

Web App DevOps Lab

This lab will step through the key elements in setting up a DevOps pipeline for Azure using the DevOps tools for Azure - Visual Studio Team Services (VSTS).

Background information

[What is DevOps?](#)

[How Microsoft does DevOps](#)

Overall flow

- Preparing for the lab
- Creating the project
- Continuous Integration
- Create an Azure Web App
- Continuous Deployment
- Infrastructure as Code
- Automated Testing with Selenium
- Monitoring with Application Insights

Preparing for the lab

For this lab you will require:

- A Visual Studio Team Services (VSTS) account (free)
- An Azure subscription (your own or a free trial)
- Visual Studio 2017 installed (optional)

Use the same account (login/email and password) for both VSTS and Visual Studio.

VSTS supports any app and doesn't require the use of Visual Studio, .NET or other Microsoft languages. At the bottom of this page there are links to labs that work through implementing DevOps with Node, Java, Eclipse, IntelliJ and Docker.

If you are a student you can create an Azure Student account by completing the forms at these links:

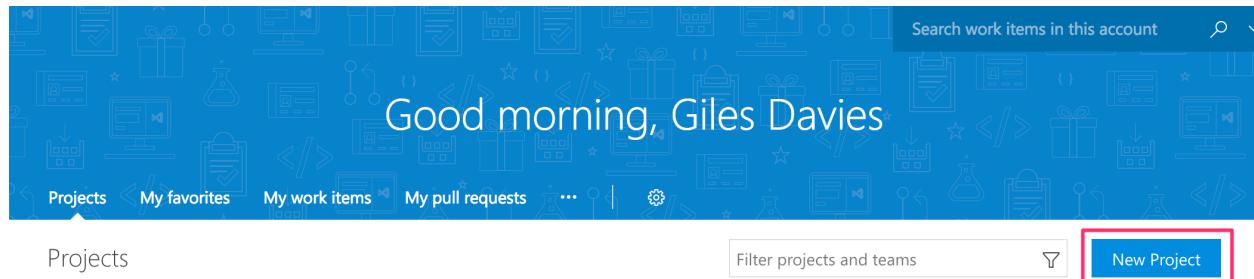
- [Register for Imagine](#)
- [Sign up for Azure for students](#)

If you don't have one, create a [VSTS account](#). Guidance on creating an account

Lab 1: Creating the project

Task 1: Create the VSTS Team Project

1. Open a browser and navigate to your VSTS account
<https://youraccountname.visualstudio.com>.
2. In your VSTS account select New Project.



3. Give the project a name, e.g. Web App. Make sure that version control is set to Git and click Create. You could equally choose to use TFVC but this lab has documented the steps for using Git.



Create new project

Projects contain your source code, work items, automated builds and more.

Project name *

 ✓

Description

Version control



Work item process

 ?CreateCancel

4. Your VSTS team project has been created. You can add other people to the team if you want.

The screenshot shows the VSTS interface for a 'Web App' project. At the top, there's a navigation bar with links like 'Web App', 'Dashboards', 'Code', 'Work', etc. Below the navigation is a search bar for work items. On the left, there's a sidebar with a 'Members (1)' section where a user icon has a red box around it. The main content area has a heading 'Get started with your new project!' and a 'Clone to your computer' section. It shows cloning options via HTTPS or SSH, and a 'Clone in Visual Studio' button which is also highlighted with a red box. Below these are several cloning methods listed as bullet points. To the right of the cloning section is a 'Activity' summary for the last 7 days.

You now have a VSTS team project. The next step is to create a web application.

Task 2: Creating the Visual Studio Web Application

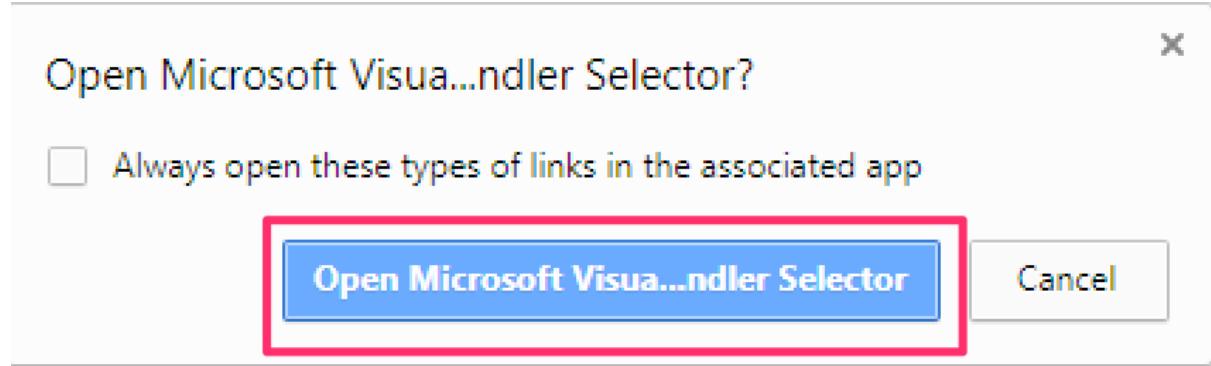
1. To clone the (empty) Git repository into Visual Studio there are two main options - cloning from within the browser or from within Visual Studio.

Cloning from within the browser.

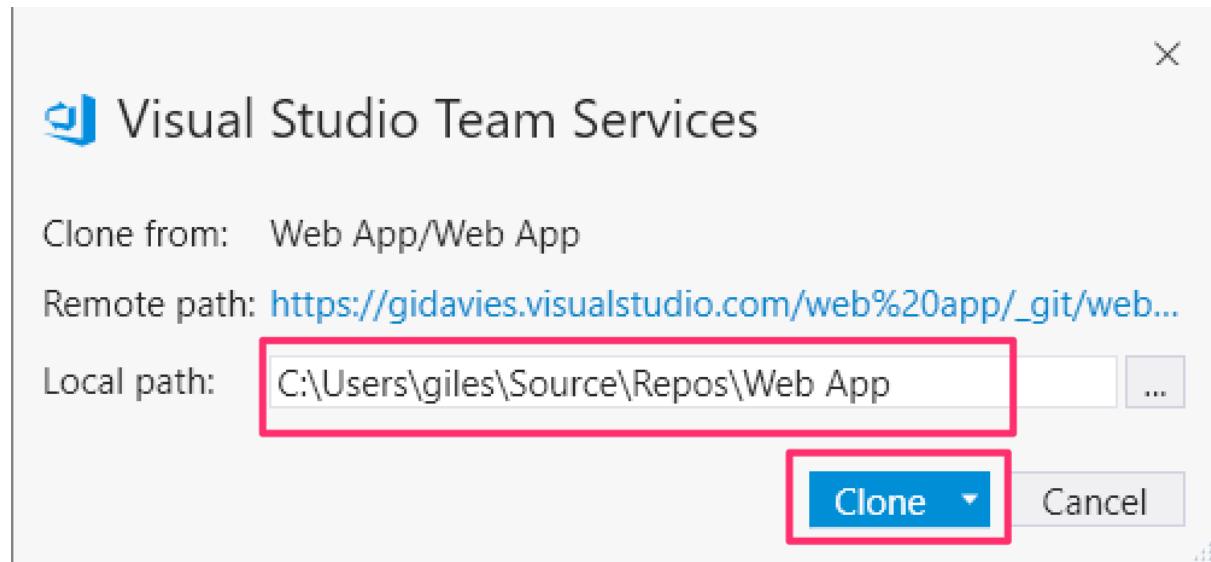
1. Click on the Clone in Visual Studio button.

This screenshot is identical to the one above, showing the 'Clone in Visual Studio' button highlighted with a red box. The rest of the interface, including the sidebar and activity summary, remains the same.

2. You may be prompted to confirm that you want to open Visual Studio (exact look and feel will depend on the browser you're using). If so, agree to open Visual Studio.



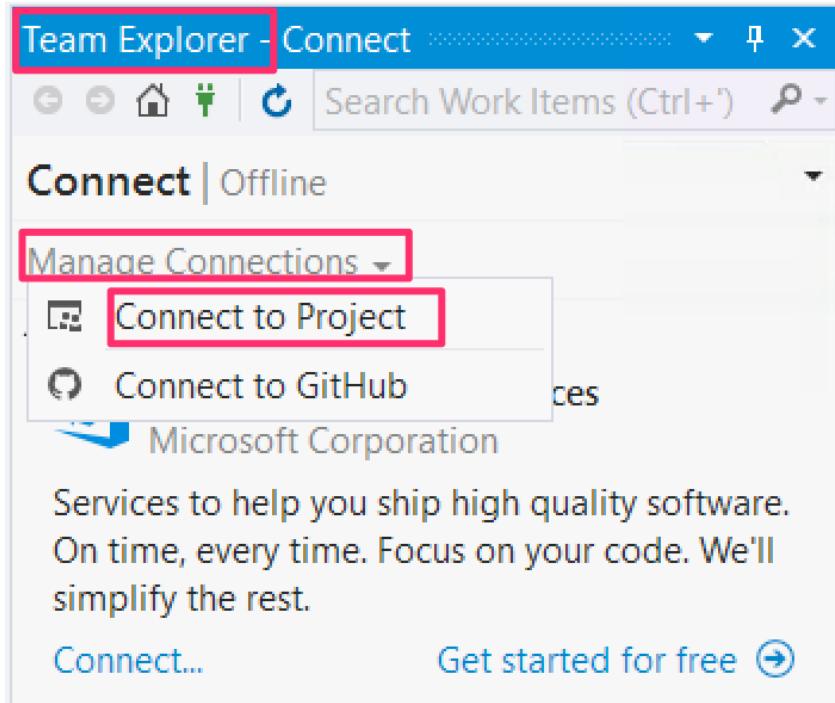
3. Visual Studio will then prompt to clone the VSTS remote Git repo to your local machine. Change the local path as required and then click Clone.



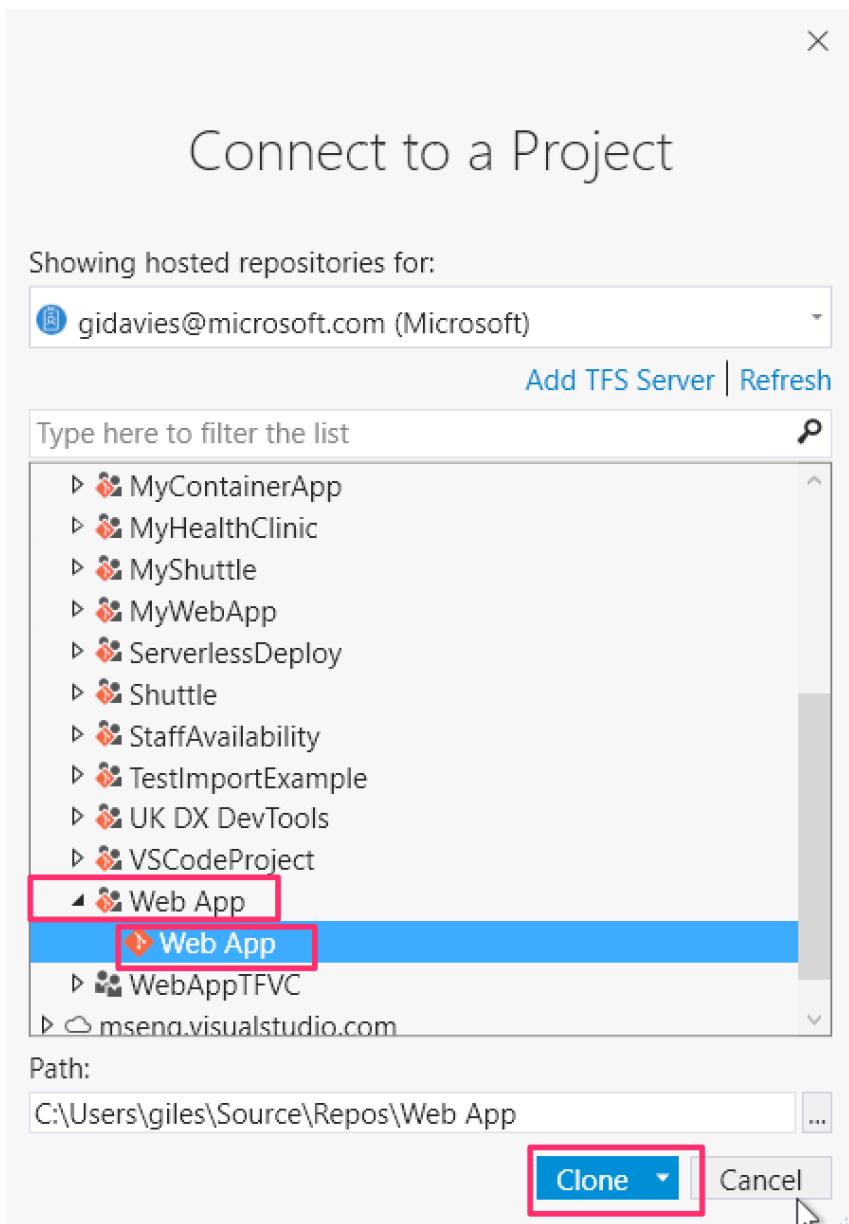
If this doesn't work then follow the Cloning from within Visual Studio steps.

Cloning from within Visual Studio

1. Open the Team Explorer view (bottom left hand side next to the Solution Explorer). Then select Manage Connections and Connect to Project.



2. Select the Web App project (you may only have one project listed) expand it to show the Web App repository and then click Clone.



Signing in to Visual Studio

If you haven't used Visual Studio on the machine before you will be prompted to sign in when Visual Studio starts. Sign in with the account that you are using for VSTS.

Visual Studio

Welcome!

Connect to all your developer services.

Sign in to start using your Azure credits, publish code to a private Git repository, sync your settings, and unlock the IDE.

