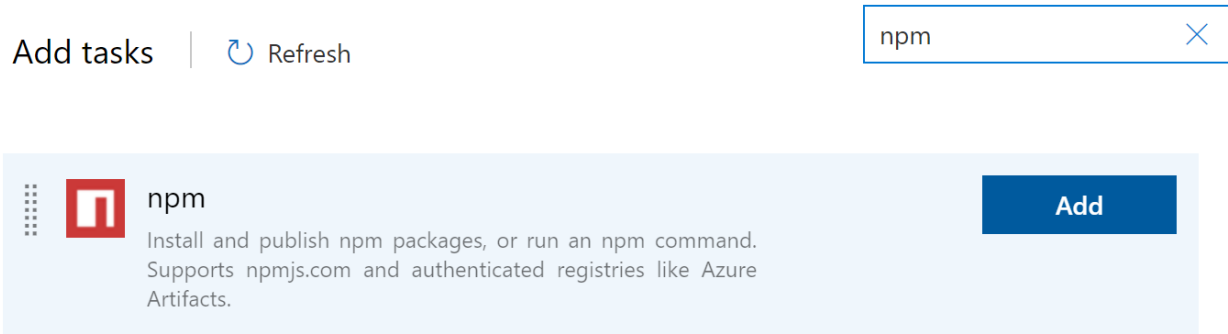


8. Search and click on **"Add"** for the following tasks:

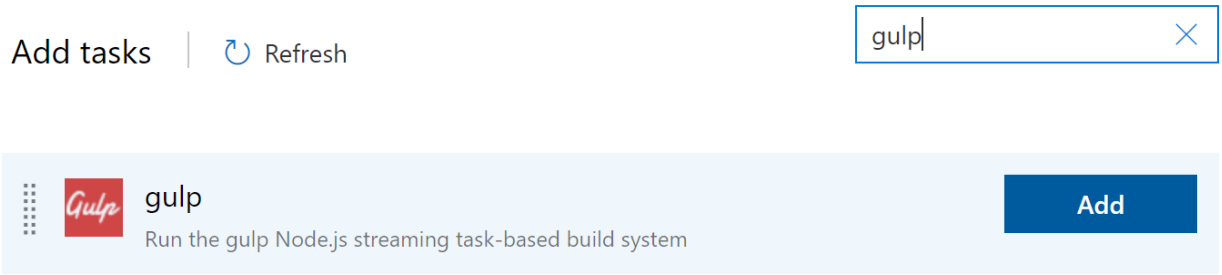
   a. Use. NET Core



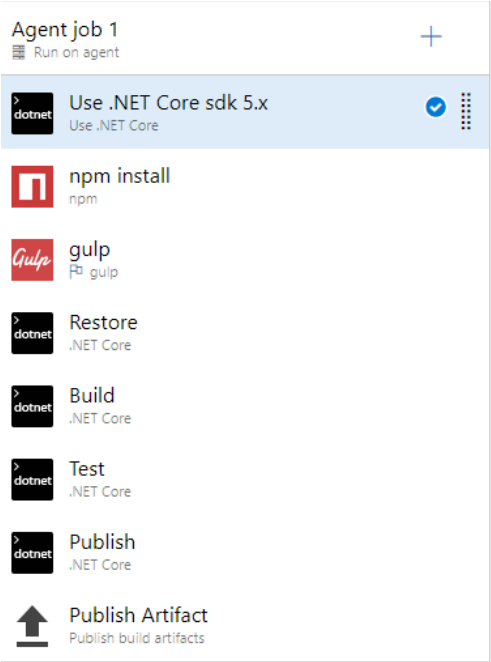
   b. npm



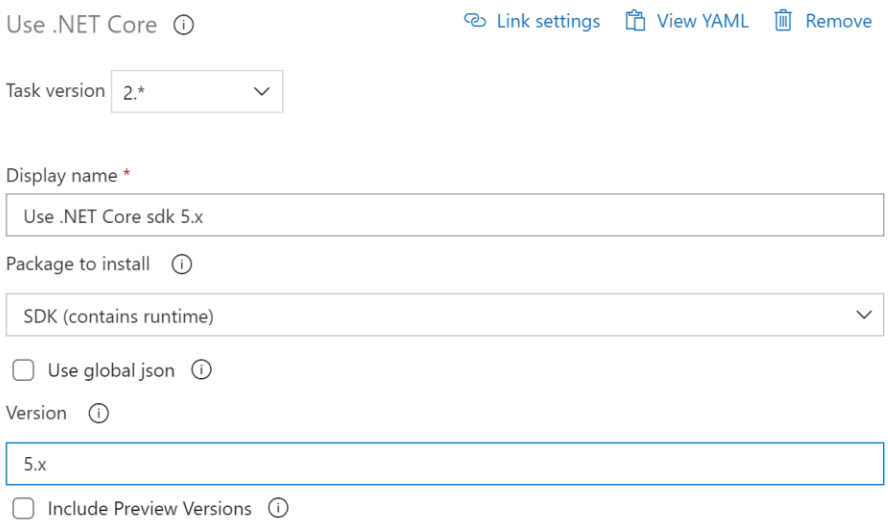
   c. gulp



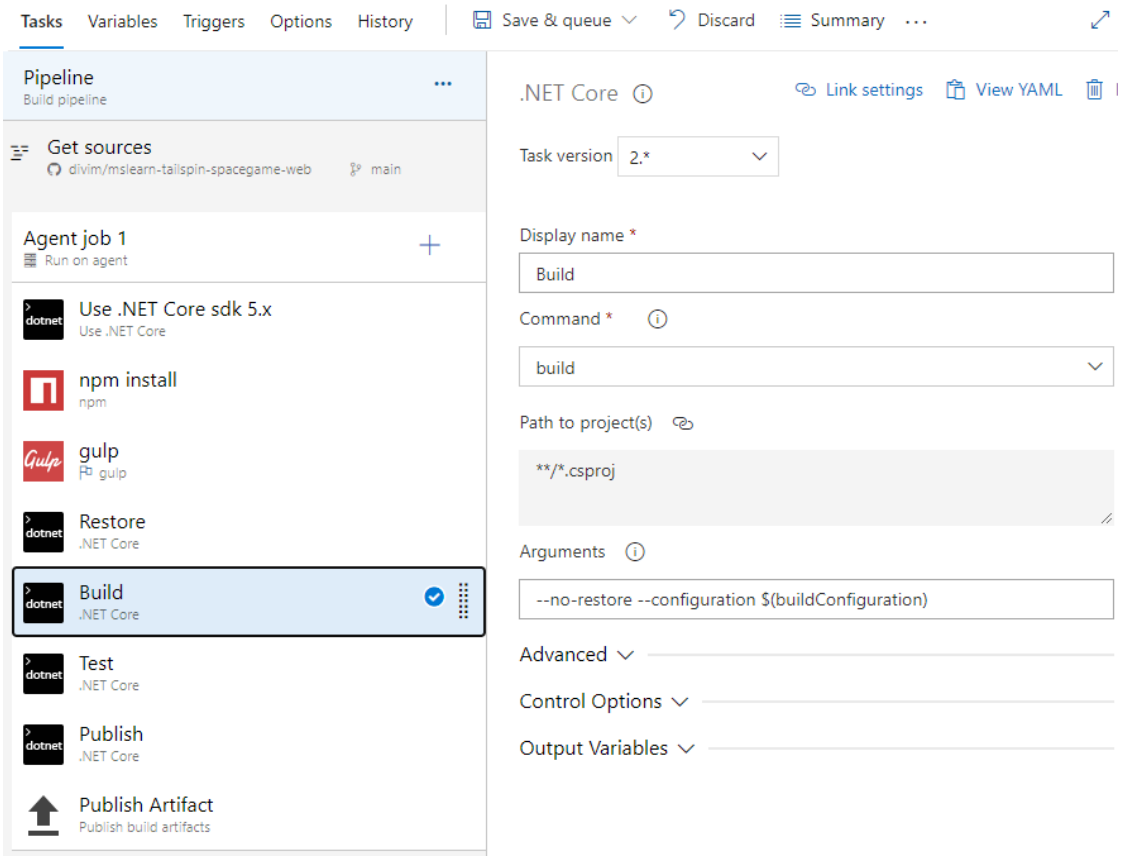
9. Your pipeline will now look like this:



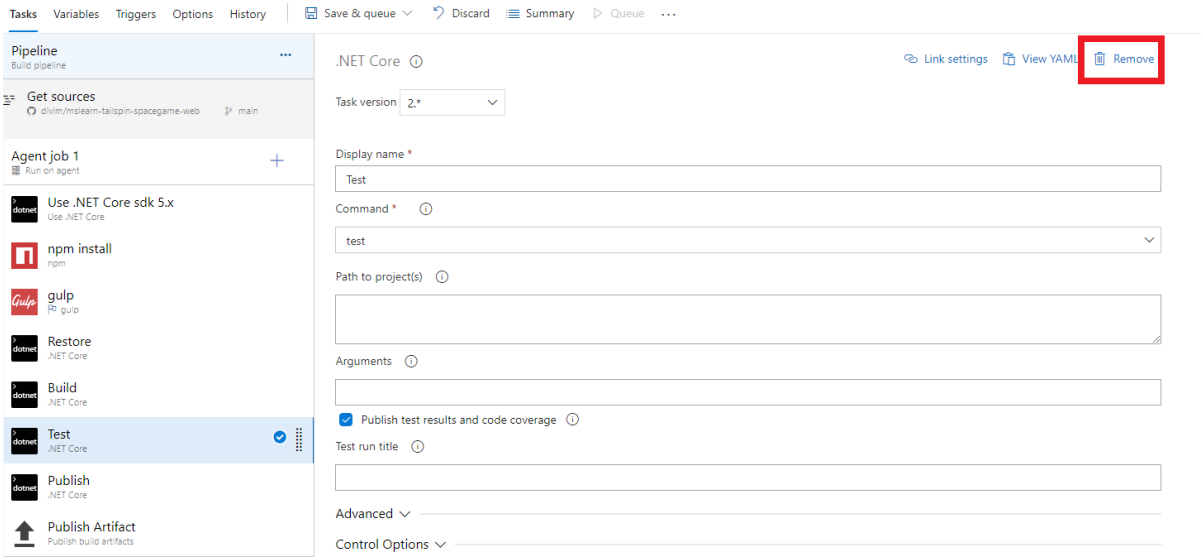
10. Click on the "Use .NET Core sdk" task and write the version as "**5.x**".



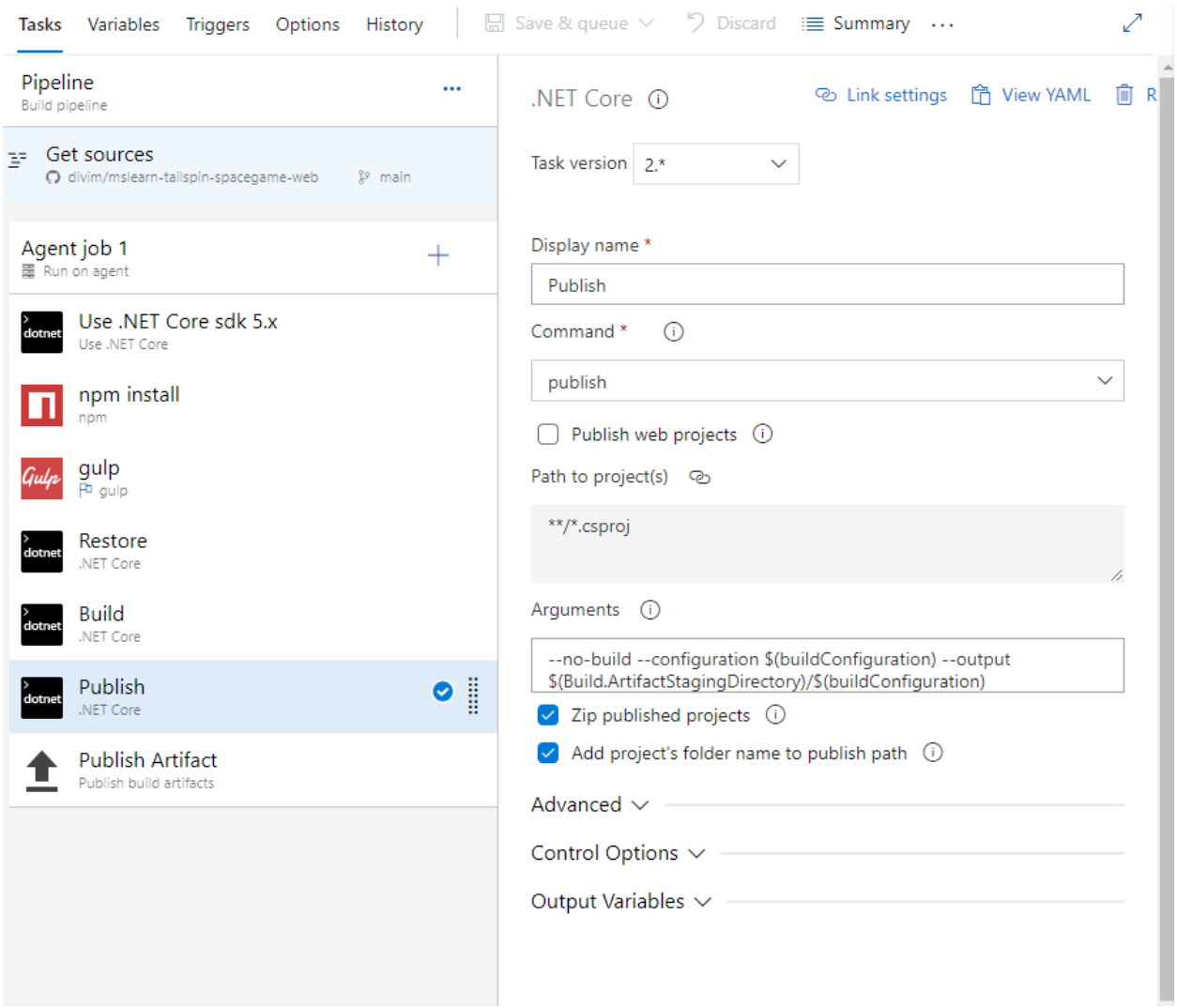
11. Click on the "**Build**" task and edit the "**Arguments**" as follows: `--no-restore --configuration $(buildConfiguration)`



