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.

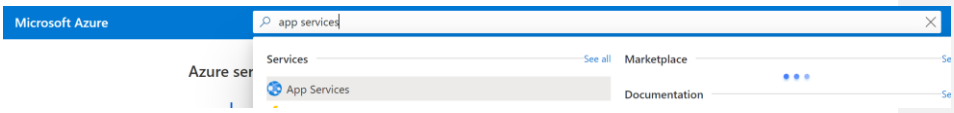
Pre-requisites

1. Visual Studio Code application
2. A GitHub account
3. An Azure DevOps setup
4. An active Azure subscription
5. .NET 5.0 SDK
6. Git

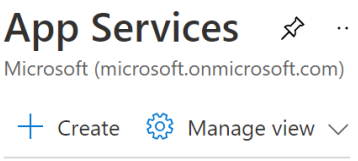
Lab 101: Azure App Service

Create an Azure App Service

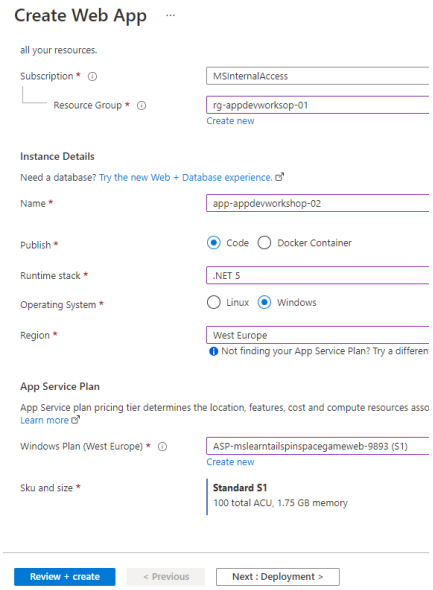
1. Go to the [Azure Portal]
2. Search for **App Services** at the search bar on top



3. Click on **+ Create**



4. Under **Basics** tab, enter the following details:
    a. Select the relevant resource group 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. Go through the next tabs to understand settings. You don't have to change any other settings (accept defaults for all other settings).
6. Click on **Review + Create**
7. Click on **Create**

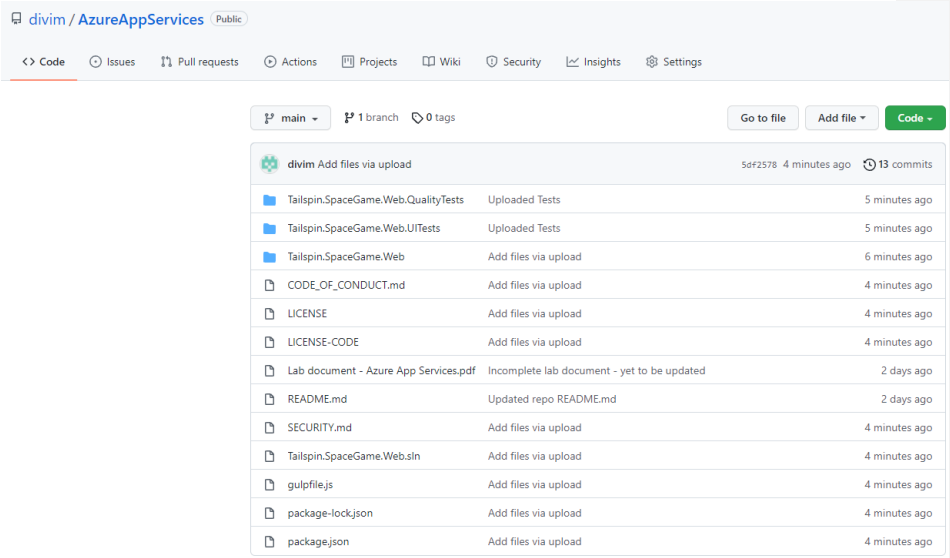


8. Once the deployment is complete, click on “Go to resource”.
9. From the overview tab, click on the “URL” to view the default web application that gets deployed.

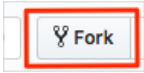
10. You should be able to view a welcome page.

Fork the web project to your GitHub account

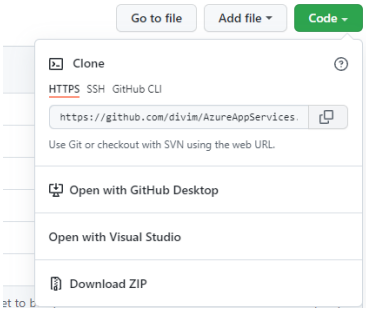
1. Go to GitHub and sign in
2. Visit the Azure App Services workshop repo (ASP.Net Core application)



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

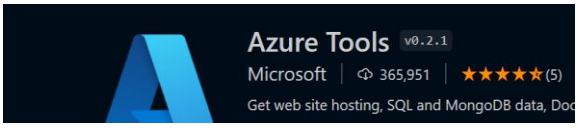


4. Go to your fork of the Space Game project. The forked repository will be saved as *"<your-account-name>/AzureAppServices*".
5. Select "**Code**". Under the "**HTTPS**" tab, copy the URL.

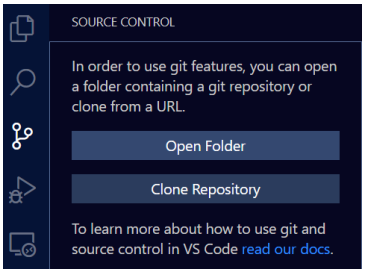


6. Open Visual Studio code.
7. Install the necessary extensions to visual studio code.

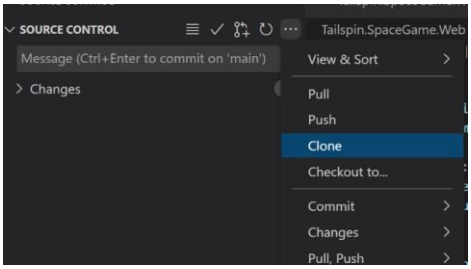
   Click on the **Extensions** button (or **CTRL+Shift+X)** on the left side ribbon, search for "**Azure Tools**" extension (from Microsoft) and install it.



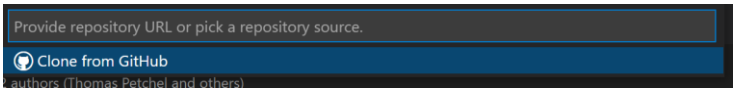
8. Under **Source Control**. Click on **Clone Repository**.



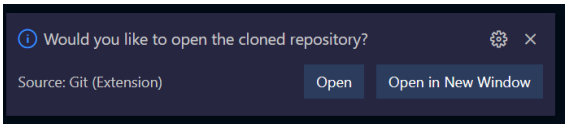
or



9. Click on **Clone from GitHub**



10. Sign-in and choose **AzureAppServices.**
11. Select a folder on your local machine to clone the files.
12. Once the clone is complete, in the visual studio code's prompt for opening the repository, click "**Open**".



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

14. 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 "username"
$ git config --global user.email "email"
```

Commented [MS1]: git config --global user.name "username"

quotation mark should be used instead of left and right quotation mark.. This is a unicode confilct

Commented [MS2]: git config --global user.email "email"

Run and validate the Project Locally.

1. Open the terminal by going to “View > Terminal” or “Ctrl + `” and switch the directory to **Tailspin.SpaceGame.Web** folder using the **cd** command.

$ cd <path to Tailspin.SpaceGame.Web>

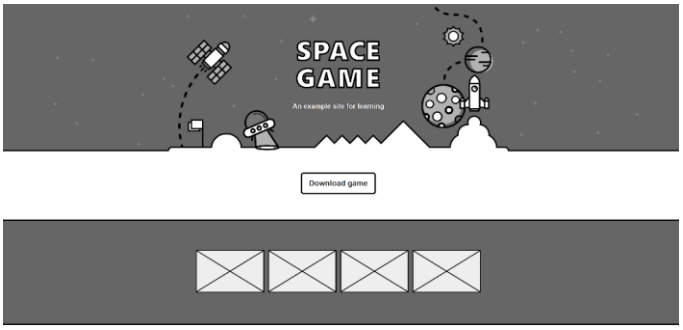
2. Enter the following command on the terminal to make sure the code runs on your local host:

   $ dotnet build -c Release

   $ dotnet run -c Release --no-build --project Tailspin.SpaceGame.Web.csproj

   Navigate to your http://localhost:5000

```
PS C:\Users\divimishra\mslearn-tailspin-spacegame-web-deploy> dotnet run --configuration Release --no-build --project .\Tailspin.SpaceGame.Web\
Hosting environment: Production
Content root path: C:\Users\divimishra\mslearn-tailspin-spacegame-web-deploy\Tailspin.SpaceGame.Web
Now listening on: http://localhost:5000
Now listening on: https://localhost:5001
Application started. Press Ctrl+C to shut down.
```



Ctrl+C when you’re done.

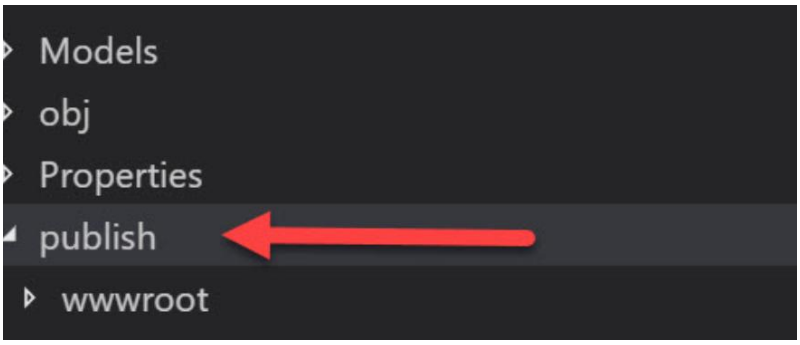
Publish the project to Azure App Service

1. Click on the Azure icon in Visual Studio Code’s left side ribbon

2. Sign-in to your Azure Account when prompted.
3. Drill down into your subscription -> resource group -> App service (that you created in step 1).
4. Visual studio code will prompt you to install the app service extension. Click “Install” to Install the extension.
5. Once the extension is installed, go back to your file explorer.
6. Go back to your terminal and enter the following command:

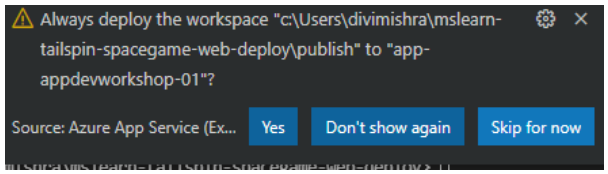
   $ dotnet publish -c Release -o ./publish

   You will notice that a new publish folder has been created



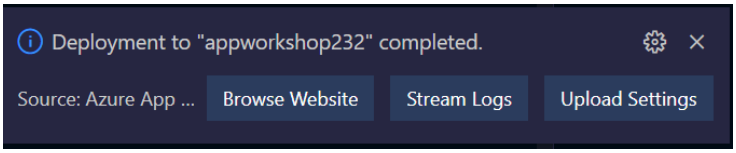
7. Right click on the **publish** folder.
8. Select **Deploy to web app**… and select the right subscription.
9. Select the Azure App service you have created.
10. Select **Deploy** to confirm.

11. If you get the following prompt, click on “**Don’t show again**”.



12. You can view the deployment status using the output window.

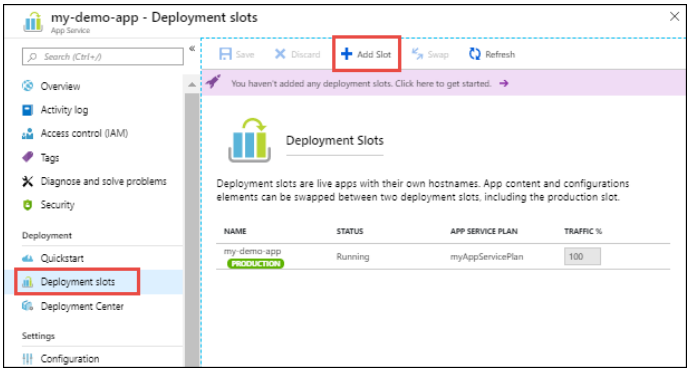
13. Once deployment is completed, select “Browse Website” button to open the azure app services website.



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

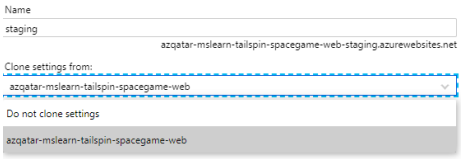
Create deployment slots

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.

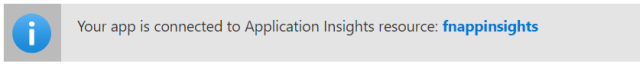
    Note: If you would like to add an access rule restriction for the staging environment, please follow the steps from this document: [Azure App Service access restrictions - Azure App Service | Microsoft Docs]

Lab 102: Monitor and Scale your application

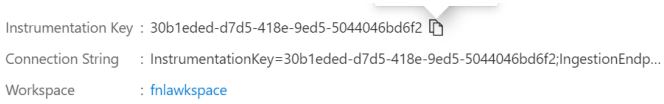
Enabling live telemetry through instrumentation key using VS Code

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 (if you enabled application insights just now).
5. Click on the application insights link to open the application insights instance.

Link to an Application Insights resource



6. Copy the instrumentation key from the application insights overview page.



7. On the VS Code terminal, navigate to /Tailspin.SpaceGame.Web using the cd command.

   $ cd /Tailspin.SpaceGame.Web

8. Run the following command:

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

   Note: You will now see a new package reference in Tailspin.SpaceGame.Web.csproj.

9. 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();
        }
```

Save all the changes (Ctrl + S).