[Learn more](#)

[Sign in](#)

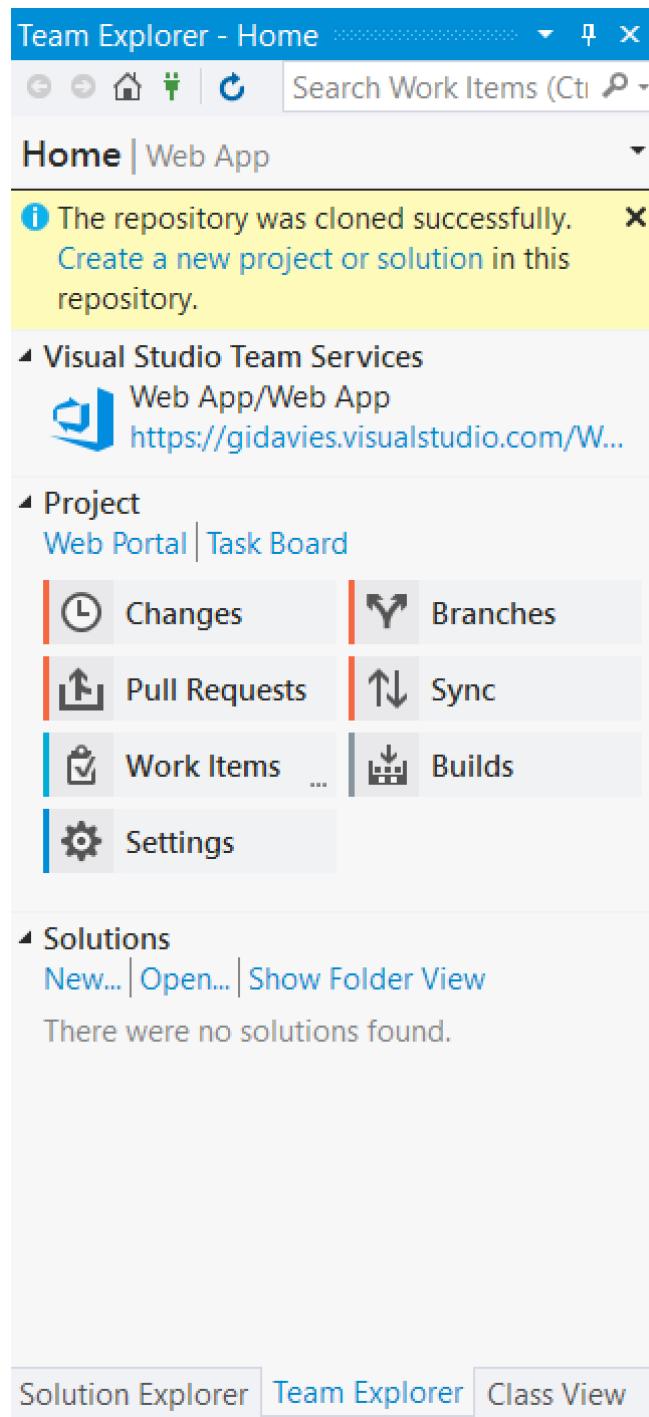
Don't have an account? [Sign up](#)

[Not now, maybe later.](#)

x

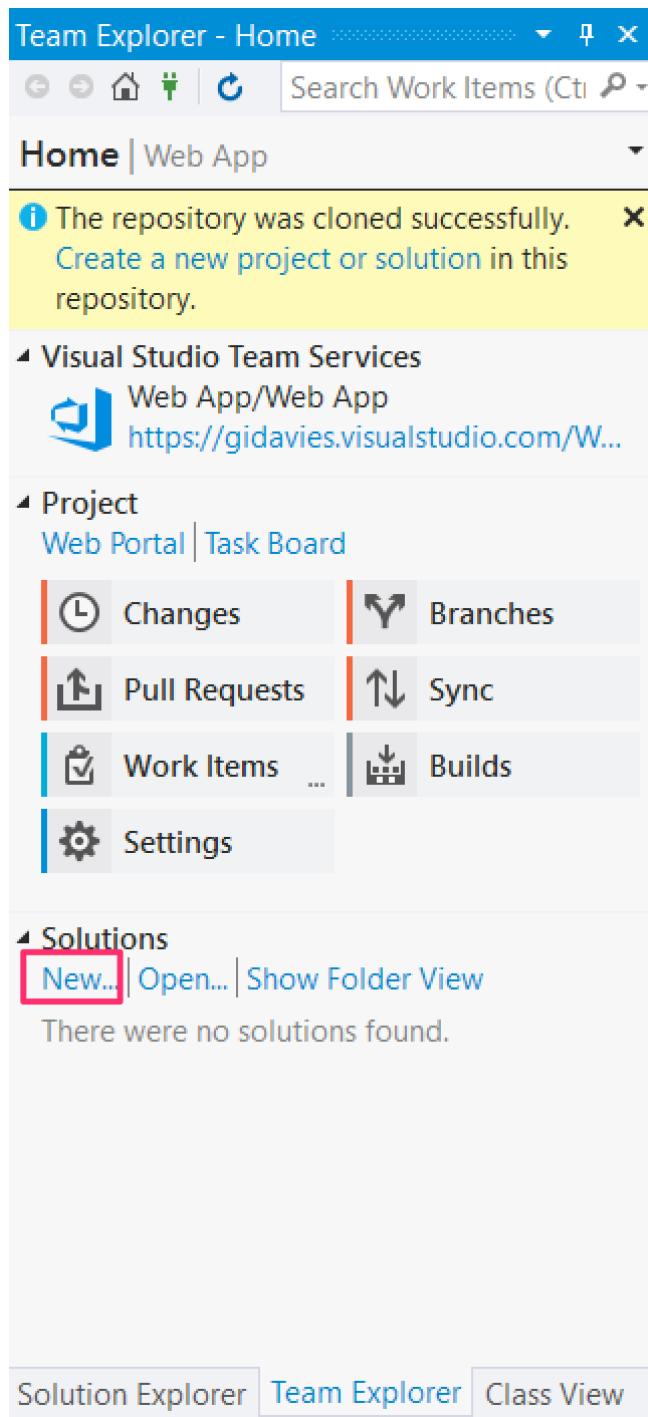


2. If you now open the Team Explorer (bottom right hand corner) you will see that the

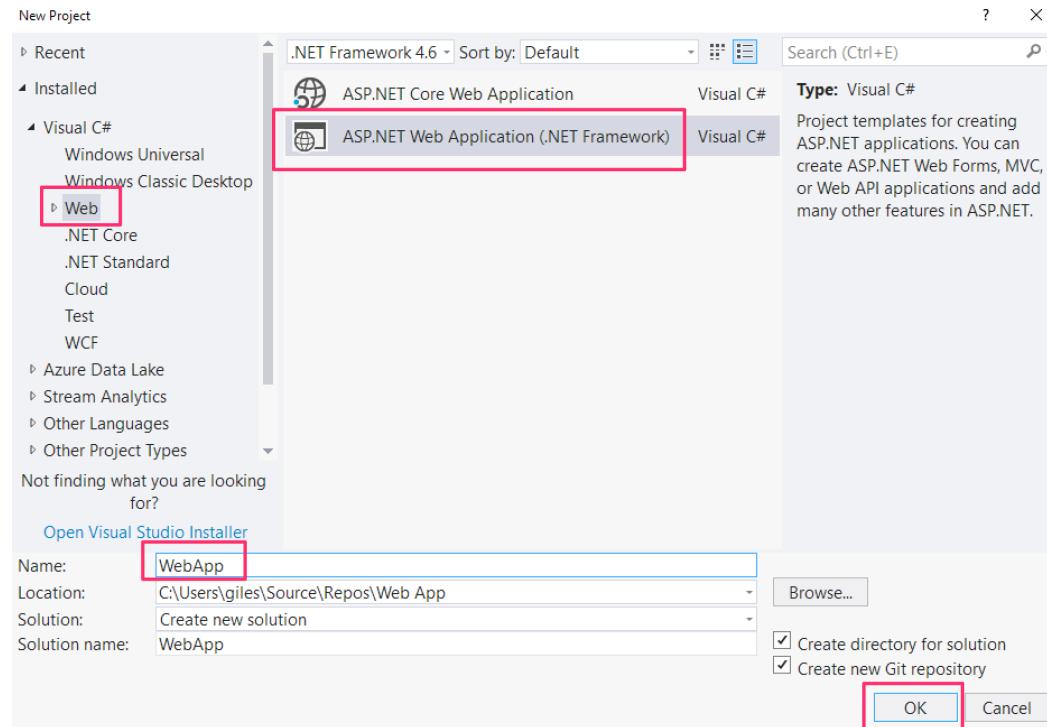


repository has been cloned successfully.

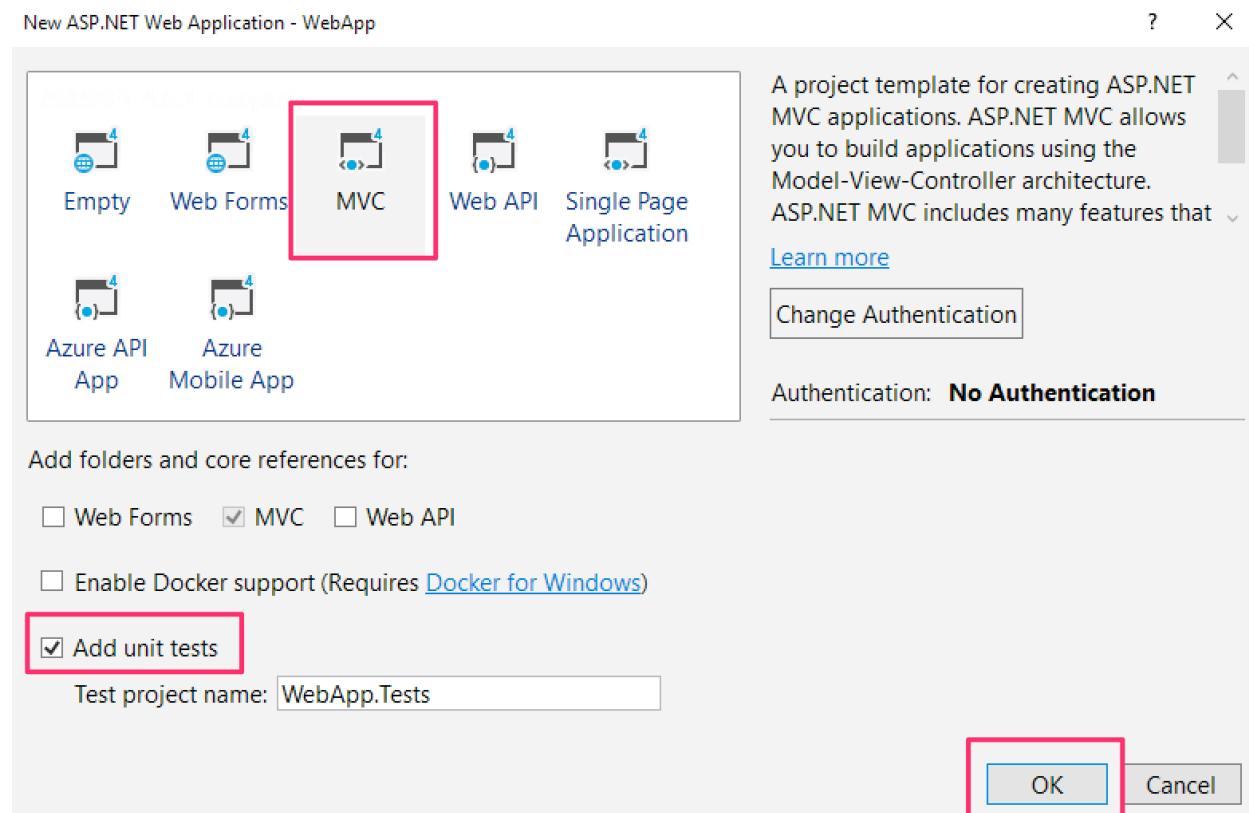
3. Select New in the Solutions area of Team Explorer.



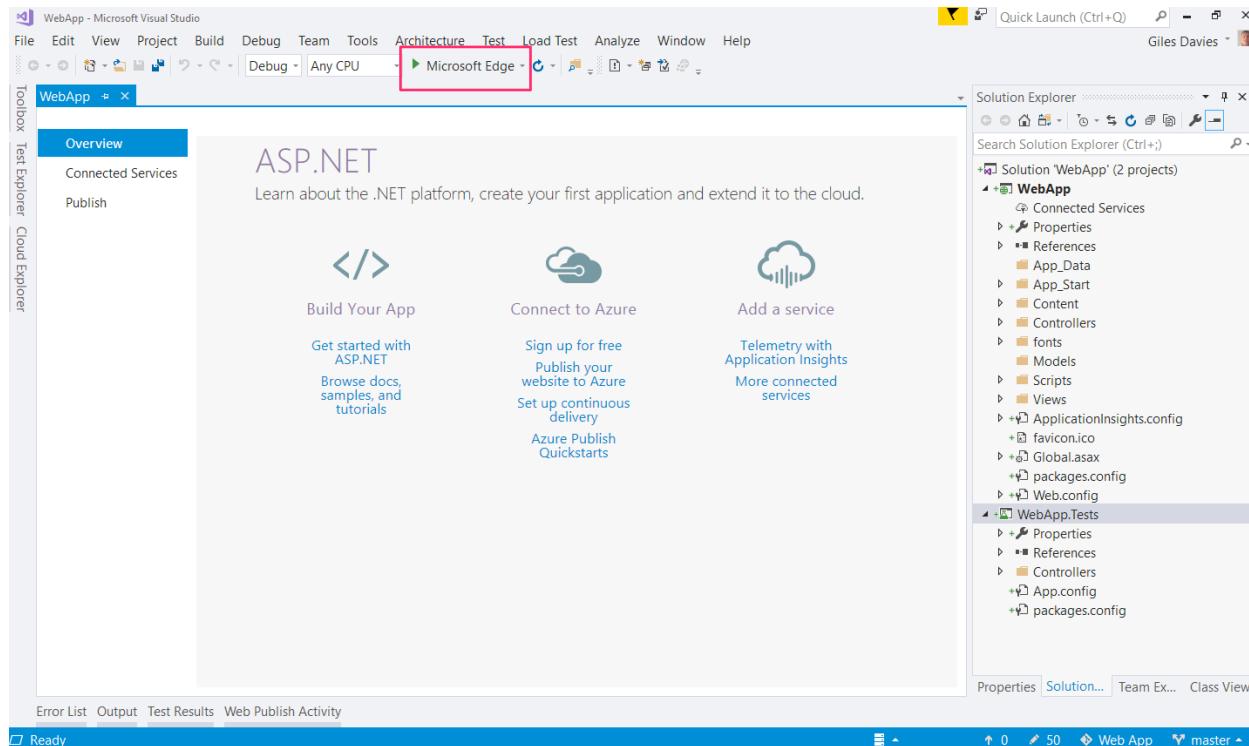
4. Select Web | ASP.NET Web Application, change the name as required and click OK.



5. Ensure that MVC is selected, disable Enable Docker if checked and check Add unit tests. Click OK.



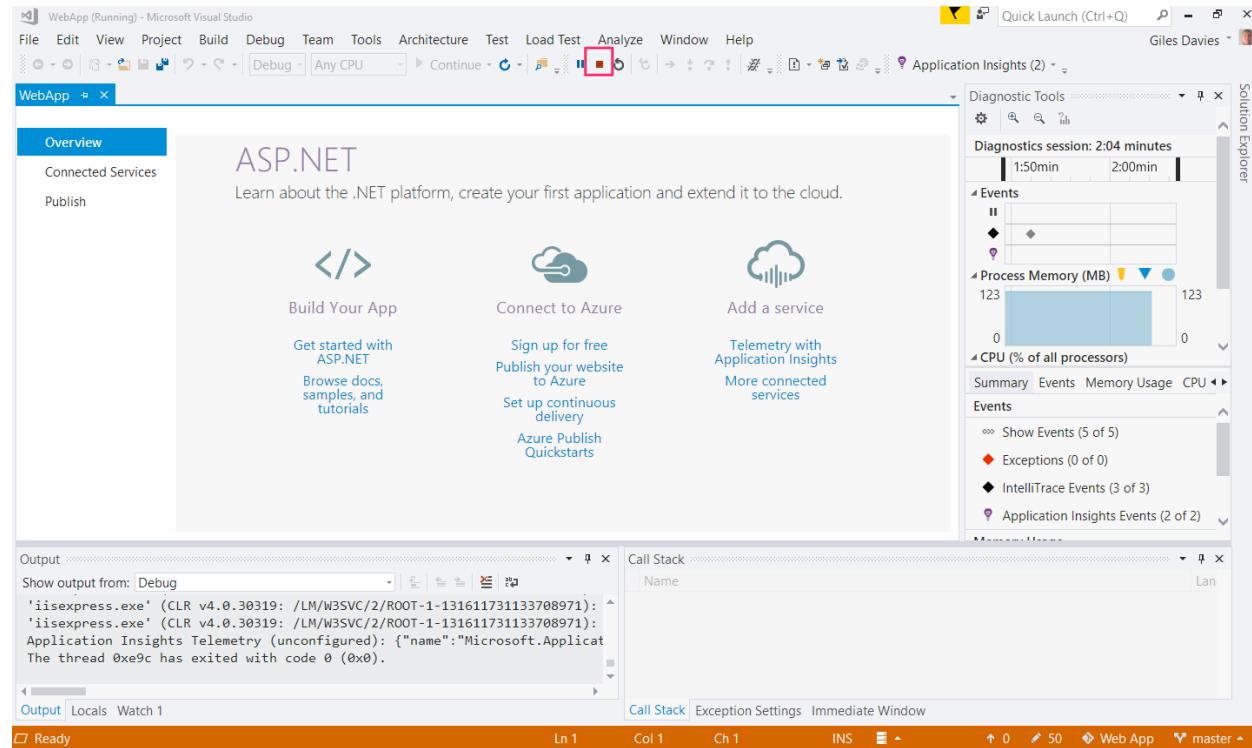
6. When the project has been created, click the run button (green arrow) or F5 to build and launch the web application locally.