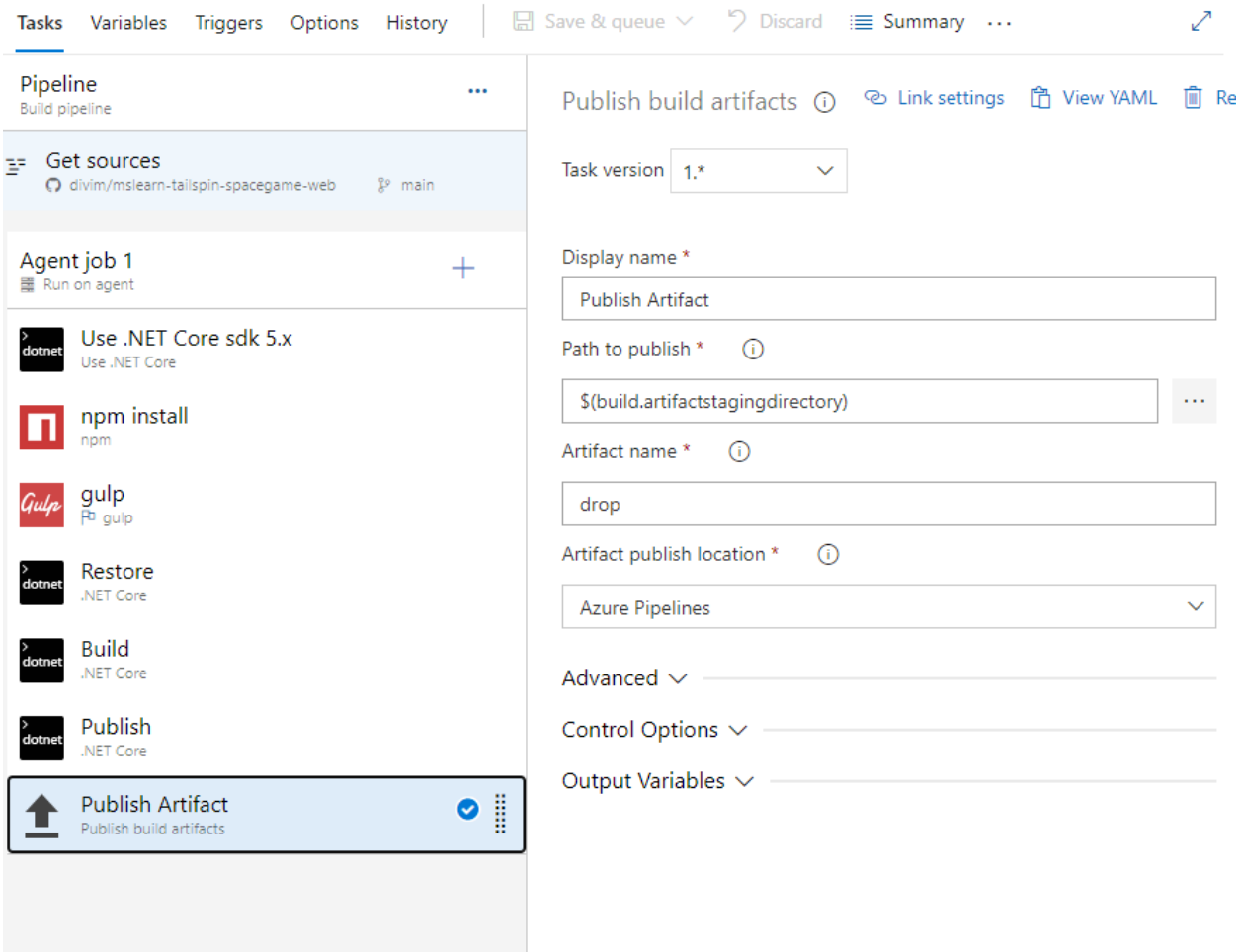
12. Click on the **Test** task and delete it by clicking on "**Remove**". We will be creating unit tests and adding it to the pipeline in the lab after this.



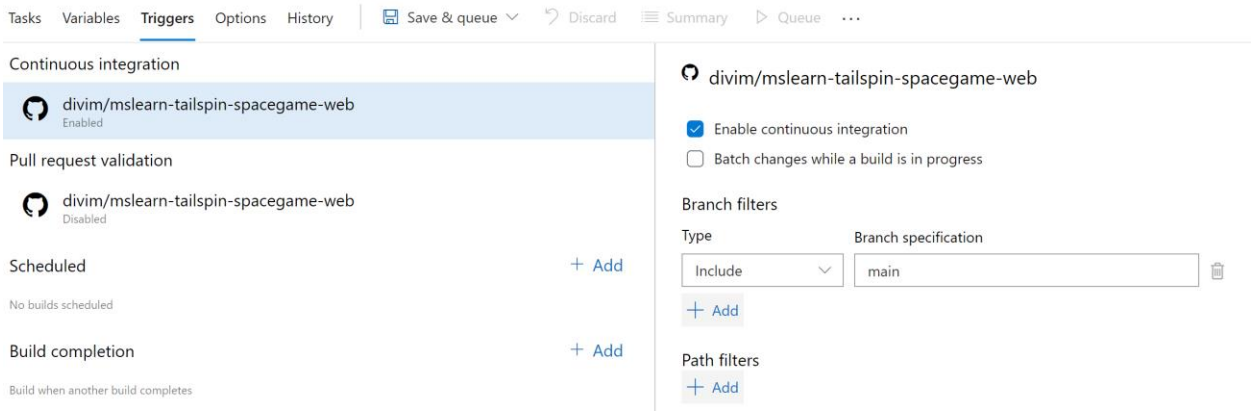
13. Click on the **Publish** task and edit it as follows:
    a. Unselect the "**Publish web projects"** button.
    b. Edit the arguments as follows: `--no-build --configuration $(buildConfiguration) --output $(Build.ArtifactStagingDirectory)/$(buildConfiguration)`



14. Click on the **Publish Artifact** task. Under **Control Options** > **Run this task**, click on the option "**Only when all previous tasks have succeeded".**

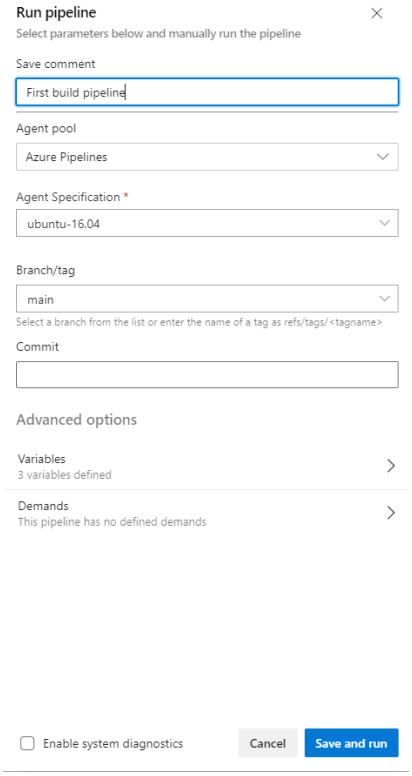


15. Under the **Triggers** section, **enable continuous integration**

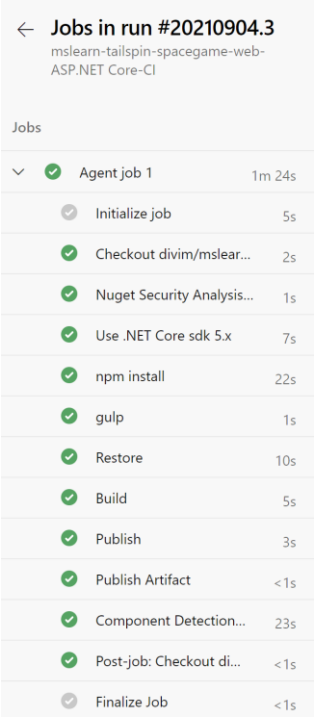


16. Under the "Save & Queue", click on **Save & Queue.**

17. Add a relevant comment and click on **Save & Run.**



18. Observe the tasks running:



You have successfully created your build pipeline.

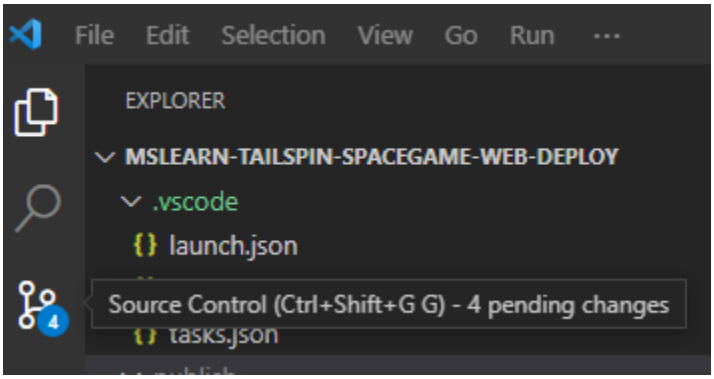
Source Control with GitHub

We just ran the pipeline manually by clicking on Save & Queue. In a real life scenario, we would like this pipeline to run every time we commit changes to our source code on GitHub.

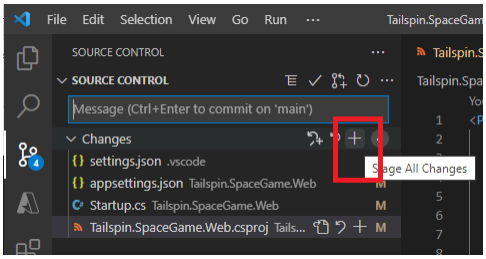
Recall that we had made some changes to the VS Code when we added the Application Insights instrumentation key.

Let's now push it to GitHub.

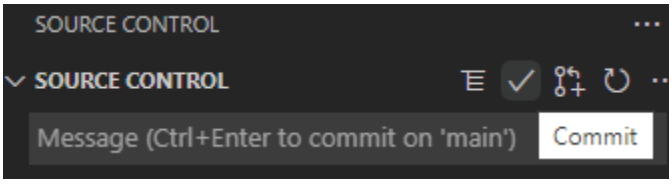
1. Navigate back to your VS Code and notice that your **Source Control** tab highlights pending changes.