10. You now specify the instrumentation key. Under the appsettings.json file, edit the following:

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

Note:

- Ensure that your repository is a private repository as you include the instrumentation key within the code itself
- Alternatively, you can access the instrumentation as an environment variable (App Service > Configuration > App Settings). The syntax for accessing the app settings:

```
using System.Configuration;
string value = ConfigurationManager.AppSettings["key"];
```

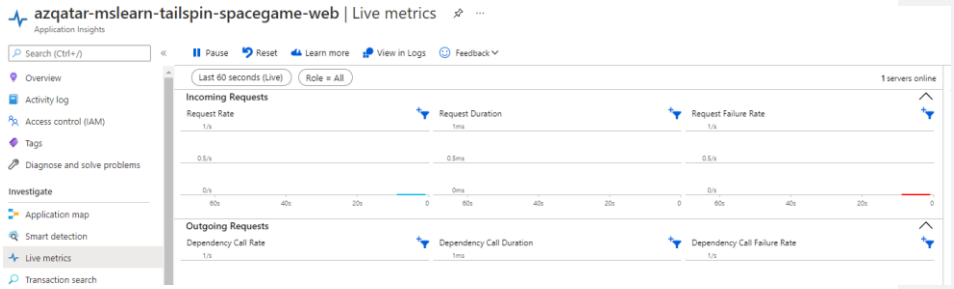
11. Save all changes (Ctrl + S).
12. On the terminal:

    $ cd .. (to go back to the root directory of your web app code)

    $ dotnet build -c Release

    $ dotnet publish -c Release -o ./publish

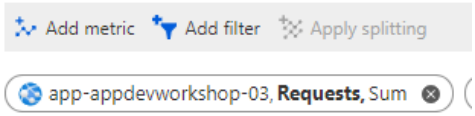
13. Right-click on the **publish** folder and click on **Deploy to web app** for the changes to be reflected. (Soon, you'll deploy a CI/CD pipeline using Azure Pipelines to automate this process and other processes for you).
14. To verify that the application insights package was configured accurately, 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. Go back to your Azure App Service.
2. Under **Monitoring,** click on **Metrics**
3. Select **Requests** as your metric, and **Sum** as your aggregation.
4. Select **Add metric** from the top left.

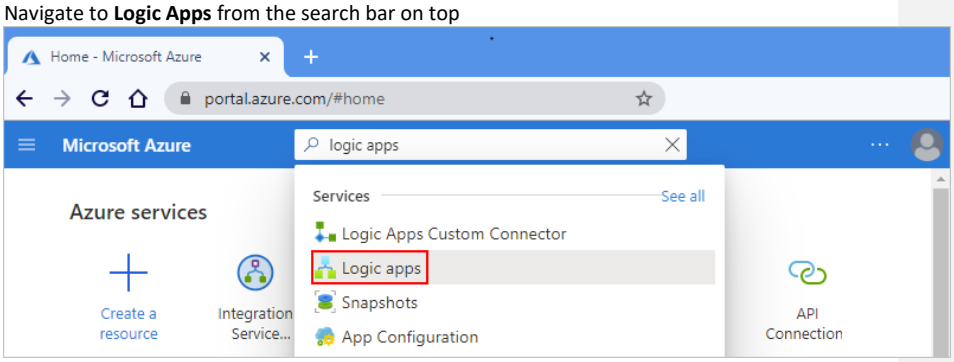


5. Select metrics of your choice and display them into meaningful bar representations. For example:
    a. Select Add metric, and under the Metric dropdown list, select CPU Time. For Aggregation, select Sum.
    b. Select Add metric, and under the Metric dropdown list, select Http Server Errors. For 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.
6. Select "**Pin to Dashboard**". Click on the notification for viewing your dashboard. This is for developing shared dashboards to share with team members.

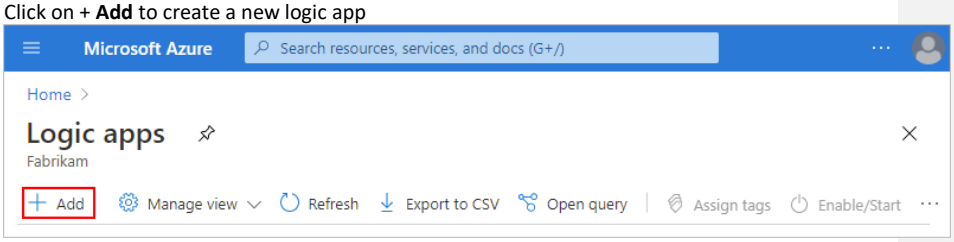
Configure alerts with Azure Logic Apps for Azure App Service

Creating the Logic App

1. Open Azure Portal in a new tab
2. Navigate to **Logic Apps** from the search bar on top



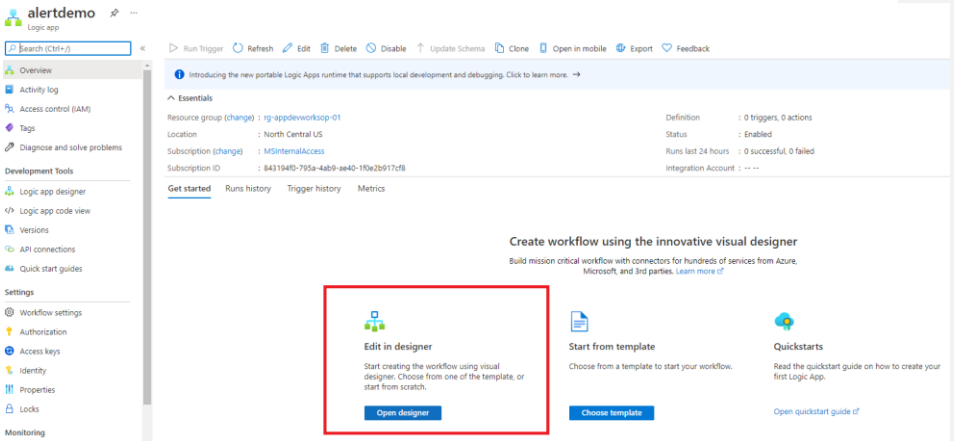
3. Click on + **Add** to create a new logic app



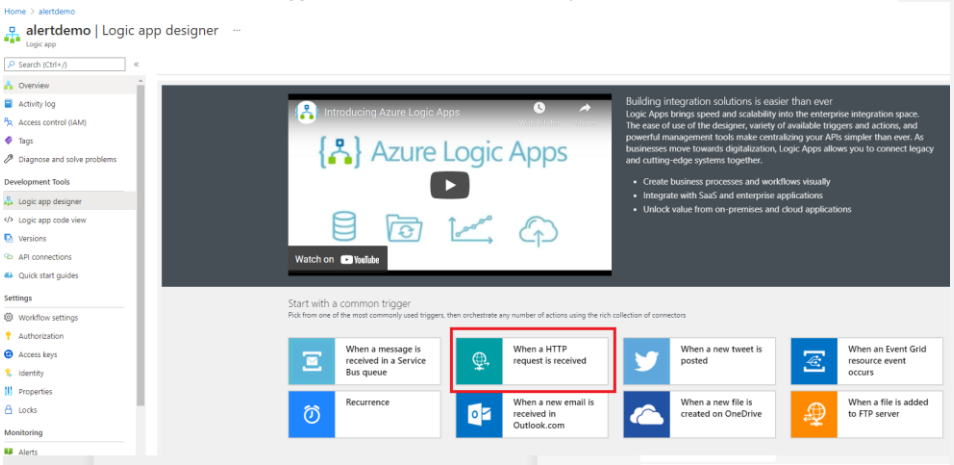
4. Select your resource group
5. Under **Instance Details: T**ype: **Consumption**

   Consumption vs Standard instance details:

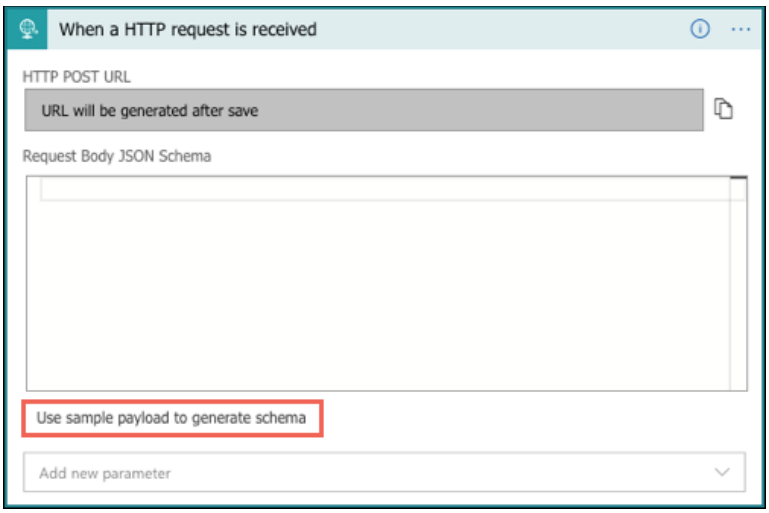
   - **Consumption**:
     - This logic app resource type runs in global, multi-tenant Azure Logic Apps
     - Pay-for-what-you-use pricing model
     - Only one workflow
   - **Standard**: This logic app resource type runs in single-tenant Azure Logic Apps
     - Pricing is based on a hosting plan
     - Portable runtime runs anywhere (Azure, containers, on-prem, etc.)
     - Can have multiple stateful and stateless workflows
6. Select **Review + Create** and then **Create.**
7. Once the logic app resource has been created, click on **Go to resource.**
8. Click on **Open designer**



9. Under **Start with a common trigger,** Click on **When a HTTP request is received**



10. Click on **Use sample payload to generate schema**



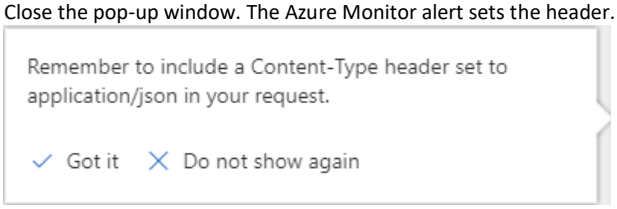
11. Replace the Request Body JSON Schema with the following code:

```
{
        "schemaId": "Microsoft.Insights/activityLogs",
        "data": {
            "status": "Activated",
            "context": {
            "activityLog": {
                "authorization": {
                "action": "microsoft.insights/activityLogAlerts/write",
                "scope": "/subscriptions/…"
                },
                "channels": "Operation",
                "claims": "…",
                "caller": "logicappdemo@contoso.com",
                "correlationId": "91ad2bac-1afa-4932-a2ce-2f8efd6765a3",
                "description": "",
                "eventSource": "Administrative",
                "eventTimestamp": "2018-04-03T22:33:11.762469+00:00",
                "eventDataId": "ec74c4a2-d7ae-48c3-a4d0-2684a1611ca0",
                "level": "Informational",
                "operationName": "microsoft.insights/activityLogAlerts/write",
                "operationId": "61f59fc8-1442-4c74-9f5f-937392a9723c",
                "resourceId": "/subscriptions/…",
                "resourceGroupName": "LOGICAPP-DEMO",
                "resourceProviderName": "microsoft.insights",
                "status": "Succeeded",
                "subStatus": "",
                "subscriptionId": "…",
```

```
                    "submissionTimestamp": "2018-04-03T22:33:36.1068742+00:00",
                    "resourceType": "microsoft.insights/activityLogAlerts"
                }
            },
            "properties": {}
        }
    }
```

12. Click on **Done.**

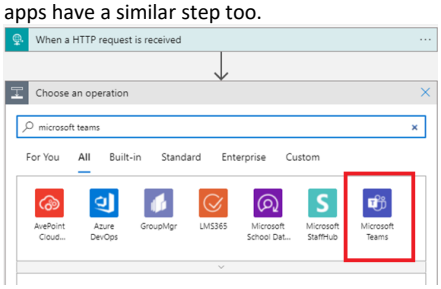
13. Close the pop-up window. The Azure Monitor alert sets the header.



14. Click on + **New step**

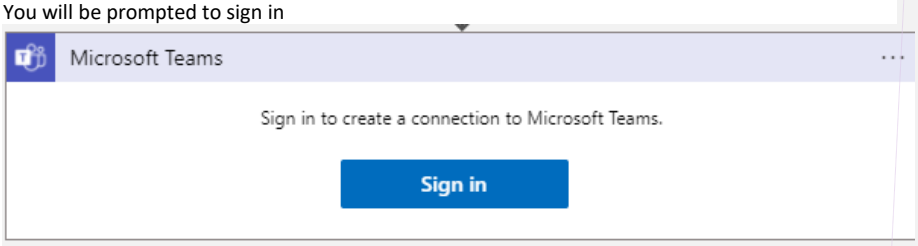
15. Search for **Microsoft Teams**

Note: Please search for your respective team collaboration platform. Example: Microsoft Teams, Skype for Business, Slack, etc. This lab shows steps for Microsoft Teams, however, the other apps have a similar step too.



16. Choose **Microsoft Teams – Create a chat** action

17. You will be prompted to sign in



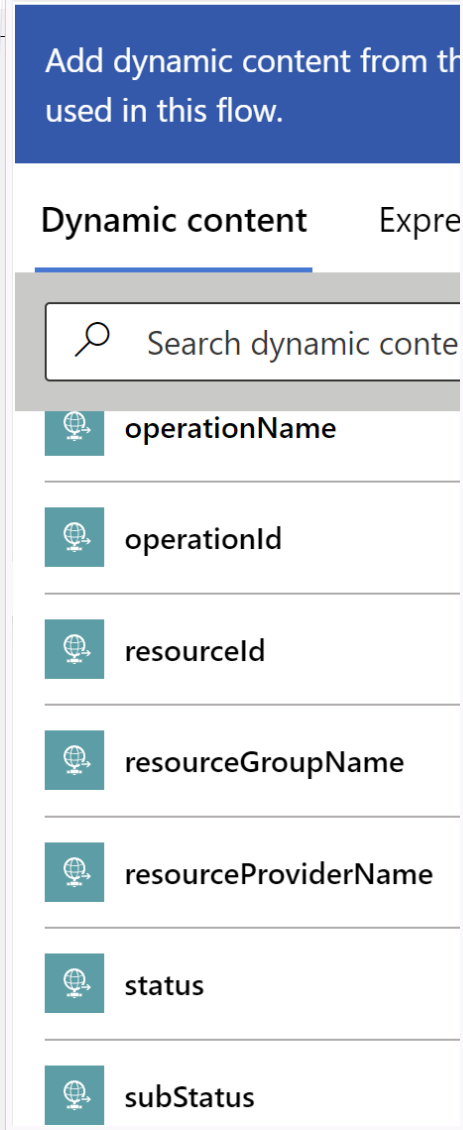
18. Enter your teammates' email addresses.