7. The web application will launch locally. Take a quick look and then close the browser.

The screenshot shows a web browser window with the URL 'localhost:2767/' in the address bar. The page content is the 'ASP.NET' Getting Started page, featuring the title 'ASP.NET', a brief description, and a 'Learn more »' button. Below this, there are three main sections: 'Getting started', 'Get more libraries', and 'Web Hosting', each with a brief description and a 'Learn more »' button. At the bottom of the page, there is a copyright notice: '© 2018 - My ASP.NET Application'.

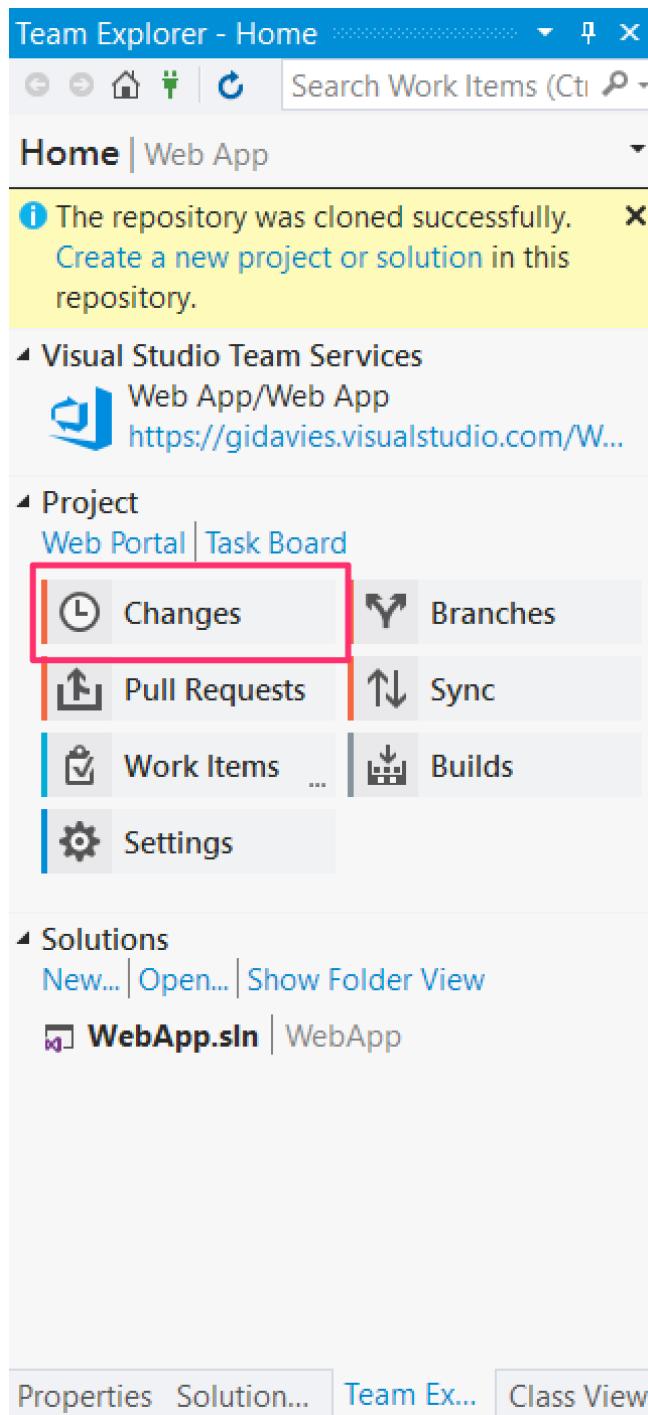
8. To stop the debug session click the stop button or Shift + F5 in Visual Studio.



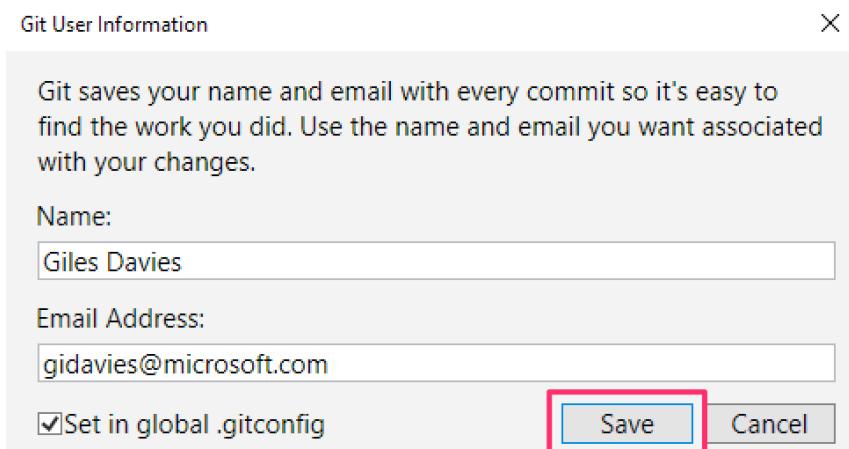
You now have a web application. The next step is to add the application to Git so that it is under source control.

Task 3: Committing the new Web App to source control

1. In the Visual Studio Team Explorer, select Changes.

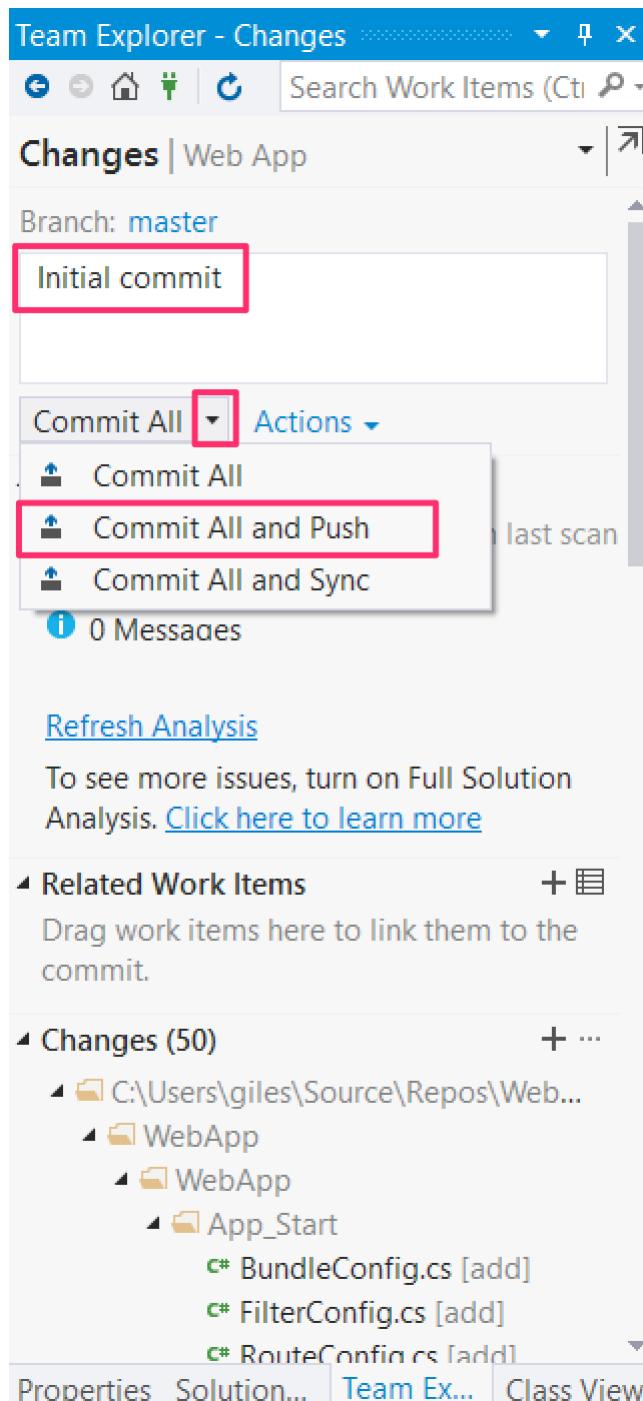


2. You may be prompted to enter or confirm your Git User Information. Just click Save and you



won't be prompted again.

3. Add a commit comment, such as Initial commit, then select Commit All and Push.



4. When completed you can go back to VSTS in the browser, select Code and you will see your web application. Take a look at the history to see your initial commit.

The screenshot shows the VSTS interface with the 'Code' tab selected. On the left, there's a navigation tree for a 'Web App' project, with 'WebApp' expanded to show 'WebApp', 'WebApp.Tests', 'WebApp.sln', '.gitattributes', and '.gitignore'. The main area displays a commit history for the 'master' branch. The first three commits are listed:

Name	Last change	Commits
WebApp	just now	4c6ce905 Initial commit Giles Davies
.gitattributes	just now	4c6ce905 Initial commit Giles Davies
.gitignore	just now	4c6ce905 Initial commit Giles Davies

A 'Set up build' button is visible in the top right corner.

You now have a web application, committed to source control in VSTS. The next step is to add Continuous Integration to the VSTS project.

Optional: Create a new Dashboard in VSTS:

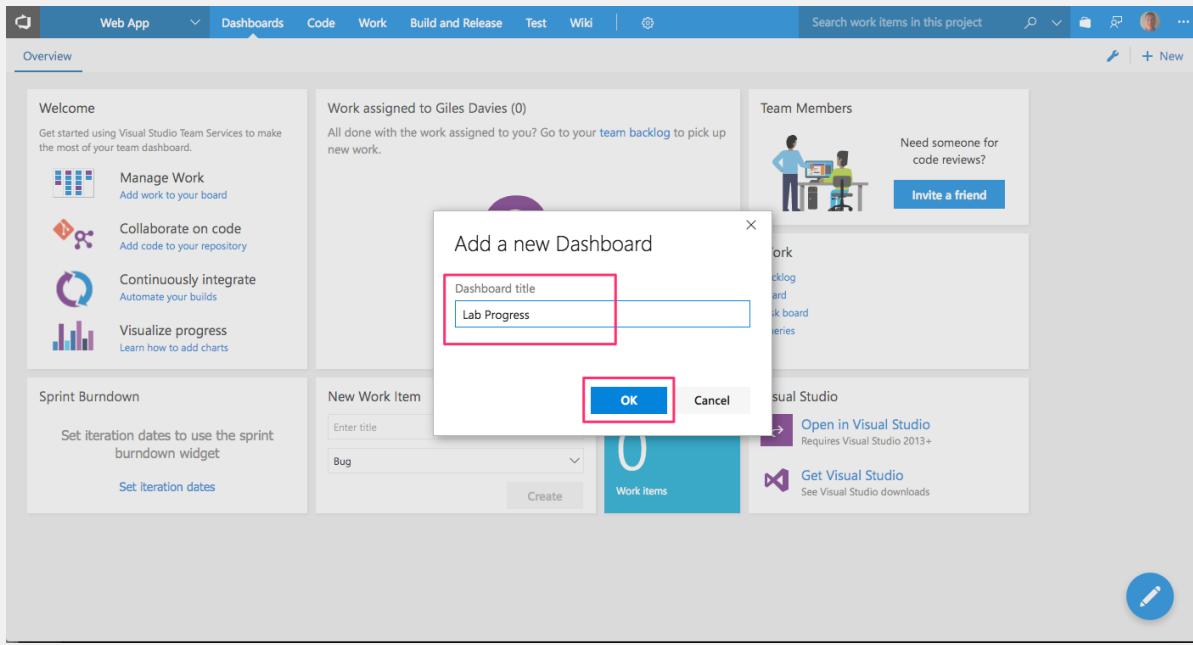
- In your VSTS project, select Dashboard and then New:

The screenshot shows the 'Dashboards' screen in VSTS. A red box highlights the 'Dashboards' tab in the top navigation bar. A red box also highlights the '+ New' button in the top right corner. The dashboard itself is divided into several sections:

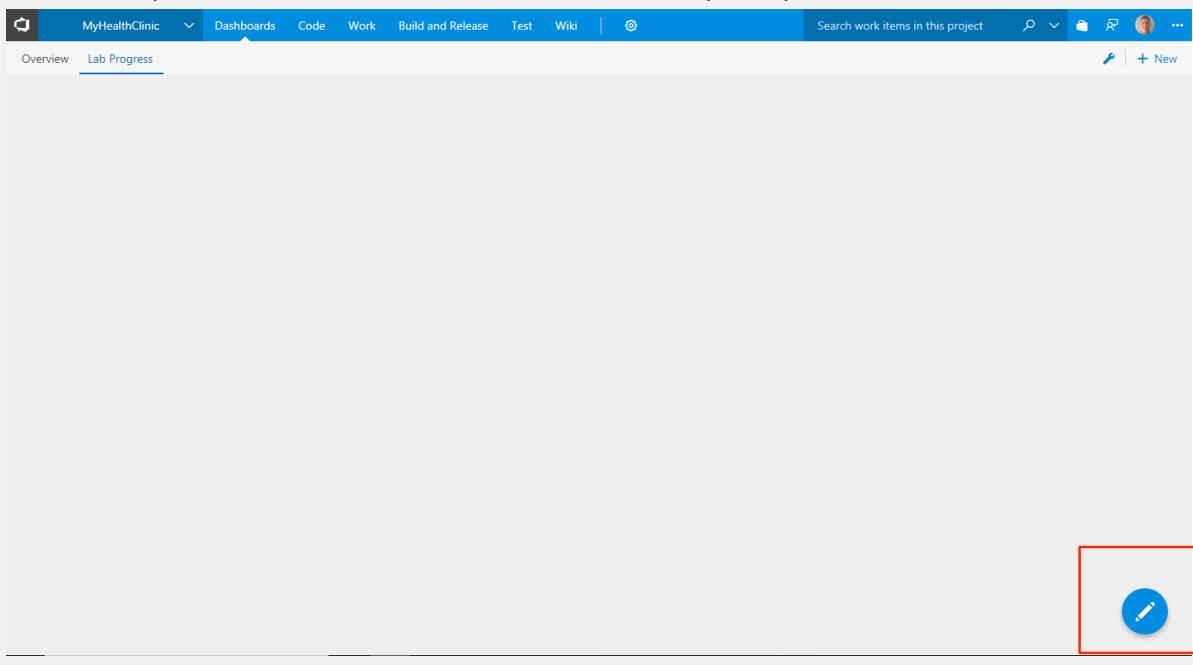
- Welcome:** Get started using Visual Studio Team Services to make the most of your team dashboard. Includes links to 'Manage Work', 'Collaborate on code', 'Continuously integrate', and 'Visualize progress'.
- Work assigned to Giles Davies (0):** All done with the work assigned to you? Go to your [team backlog](#) to pick up new work.
- Team Members:** Shows two team members and a link to 'Invite a friend'.
- Work:** Includes links to 'Backlog', 'Board', 'Task board', and 'Queries'.
- Sprint Burndown:** Set iteration dates to use the sprint burndown widget. Includes a link to 'Set iteration dates'.
- New Work Item:** A form to enter a title and type (set to 'Bug') with a 'Create' button.
- Work in Progress:** Shows 0 work items.
- Visual Studio:** Links to 'Open in Visual Studio' (Requires Visual Studio 2013+) and 'Get Visual Studio' (See Visual Studio downloads).

A large blue pencil icon is located in the bottom right corner of the dashboard area.