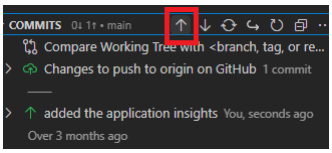
2. Click on the **Source Control** tab
3. Click on **+** for Staging all the changes



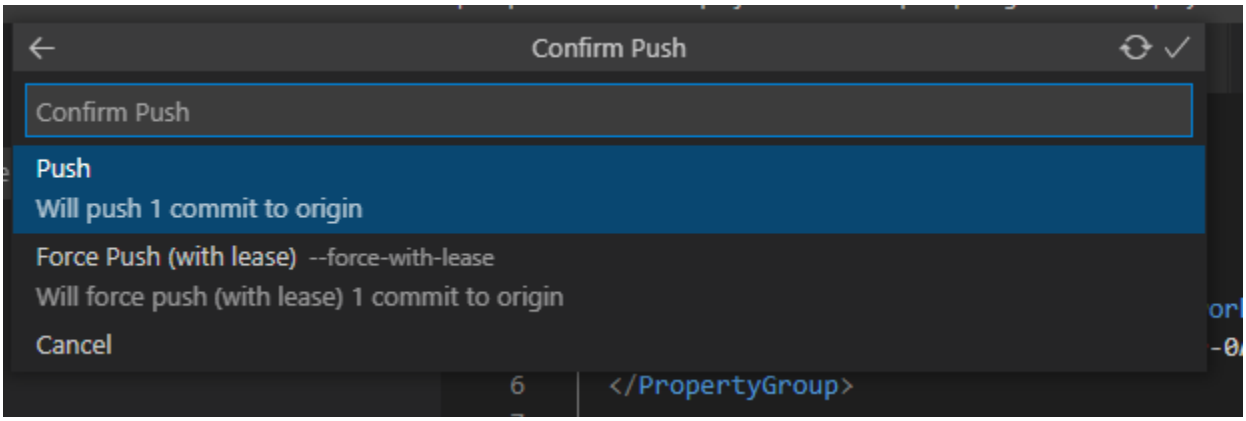
4. Click on the tic mark for Committing the changes



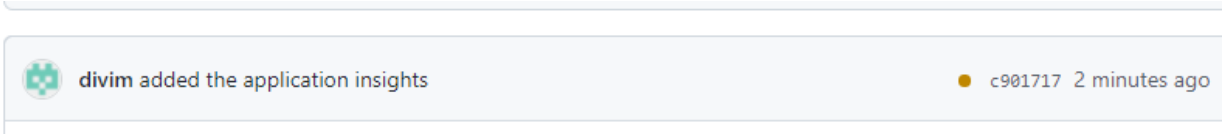
5. Enter a relevant message, such as "added app insights instrumentation key"
6. Under **COMMITS** (at the bottom), click on the UP arrow. This will **Push** all the changes to GitHub.



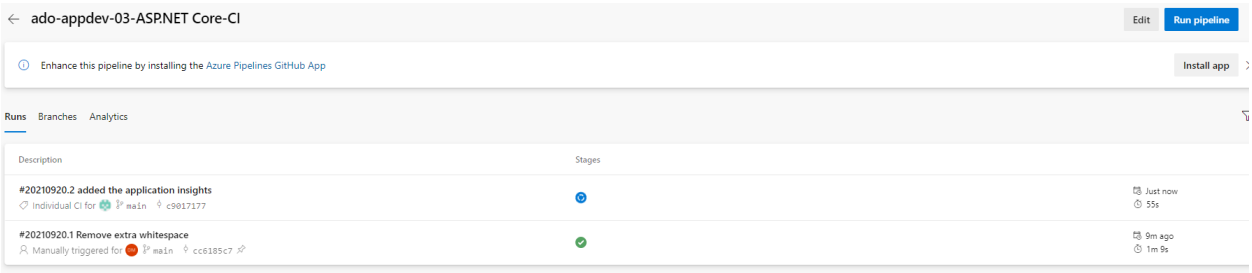
7. Confirm the Push.



8. Navigate to your GitHub repo on github.com. Validate the push.



9. Now that you have pushed new code to GitHub, your build pipeline has been triggered. Navigate back to Azure Pipelines. You will see that your pipeline has been triggered – this time because of a push to the main branch.

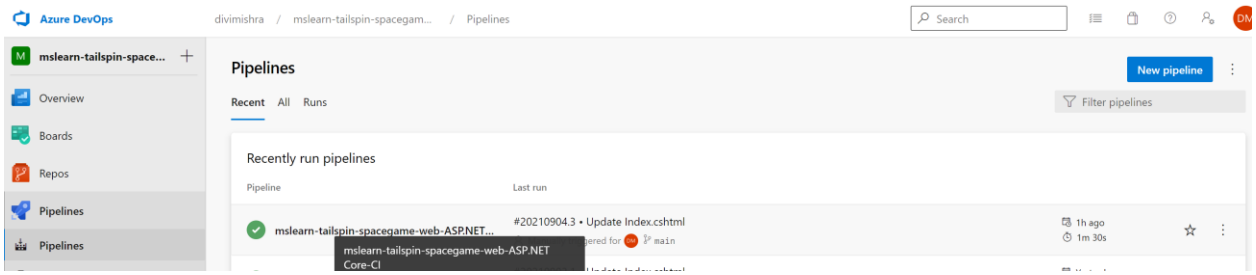


Note: You have now configured a way to reflect changes from your VS Code -> GitHub -> Azure Pipelines. In **Lab 05**, we will complete the pipeline as VS Code -> GitHub -> Azure Pipelines -> Azure App Services.

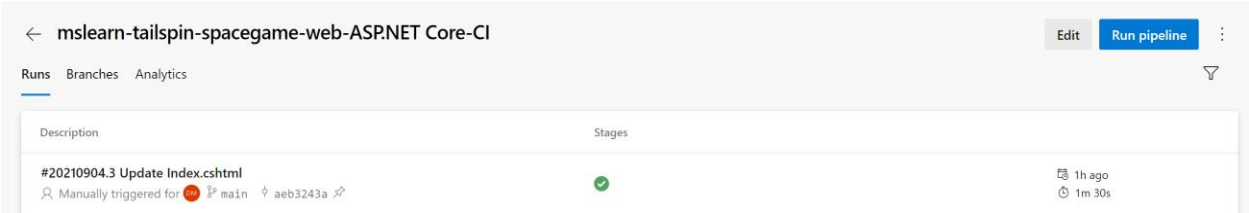
Add unit tests

Adding the test job to the pipeline

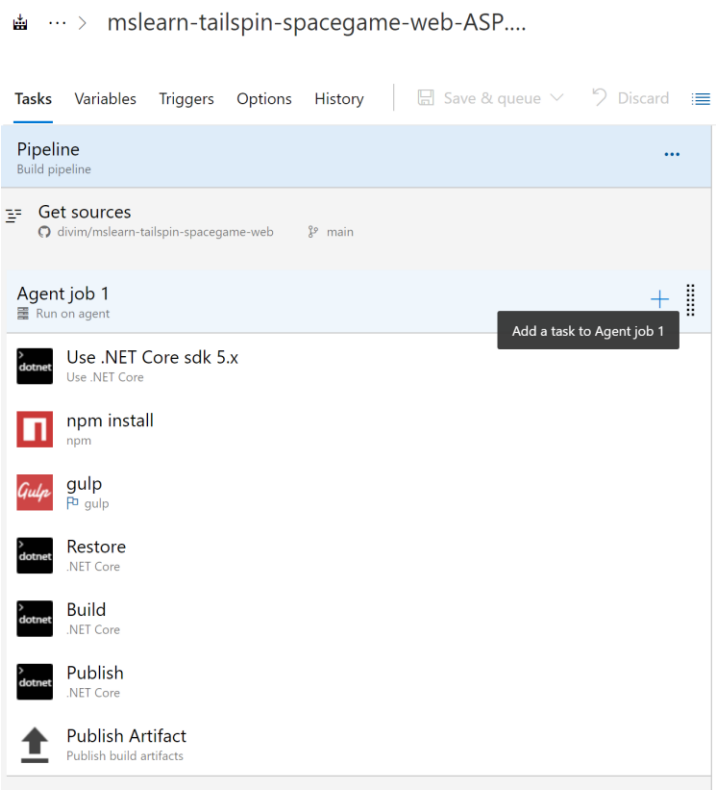
1. Open your Pipeline on dev.azure.com again.



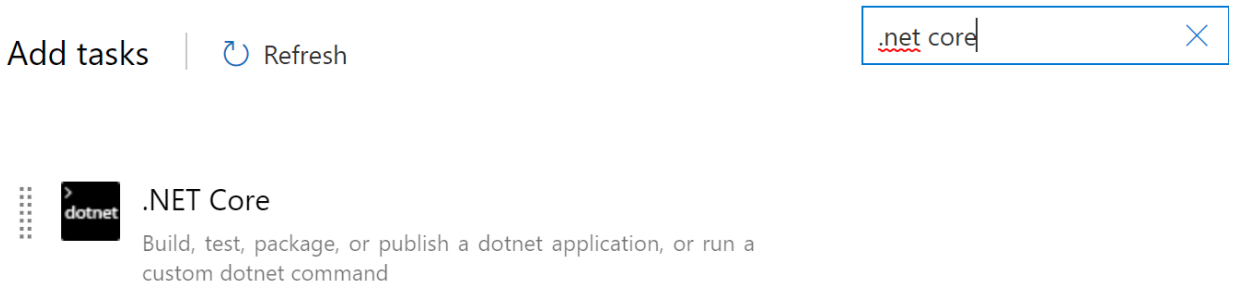
2. Click on **Edit** from the top right to edit the pipeline.



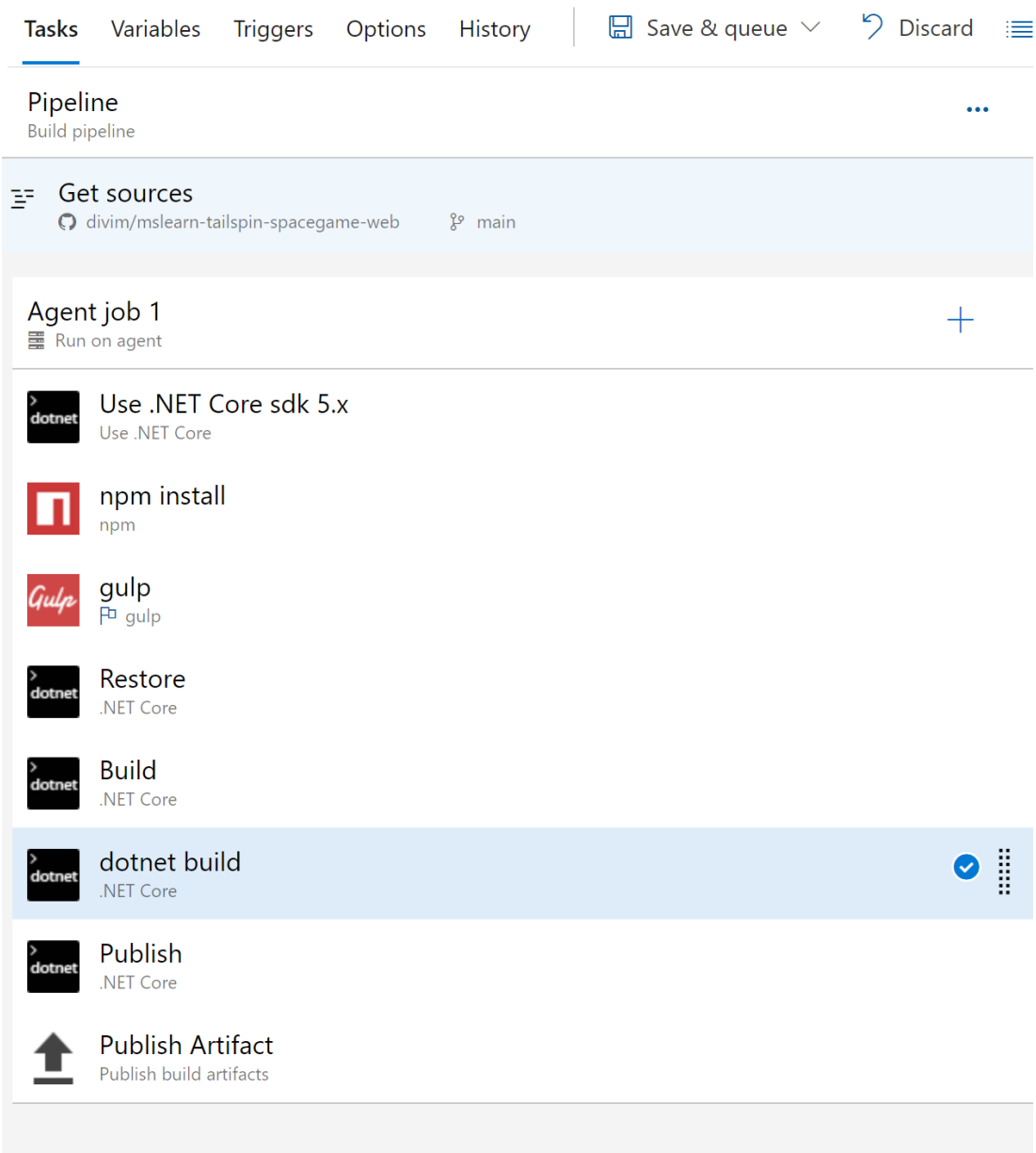
3. Click on "+" to add a task to the job agent.