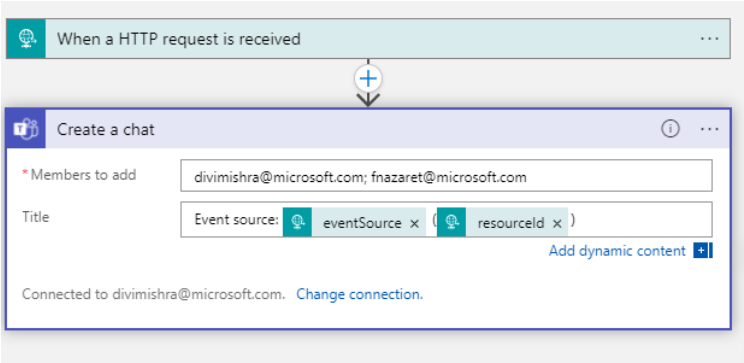
For Title, type:

Event Source: <eventSource> <resourceID>

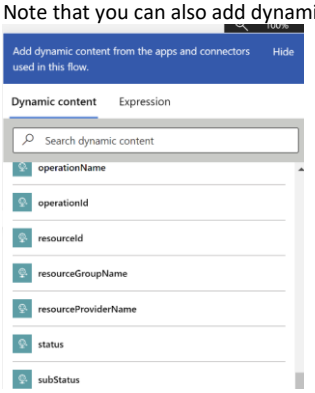
Here, **eventSource** and **resourceID** is dynamic content

Commented [MS3]: I think it's good to mention the left pane where they can add the Dynamic content.





Note that you can also add dynamic content as a part of this action:

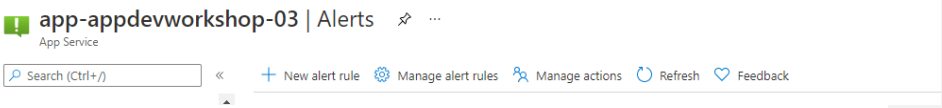


19. Click on **Save.** (You can click on **Run Trigger** for a manual run to check if it works).

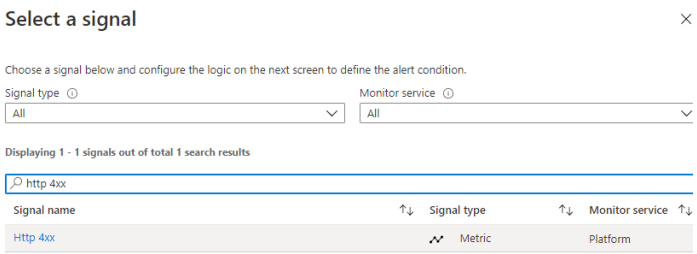
Creating the alert

1. Navigate back 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 400 or 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 **Add Action Groups.**

a. Click on **+ Create action group**

b. Under **Basics > instance details**, name your action group something relevant. For example, **dotnetapp_alertgroup**

c. Under **Actions**:

i. Action type: **Logic App**

ii. Select the logic app you just created

iii. Name: **msteamschat**

d. Select **Review + Create**

e. Select **Create**

7. Select the alert rule name, description, severity.

Note: 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.

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)”.

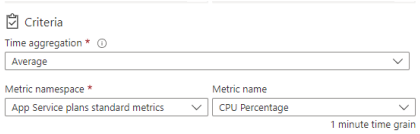
There are 2 ways to scale your application: manually or automatically. We certainly recommend autoscaling for unexpected spikes in traffic, errors, and to respond to your organization’s monitoring strategy.

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 Auto-scale):**

When scaling automatically: decide if you want to scale based on a metric or scale to a specific instance count (when no other scale conditions are matched). You decide the minimum, maximum and default instances that you’d like to have for your ASP.

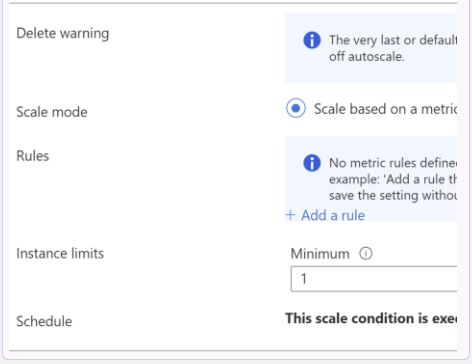
Here, we scale based on a metric:

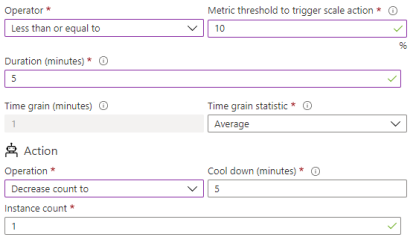
a. Add a rule to set instance count to 1 if the CPU percentage is less than 10%. Explore through the other standard metrics that you can also leverage.



You can define the operation to increase/decrease the count of instances according to your scaling rule.

Commented [MS4]: Adding more guidance on how to create rules in this windows





Best practice: for every scale out rule, there must be a scale in rule (and vice versa).

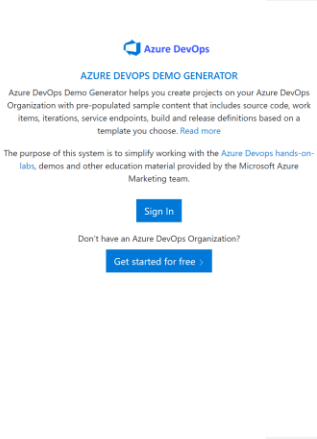
b. Click on Save.

c. Review your dashboard to monitor performance.

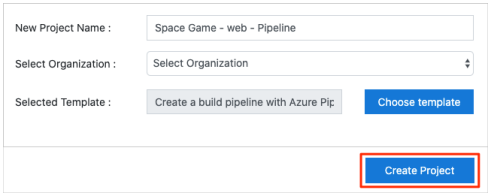
Lab 103: Continuous Integration with Azure DevOps

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]



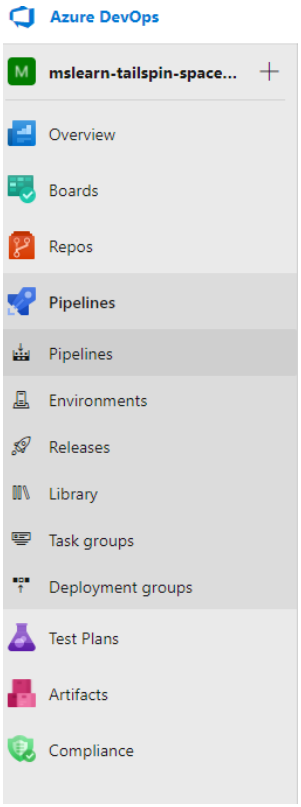
2. Sign in and accept the usage terms.
3. Create a new project with your Azure DevOps (ADO) organization and a project name. For Selected template: under MS learn, chose “Create a build pipeline with Azure Pipelines”. Finally, create project.



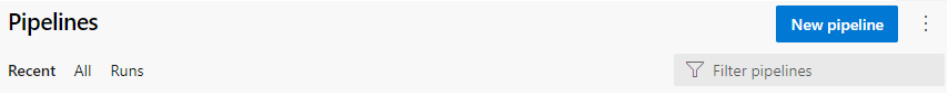
4. Once the deployment is done, select “Navigate to the project”

Create a build pipeline with Azure Pipelines

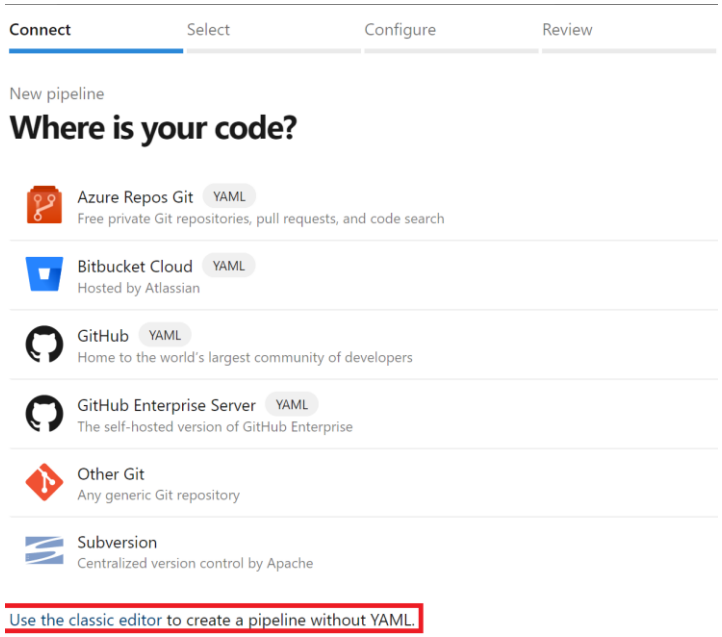
1. Once your project is deployed, navigate to **Pipelines**.



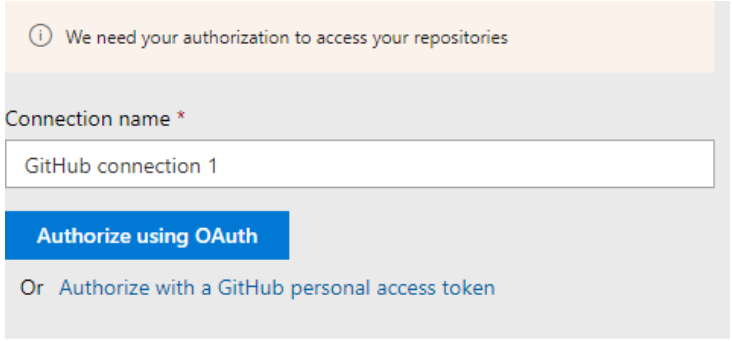
2. Create a new pipeline.



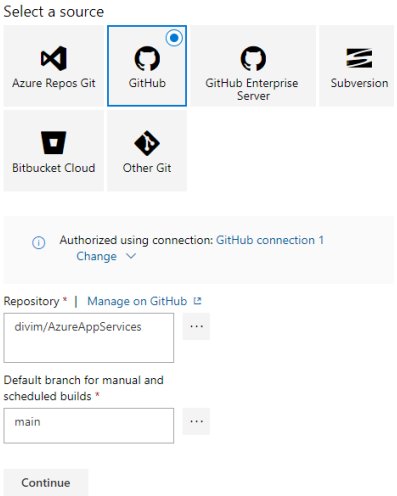
3. Use the classic editor option at the bottom.



4. Select your source as GitHub. You will be required to sign in. You can use **OAuth** .

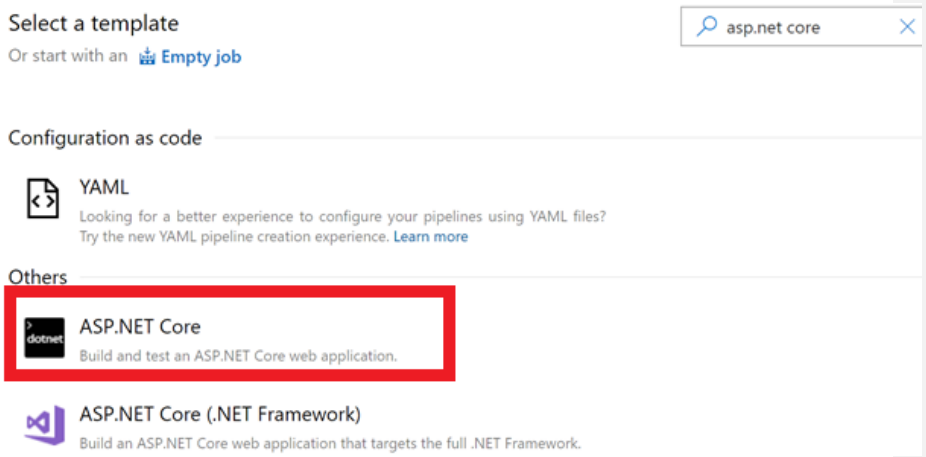


5. Select your forked repository and the default branch as **main**.



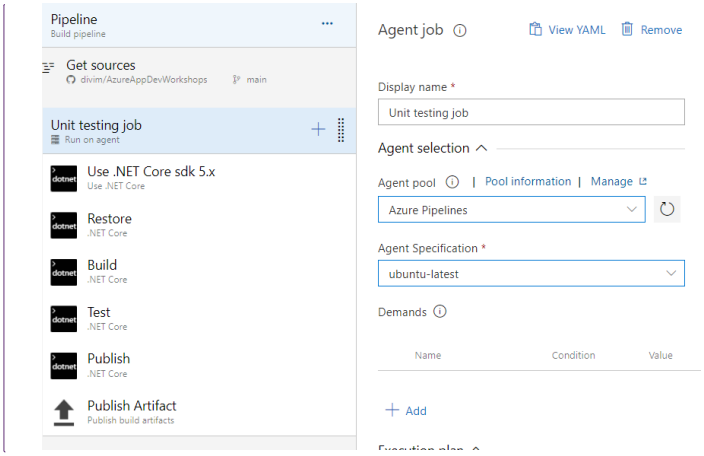
Select “Continue”.

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

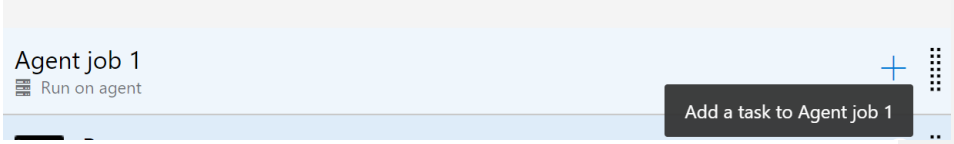


7. Your pipeline currently looks like this:
    a. Please ensure to use the “ubuntu-latest” specification for the Azure pipeline agent pool’s specification

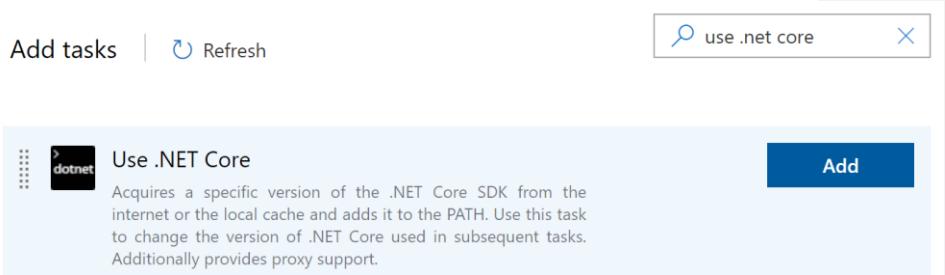
Commented [MS5]: Ubuntu-v16 is not exist anymore, better to mention the latest version instead.