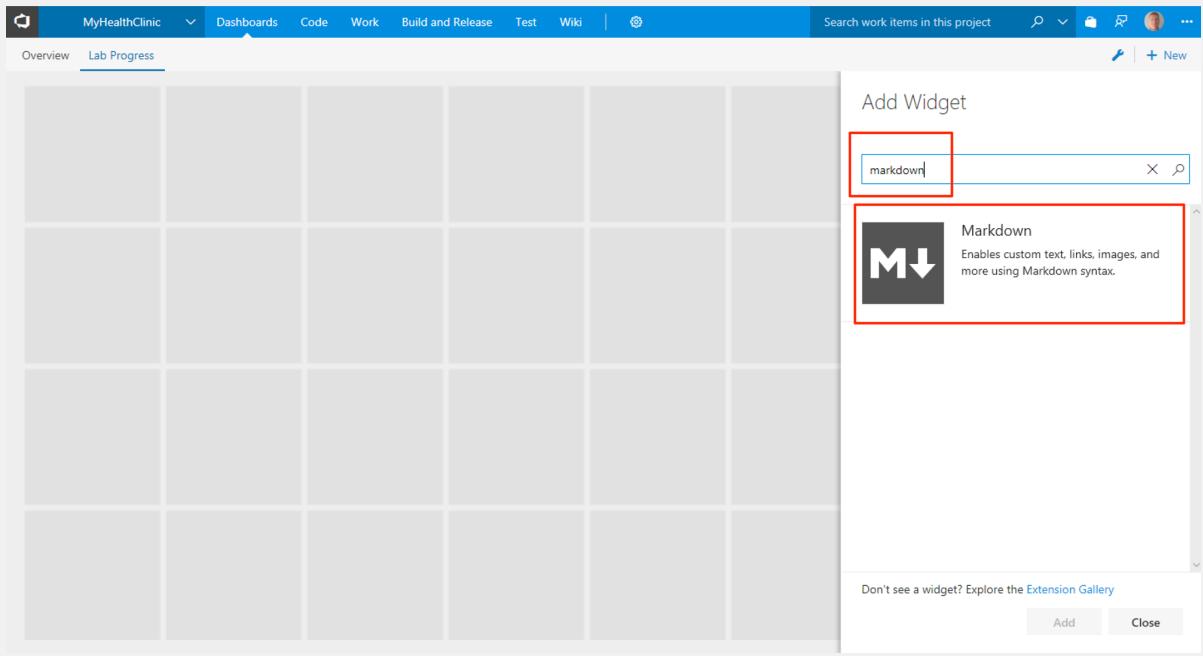
- Name the new Dashboard Lab Progress:



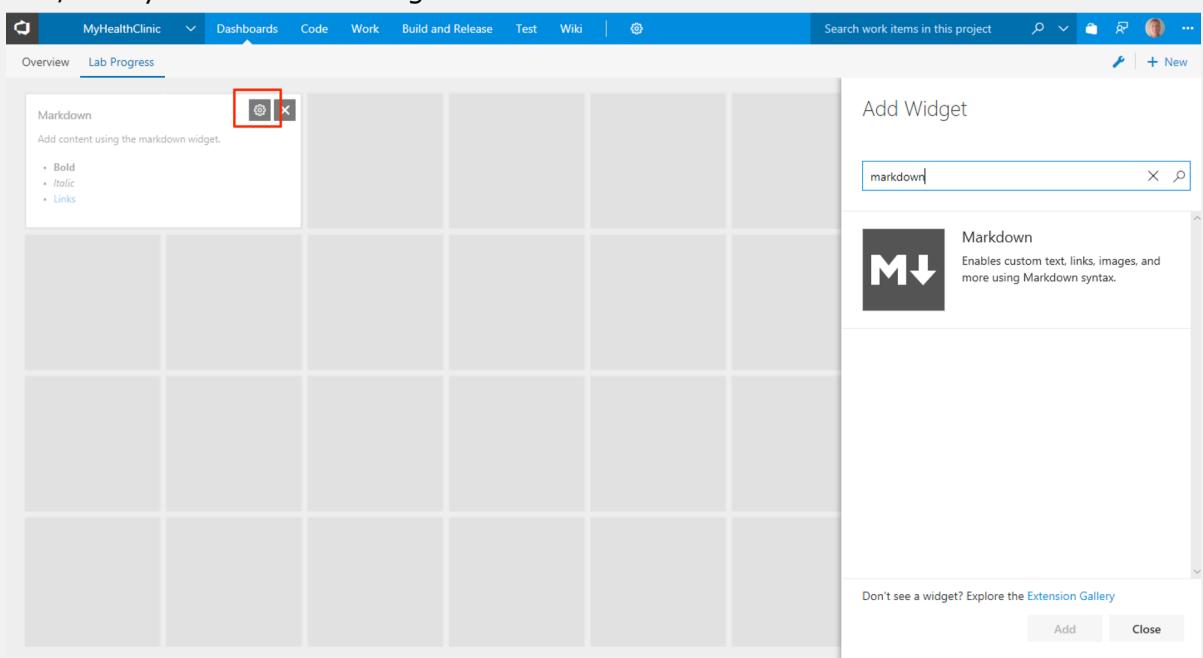
- Select the pencil icon in the bottom left, followed by the plus button:



- Search for and select the Markdown widget:



- Configure the Markdown widget and add the names of your team, project vision and/or any other info that might be useful:



- Save the changes, close the widget gallery and save the dashboard by clicking on the blue edit button in the bottom right hand corner.

Lab 2: Continuous Integration

Continuous Integration is a key DevOps practice to build, test and create the software to later deploy.

Task 1: Set up the Continuous Integration definition

1. Navigate to the VSTS project Code hub, and select Set up build.

The screenshot shows the VSTS Code hub interface. The top navigation bar has tabs for 'Web App', 'Dashboards', 'Code' (which is highlighted with a red box), 'Work', 'Build and Release', 'Test', and 'Wiki'. Below the navigation is a search bar 'Search work items in this project'. On the right side of the header are icons for 'Fork', 'Clone', and user profile. The main area shows a file tree for a 'Web App' project with files like 'WebApp', 'WebApp.Tests', 'WebApp.sln', '.gitattributes', and '.gitignore'. At the bottom right of the main area is a large blue button labeled 'Set up build' with a red box around it.

2. There are a range of build templates available, including non-Microsoft technologies but for this example select the ASP.NET template and click Apply.

The screenshot shows the VSTS Build templates page. The top navigation bar includes 'Builds', 'Releases', 'Packages', 'Library', 'Task Groups', and 'Deployment Groups'. The main content area has a large circular arrow icon and a section titled 'Choose a template'. Below it, a note says: 'Choose a template that builds your kind of app. Don't worry if it's not an exact match; you can add and customize the tasks later.' To the right, there is a search bar 'Search' and a section titled 'Select a template' with the sub-instruction 'Or start with an Empty process'. A 'Config as code' section shows a 'YAML' option. The 'Featured' section lists several templates: '.NET Desktop', 'ASP.NET' (which is highlighted with a red box), 'ASP.NET Core', 'ASP.NET Core (.NET Framework)', 'Azure Web App', and 'Universal Windows Platform'. An 'Apply' button is located at the bottom right of the 'ASP.NET' template card.

3. The template creates a build definition with a number of tasks added. Select the Process task which states that some settings need attention. You need to select the build agent where you want to run this build. You can choose to run the builds using an on-premise agent or use the agents hosted on Azure. We will use the Hosted VS2017 agent as it has the .NET framework and all other components that are required to build the app. Check that the agent

is set to Hosted 2017.

The screenshot shows the 'Build process' configuration for a 'Web App-ASP.NET-CI' pipeline. The 'Agent queue' dropdown is highlighted with a red box and shows 'Hosted VS2017' selected. Other options like 'Unlink all' and 'Manage' are visible. The pipeline consists of several tasks: 'Get sources' (highlighted with a red box), 'Use NuGet 4.3.0', 'NuGet restore', 'Build solution', 'Test Assemblies', 'Publish symbols path', and 'Publish Artifact'. The 'Get sources' task has a warning icon indicating 'Some settings need attention'.

4. The Get sources task will be showing red to indicate that something needs completing. Click on the Get sources task, and set the source to VSTS Git. This is telling the build that it will get the source code to build from Git hosted in VSTS, although there are many other options. Keep the default settings for VSTS Git.

The screenshot shows the 'Select a source' dialog in VSTS. The 'VSTS Git' option is highlighted with a red box. Other options shown are 'TFVC', 'GitHub', and 'External Git'.

5. The template restores any dependencies using NuGet, builds the solution, runs any unit tests and then publishes the output. This should be ready to use, so for now test the build by clicking Save & Queue.

The screenshot shows the 'Build process' configuration for a 'Web App-ASP.NET-CI' pipeline. The 'Save & queue' button is highlighted with a red box. The pipeline tasks are identical to the previous screenshot: 'Get sources', 'Use NuGet 4.3.0', 'NuGet restore', 'Build solution', 'Test Assemblies', 'Publish symbols path', and 'Publish Artifact'. The 'Get sources' task still has a warning icon.

6. The next window allows you to change some inputs into the build, but just click Save &

Save build definition and queue

Save comment

Agent queue

Hosted VS2017

Branch

master

Commit

Variables Demands

BuildConfiguration	release
BuildPlatform	any cpu
system.debug	false

+ Add

Save & queue Cancel

Queue.

7. You should now see that a build has been queued. Click on the build number to watch the build in progress.

Web App

Builds Releases Packages Library Task Groups Deployment Groups*

Web App-ASP.NET-CI

Build #20180123.1 has been queued.

Tasks Variables Triggers Options Retention History

Process Build process

Get sources Web App master

Phase 1 Run on agent

Name * Web App-ASP.NET-CI

Agent queue * Hosted VS2017

8. Observe the build progressing. It will typically take a few minutes.

Build Started

Phase 1 8
Running for 74 seconds (Hosted Agent)

```
D:\a\_tasks\VSSBuild_71a9a2d3-a98a-4caa-96ab-affca411ecda1.126.0\ps_modules\MSBuildHelpers\vswhere.exe" -version [15.0,16.0] -l "C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise\MSBuild\15.0\Bin\msbuild.exe" "d:\a\1\s\WebApp\WebApp.sln" /nologo /d:CentralLogger,D:\a\1\tasks\VSSBuild_71a9a2d3-a98a-4caa-96ab-affca411ecda1.126.0\ps_modules\MSBuildHelpers\Microsoft.TeamFoundation.DistributedTask.MSBuild.Logger.dll","RootDetailId=b5d98c1e18ccfc83ba|SolutionDir=d:\a\1\s\WebApp" /ForwardingLogger,"D:\a\1\tasks\VSSBuild_71a9a2d3-a98a-4caa-96ab-affca411ecda1.126.0\ps_modules\MSBuildHelpers\Microsoft.TeamFoundation.DistributedTask.MSBuild.Logger.dll" /p:DeployOnBuild=true /p:PackageAsSingleFile=true /p:SkipInvalidConfigurations=true /p:PackageLocation="d:\a\1\\" /p:platform="any cpu" /p:configuration="Release" /p:VisualStudioVersion="15.0" /p:MSDeployUserAgent="VSTS_59c82c86-a75a-4dad-96998dd9e93a_build_20_85" Building the projects in this solution one at a time. To enable parallel build, please add the "/m" switch. Build started 1/23/2018 10:27:20 AM. Project "d:\a\1\s\WebApp\WebApp.sln" on node 1 (default targets). ValidateSolutionConfiguration: Building solution configuration "release|any cpu". Project "d:\a\1\s\WebApp\WebApp.sln" (1) is building "d:\a\1\s\WebApp\WebApp\WebApp.csproj" (2) on node 1 (default targets). PrepareForBuild: Creating directory "bin".
```

9. When the build completes click on the build number to see the build log. The summary tab shows who made what changes and when, as well as unit test results.

Build succeeded

Build 20180123.1 8
Ran for 2.8 minutes (Hosted VS2017), completed 14 seconds ago

Total tests	Failed tests
3 (+3)	0 (+0)
Passed (3)	Failed (0)
New (0)	Existing (0)
Others (0)	

Pass percentage: 100% (+100%)

Run duration: 10s 797ms (+10s 797ms)

Code Coverage: No build code coverage data available.

Tags: Add tag...

10. Click on the artifacts tab and the Explore button to look at the output of the build.

Build succeeded

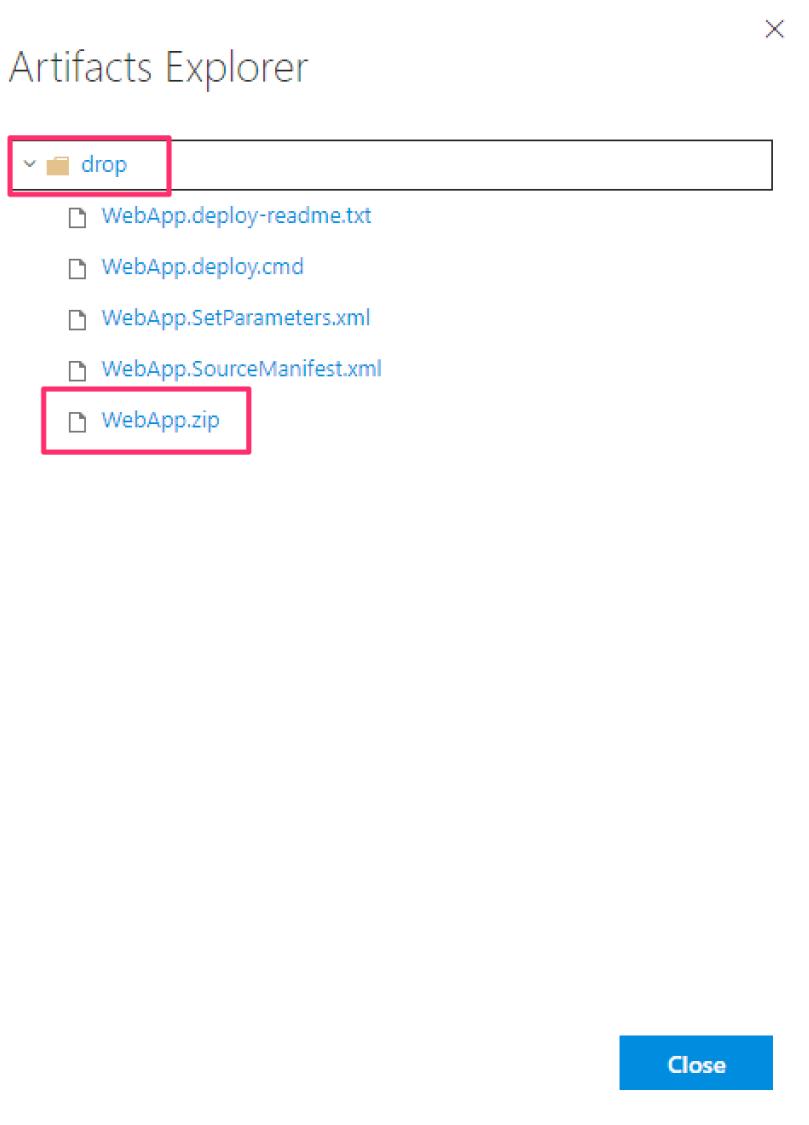
Build 20180123.1 8
Ran for 2.8 minutes (Hosted VS2017), completed 14 seconds ago

Artifacts

Name
drop

Download Explore

11. Expand the drop folder and notice that there is a zip file created from the build task in the build definition. This is the web application, packaged as a zip file, which is an easy way to



deploy to Azure. Click Close.

Close

You now have a working build definition for the web application. The next step is to set it up with a Continuous Integration trigger and test it.

Task 2: Enable Continuous Integration

1. Hover your mouse over the build definition and you will see three dots. Click on the dots and select Edit.

The screenshot shows the Microsoft DevOps interface under the 'Build and Release' tab. In the 'Build Definitions' section, a build named 'Web App-ASPNET-CI' is listed. A context menu is open over this build, with the 'Edit...' option highlighted by a red box.

2. Select Triggers and check Enable continuous integration. Save (but not queue).

The screenshot shows the 'Triggers' tab for the 'Web App-ASP.NET-CI' build definition. Under the 'Continuous integration' section, the 'Web App' trigger is selected and has 'Enabled' checked. The 'Enable continuous integration' checkbox is also checked and highlighted by a red box.

3. Test the Continuous Integration trigger by returning to Visual Studio and making a change. For example open the WebApp/Views/Home/Index.cshtml and make a change such as

changing the heading for the home page. Save the changes.