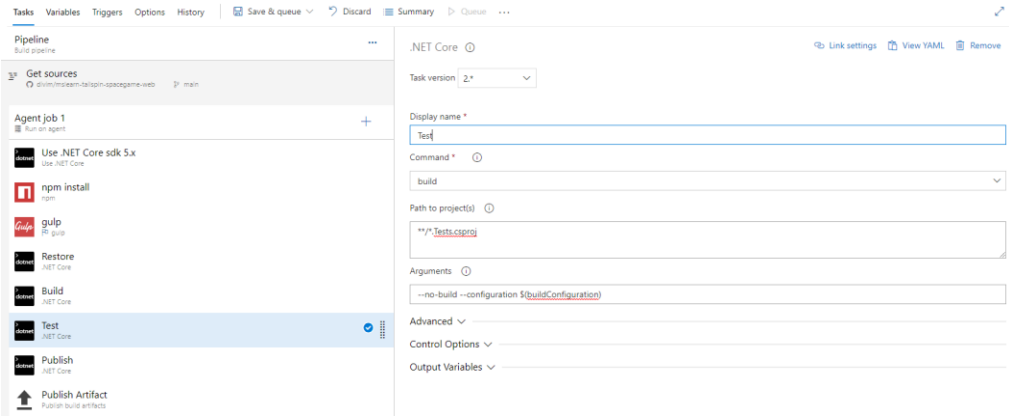
4. Search for "**.NET Core**". Click on "**Add**".



5. Drag the task between "**Build"** and "**Publish**". Your pipeline now looks like this:



6. Click on the task to edit it.
    a. Under Display name, type **Test**
    b. Under Command, select **test**
    c. Under path to project, type `**/*.Tests.csproj`
    d. Under arguments, type `--no-build --configuration $(buildConfiguration)`

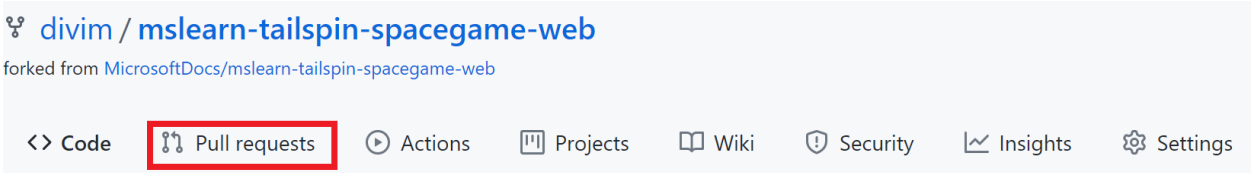


7. Click on **Save** (NOT Save & Queue).

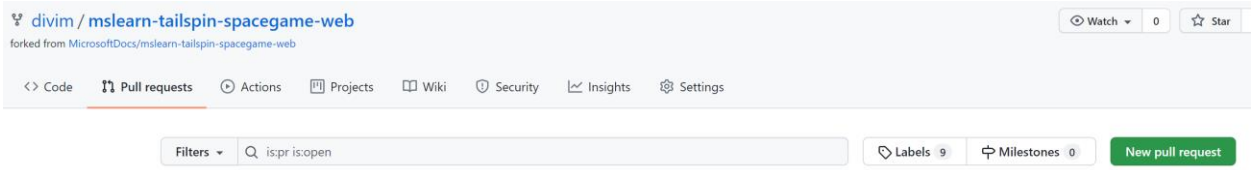
Pushing the tests files to the GitHub repository

The tests are stored in the **unit-tests** branch. Hence, we need to push these tests to the **main** branch.

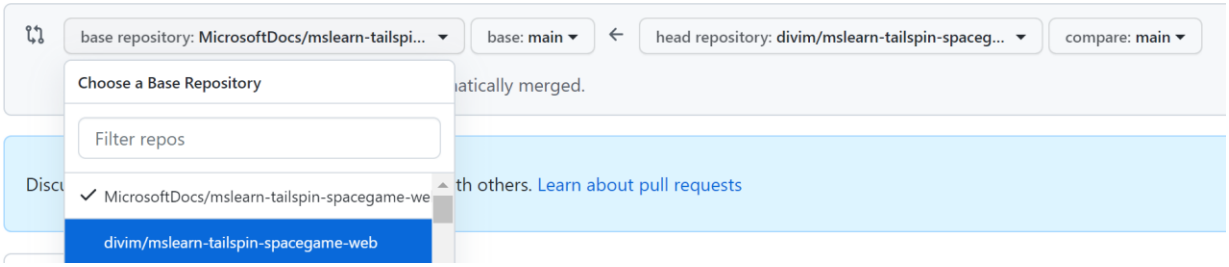
1. Navigate to your GitHub repository. Click on **Pull requests.**



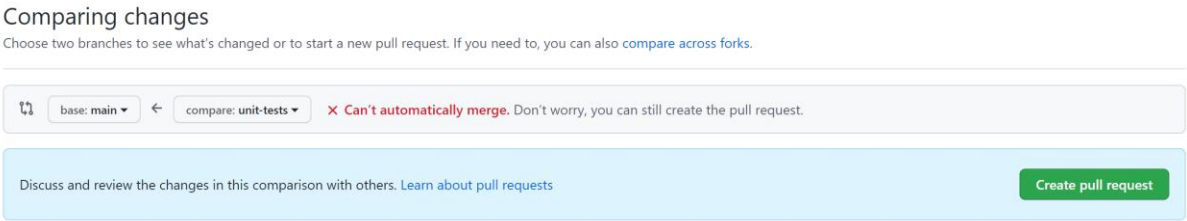
2. Click on **New Pull Request**



3. Change the default base repository. Select your forked repo. (Example: <yourname>/mslearn-tailspin-spacegame-web

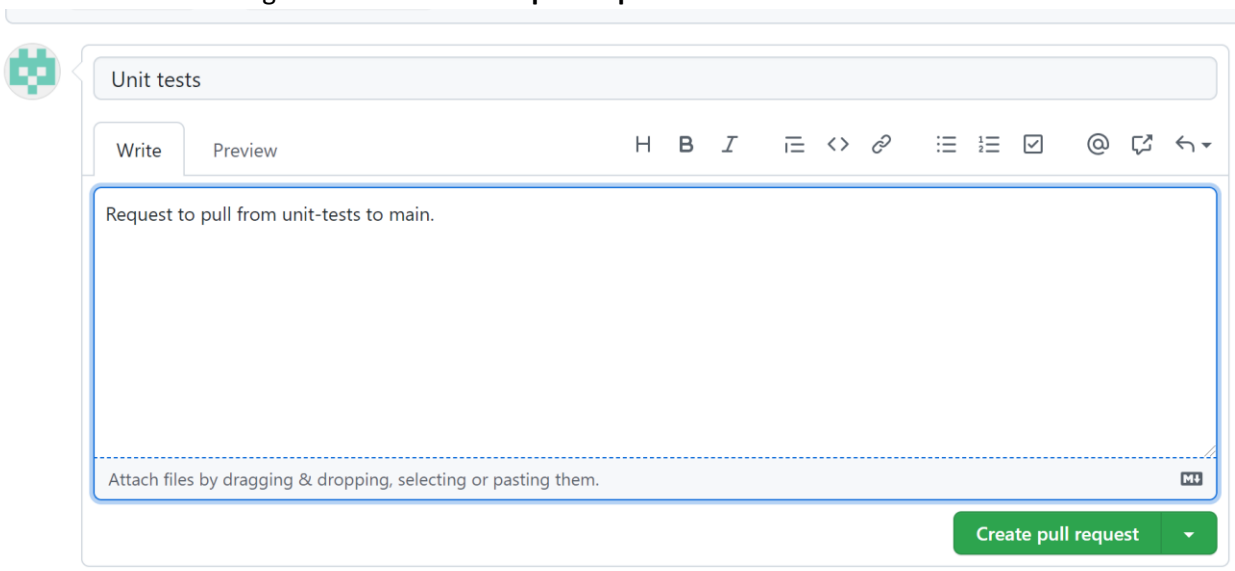


4. Leave the base branch as **main** and change the compare branch to **unit-tests**



(If you get an error "Can't automatically merge.": ignore and proceed creating pull request.)

5. Write a relevant message and click on **Create pull request.**



6. (***OPTIONAL STEP***) If you have conflicts on your pull request, scroll down and click on **Resolve conflicts.**

When you open the file with conflicts in your text editor, you'll see the changes from the HEAD or base branch after the line <<<<<<< HEAD. Next, you'll see =======, which divides your changes from the changes in the other branch, followed by >>>>>>> BRANCH-NAME.

```
11  <<<<<<< unit-tests
12    <ItemGroup>
13      <Content Remove="SampleData\profiles.json" />
14      <Content Remove="SampleData\scores.json" />
15      <Content Remove="SampleData\profiles.json" />
16      <Content Remove="SampleData\scores.json" />
17    </ItemGroup>
18    <ItemGroup>
19      <Content Update="SampleData\profiles.json">
20        <CopyToOutputDirectory>Always</CopyToOutputDirectory>
21      </Content>
22      <Content Update="SampleData\scores.json">
23        <CopyToOutputDirectory>Always</CopyToOutputDirectory>
24      </Content>
25    </ItemGroup>
26    <ItemGroup>
27      <EmbeddedResource Include="SampleData\profiles.json">
28        <CopyToPublishDirectory>PreserveNewest</CopyToPublishDirectory>
29      </EmbeddedResource>
30      <EmbeddedResource Include="SampleData\scores.json">
31        <CopyToPublishDirectory>PreserveNewest</CopyToPublishDirectory>
32      </EmbeddedResource>
33  =======
34
35    <ItemGroup>
36      <PackageReference Include="Microsoft.ApplicationInsights.AspNetCore" Version="2.18.0" />
37  >>>>>>> main
```

Edit the file and choose the part of the file you'd like to keep. Then, click on **Mark as resolved.**

Tailspin.SpaceGame.Web/Tailspin.SpaceGame.Web.csproj 1 conflict Prev Next Mark as resolved

```
1   <Project Sdk="Microsoft.NET.Sdk.Web">
2
3     <PropertyGroup>
4       <TargetFramework>net5.0</TargetFramework>
5       <ProjectGuid>{A0C4E31E-AC75-4F39-9F59-0AA19D9B8F46}</ProjectGuid>
6     </PropertyGroup>
7
8     <ItemGroup>
9       <Folder Include="wwwroot\images\avatars\" />
10    </ItemGroup>
11
12    <ItemGroup>
13      <Content Remove="SampleData\profiles.json" />
14      <Content Remove="SampleData\scores.json" />
15      <Content Remove="SampleData\profiles.json" />
16      <Content Remove="SampleData\scores.json" />
17    </ItemGroup>
18    <ItemGroup>
19      <Content Update="SampleData\profiles.json">
20        <CopyToOutputDirectory>Always</CopyToOutputDirectory>
21      </Content>
22      <Content Update="SampleData\scores.json">
23        <CopyToOutputDirectory>Always</CopyToOutputDirectory>
24      </Content>
25    </ItemGroup>
26    <ItemGroup>
27      <EmbeddedResource Include="SampleData\profiles.json">
28        <CopyToPublishDirectory>PreserveNewest</CopyToPublishDirectory>
29      </EmbeddedResource>
30      <EmbeddedResource Include="SampleData\scores.json">
31        <CopyToPublishDirectory>PreserveNewest</CopyToPublishDirectory>
32      </EmbeddedResource>
33    <ItemGroup>
34      <PackageReference Include="Microsoft.ApplicationInsights.AspNetCore" Version="2.18.0" />
35    </ItemGroup>
36  </Project>
```