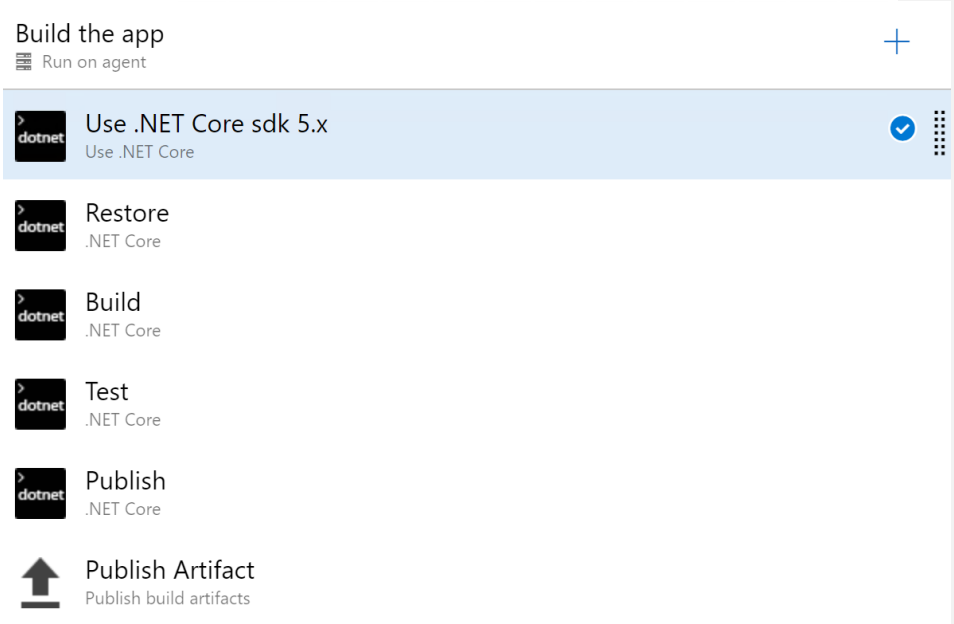
8. Click on the "+" to add an agent job.



9. Search and click on **"Add"** for the following task: **Use. NET Core**



10. Your pipeline will now look like this:



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

Use .NET Core

Link settings | View YAML | Remove

Task version 2.*

Display name *

Use .NET Core sdk 5.x

Package to install

SDK (contains runtime)

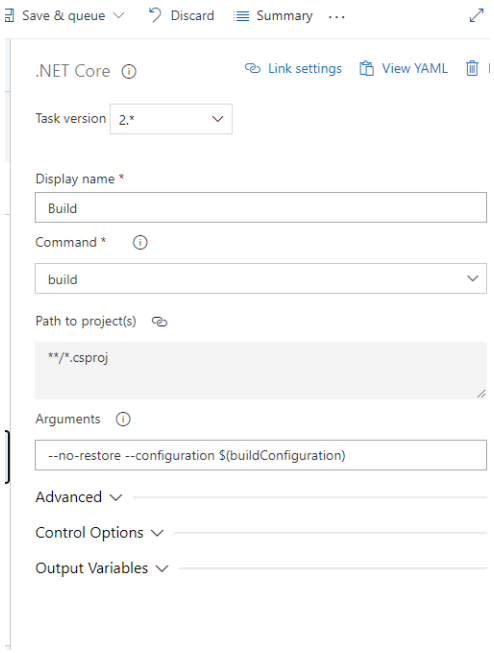
- [ ] Use global json

Version

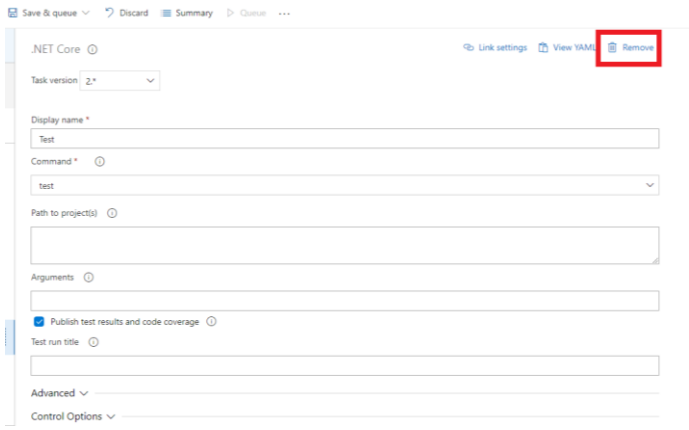
5.x

- [ ] Include Preview Versions

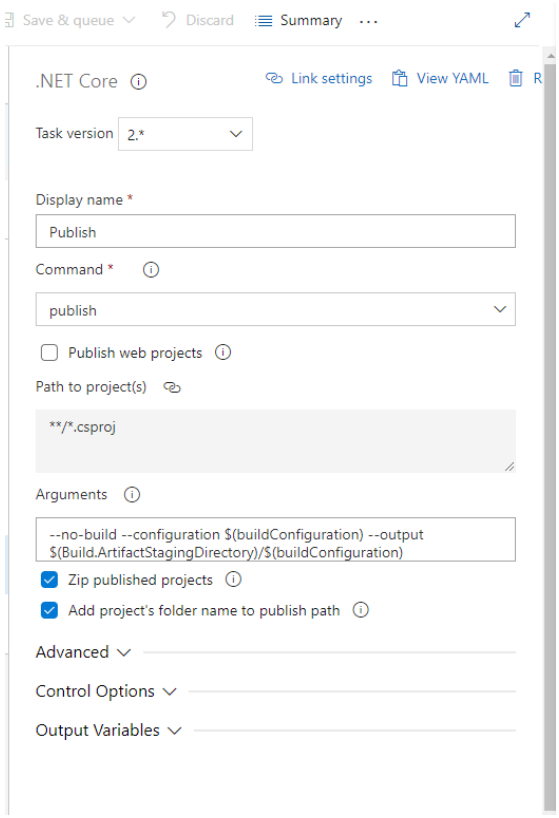
12. Click on the "**Build**" task and edit the "**Arguments**" as follows: `--no-restore --configuration $(buildConfiguration)`



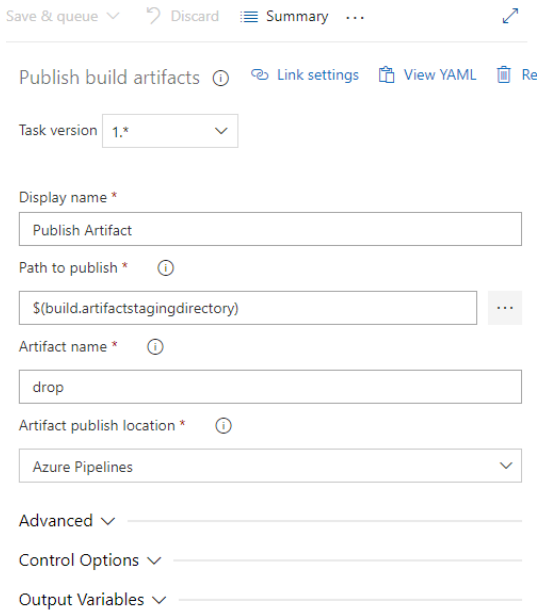
13. Click on the **Test** task and delete it by clicking on "**Remove**". We will be creating unit tests and adding it to the pipeline in the lab after this.



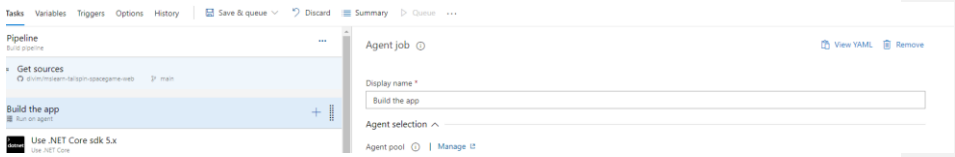
14. Click on the **Publish** task and edit it as follows:
    a. Unselect the "**Publish web projects"** button.
    b. Edit the arguments as follows: `--no-build --configuration $(buildConfiguration) --output $(Build.ArtifactStagingDirectory)/$(buildConfiguration)`



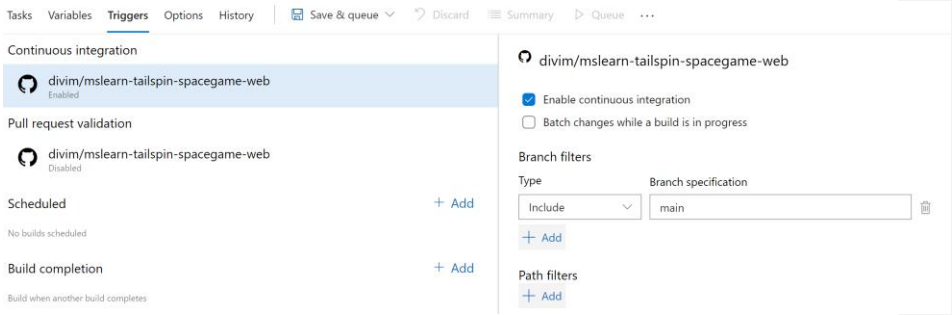
15. Click on the **Publish Artifact** task. Under **Control Options** > **Run this task**, click on the option **"Only when all previous tasks have succeeded".**



16. Click on **Agent Job 1** and rename it to **Build the app**

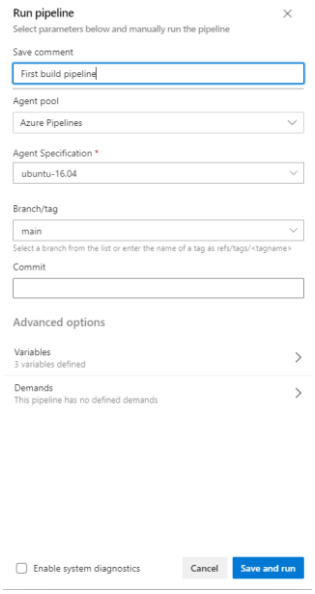


17. Under the **Triggers** section, **enable continuous integration**



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

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



20. Observe the tasks running.

    You have successfully created your build pipeline.

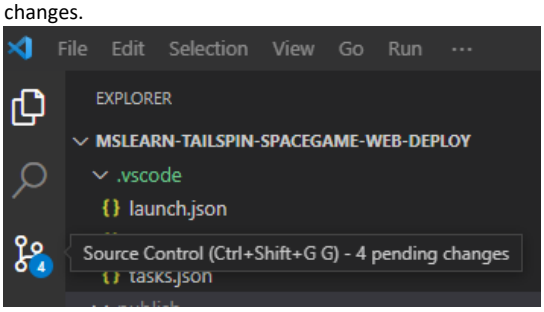
Source Control with GitHub

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.

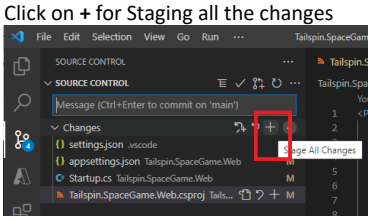
Recall that we had made some changes to the VS Code when we added the Application Insights instrumentation key.

Let's now push it to GitHub.

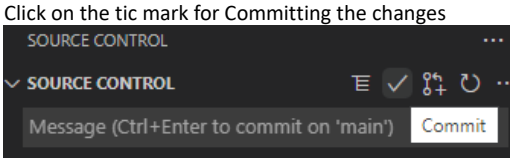
1. Navigate back to your VS Code and notice that your **Source Control** tab highlights pending changes.



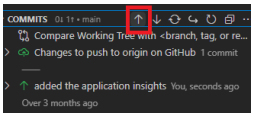
2. Click on the **Source Control** tab
3. Click on **+** for Staging all the changes



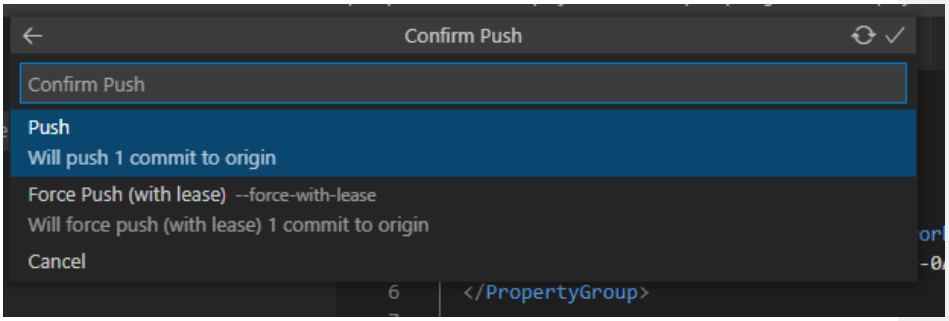
4. Click on the tic mark for Committing the changes



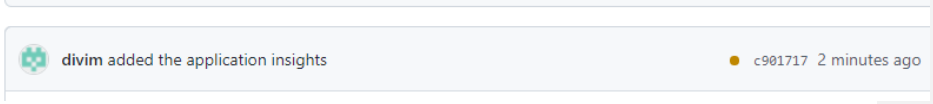
5. Enter a relevant message, such as "added app insights instrumentation key"
6. Under **COMMITS** (at the bottom), click on the UP arrow. This will **Push** all the changes to GitHub.



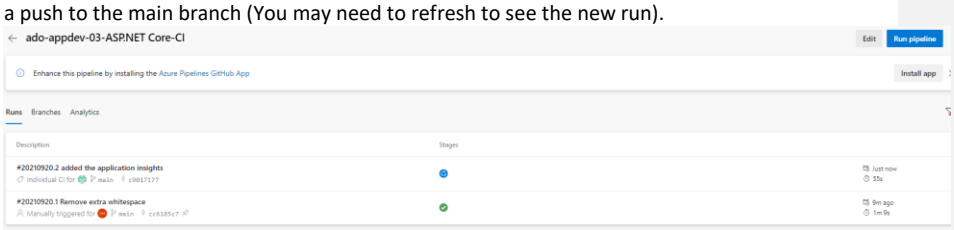
7. Confirm the Push.



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 (You may need to refresh to see the new run).



Note:

You have now configured a way to reflect changes from your VS Code -> GitHub -> Azure Pipelines.