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Title Bar:** WebApp - Microsoft Visual Studio
- Menu Bar:** File, Edit, View, Project, Build, Debug, Team, Tools, Architecture, Test, Load Test, Analyze, Window, Help
- Toolbar:** Standard icons for file operations.
- Solution Explorer:** Shows the solution structure for 'WebApp' (2 projects). The 'Views' folder under 'WebApp' is highlighted with a red box, containing 'Home' (with 'About.cshtml', 'Contact.cshtml', and 'Index.cshtml'), 'Shared' (with 'ViewStart.cshtml'), and 'Web.config'. Other files like 'ApplicationInsights.config', 'favicon.ico', 'Global.asax', 'packages.config', 'Web.config', 'WebApp.Tests', and 'WebApp.Properties' are also listed.
- Toolbox:** Standard .NET development tools.
- Test Explorer:** Shows no results.
- Cloud Explorer:** Shows no results.
- Code Editor:** Displays the 'Index.cshtml' file content. The code includes a title, a jumbotron with a heading and a link to the ASP.NET website, and three main sections: 'Getting started', 'Get more libraries', and 'Web Hosting', each with its own descriptive paragraph and a link to the Microsoft website.
- Status Bar:** Shows '100 %' completion, 'Giles Davies, 1 hour ago | 1 author, 1 change', and navigation keys (Ln 6, Col 19, Ch 19, INS).
- Bottom Navigation:** Error List, Output, Test Results, Web Publish Activity.

4. In the Team Explorer, return to the Changes hub to see the files you've changed, and add a

The screenshot shows the 'Changes' hub in Team Explorer. A commit message 'Changed home page title' is highlighted with a red box. A context menu is open over this message, with the 'Commit All and Push' option highlighted and also boxed in red. Other options in the menu include 'Commit All' and 'Commit All and Sync'. Below the commit message, there's a section for 'Related Work Items' and 'Changes (1)'. The 'Changes (1)' section lists a file named 'Index.cshtml'.

comment and select Commit All and Push.

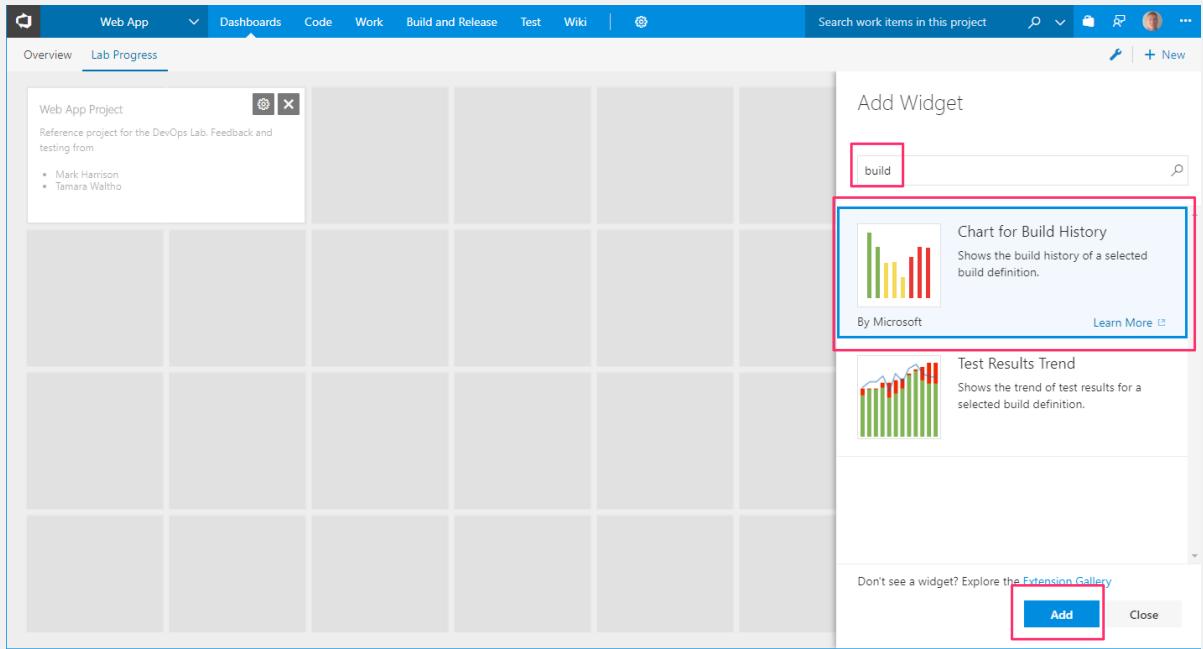
5. In VSTS, navigate to the Builds (Build & Release | Builds) and you should now see a build in progress. If you want to click on the build number to watch the build.

The screenshot shows the 'Builds' tab in the VSTS 'Build and Release' interface. A build named 'Web App-ASP.NET-CI' is listed with the status 'in progress'. The build was triggered by 'Giles Davies requested just now'. The 'Builds' tab is highlighted with a red box.

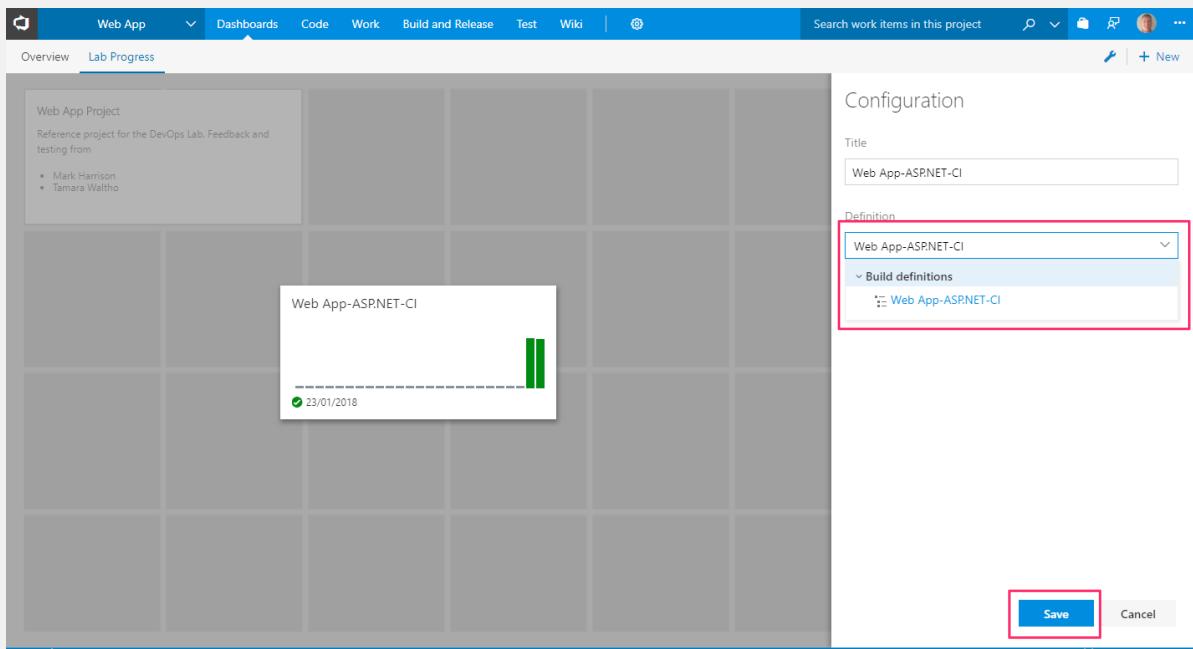
You now have a build triggered whenever you make a change to the code and push that change to Git in VSTS - Continuous Integration is in place for the project.

Optional: Add a Build History widget to the Lab Progress dashboard by:

- Searching for and adding the build history widget:



- Ensuring that it is configured to the build definition created in the preceding steps:



- Save the changes, close the widget gallery and save the dashboard by clicking on the blue edit button in the bottom right hand corner.

Lab 3: Create an Azure Web App

[Azure Web Apps](#) is an Azure service for hosting web applications. In this lab you'll create the Azure Web App into which you will later deploy the web application using Continuous Deployment.

1. In a browser go to the Azure Portal at <http://portal.azure.com>.

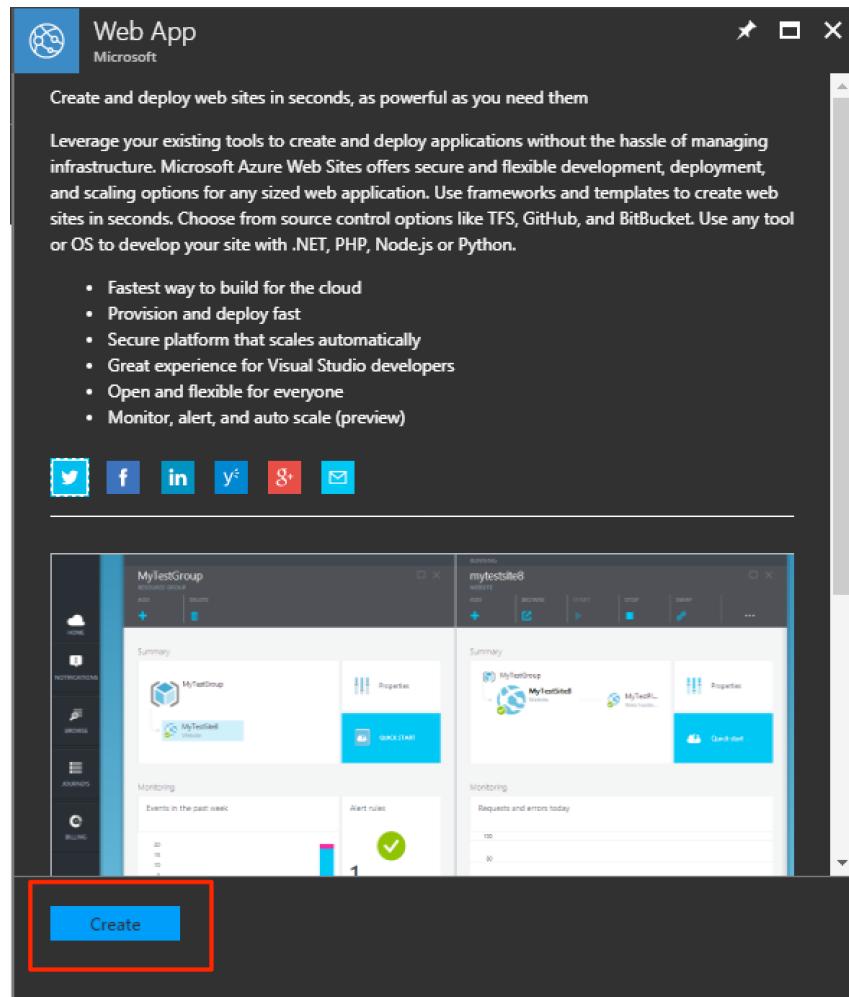
2. Select Create a resource and enter web app into the search field:

The screenshot shows the Microsoft Azure 'New' blade. On the left, there's a sidebar with a 'Create a resource' button highlighted by a red box. Below it are links for 'All services', 'FAVORITES' (Dashboard, Recent, DevOps Projects, Resource groups, App Services, Log Analytics), and a 'Popular' section with links like 'Get started', 'Recently created', 'Compute', 'Networking', 'Storage', 'Web + Mobile', and 'Containers'. In the center, there's a search bar with 'web app' typed into it, also highlighted by a red box. To the right, there's a 'Azure Marketplace' section with cards for 'Windows Server 2016 VM', 'Ubuntu Server 16.04 LTS VM', 'Web App', and 'SQL Database', each with a 'Quickstart tutorial' link.

3. Press enter and select Web App from the list:

The screenshot shows the Microsoft Azure Marketplace search results for 'web app'. The search bar at the top has 'web app' typed into it. The results table has columns for 'NAME', 'PUBLISHER', and 'CATEGORY'. There are two items listed: 'Web App' (Publisher: Microsoft, Category: Web + Mobile) and 'Web App Bot' (Publisher: Microsoft, Category: AI + Cognitive Services). The 'Web App' item is highlighted by a red box.

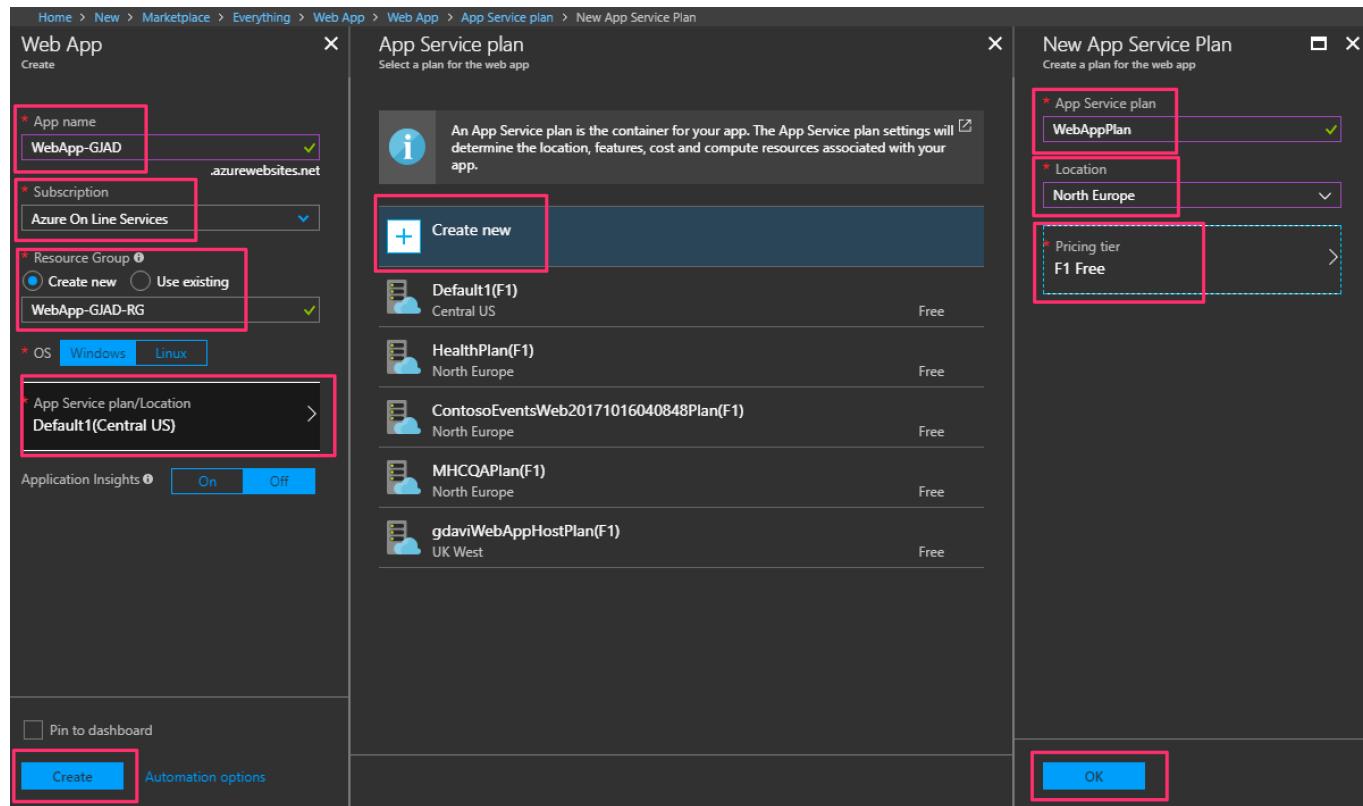
NAME	PUBLISHER	CATEGORY
Web App	Microsoft	Web + Mobile
Web App Bot	Microsoft	AI + Cognitive Services



4. Click Create:

5. Complete the highlighted fields as follows:

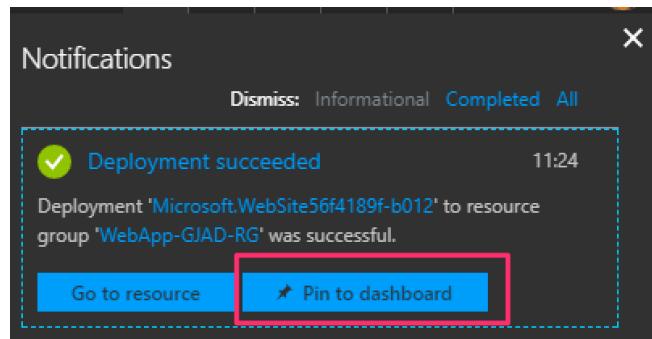
- App name: Choose a unique name that will be the URL for the web application such as WebApp plus your initials
- Subscription: If you have more than one subscription, ensure that you choose the correct one for this lab
- Resource Group: Create a new resource group for your web app
- OS: Leave this as the default of Windows.
- App Service Plan/Location: Click on this to create a new App Service Plan. Complete these fields:
 - App Service plan: Enter a name, such as WebAppPlan
 - Location: Select an Azure region close to you
 - Pricing tier: Click on this, and select the F1 Free tier



6. Click OK to save the App Service Plan.

7. Click Create to save and create the Web App.

8. After a short time (approx. 1-2 mins) you should see a notification that the Web App has been successfully created. You may want to pin the web app to your Azure dashboard for easy location later on:



9. Confirm that your Web App is created by selecting Go to resource.

10. Click on the URL:

The screenshot shows the Azure portal's 'WebApp-GJAD' app service page. On the left, there's a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Deployment, Quickstart, Deployment credentials, and Deployment slots. The main panel displays details about the app service, including its resource group ('WebApp-GJAD-RG'), status ('Running'), location ('North Europe'), subscription ('Azure On Line Services'), and OS name ('Windows Server 2016'). At the top right, there are buttons for Browse, Stop, Swap, Restart, Delete, Get publish profile, and Reset publish profile. A prominent red box highlights the 'URL' field, which contains the value 'https://webapp-gjad.azurewebsites.net'. Below the URL, there's a section for App Service plan/pricing tier, showing 'WebAppPlan (Free: 0 Small)'.