Then, click on **Commit merge.**

Unit tests #7

Resolving conflicts between `unit-tests` and `main` and committing changes → `unit-tests` Commit merge

2 conflicting files

Tailspin.SpaceGame.Web.csproj
...ame.Web/Tailspin.SpaceGame.Web.csproj

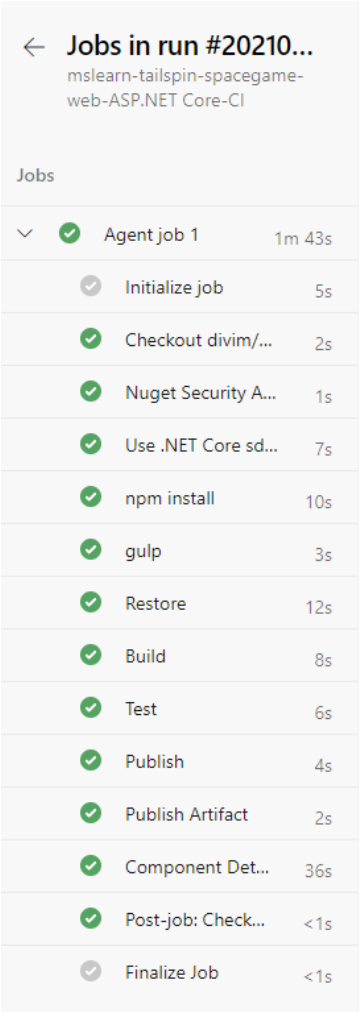
Index.cshtml
...SpaceGame.Web/Views/Home/Index.cshtml

Tailspin.SpaceGame.Web/Views/Home/Index.cshtml ✓ Resolved

```
1   -@model TailSpin.SpaceGame.Web.Models.LeaderboardViewModel
2   @{
3       ViewData["Title"] = "Home Page";
4   }
5   <section class="intro">
6       <div class="container">
7           <img class="title" src="/images/space-game-title.svg" alt="Space Game">
8           <p>Welcome to the official Space Game site!</p>
```

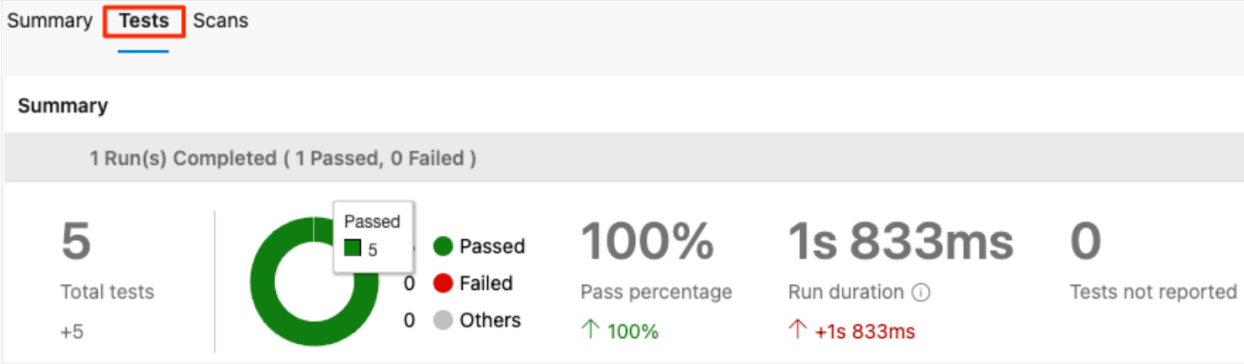
Your test files are now on the main branch.

8. Since you committed the tests to the main branch, the pipeline has been triggered again. This time, it includes the **Test** job.
9. Navigate to the latest pipeline run. Click on the **Agent Job** and observe the tasks run.



Your pipeline now contains **unit testing**.

10. Navigate back to the **pipeline summary**.
11. Move to the **Tests** tab to view a summary of the test run.

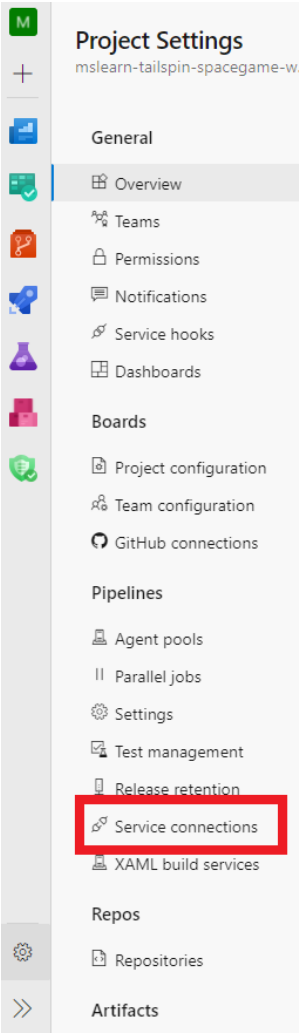


Note: You must ideally add functional and non-functional test to your build pipeline. Read more about such tests: [Run Functional Tests task - Azure Pipelines | Microsoft Docs]

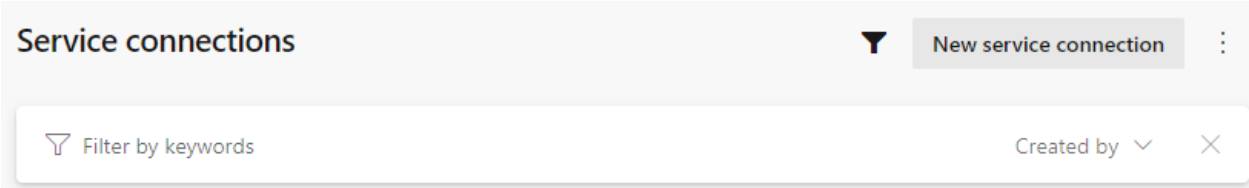
Lab 05: Zero-downtime app deployment with release pipeline

Deploy to staging slot from build pipeline

1. Go back to your build pipeline in dev.azure.com.
5. At the bottom left, click on **Project Settings.** Under Pipelines, navigate to Service connections.

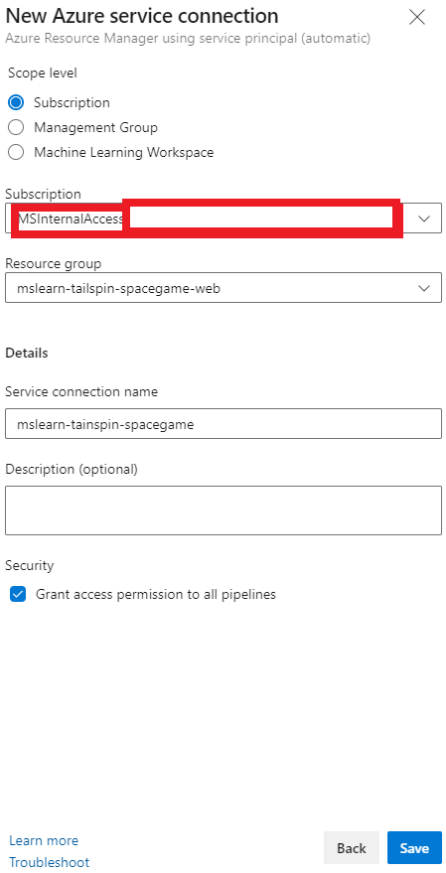


6. Click on **New Service Connection**



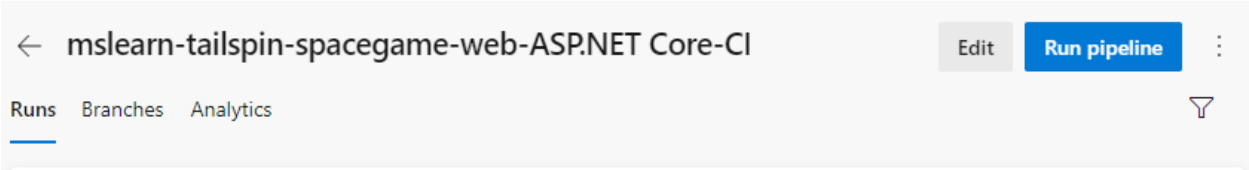
7. Click on **Azure Resource Manager.** Select Next.
8. Click on **Service Principal (Automatic).** Select Next.
9. Define your Subsciption and Resource group that contains your app service.
10. Give a relevant service connection name.

11. Select **Grant access permission to all pipelines.**



12. Select **Save.**

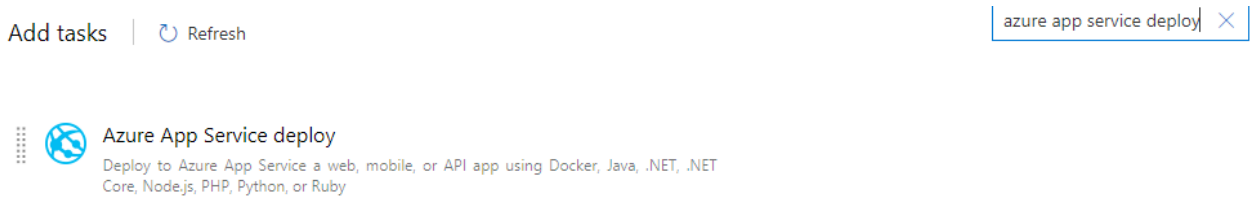
13. Navigate back to your pipeline and select **Edit.**



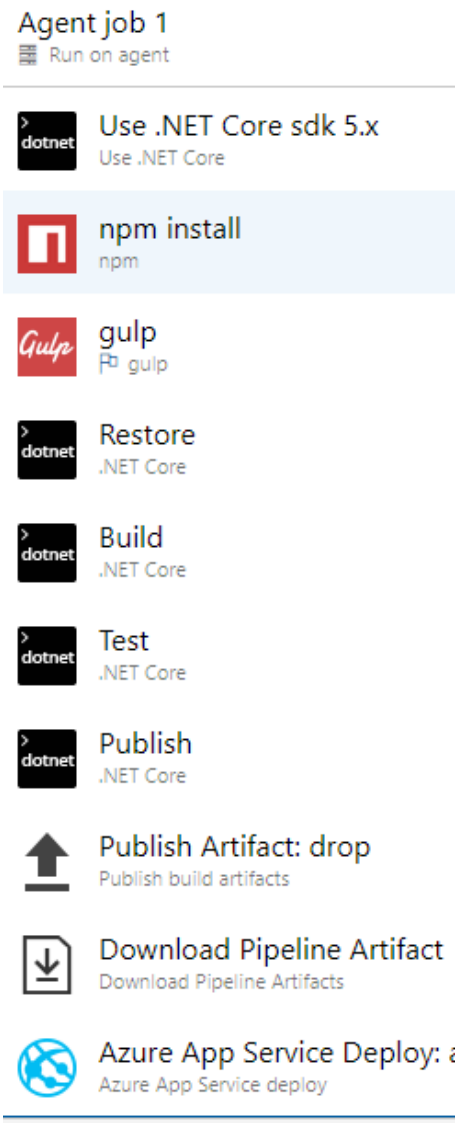
14. Add the following agent jobs:

    a. Download pipeline artifact

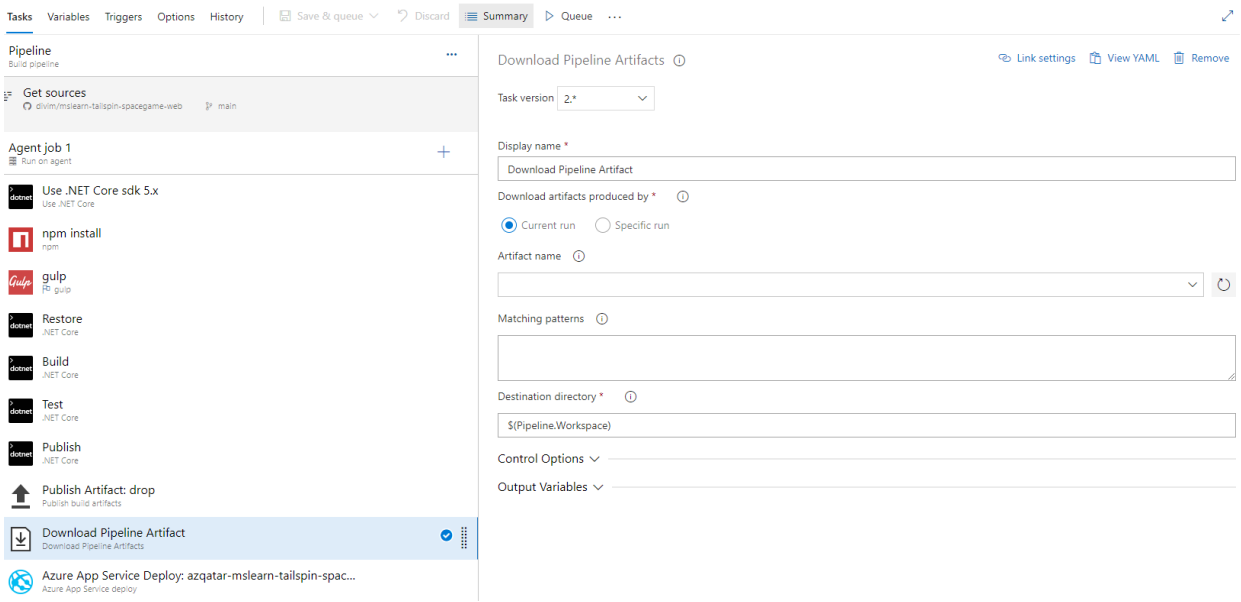
    b. Azure App Service Deploy



15. Adjust the tasks for it to be in the following order:

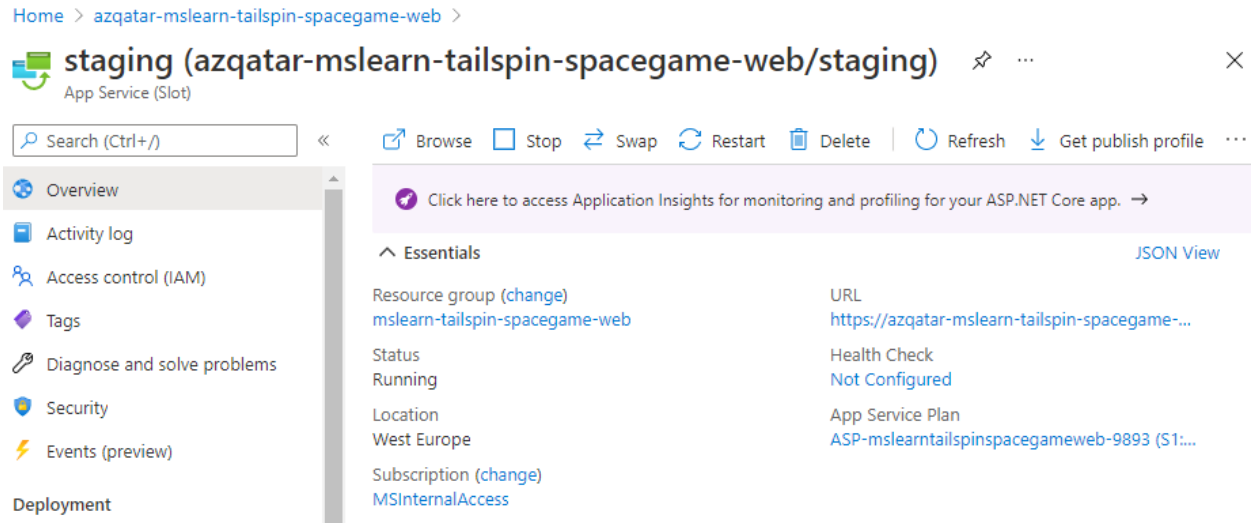


16. Click on the **Download Pipeline Artifact** task to edit the task as follows:
    a. Destination directory: $(Pipeline.Workspace)
    b. Leave the Artifact name and matching patterns empty

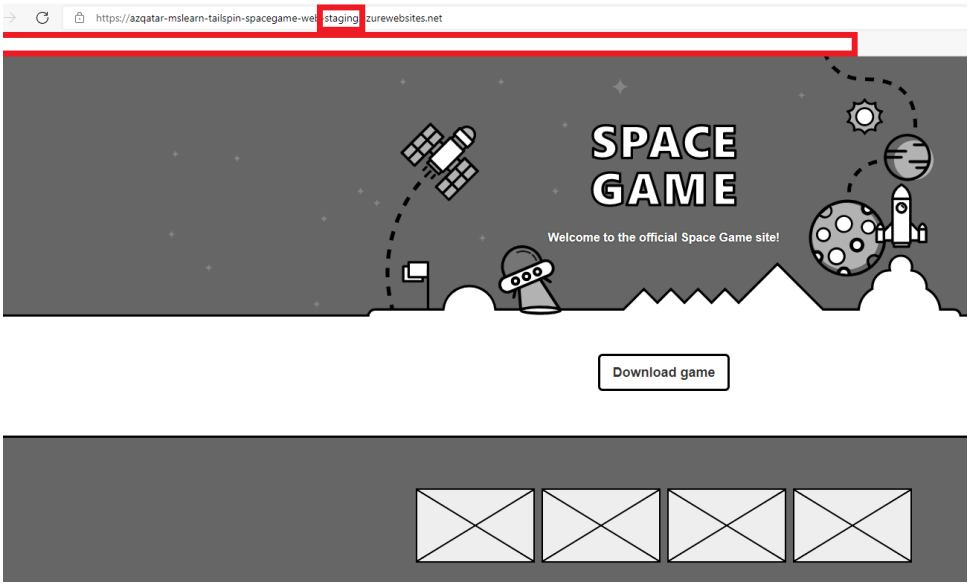


17. Click on **Azure App Service Deploy** to edit it as follows:
    a. Connection type: Azure Resource Manager
    b. Susbcription: <name-of-your-service-connection>
    c. App Service type: *as defined during your app service definition above*
    d. App Service name: *as defined during your app service definition above*
    e. Select **Deploy to Slot or App Service Environment**
    f. Resource group: RG of your app service
    g. Slot: staging
    h. Package or folder**: $(Pipeline.Workspace)/drop/Release/Tailspin.SpaceGame.Web.zip**
18. Select **Save & Queue.**
19. Select **Save & Run.**
20. Once the pipeline is done running, go back to Azure Portal > App Service > your app service > Deployment Slots
21. Click on the **staging slot**

22. Click on **Browse**



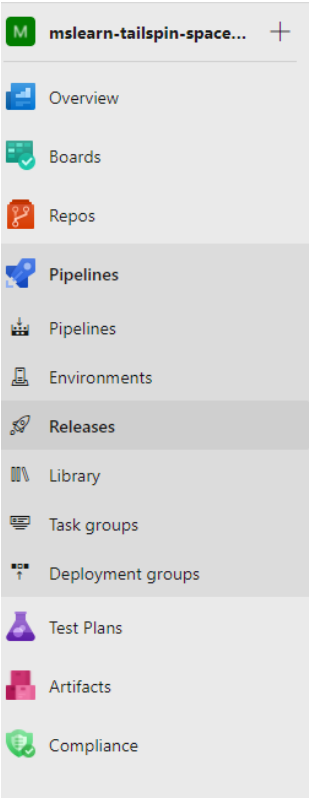
23. Your app has been deployed successfully to the staging slot. Note that the name of the URL is simply <app-service-name>-staging.azurewebsites.net



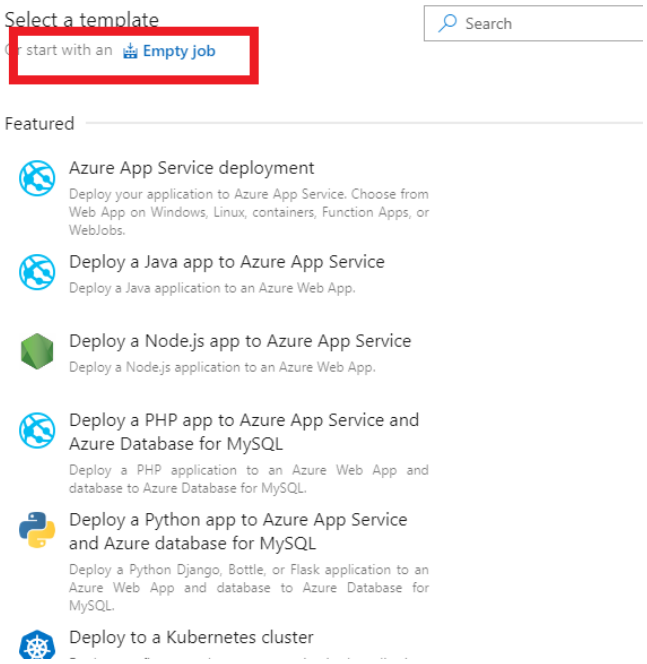
24. Simply remove the “-staging” from the URL to view the production slot website.

Build a release pipeline: zero-downtime deployment with slot swapping

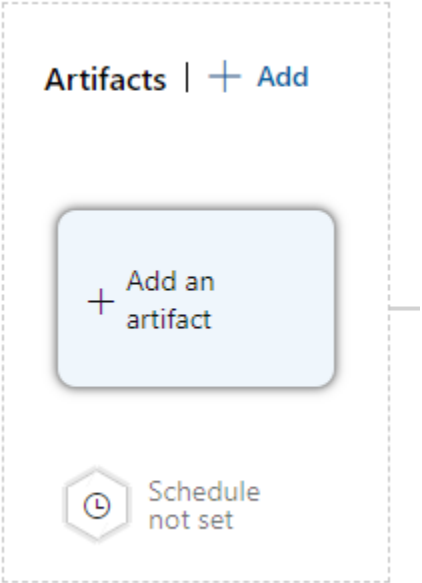
1. Navigate to your Azure DevOps project > **Pipelines > Releases**



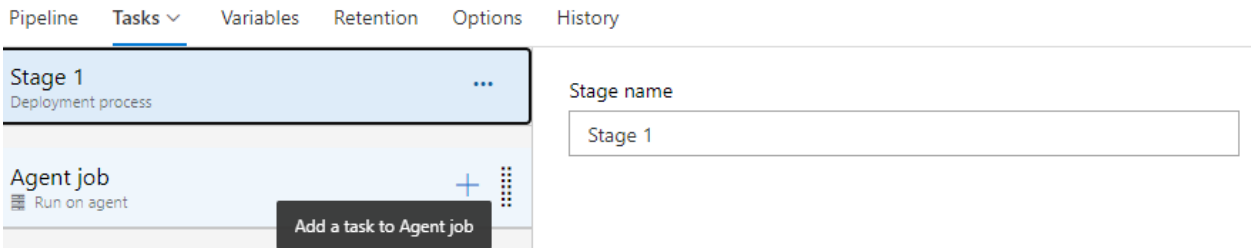
2. Select “New pipeline”.
3. For the template, **start with an empty job**



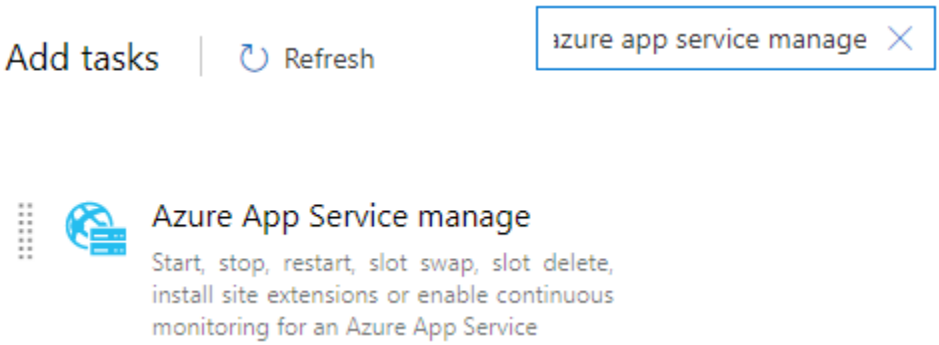
4. Click on **Add an artifact**



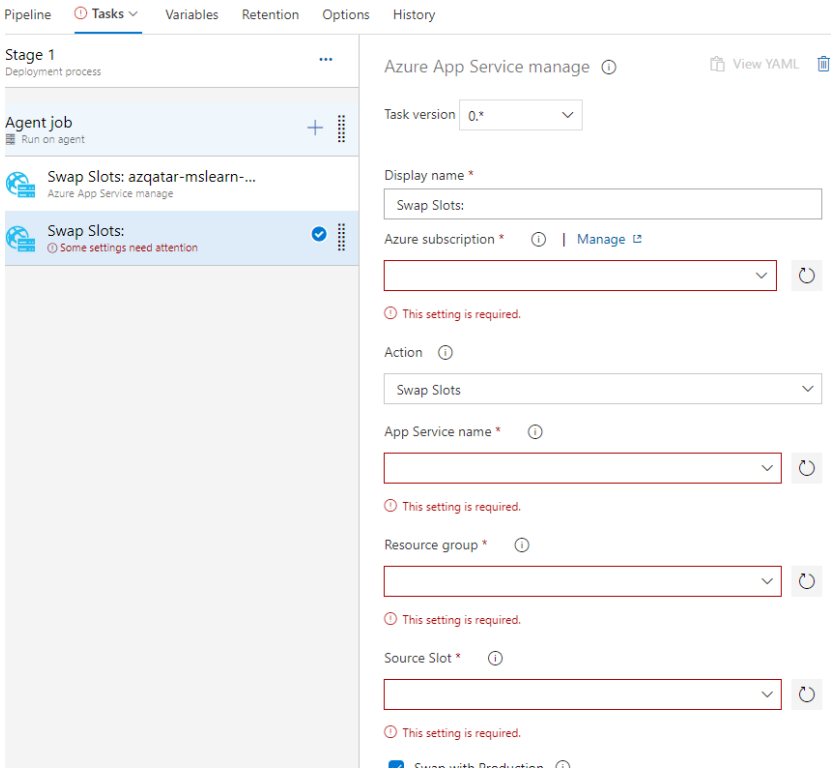
5. Select your source type as **Build.**
6. Select your Source build pipeline from the previous lab.
7. Leave the default values for the default version and source alias.
8. Select **Add.**
9. Select the **Stage 1.**
10. Add a task to agent job by clicking on the "+".