In the next two labs, we will complete the pipeline as VS Code -> GitHub -> Azure Pipelines **-> Azure App Services.**

Lab 104: Testing with Azure Pipelines

Run unit tests locally

1. Open your VS Code workspace
2. Move to the testing directory:
   $ cd .\Tailspin.SpaceGame.Web.Tests\
3. Run the tests locally:

   $ dotnet test

```
Test run for C:\Users\divimishra\OneDrive - Microsoft\Desktop\App Innovation Workshops\Personal\AzureAppServices\Tailspin.SpaceGame.Web.Tests\bin\Debug\net5.0\Ta
Microsoft (R) Test Execution Command Line Tool Version 16.11.0
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test execution, please wait...
A total of 1 test files matched the specified pattern.

Passed!  - Failed:     0, Passed:     5, Skipped:     0, Total:     5, Duration: 70 ms - Tailspin.SpaceGame.Web.Tests.dll (net5.0)
```

Run Code Coverage test locally

1. On your terminal, run the following commands:
    a. *Return to the parent directory of **Azure App Service** using the cd command*
    b. *Create a local tool manifest file:*
       $ **dotnet new tool-manifest**
    c. *Install ReportGenerator:*
       $ **dotnet tool install dotnet-reportgenerator-globaltool**

       Warning: You may receive an error System.UnauthorizedAccessException error. You can ignore that and proceed with the next steps

    d. *Add the coverlet.msbuild package to the project (please ensure that you're at the 'Azure App Service' directory for this):*
       $ **dotnet add Tailspin.SpaceGame.Web.Tests package coverlet.msbuild**
    e. *Run the command for unit test and code coverage (the /p: tells coverlet what format to use and where to place the results):*
       $ dotnet test Tailspin.SpaceGame.Web.Tests/Tailspin.SpaceGame.Web.Tests.csproj --configuration Release /p:CollectCoverage=true /p:CoverletOutputFormat=cobertura /p:CoverletOutput=./TestResults/Coverage/

       After running this command, you'll see the following:

```
Starting test execution, please wait...
A total of 1 test files matched the specified pattern.

Passed!  - Failed:     0, Passed:     5, Skipped:     0, Total:     5, Duration: 102 ms - Tailspin.SpaceGame.Web.Tests.dll (net5.0)

Calculating coverage result...
  Generating report '.\TestResults\Coverage\coverage.cobertura.xml'

+----------------------------+-------+--------+--------+
| Module                     | Line  | Branch | Method |
+----------------------------+-------+--------+--------+
| Tailspin.SpaceGame.Web     | 11.45% | 0%     | 18.91% |
+----------------------------+-------+--------+--------+
| Tailspin.SpaceGame.Web.Views | 0%    | 0%     | 0%     |
+----------------------------+-------+--------+--------+

+---------+-------+--------+--------+
|         | Line  | Branch | Method |
+---------+-------+--------+--------+
| Total   | 7.77% | 0%     | 14.58% |
+---------+-------+--------+--------+
| Average | 5.72% | 0%     | 9.45%  |
+---------+-------+--------+--------+
```

    f. *Use ReportGenerator to convert Cobertura to HTML*
       $ **dotnet tool run reportgenerator -reports:./Tailspin.SpaceGame.Web.Tests/TestResults/Coverage/coverage.cobertura.xml -targetdir:./CodeCoverage -reporttypes:HtmlInline_AzurePipelines**

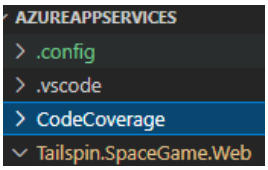
2. You will notice a new folder called CodeCoverage created

Commented [MS6]: This command gives the following error:

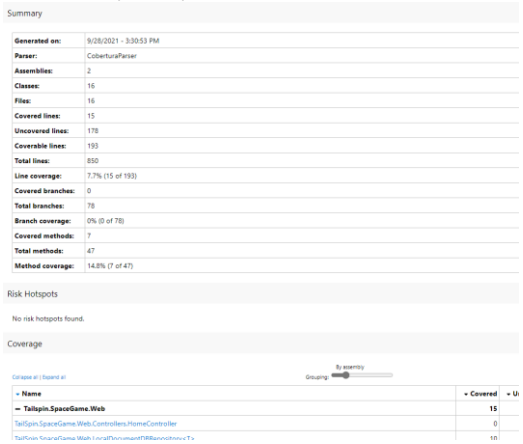
```
PS C:\Users\msoultan\source\repos\AzureAppDevWorkshops\Azure App Service\Tailsp
Could not find project or directory `Tailspin.SpaceGame.Web.Tests`.
```

It should be replaced by
dotnet add package coverlet.msbuild

Commented [MS7]: Wasn't able to execute this command, as it depends on the previous one



3. Right click on CodeCoverage's **Index.htm** file and open it in Explorer to view the file in your browser.
   You will see your report:



Summary

Generated on:	9/28/2021 - 3:30:53 PM
Parser:	CoberturaParser
Assemblies:	2
Classes:	16
Files:	16
Covered lines:	15
Uncovered lines:	178
Coverable lines:	193
Total lines:	850
Line coverage:	7.7% (15 of 193)
Covered branches:	0
Total branches:	78
Branch coverage:	0% (0 of 78)
Covered methods:	7
Total methods:	47
Method coverage:	14.8% (7 of 47)

Risk Hotspots

No risk hotspots found.

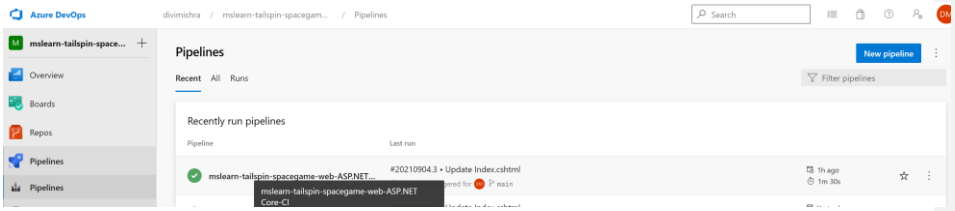
Coverage

Name	Covered
Tailspin.SpaceGame.Web	15
TailSpin.SpaceGame.Web.Controllers.HomeController	0

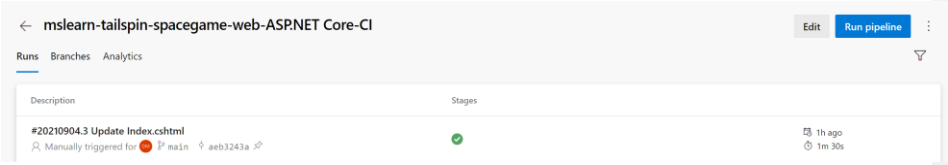
4. Commit your changes to your GitHub repository.

Add tests to Azure Pipeline

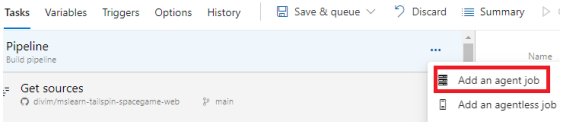
1. Open your Pipeline on dev.azure.com again.



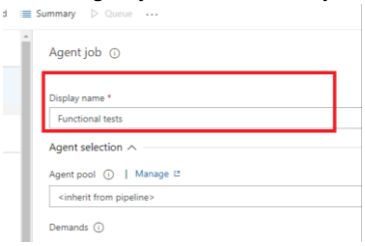
2. Click on **Edit** from the top right to edit the pipeline.



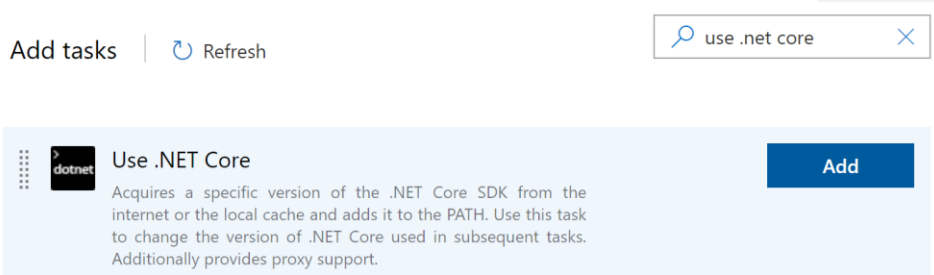
3. Select the three dots next to **Pipeline** and click on **Add an agent job**



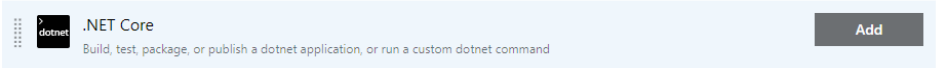
4. Click on **Agent job** and rename the job to **Test the app.** Leave the rest as default.



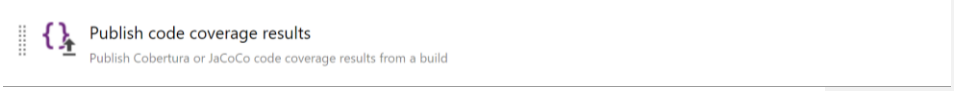
5. Click on the **+** to add a new task to the Functional tests job
6. Add the following jobs:
    a. Use. NET Core



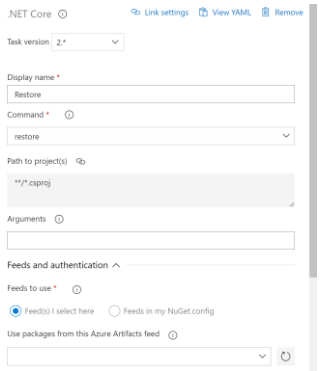
    b. .NET Core **: Do this 2 times!**



    c. Publish code coverage results



7. Click on the **Use .NET Core** task
    a. Version: **5.x**
8. Click on the **.NET Core #1** task
    a. Display Name: **Restore**
    b. Command: **restore**
    c. Path to project: ****/*.csproj**



9. Click on the next **.Net Core** task:
    a. Display name: **Test the app**
    b. Command: **test**
    c. Path to project: ****/*.Tests.csproj**
    d. Arguments: **--configuration $(buildConfiguration) /p:CollectCoverage=true /p:CoverletOutputFormat=cobertura /p:CoverletOutput=$(Build.SourcesDirectory)/TestResults/Coverage/**
    e. **Select** the Publish test results and code coverage option

10. Click on the **Publish code coverage task**:
    a. Code coverage tool: **Cobertura**
    b. Summary file: **$(Build.SourcesDirectory)/**/coverage.cobertura.xml**

Publish code coverage results

Task version 1.*

Display name *

Publish code coverage from $(Build.SourcesDirectory)/**/coverage.cobertura.xml

Code coverage tool *

Cobertura

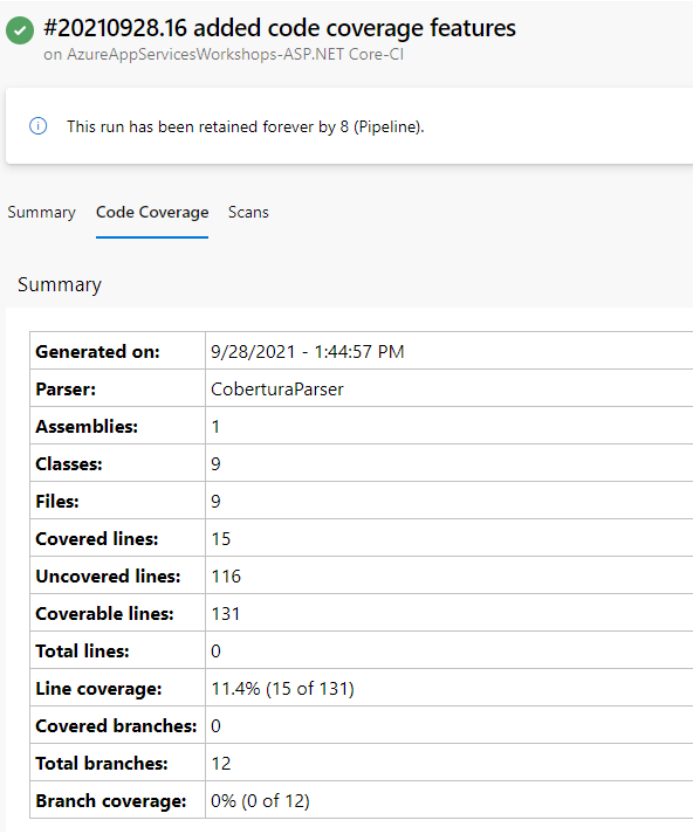
Summary file *

$(Build.SourcesDirectory)/**/coverage.cobertura.xml

11. Click on **Save & Queue** to save your progress and manually trigger the pipeline.
12. Navigate back to the **pipeline summary** of your latest run. Here, you will see the Test and coverage section:

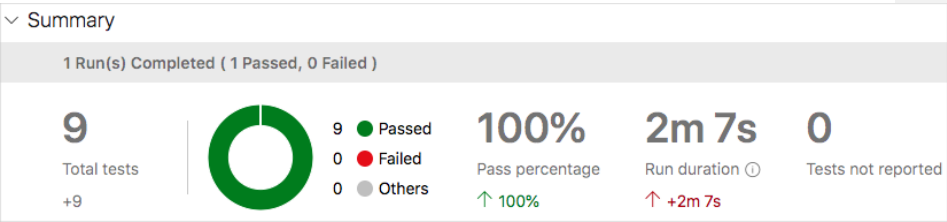
Commented [MS8]: The testing task gives the following error

```
The argument /home/vsts/work/1/s/Azure App Service/Tailspin.SpaceGame.Web.Tests/bin/Re
##[error]Error: The process '/opt/hostedtoolcache/dotnet/dotnet' failed with exit code
##[warning]No test result files were found.
##[warning].NET 5 has some compatibility issues with older Nuget versions(<=5.7), so i
Info: Azure Pipelines hosted agents have been updated and now contain .Net 5.x SDK/Run
##[error]Dotnet command failed with non-zero exit code on the following projects : /ho
Finishing: Test the app
```



Generated on:	9/28/2021 - 1:44:57 PM
Parser:	CoberturaParser
Assemblies:	1
Classes:	9
Files:	9
Covered lines:	15
Uncovered lines:	116
Coverable lines:	131
Total lines:	0
Line coverage:	11.4% (15 of 131)
Covered branches:	0
Total branches:	12
Branch coverage:	0% (0 of 12)

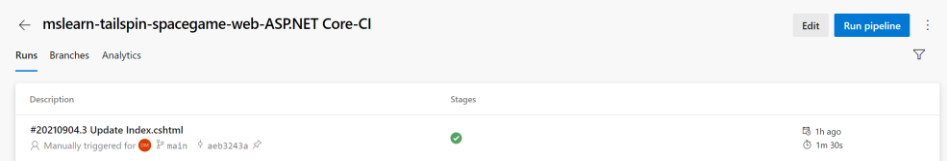
13. Move to the **Tests** tab to view a summary of the test run.