11. Your Web App should open in the browser and you will see something like this:

The screenshot shows a web browser displaying the Microsoft Azure landing page for a Web App Service. The page has a blue header with the Microsoft Azure logo. The main content area is blue and features the text 'Your App Service app is up and running' in large white font. Below this, it says 'Go to your app's [Quick Start](#) guide in the Azure portal to get started or read our [deployment documentation](#)'. A red box highlights the URL in the browser's address bar, which is 'https://webapp-gjad.azurewebsites.net'.

The exact page details will change over time but this now confirms that you have created a Web App in Azure. In the next lab we will deploy the web application into the newly created Azure Web App.

Optional: Add an Embedded Webpage widget to the Lab Progress dashboard by:

- Searching for and adding the Embedded Webpage widget:

The screenshot shows the Microsoft DevOps Lab interface. On the left, there's a navigation bar with 'Web App', 'Dashboards', 'Code', 'Work', 'Build and Release', 'Test', 'Wiki', and a search bar. Below the navigation bar, there are two cards: 'Web App Project' and 'Web App-ASP.NET-CI'. On the right, a modal window titled 'Add Widget' is open. Inside the modal, there's a search bar with the word 'web' typed into it. Below the search bar, there's a list of widgets. One widget, 'Embedded Webpage', is highlighted with a red box. This widget has a blue icon with '</>' symbols and the text 'Embed an external webpage on your dashboard within an iframe.' Below the widget, it says 'By Microsoft' and 'Learn More'. At the bottom of the modal, there's a 'Don't see a widget? Explore the Extension Gallery' link, an 'Add' button, and a 'Close' button.

- Add the URL for your web app created in the preceding steps:

The screenshot shows the 'Configuration' section of a dashboard. In the 'URL' field, the URL 'https://webapp-gjad.azurewebsites.net/' is entered. A red box highlights this URL field. In the bottom right corner, there is a blue 'Save' button and a grey 'Cancel' button.

- Save the changes, close the widget gallery and save the dashboard by clicking on the blue edit button in the bottom right hand corner.

Lab 4: Continuous Deployment

Continuous Deployment is another key practice within DevOps to enable the continuous delivery of value (in this example the web application) to end users.

Task 1: Create the release pipeline

1. Open the last build log in VSTS by navigating to Build and Release, then Builds and click on the latest build number. Then click on Release above the build summary.

The screenshot shows the 'Builds' tab selected in the VSTS navigation bar. Below it, a build summary card for 'Web App-ASP.NET-CI / Build 20180123.2' is displayed. The card shows a green 'Build succeeded' status. A red box highlights the 'Release' button in the top right corner of the card. The card also displays details like 'Build details', 'Test Results', and 'Code Coverage'.

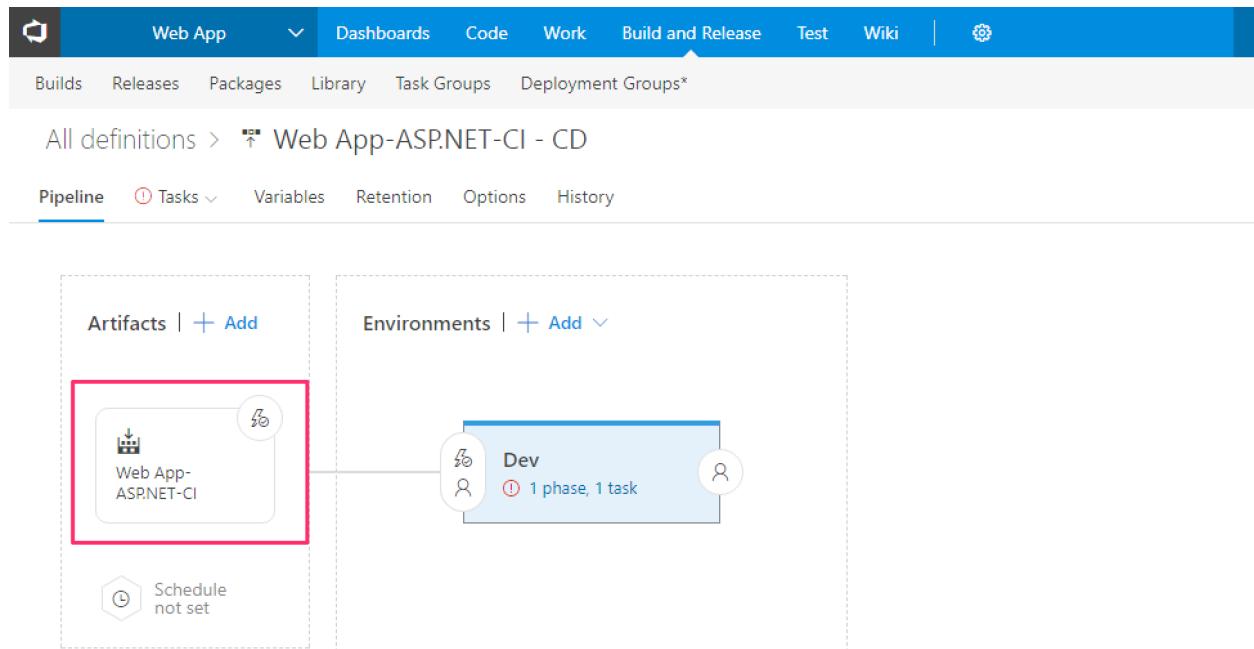
2. A new release pipeline is created, and you can apply a template to it. If you scroll through the list you'll see that there are many templates. As we are going to deploy a web application into an Azure App Service, select the Azure App Service Deployment template.

The screenshot shows the 'Builds' tab selected in the top navigation bar. Below it, the 'Pipeline' tab is active. On the left, there's a 'Artifacts' section with a 'Web App-ASP.NET-CI' artifact and a 'Schedule not set' button. To its right is an 'Environments' section with a 'Environment 1' card labeled 'Select template'. On the right side, a 'Select a Template' pane is open, showing a list of templates under 'Featured'. The first template, 'Azure App Service Deployment', is highlighted with a red border and has a blue 'Apply' button. Other listed templates include 'Deploy Node.js App to Azure App Service', 'Deploy PHP App to Azure App Service', 'IIS Website and SQL Database Deployment', 'Azure Cloud Service Deployment', 'Azure App Service Deployment with Continuous Monitoring', and 'Azure App Service Deployment with Performance Test'. A search bar at the top of the pane has the word 'Empty process' in it.

3. A release pipeline may have many environments. For now we will just have one, and the first environment to deploy to is typically a shared development environment, so call it Dev or similar. Click the close button.

The screenshot shows the 'Builds' tab selected in the top navigation bar. Below it, the 'Pipeline' tab is active. On the left, there's a 'Artifacts' section with a 'Web App-ASP.NET-CI' artifact and a 'Schedule not set' button. To its right is an 'Environments' section with a 'Dev' environment card labeled '1 phase, 1 task'. On the right side, a 'Environment' configuration pane is open. It shows a 'Properties' section with 'Name and owners of the environment'. Under 'Environment name', the value 'Dev' is entered and highlighted with a red border. Under 'Environment owner', the value 'Giles Davies' is listed. A red box highlights the close ('X') button in the top right corner of the configuration pane.

4. Notice that there is already an Artifact to deploy. This has been setup for us after clicking Release in the build log. Click on the Artifact to see the details.



5. This shows that the Artifact is the latest version of the output of the CI build that was created earlier. That output includes the zip file, which is the web application to be deployed.

Close the window.



Artifact

Build - Web App-ASP.NET-CI

Delete ⋮

Project ⓘ

Web App

Source (Build definition) ⓘ

Web App-ASP.NET-CI

Default version * ⓘ

Latest

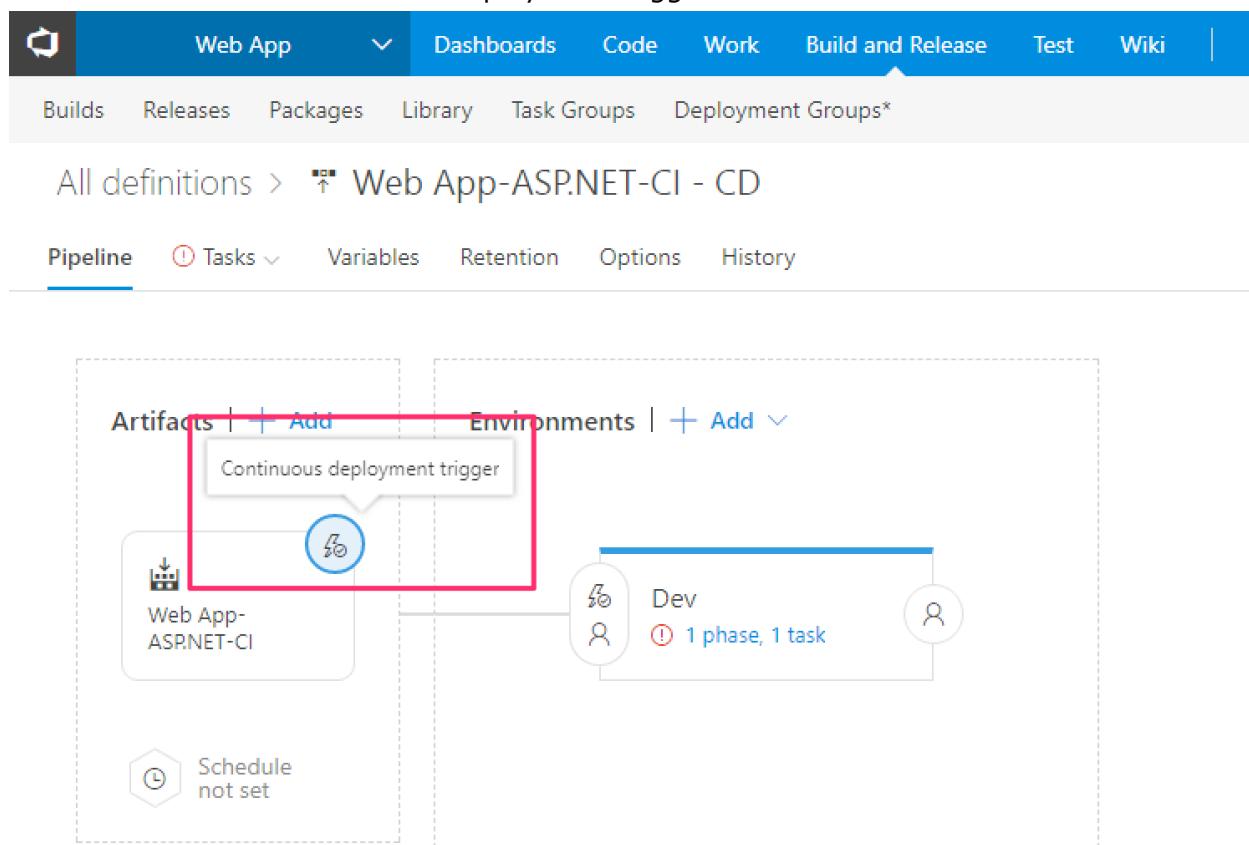


Source alias ⓘ

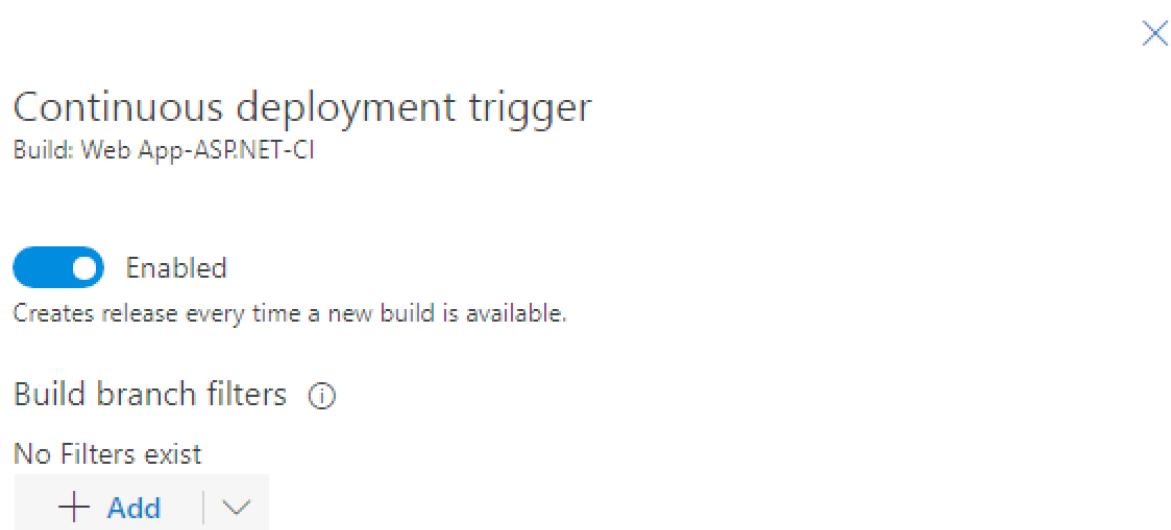
Web App-ASP.NET-CI

ⓘ The artifacts published by each version will be available for deployment in Release Management. The latest successful build of **Web App-ASP.NET-CI** published the following artifacts: **drop**.