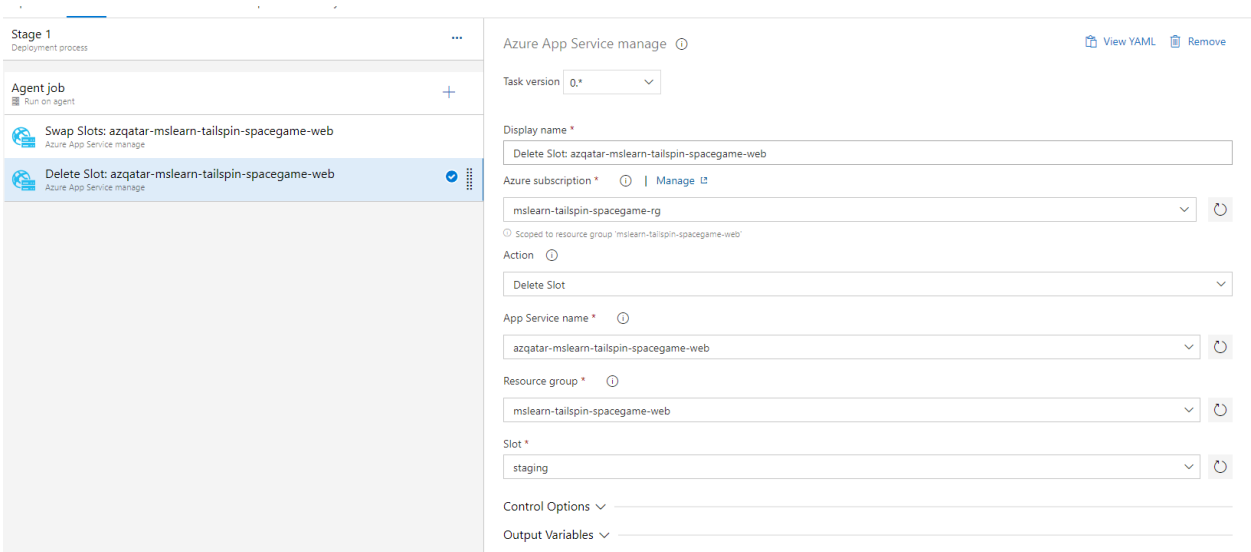
11. Search for **Azure App Service manage** and click **Add.**



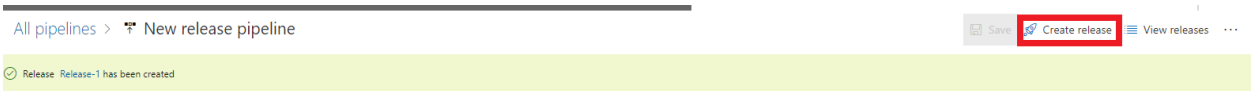
12. Click on the task to edit the task as follows:
    a. Select your service connection under Azure Subsciption.
    b. Select the action as **Swap slots.**
    c. Select your app service for the app service name and its associated resource group.
    d. Select **staging** as your source slot.
    e. Select **Swap with production.**
13. Add another task for **Azure App Service manage.**



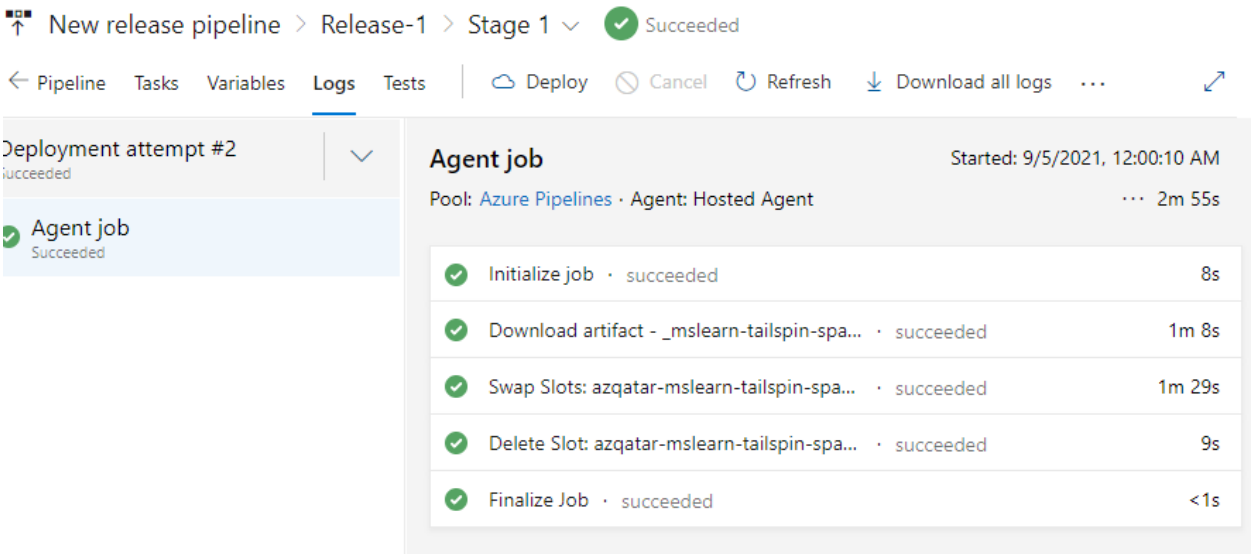
14. Select the task to edit it as follows:
    a. Select your service connection for the Azure subscription
    b. Select your action as **Delete Slot**
    c. Select your app service name and resource group
    d. Select the slot to be deleted as **staging**



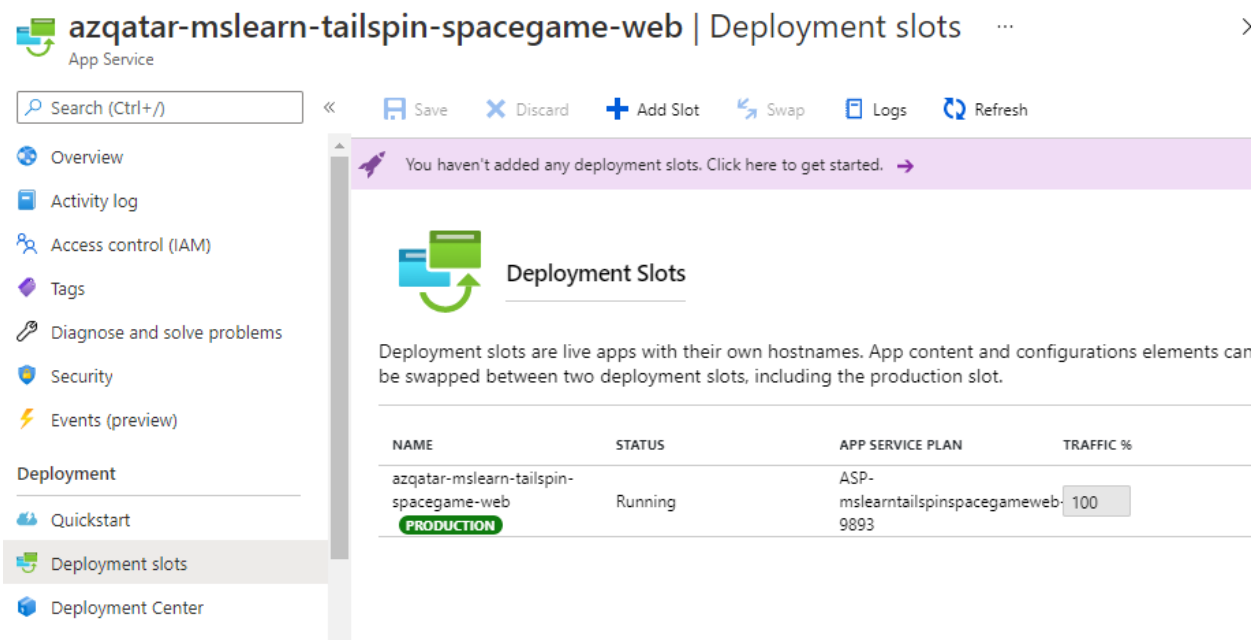
15. Select **Save.**
16. Select **Create release.**



17. Observe the tasks on the release pipeline.



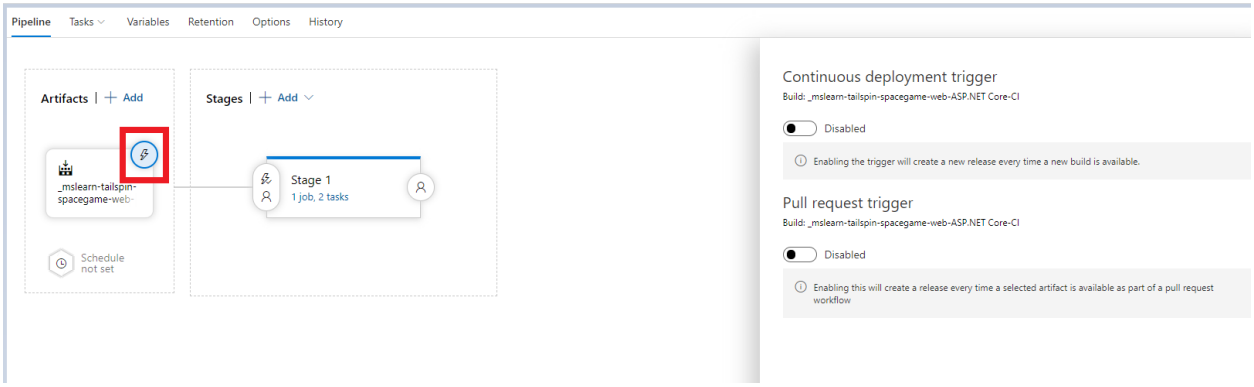
18. Once the pipeline has been completed, navigate back to Azure Portal.
19. Under deployment slots, you will notice that the slot is now deleted.



Now, when you push a commit to GitHub's main branch, your build pipeline will be triggered and deploy the app to your staging slot. If successful, your release pipeline will triggered to swap the staging and production slot. Finally, the pipeline will delete the staging slot to stop incurring any charges.

Notes:

You can choose to enable or disable continuous deployment for your release pipeline. This will allow you to choose whether you want the release to happen automatically or with a manual check.



Optional labs

Please note that these optional labs will take more time than the other labs. These labs may also require a higher SKU for the app service plan used.