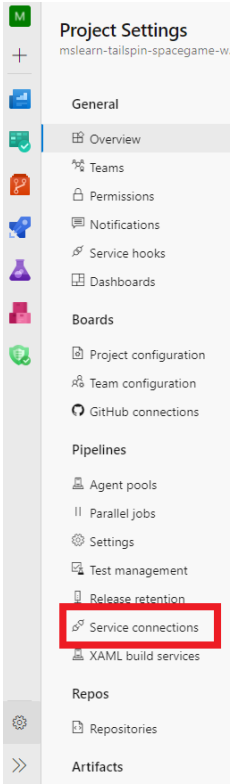
Lab 105: Zero-downtime app deployment with release pipeline

Deploy to staging slot from build pipeline

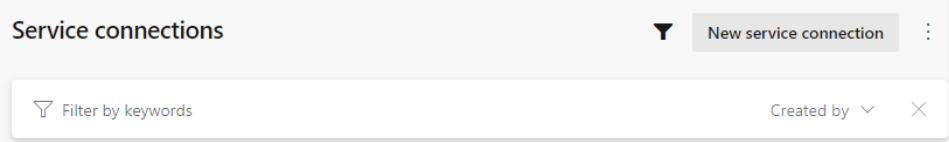
1. Go back to your build pipeline in dev.azure.com.
2. Click on **Edit** from the top right to edit the pipeline.



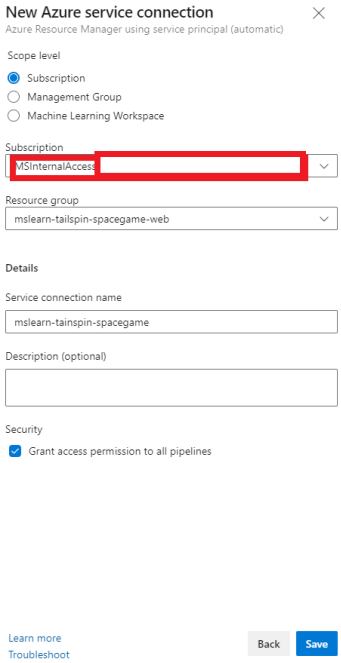
3. At the bottom left, click on **Project Settings.** Under Pipelines, navigate to Service connections.



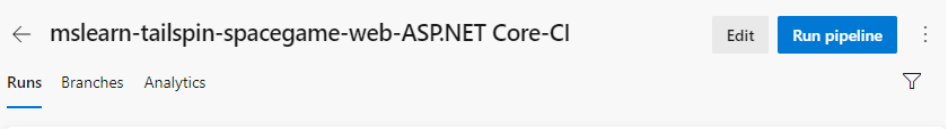
4. Click on **New Service Connection**



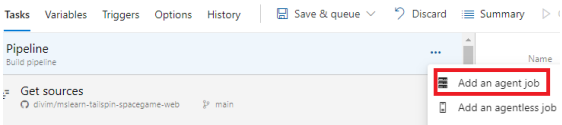
5. Click on **Azure Resource Manager.** Select Next.
6. Click on **Service Principal (Automatic).** Select Next.
7. Define your Subsciption and Resource group that contains your app service.
8. Give a relevant service connection name.
9. Select **Grant access permission to all pipelines.**



10. Select **Save.**
11. Navigate back to your pipeline and select **Edit.**

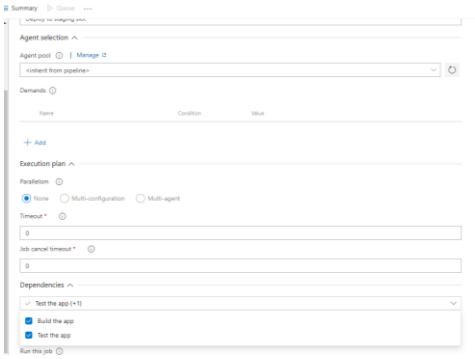


12. Select the three dots next to **Pipeline** and click on **Add an agent job**



13. Click on **Agent job**

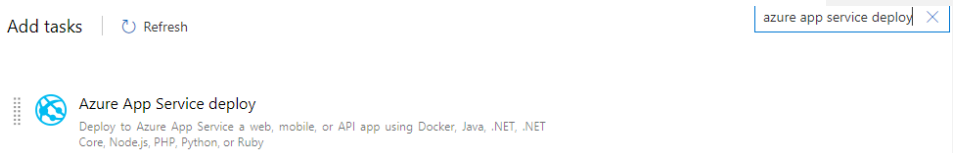
- **a.** **R**ename the job to **Deploy to Staging environment**
- b. Choose dependencies as both the other build pipelines:



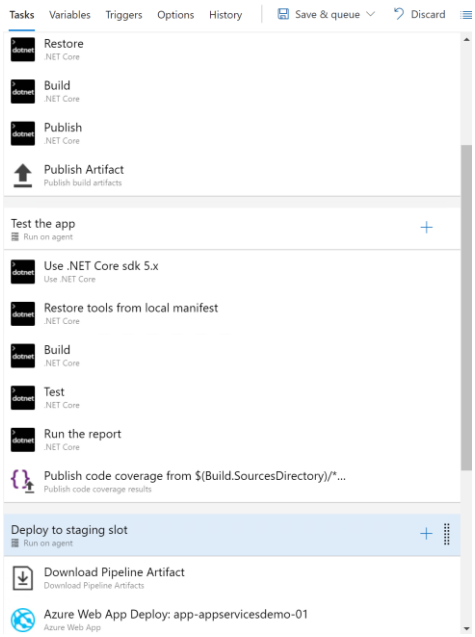
14. Add the following agent jobs:

- a. Download pipeline artifact

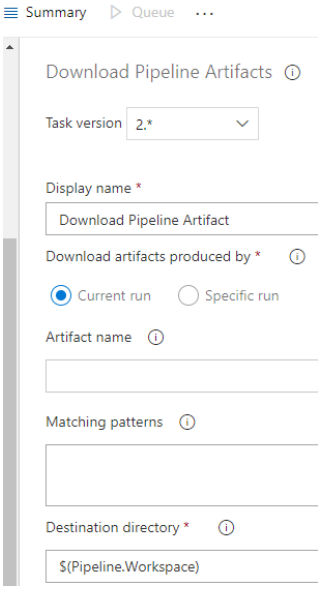
- b. Azure App Service Deploy



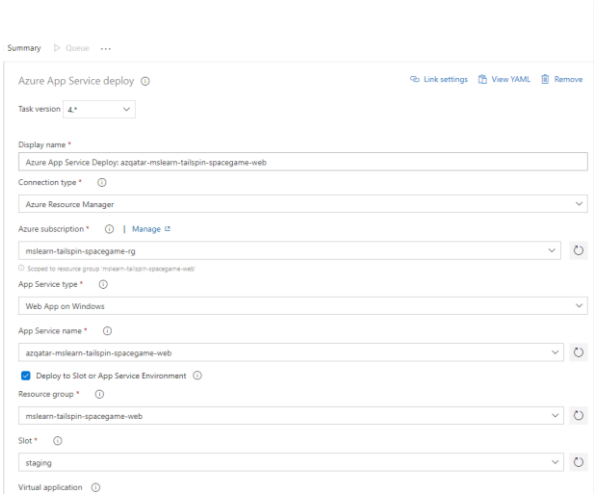
15. Ensure that your pipeline now looks like this:



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)/**/Tailspin.SpaceGame.Web.zip**



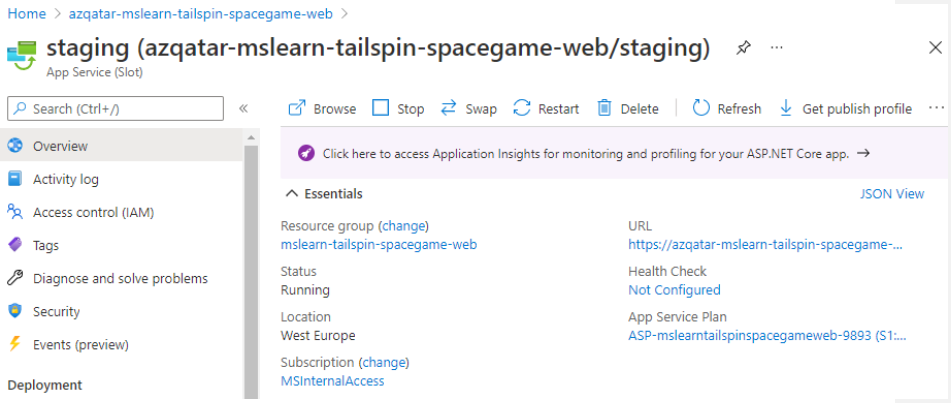
18. Select **Save & Queue.**

19. Select **Save & Run.**

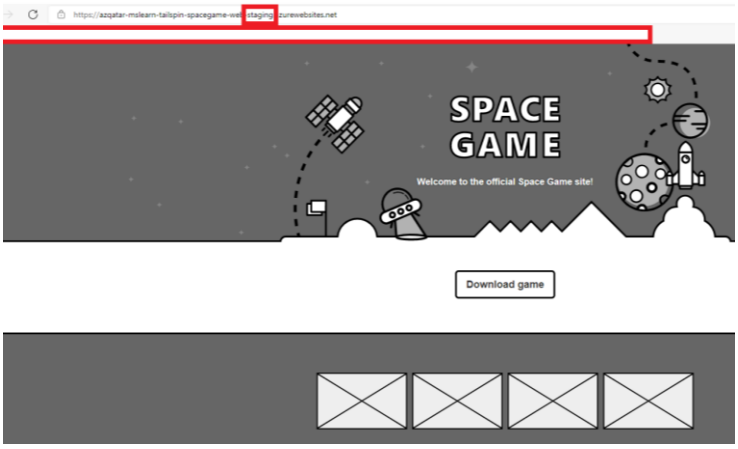
20. Once the pipeline is done running, go back to Azure Portal > App Service > your app service > Deployment Slots

21. Click on the **staging slot**

22. Click on **Browse**



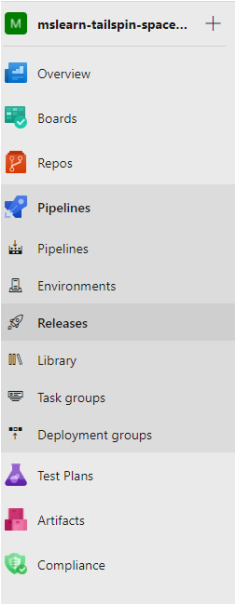
23. Your app has been deployed successfully to the staging slot. Note that the name of the URL is simply <app-service-name>-staging.azurewebsites.net



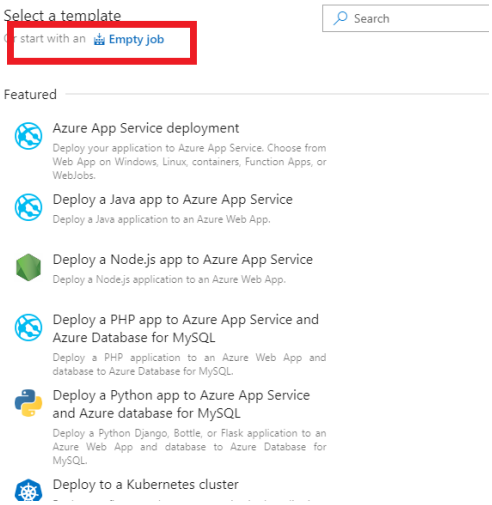
24. Simply remove the “-staging” from the URL to view the production slot website.

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

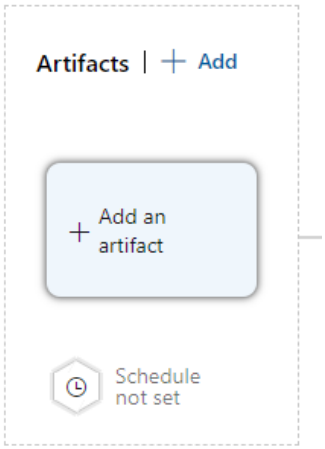
1. Navigate to your Azure DevOps project > **Pipelines > Releases**



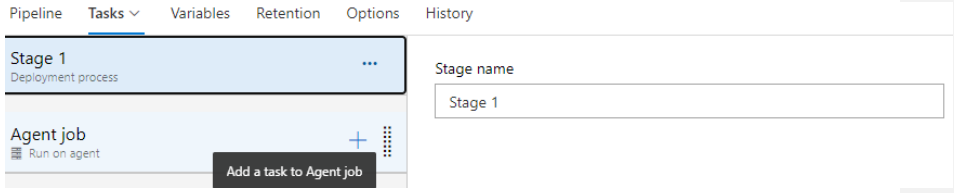
2. Select "New pipeline".
3. For the template, **start with an empty job**



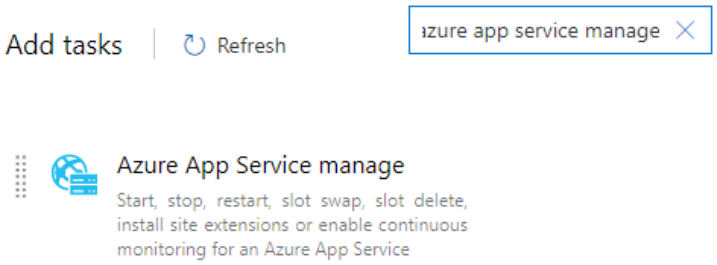
4. Click on **Add an artifact**



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 "+".