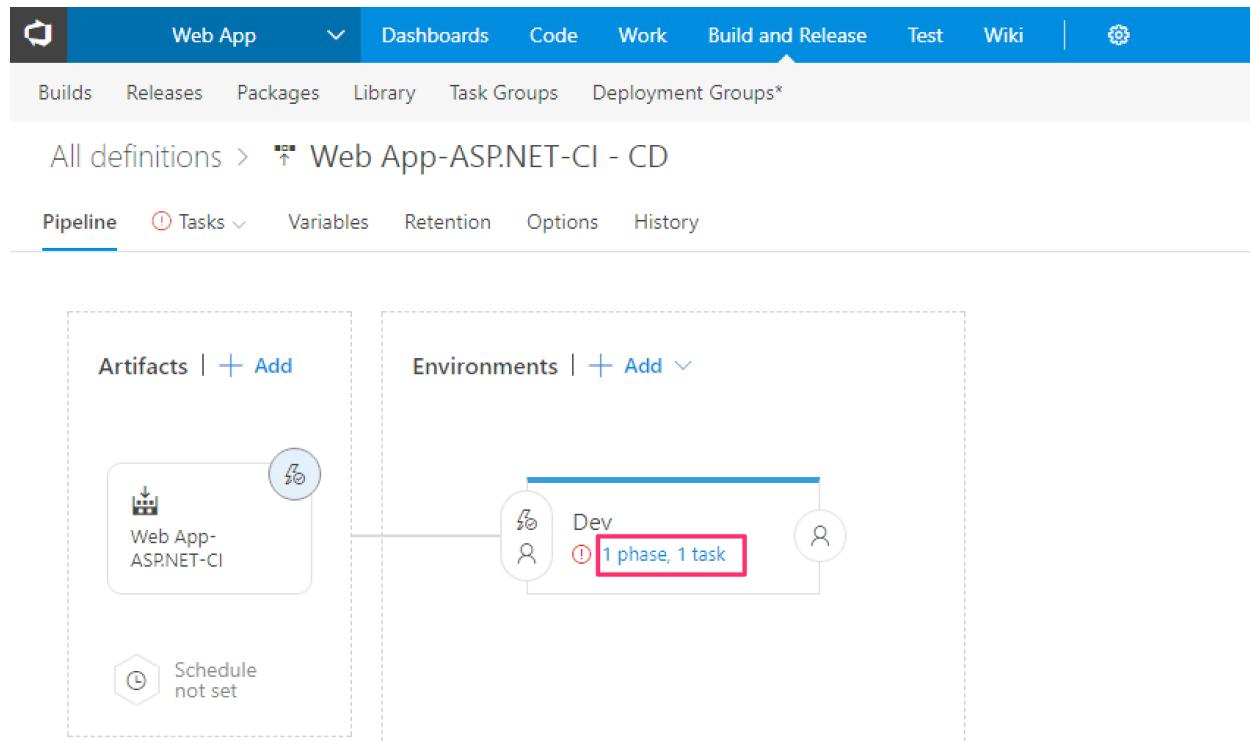
6. Click on the Artifact Continuous deployment trigger.



7. This is where you can enable or disable Continuous Deployment - i.e. whenever a new build is created, this release pipeline is triggered. Notice that it should already be set to Enabled (set it if not). Close the window.



8. Now click on the phase and tasks in the Dev environment. This is where we will configure how to do the deployment into Azure.

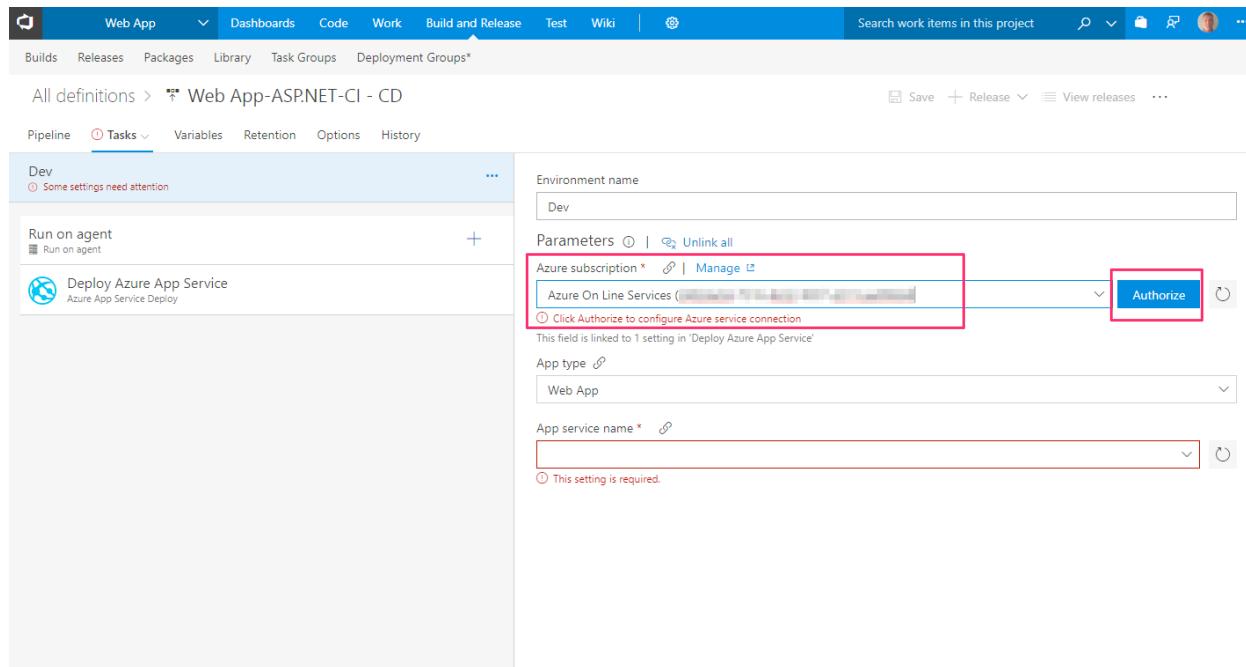


9. The first areas to address in the environment are Azure settings.

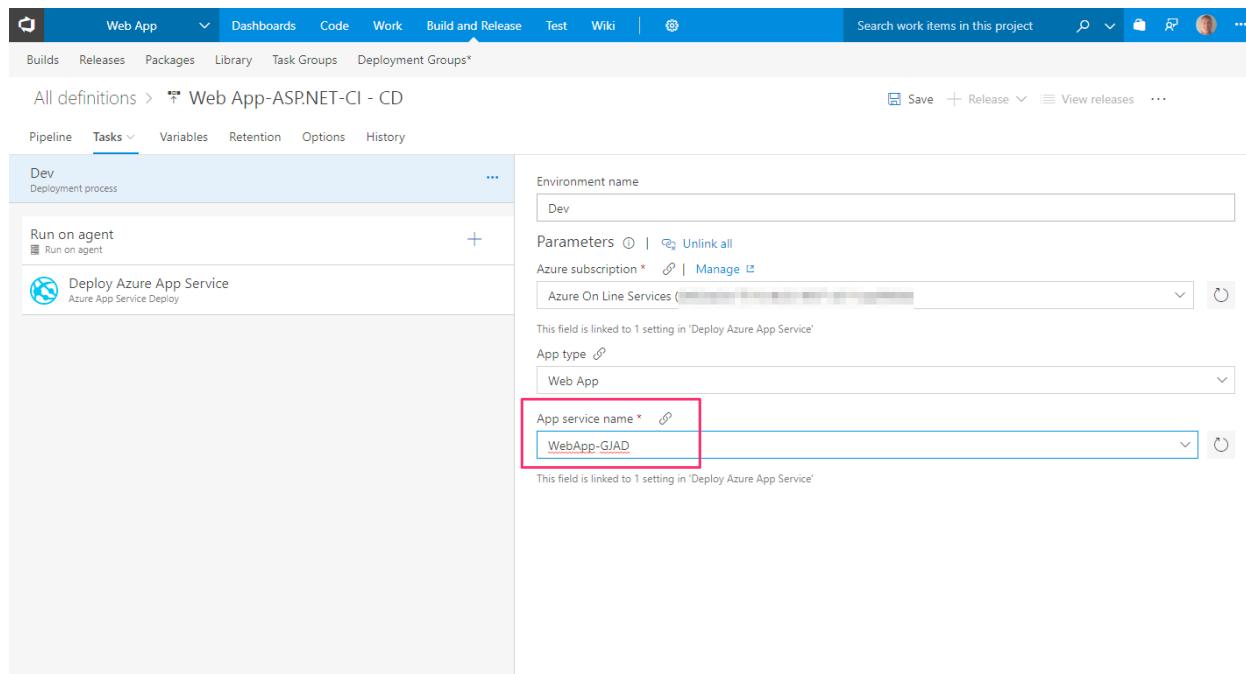
The screenshot shows the "Environment settings" page for the "Dev" environment.

- Environment name:** Dev
- Parameters:** Some settings need attention
- Azure subscription:** This setting is required.
- App type:** Web App
- App service name:** This setting is required.

10. In the Azure subscriptions list, choose the relevant subscription and click Authorize. This is to create an authorised connection to Azure using that subscription.



- Having authorised the subscription you should be able to see and select the Web App that you created in the preceding step. This is the target Web App that we will deploy to.



- Click the Run on agent step and confirm that the agent is set to the same as the build - the Hosted VS2017 agent.

Agent phase ①

Display name * Run on agent

Agent selection ^

Agent queue ① | Manage ↗ Hosted VS2017

Demands ①

Execution plan ^

Parallelism ①

None Multi-configuration Multi-agent

Timeout * ① 0 Deployment job cancel timeout in minutes * ①

13. Click Deploy Azure App Service task. You shouldn't need to change anything here but note that this uses the Azure subscription to deploy to the App Service, and the package to be deployed is a zip file, as created in the CI build. Click Save.

Azure App Service Deploy ①

Version 3.*

Display name * Deploy Azure App Service

Azure subscription * ↗ | Manage ↗ Azure On Line Services ()

App type * ↗ Web App

App Service name * ↗ WebApp-GJAD

Deploy to slot ①

Virtual application ①

Package or folder * ↗ \$(System.DefaultWorkingDirectory)***.zip

File Transforms & Variable Substitution Options ↴

You have now created a Release Pipeline, configured to Continuously Deploy whenever there is a new build. The next step is to test the overall flow.

Task 2: Test the release pipeline

1. Test the release pipeline by returning to Visual Studio and making a change. For example open the WebApp/Views/Home/Index.cshtml again and make another change such as

changing the heading for the home page. Save the changes.

The screenshot shows the Microsoft Visual Studio interface. The main window displays the `Index.cshtml` file under the `WebApp` project. The code contains the following HTML and C# code:

```
@{
    ViewBag.Title = "Home Page";
}

<div class="jumbotron">
    <h1>My Web App</h1>
    <p class="lead">ASP.NET is a free web framework for building great Web sites and Web applications. It's open source, powerful, and provides tools to help you build, deploy, and manage your application quickly and easily.</p>
    <div class="row">
        <div class="col-md-4">
            <h2>Getting started</h2>
            <p>ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.</p>
            <p><a href="https://asp.net" class="btn btn-primary btn-lg">Learn more &gt;</a></p>
        </div>
        <div class="col-md-4">
            <h2>Get more libraries</h2>
            <p>NuGet is a free Visual Studio extension that makes it easy to add, remove, and update packages in your project.</p>
            <p><a href="https://go.microsoft.com/fwlink/?LinkId=301866" class="btn btn-default">Learn more &gt;</a></p>
        </div>
        <div class="col-md-4">
            <h2>Web Hosting</h2>
            <p>You can easily find a web hosting company that offers the right mix of features and price to fit your needs.</p>
            <p><a href="https://go.microsoft.com/fwlink/?LinkId=301867" class="btn btn-default">Learn more &gt;</a></p>
        </div>
    </div>

```

The `Views` folder in the Solution Explorer is highlighted with a red box. It contains the files `About.cshtml`, `Contact.cshtml`, and `Index.cshtml`.

2. In the Team Explorer, return to the Changes hub to see the files you've changed, and add a

Team Explorer - Changes

Changes | Web App

Branch: master

Changed home page title

Commit All Actions

- Commit All
- Commit All and Push**
- Commit All and Sync

0 Messages

Refresh Analysis

To see more issues, turn on Full Solution Analysis. [Click here to learn more](#)

Related Work Items

Drag work items here to link them to the commit.

Changes (1)

- C:\Users\giles\Source\Repos\Web Ap...
Index.cshtml

comment and select Commit All and Push.

Properties Solution... Team Ex... Class View

3. In VSTS, navigate to the Builds (Build & Release | Builds) and you should now see a build in progress. Click on the build number to watch the build.

Web App

Builds Releases Packages Library Task Groups Deployment Groups*

Build Definitions

Mine Definitions Queued

Build ID or build number + New + Import Security Help

Requested by me	Status	Triggered by	History
Web App-ASP.NET-CI : #20180123.2 Giles Davies requested just now	► in progress	Changed home page title 6e53d15 in master	[History]

4. When the build has completed, open the build log summary (Build and Release | Builds | click the ... by the build and View build results) and on the right hand side scroll down to see the Deployments area. You should see that a release to Dev is in progress and therefore Continuous Deployment has been triggered.

The screenshot shows the Azure DevOps Build Log Summary page for a build named "Build 20180123.3". The build status is "Build succeeded". The deployment section shows a deployment to the "Dev" environment, which is currently in progress. The deployment step "Download artifact - Web App-ASPNE..." is highlighted with a red box.

5. Click on the Dev link in Deployments to see the release logs. Note that the zip file (artifact) is downloaded from the build (not rebuilt) and then deployed to Azure.

The screenshot shows the Azure DevOps Build and Release interface. The "Releases" tab is selected. A release definition named "Web App-ASPNET-CI - CD / Release-1" is shown. The "Logs" tab is selected, displaying the release logs. The logs show the deployment steps, including "Download artifact - Web App-ASPNE..." and "Deploy Azure App Service", both of which are highlighted with red boxes.

6. Go to the web app URL (as per Lab 3 Step 10 above) and load your newly deployed web application.

New Title

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

[Learn more »](#)

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

[Learn more »](#)

Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

[Learn more »](#)

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

[Learn more »](#)

© 2018 - My ASP.NET Application

You have now deployed a web application into a live Azure site using a DevOps release pipeline triggered by committing a code change. If you want, make one or more other changes in the code, commit and push and see those changes built and deployed into your web application.

Optional: Add a Release Definition Overview widget to the Lab Progress dashboard by:

- Searching for and adding the Release Definition Overview widget:

Web App

Overview Lab Progress

Web App Project

Reference project for the DevOps Lab. Feedback and testing from:

- Mark Harrison
- Tamara Walther

23/01/2018

Web App-ASP.NET-CI

23/01/2018

Microsoft Azure

Your App Service app is up and running

Go to your app's Quick Start guide in the Azure portal to get started [View quick start](#)

We would love to meet you! The App Service team will be present at the upcoming events:

Name	Location	Dates	Info
Tech Summit - Bangalore	Bangalore, India	January 24-25	(1)
Tech Summit - Cape Town	Cape Town, South Africa	February 19-20	(1)
Tech Summit - Madrid	Madrid, Spain	February 26-27	(1)
Tech Summit - Amsterdam	Amsterdam, Netherlands	March 4-5	(1)
Tech Summit - Paris	Paris, France	March 11-12	(1)
Tech Summit - San Francisco	San Francisco, CA	March 18-19	(1)
Tech Summit - Stockholm	Stockholm, Sweden	April 12-13	(1)
Tech Summit - Vienna	Vienna, Austria	April 25-26	(1)

Add Widget

release

Burnup

Displays burnup across multiple teams and multiple sprints. Create a release burnup or bug burnup.

Deployment status

Shows the deployment and test status of a branch across the environments in your release definitions.

Release Definition Overview

Shows the status of environments in a release definition.

By Microsoft

Learn More

Add Close

- Set the Release Definition to the release created in the lab above:

The screenshot shows the Azure DevOps interface with the 'Lab Progress' tab selected. On the left, there's a 'Release Definition Overview' card for 'Web App-ASP.NET-CI'. It lists a single release named 'Release-1' under the 'Dev' environment. To the right, a 'Configuration' pane is open, showing the 'Release Definition' field set to 'Web App-ASP.NET-CI - CD'. A red box highlights this selection. At the bottom right of the configuration pane are 'Save' and 'Cancel' buttons.

- Save the changes, close the widget gallery and save the dashboard by clicking on the blue edit button in the bottom right hand corner.