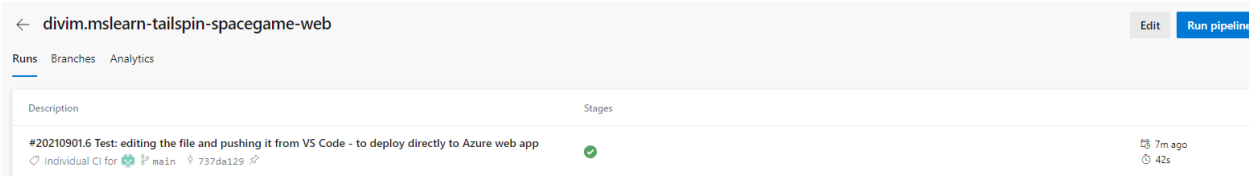
Optional Lab 01: Enabling continuous integration with YAML files

Create a build pipeline without classic editor

1. Go back to your Azure DevOps project created in Lab 01.
2. Navigate to **Pipelines** and create a pipeline.
3. Under Connect, click on **GitHub.**
4. Under Select, browse to your forked repository.
5. Under Configure, select "**ASP.NET Core**". (Please don't select ASP.NET Core (.NET Framework))
6. Select "Save and Run". Set a commit message. Select "Save and run" again.
7. Observe as all the tasks run. Your first pipeline has been created successfully.

VS Code to Azure App Services: enabling Continuous Integration

1. Open the Azure App Service resource from the Azure portal.
2. Under Deployment, go to the Deployment Center.
3. Select your source as GitHub.
4. Authorize AzureAppService to access your GitHub and select your enterprise, repository and branch.
5. Go to VS Code, make a change to the code (say index.cshtml). Then, stage, commit and push the change alongside a relevant commit message.
6. Navigate to your app in the VS Code Azure App Services Extension. Right-click and select "Browse website". You will see the changes reflected.
7. You can now edit files in VS Code. Once you commit your changes to the main branch, the build pipeline will get triggered and your changes will be reflected on successful build.



You now have a basic CI/CD pipeline from VS Code, to GitHub, and finally to Azure App Service.

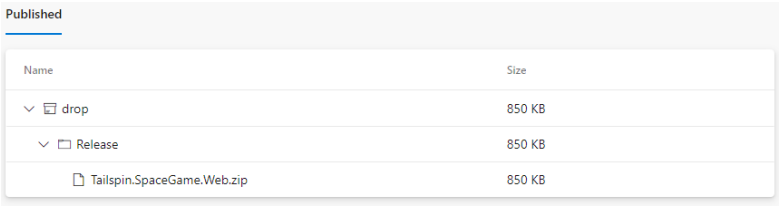
Note: As an alternative to Azure Repos, you can use GitHub Actions. Please follow this documentation on steps to configure it: [Configure CI/CD with GitHub Actions - Azure App Service | Microsoft Docs]

8. Modify **azure-pipelines.yml** file as follows to publish an artifact as a part of your build pipeline:

```
trigger:
- '*'

pool:
  vmImage: 'ubuntu-20.04'
  demands:
  - npm

variables:
  buildConfiguration: 'Release'
  wwwrootDir: 'Tailspin.SpaceGame.Web/wwwroot'
  dotnetSdkVersion: '5.x'

steps:
- task: UseDotNet@2
  displayName: 'Use .NET 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()
```

9. Push the change as a commit to the main branch. This will trigger the Azure Pipeline.
10. Go to Azure Pipelines and observe the build through the steps. You'll see that you now have a published artifact.
11. Select the artifact and expand the drop folders. You'll see a .zip file that contains your build app and its dependencies.



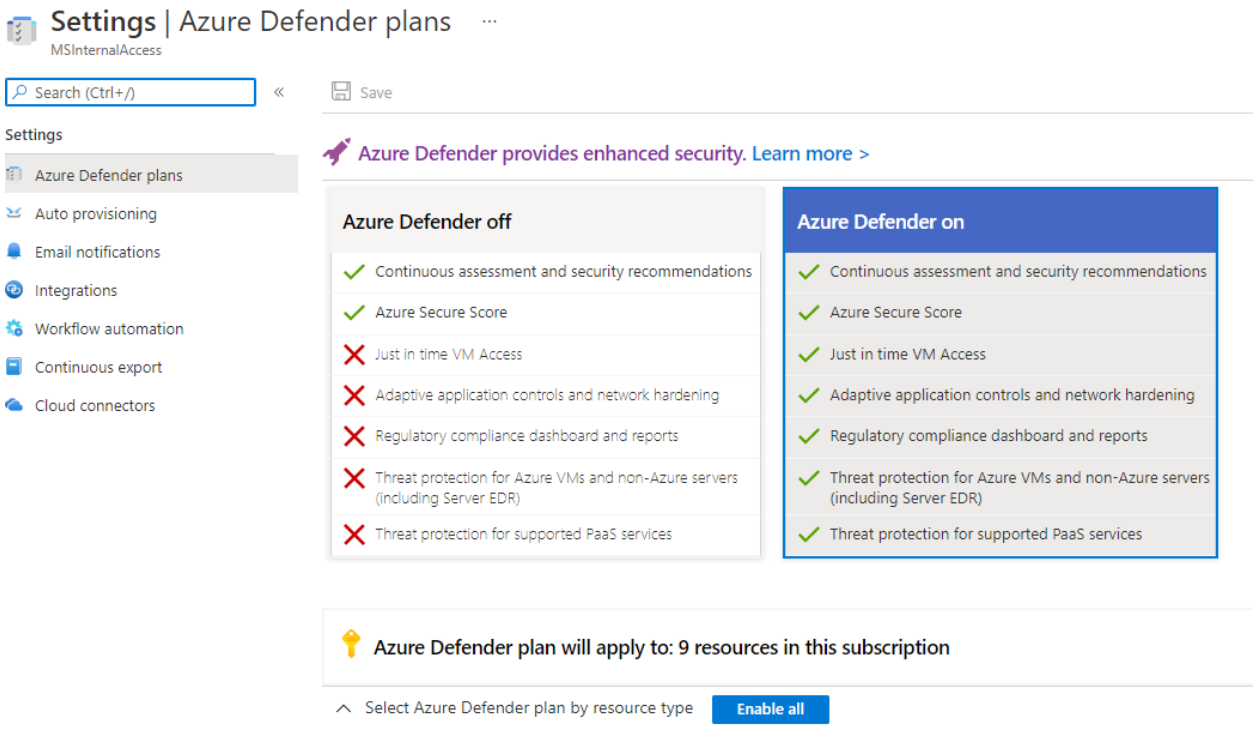
Note: You can build multiple configurations within the pipeline by defining a **template.** Read more: [Exercise - Build multiple configurations by using templates - Learn | Microsoft Docs]

Optional Lab 02: Protect App Services with other WAF Options

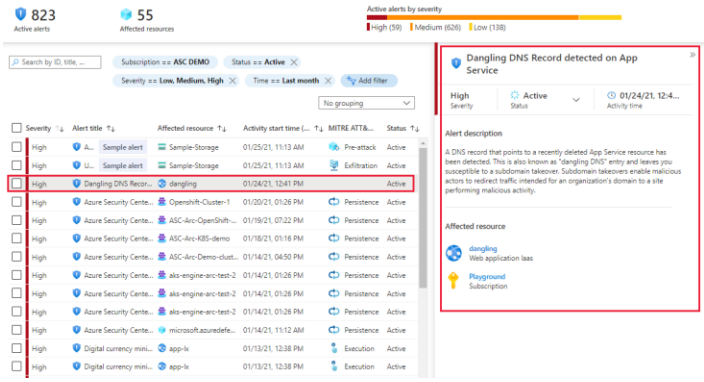
Alternatively, you can set up other WAF options from the [Azure Marketplace]. Here is a documentation on how you can integrate [a Barracuda WAF for Azure]: [Configure a WAF - Azure App Service Environment | Microsoft Docs]

Optional Lab 03: Azure Defender for App Services

1. Navigate to the Security Center from the left menu on the Azure Portal.
2. Select “Pricing & Settings” under Management.
3. Select the subscription with your application.
4. Enable Defender for all resources (strongly recommended).



5. Click on “Save” to enable Azure Defender.
6. Your app service is now under Azure Defender.



Azure Defender detects a multitude of threats to your App Service resources by monitoring:

a. the VM instance in which your App Service is running, and its management interface

b. the requests and responses sent to and from your App Service apps
c. the underlying sandboxes and VMs
d. App Service internal logs - available thanks to the visibility that Azure has as a cloud provider

You can detect DNS dangling and threats by MITRE ATT&CK tactics.

Optional Lab 04: Managing technical debt with SonarQube and Azure DevOps

Add SonarQube to your build pipeline to:

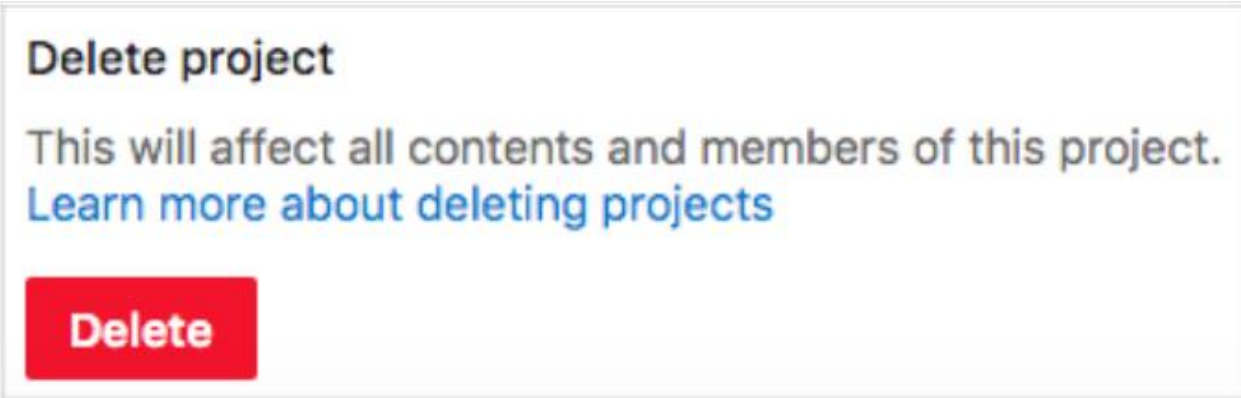
- Detect bugs
- Code smells
- Security vulnerabilities
- Centralize qualities

To analyze SonarQube projects, you must provision up your SonarQube server and set up your project.

Step-by-step lab for adding SonarQube to your build pipeline for increased testing: [Managing technical debt with SonarQube and Azure DevOps | Azure DevOps Hands-on-Labs (azuredevopslabs.com)]

Clean-up your environment

1. Delete the Azure DevOps project, including what's on Azure Boards and Azure Pipelines.
2. Navigate to dev.azure.com.
3. Navigate to your project.
4. Select Project Settings in the lower corner.
5. In the project details area, select **Delete.**



6. Enter the project name and confirm deletion.
7. Navigate to Azure Portal and delete the app service or the resource group.

Your project is now deleted.

Congratulations! You just reached the end of workshop labs.

Resources

- Intro to GitHub Lab: Introduction to GitHub | GitHub Learning Lab
- Creating pull requests, reviewing, creating issues, and other features on VS Code: Working with GitHub in Visual Studio Code.
- Importing your project to GitHub: How do I migrate an existing project to GitHub? - Learn | Microsoft Docs
- For enabling live telemetry through instrumentation key on other code frameworks: What is Azure Application Insights? - Azure Monitor | Microsoft Docs
- For enabling continuous monitoring through Azure DevOps pipeline directly: Continuous monitoring of your DevOps release pipeline with Azure Pipelines and Azure Application Insights - Azure Monitor | Microsoft Docs
- Learn more about continuous monitoring: Continuously monitor applications and services - Learn | Microsoft Docs
- Best practices for Autoscale: Best practices for autoscale - Azure Monitor | Microsoft Docs
- Confused about whether or not to choose Azure App Service? Choose your candidate with this document: Choosing an Azure compute service - Azure Architecture Center | Microsoft Docs
- Choose the right App Service plan: App Service plans - Azure App Service | Microsoft Docs
- Security recommendations for App Service: Security recommendations - Azure App Service | Microsoft Docs
- Tutorial on how to deploy ASP.NET Core with Azure SQL Database app in Azure App Service: Tutorial: ASP.NET Core with Azure SQL Database - Azure App Service | Microsoft Docs
- Tutorials to use your App Gateway to:
    1. Secure by SSL: Tutorial: Configure TLS termination in portal - Azure Application Gateway | Microsoft Docs
    2. Host multiple sites: Tutorial: Hosts multiple web sites using the Azure portal - Azure Application Gateway | Microsoft Docs
    3. Route by URL: Tutorial: URL path-based routing rules using portal - Azure Application Gateway | Microsoft Docs
    4. Redirect web traffic: Tutorial: URL path-based redirection using CLI - Azure Application Gateway | Microsoft Docs