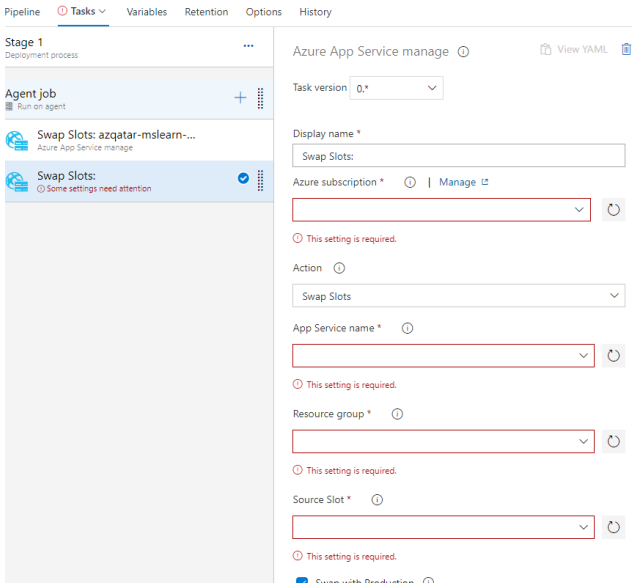


11. Search for **Azure App Service manage** and click **Add.**

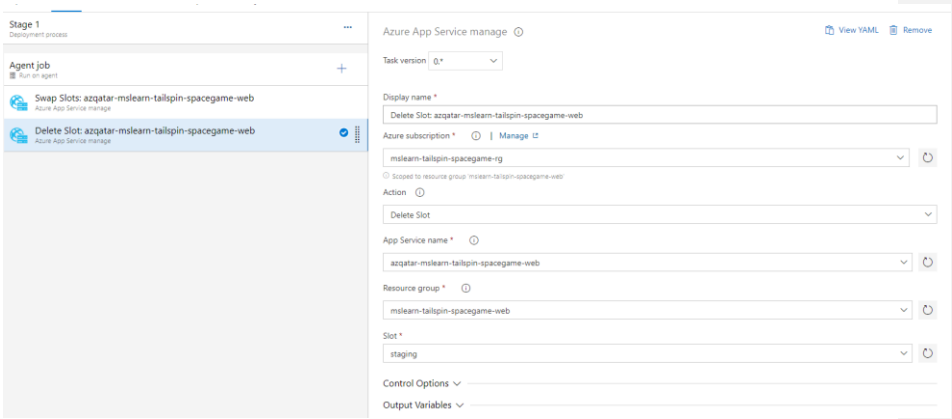


12. Click on the task to edit the task as follows:
    a. Select your service connection under Azure Subsciption.
    b. Select the action as **Swap slots.**
    c. Select your app service for the app service name and its associated resource group.
    d. Select **staging** as your source slot.
    e.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

13. Add another task for **Azure App Service manage.**

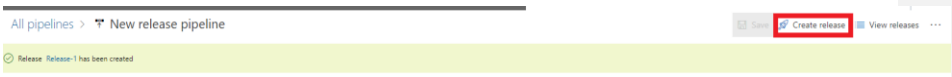


14. Select the task to edit it as follows:
    a. Select your service connection for the Azure subscription
    b. Select your action as **Delete Slot**
    c. Select your app service name and resource group
    d. Select the slot to be deleted as **staging**

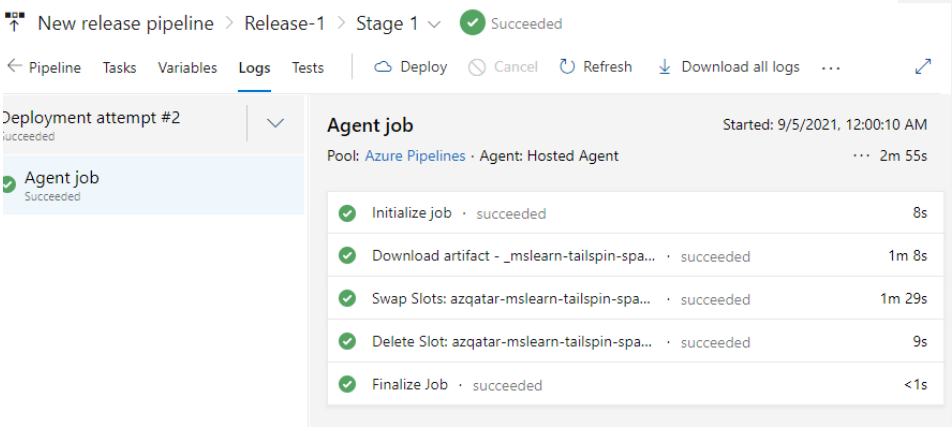


15. Select **Save.**

16. Select **Create release.**

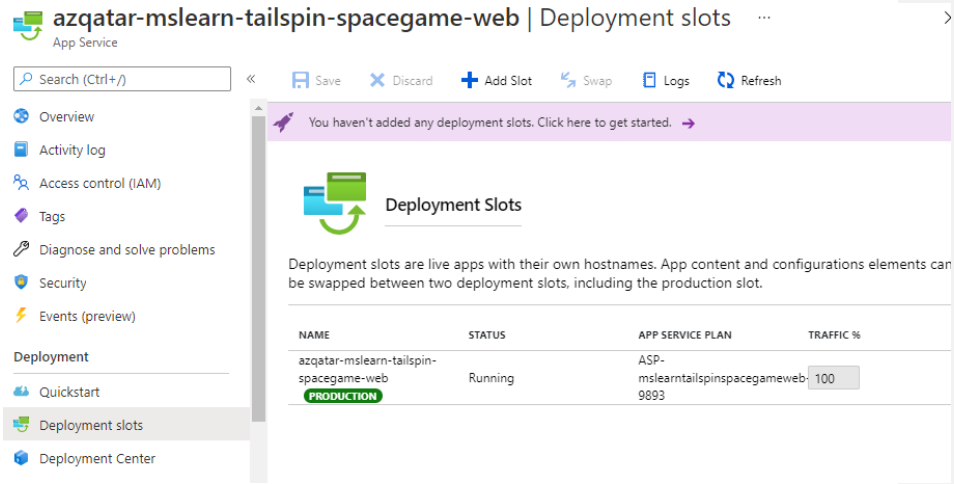


17. Observe the tasks on the release pipeline.



18. Once the pipeline has been completed, navigate back to Azure Portal.

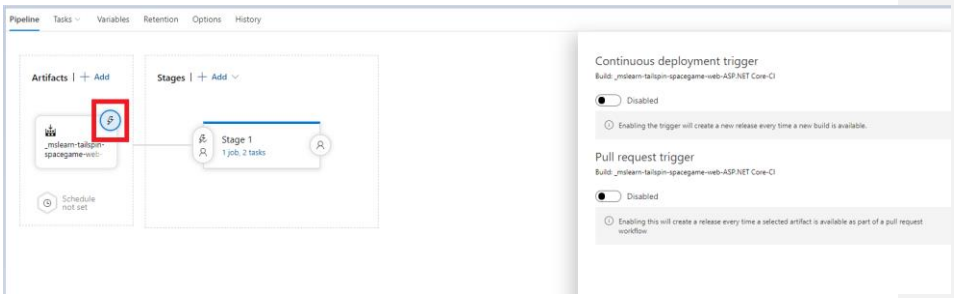
19. Under deployment slots, you will notice that the slot is now deleted.



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.



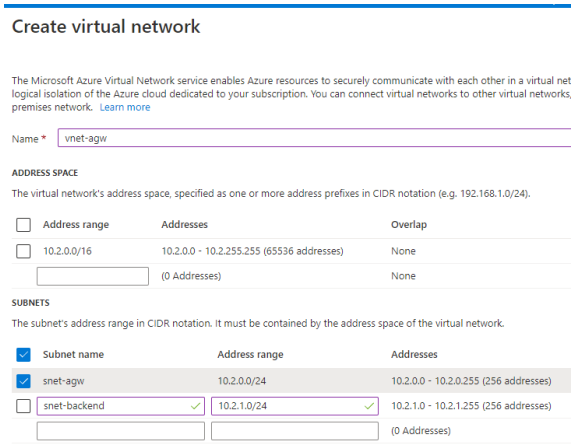
Lab 201: Protect your application

Deploy WAF with your application

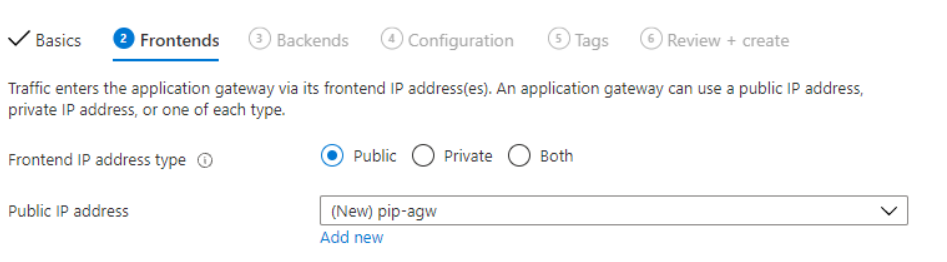
Create WAF

1. Sign in to the Azure Portal
2. On the Azure Portal Menu, select **Create a resource**
3. Select **Networking** > **Application Gateway**
4. Under the Basics tab:
    a. Resource group: Select the RG for your app service or create a new one
    b. App Gateway Name: agw-demo
    c. Tier: WAF V2
    d. Virtual network: Create new virtual network
        i. Name: vnet-agw
        ii. Address space: Leave as default
        iii. Subnets (x = value from address range):
            1. snet-agw: 10.x.0.0/24
            2. snet-backend: 10.x.1.0/24

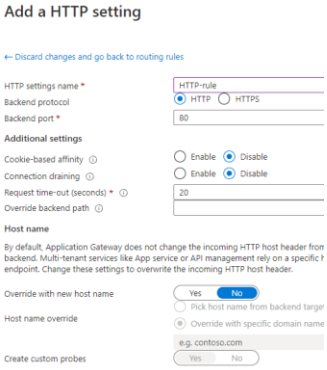
    Here is an example:



    e. Click on **OK**
5. **Frontends tab:**
    a. Type: Public
    b. Public IP Address: **Add new**

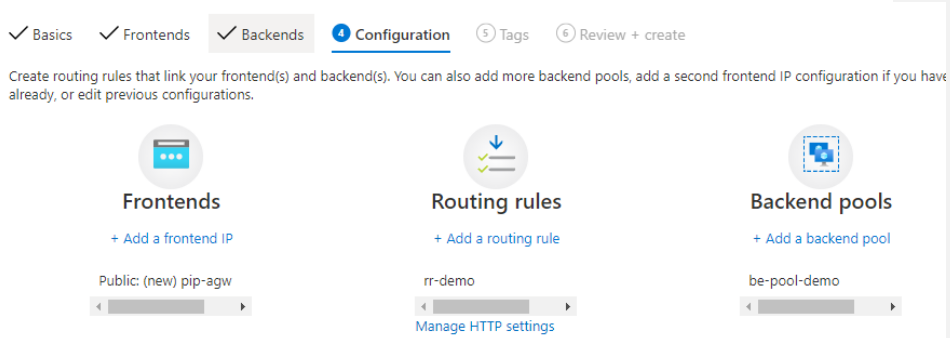


6. **Backends tab:**
    a. Add a backend pool
    b. Name: **be-pool-demo**
    c. Add backend pool without targets: **Yes**
    d. **Add** the backend pool
7. **Configuration tab:**
    a. Select **Add a routing rule**
    b. Rule name: rr-demo
    c. Listener:
        i. Listener name: **listener-demo**
        ii. Frontend IP: **Public**
        iii. Protocol: **HTTP**
        iv. Leave the rest as defaults
    d. Backend targets:
        i. Target type: **Backend pool**
        ii. Backend target: **be-pool-demo**
        iii. HTTP Settings:
            1. Add new
            2. HTTP Settings name: **HTTP-rule**
            3. Leave the rest as default



            4. Select **Add**

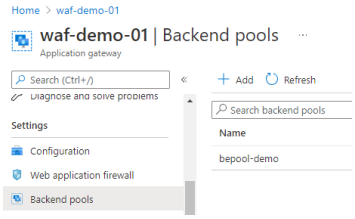
e. **Select Add**



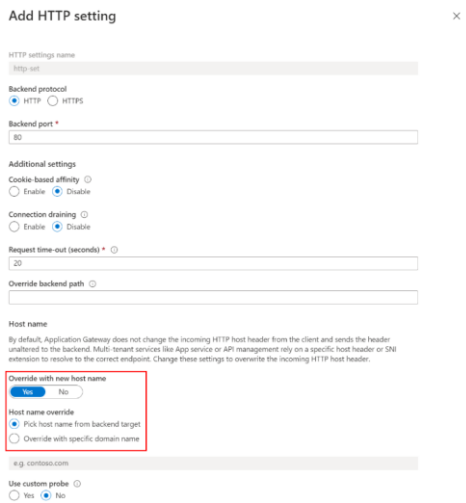
8. **Tags tab:** Add the tags relevant to your organization convention
9. Select **Review + Create.** Then, select **Create.**
   Note: The deployment of the AGW may take 20-25 mins.

Add backend pool

1. Once the deployment is done, click on **Backend pools.** Select **be-pool-demo.**



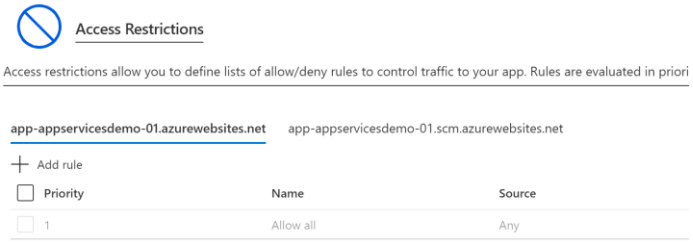
2. Under backend targets:
   a. Target types: **App Services**
   b. Target: Your app service
   c. Save.
3. Select **HTTP Settings** for your AGW. Select the existing HTTP setting.
   a. Under **Override with new host name,** select **Yes.**
   b. Under **Host name override,** select **Pick host name from backend target.**



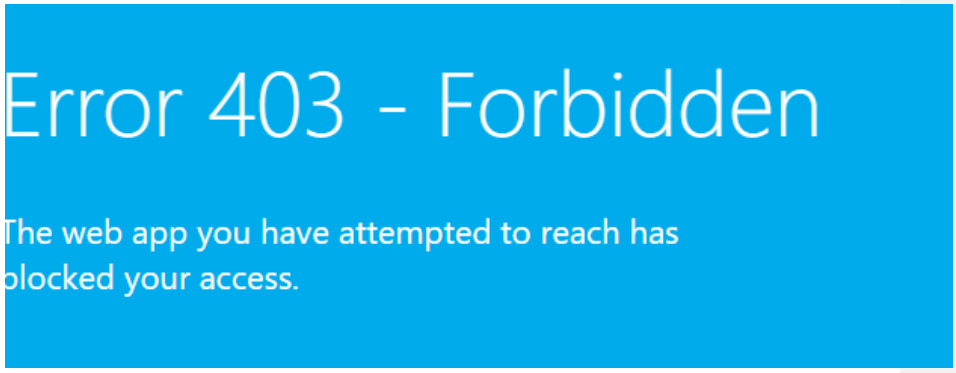
4. Once the deployment of the backend target is done, visit the frontend public IP address to see the app service deployed to it.

Set service endpoint-based rule

1. On the Azure Portal, navigate to the app service.
2. On the left pane, select **Networking**.
3. On the networking pane, select **Access Restrictions** under **Inbound traffic.**
4. Select **Add rule**



5. Enter the following details:
    a. Name: agw-service-endpoint
    b. Action: Allow
    c. Priority: 400
    d. Type: **Virtual Network**
    e. Select the virtual network and the agw's default subnet.
    f. Leave the rest as default
6. Note that if you try to now access the web app, you'll receive Error 403.

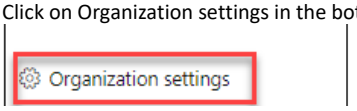


Now, you can only access the web app through the AGW.

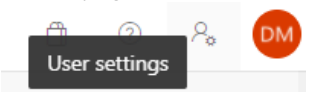
Lab 202: UI Testing with Selenium

Generate PAT for ADO

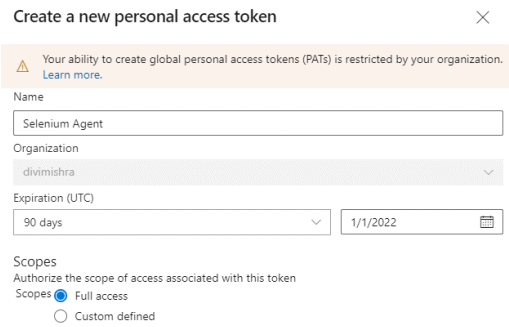
1. Navigate to your Azure DevOps homepage
2. Click on Organization settings in the bottom left.

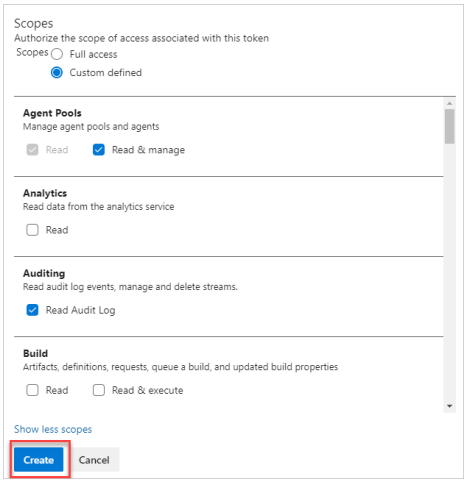


3. At the top right, click on **User Settings**



4. Click on **Personal Access Tokens**
    a. Name: **Selenium Agent**
    b. Expiration as desired
    c. Scopes:
        i. **Full access**
        ii. **(OR)** Agent pools: Read & Manage; Auditing: Read Audit Log





5. Copy the PAT and store it somewhere safe as you won't be able to see it again. You will need this for the upcoming lab.

Configure the VM agent

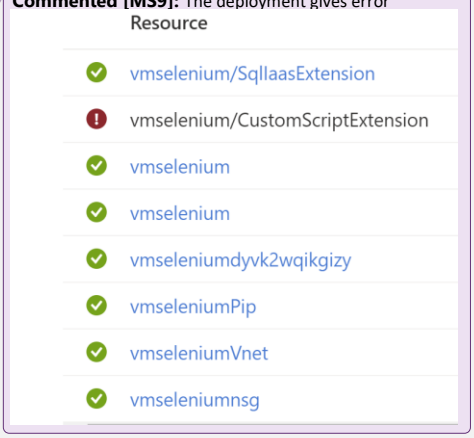
1. Click on the Deploy to Azure button below to provision a Windows Server 2016 VM along with SQL Express 2017 and browsers (Chrome and Firefox). Deploy this on the same Azure RG as you used for your web app in the previous labs.

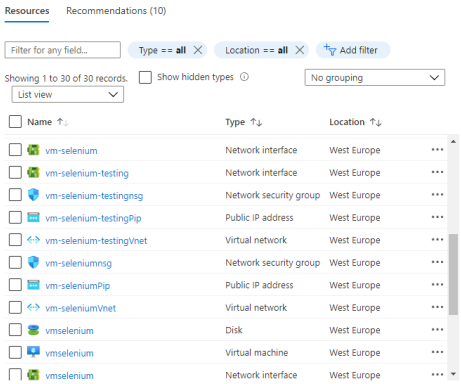
   DEPLOY TO AZURE

   This will take about 20 mins to deploy.

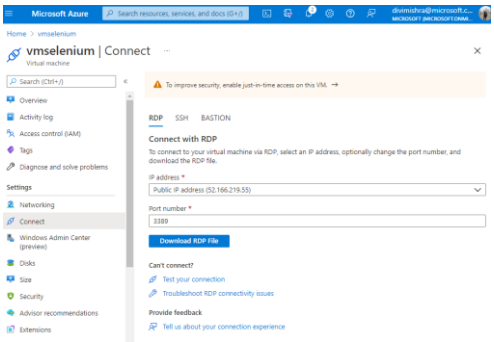
2. Navigate to your Azure Portal and open the resource group. You will notice the resources deployed:

Commented [MS9]: The deployment gives error



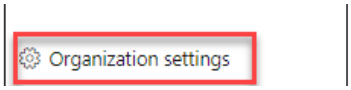


3. Select **vmselenium** virtual machine.
4. Download the RDP File for connecting to the VM.

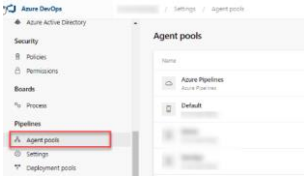


5. Connect with the following credentials:
    a. Username: vmadmin
    b. Password: P2ssw0rd@123

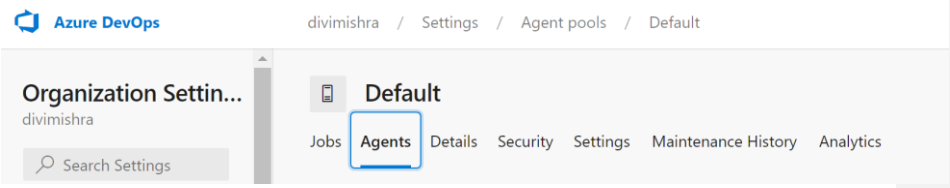
6. In the VM, open any web browser and sign in to your Azure DevOps organization.
7. Click on **Organization Settings** at the bottom left.



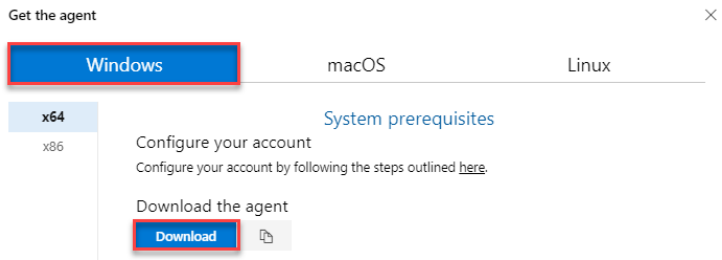
8. Select **Agent pools** under Pipelines



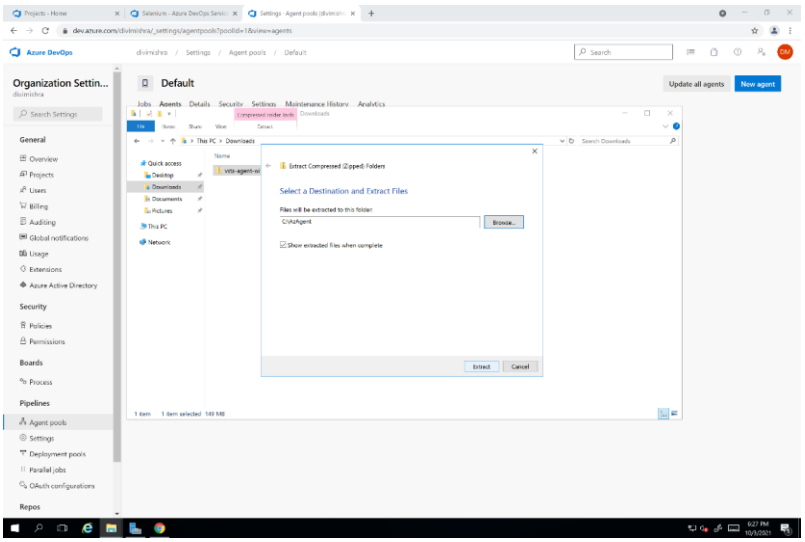
9. Select **Default** pool and select **Agents tab.** Click on **New agent.**

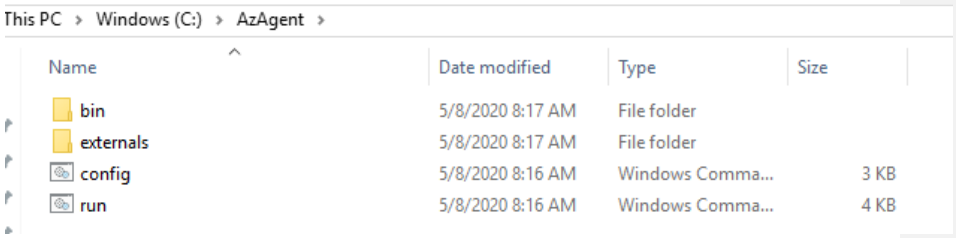


10. Under **Windows,** click on **Download.**



11. Make a directory in the C Drive named **AzAgent.**
12. Go back to your **Downloads** directory and extract the zipped folder from Step 10 on **AzAgent.**





13. Open **Powershell in administrator mode.** Change to the **C:\AzAgent** path and type **.\config.cmd**
14. Press Enter.
15. Provide the following details:
    a. **Enter server URL**: Your Azure DevOps Organization URL

b. **Authentication type**: Press the **enter** key for **PAT** as the authentication type and paste the PAT you had configured in the next prompt.

c. Let us use the default options for the rest of the configuration. Press **Enter** for all prompts until the command execution completes.

16. Once the agent is registered, type .**run.cmd** and hit **Enter** to start the agent.

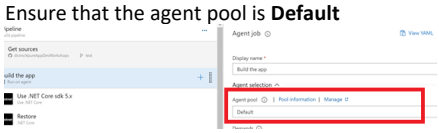
```
PS C:\AzAgent> .\config.cmd

        agent v2.166.4                    (commit efdfb40)

>> Connect:

Enter server URL > https://dev.azure.com/[redacted]
Enter authentication type (press enter for PAT) >
Enter personal access token > **************************************************
Connecting to server ...

>> Register Agent:

Enter agent pool (press enter for default) >
Enter agent name (press enter for selenium) > [redacted]
Scanning for tool capabilities.
Connecting to the server.
Successfully added the agent
Testing agent connection.
Enter work folder (press enter for _work) >
2020-05-08 08:24:14Z: Settings Saved.
Enter run agent as service? (Y/N) (press enter for N) >
Enter configure autologon and run agent on startup? (Y/N) (press enter for N) >
PS C:\AzAgent> .\run.cmd
Scanning for tool capabilities.
Connecting to the server.
2020-05-08 08:24:27Z: Listening for Jobs
```

Configure the pipeline

1. Go to the pipeline on Azure DevOps.
2. For **all agent jobs**, under Agent Selection, select the **Agent Pool** as **Default.**
3. Under **Variables,** add the following variables:

SITE_URL	https://<your-app-service-name>-01.azurewebsites.net
CHROMEWEBDRIVER	C:\SeleniumWebDrivers\ChromeDriver
EDGEWEBDRIVER	C:\SeleniumWebDrivers\EdgeDriver
GECKOWEBDRIVER	C:\SeleniumWebDrivers\GeckoDriver

4. Add another agent job.
   a. Name it **UI Testing**
   b. Ensure that the agent pool is **Default**



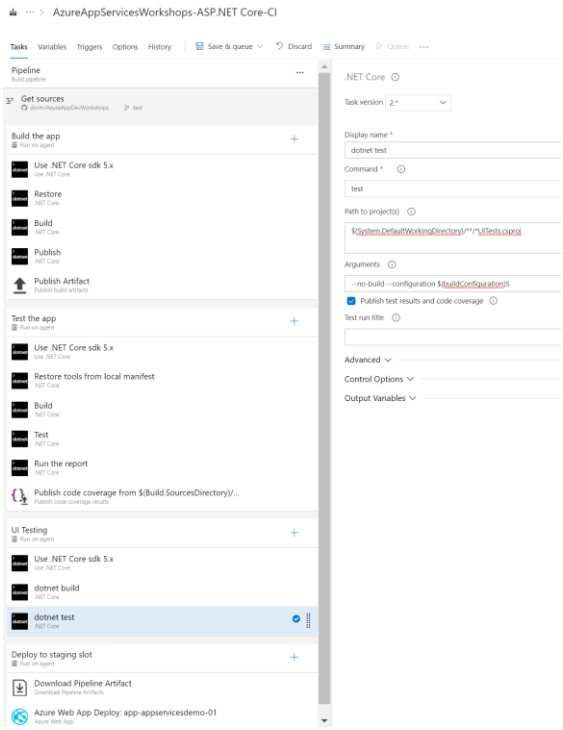
c. Add the following tasks:

   i. **Use .NET Core**
      1. Specify the SDK as 5.x

   ii. **Dotnet Build**
      1. Path to project: $**(System.DefaultWorkingDirectory)/**/*UITests.csproj**

   iii. **Dotnet Test**
      1. Path to project: **$(System.DefaultWorkingDirectory)/**/*UITests.csproj**
      2. Arguments: **--no-build --configuration $(buildConfiguration)**
      3. **Select** "Publish test results and code coverage"

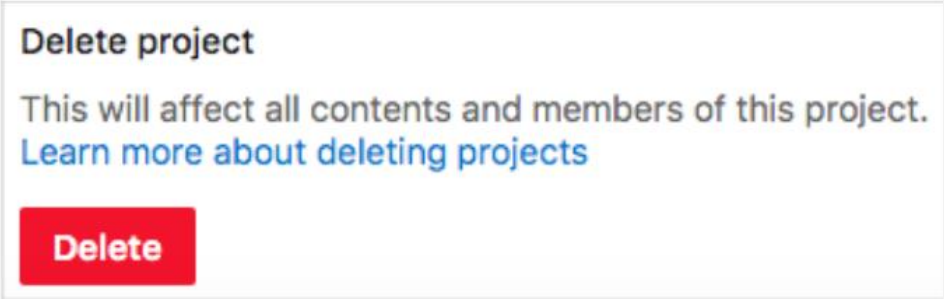
5. Ensure that this is what your pipeline looks like



6. Select **Save & Queue**
7. On the connected VM, see the UI tests executing. Note: It takes around 10 mins the first time you run it.
   In this lab, 4 UI tests are run on Chrome, Edge and Gecko.

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 resource group you have been working on.

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
- 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)