Lab 5: Infrastructure as Code

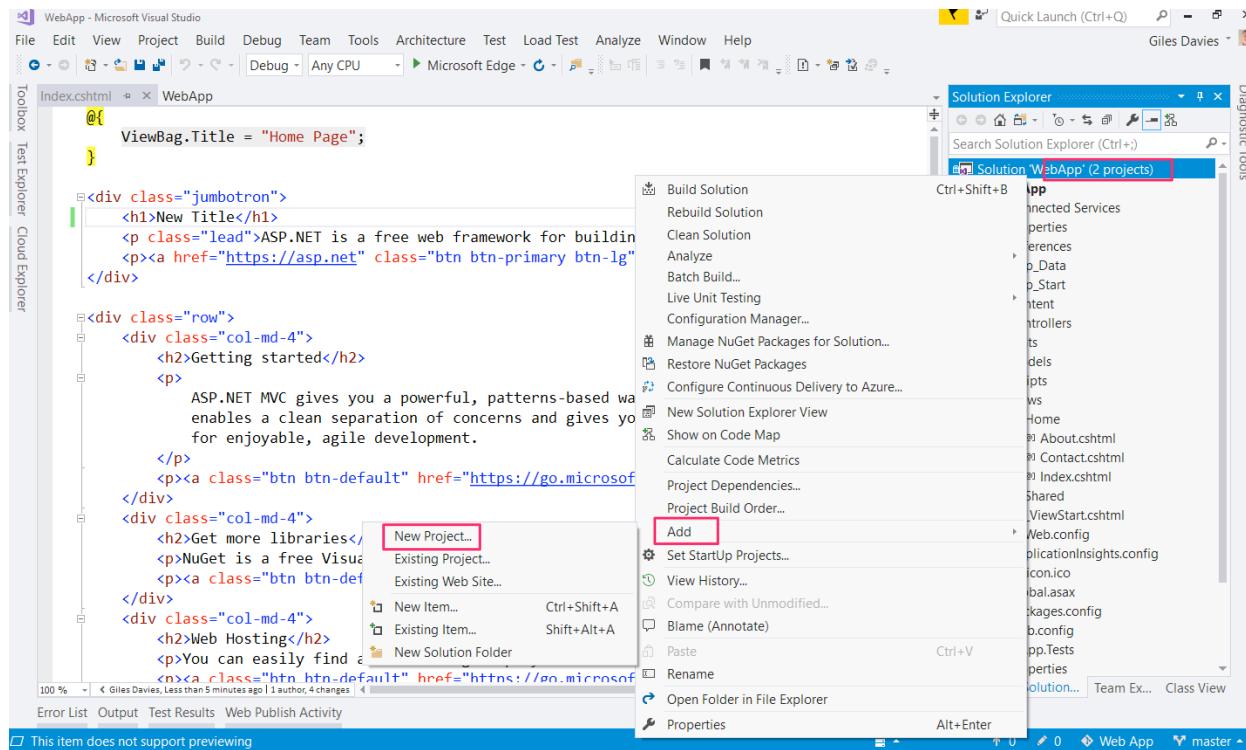
The ability to treat infrastructure (machines, networks, configuration) in the same way as code brings many benefits, but in particular allows you to create infrastructure on demand and include that in your DevOps pipeline.

Azure Resource Manager (ARM) templates are the native approach and this lab adds using ARM into the flow. The example here will allow you to create an environment in Azure on demand as part of the flow.

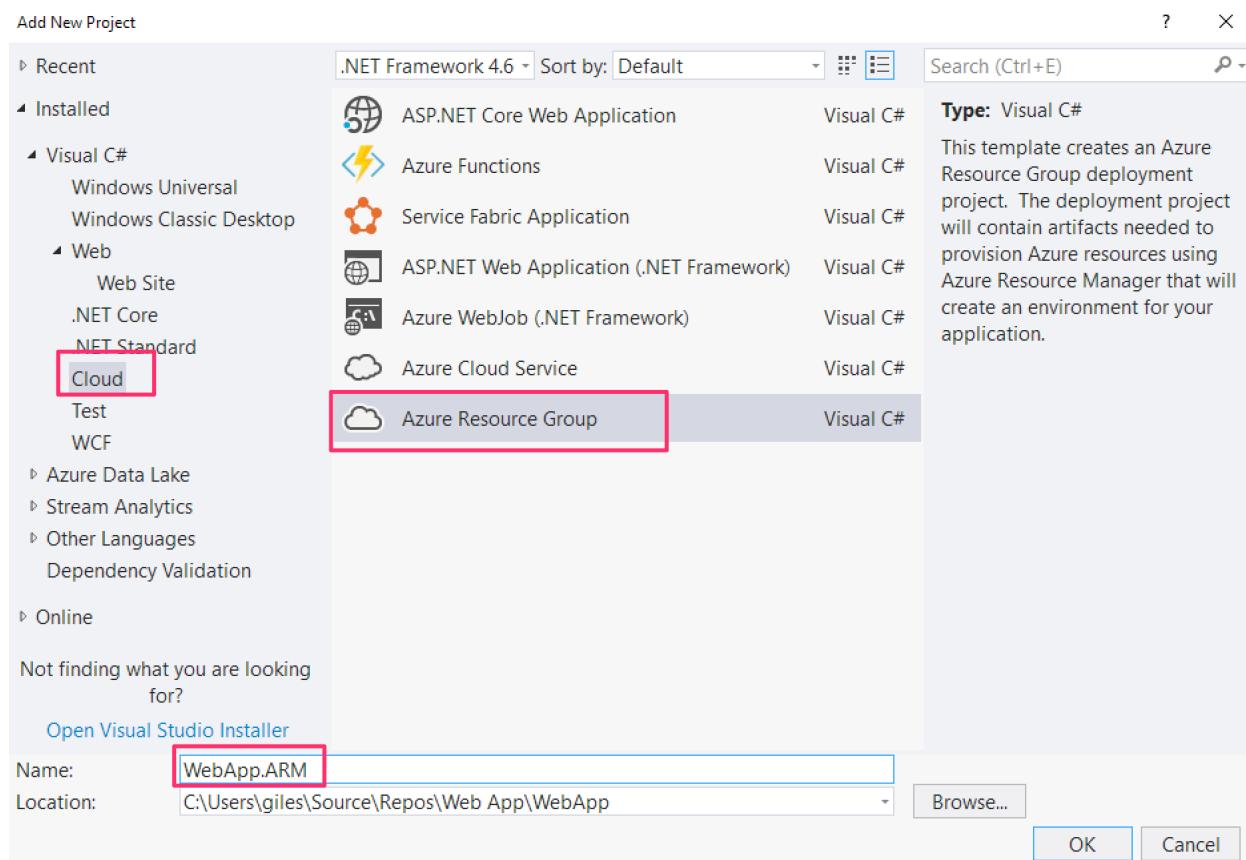
This lab will create a new test environment in Azure without needing to manually create it (via the Portal or the Command line etc.).

Task 1 - Create the ARM template for a Web App

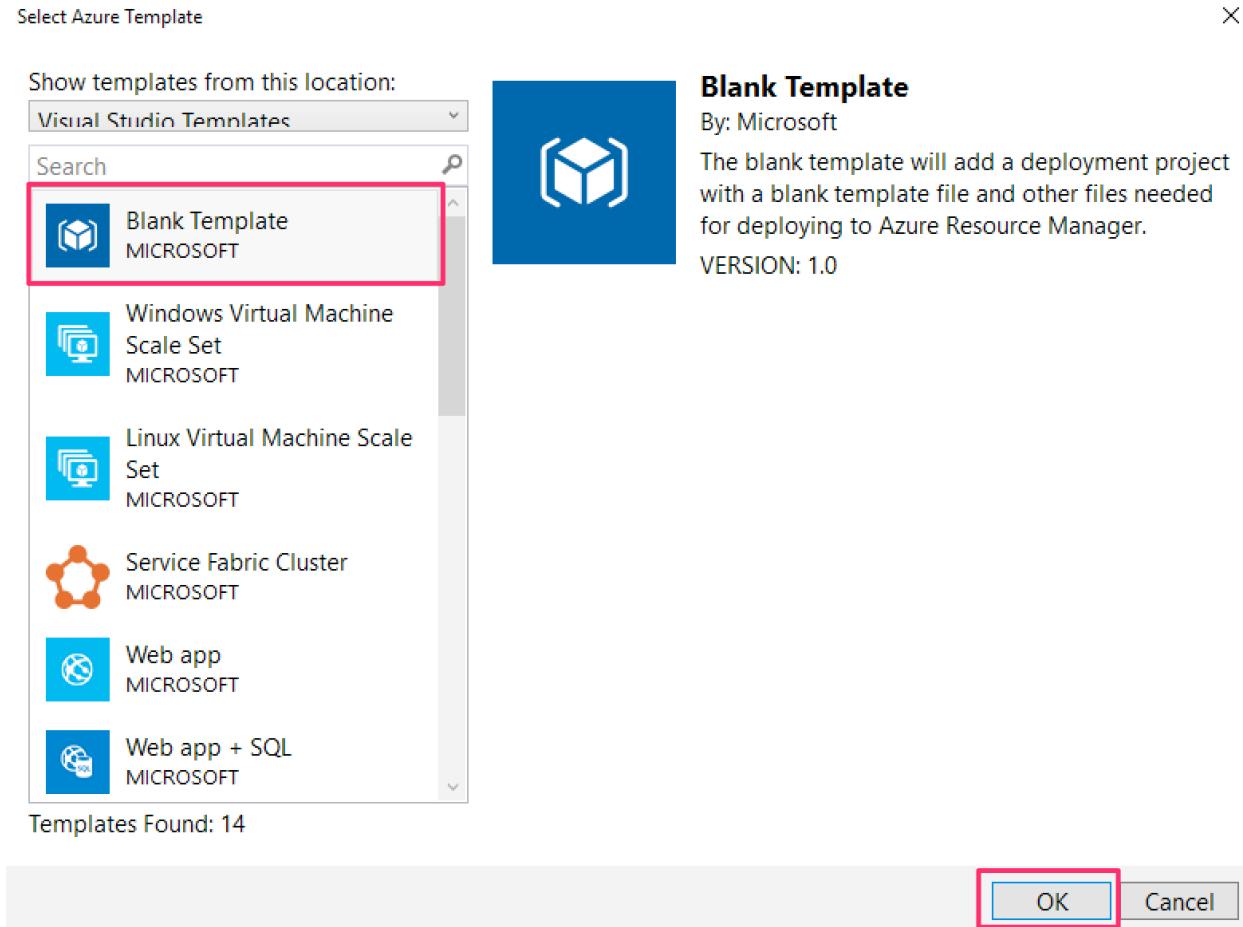
1. In Visual Studio, with the Web App solution open in Solution Explorer, right click the solution and select Add | New Project.



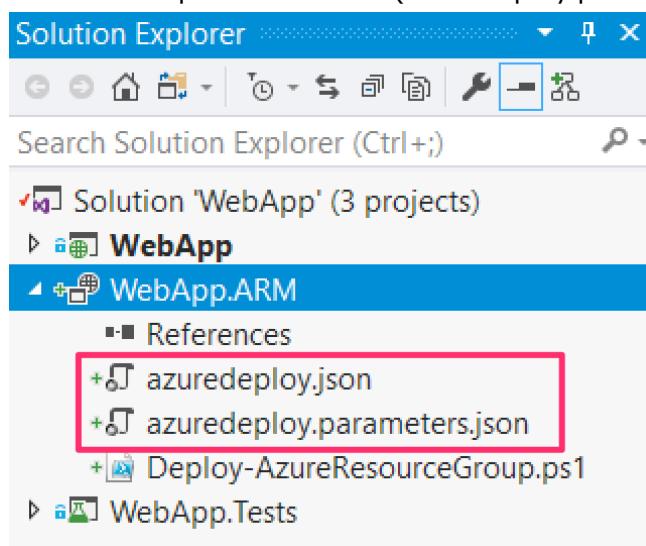
2. Select Cloud | Azure Resource Group and name the project e.g. WebApp.ARM.



3. There are a range of ARM templates to create a wide variety or resources in Azure. In this case from the Visual Studio Templates select Blank Template and click OK.



4. You now have a project in your solution containing a blank ARM template (`azuredeploy.json`) and a blank parameters file (`azuredeploy.parameters.json`).



5. Now replace the content of the files created in the previous step with pre-prepared content:

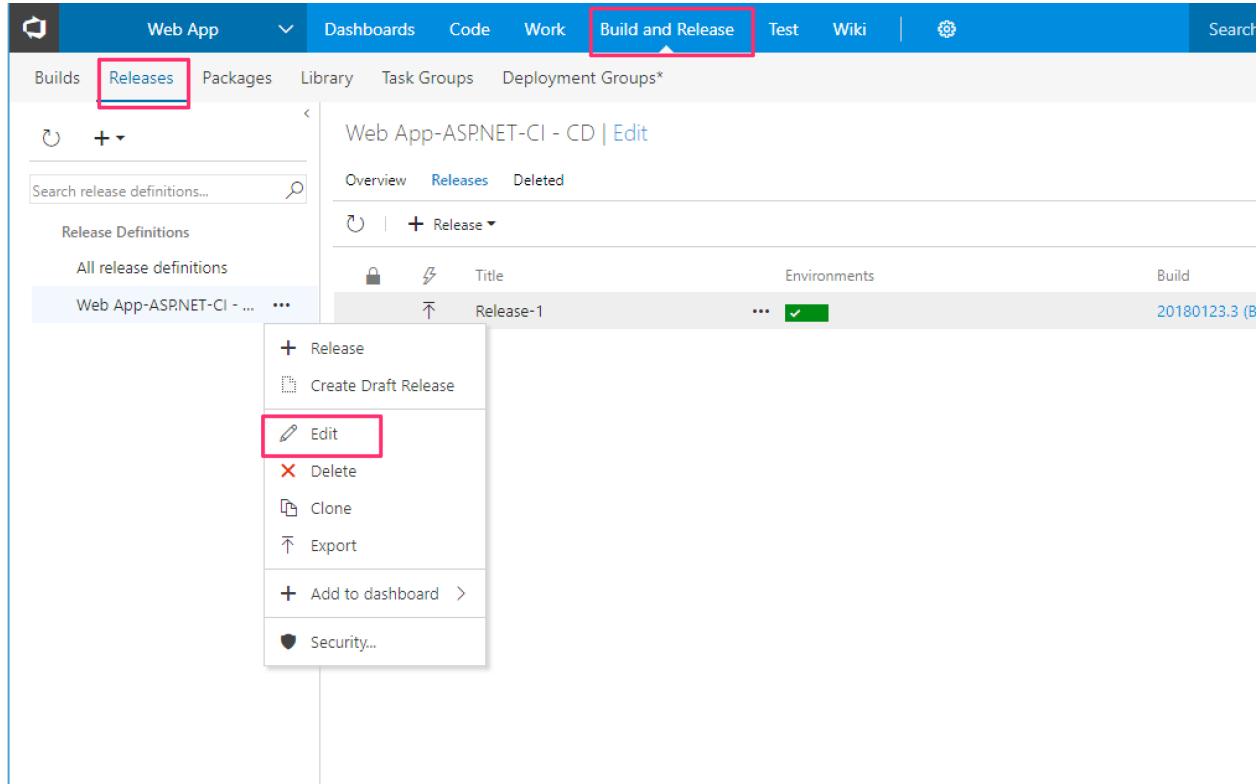
- View (or download) [this `azuredeploy.json` file](#), select all the contents of the file, and then copy and paste over the content of the `azuredeploy.json` file in Visual Studio.
- Do the same with this [`azuredeploy.parameters.json` file](#).
- Save the files in Visual Studio.

6. In Visual Studio, select the Team Explorer | Changes. Add a commit comment and select Commit All and Push. Save if prompted.

The ARM template is now added to source control, although there is no need for it to be in the same repository as the code, this is just for convenience in this lab. Note that adding the new files to source control will trigger a build and release in the background. You could temporarily turn off the CI trigger but just let it run in the background while completing the next task.

Task 2 - Update the release pipeline to provision the Web App using the ARM template.

1. In VSTS select Build and Release | Releases | your release definition and Edit.



2. Add a new artifact to the release pipeline. This means the release will get the application from the build (as a zip file) and the ARM templates directly from source control (as they

don't need to be compiled or packaged).

The screenshot shows the Azure DevOps interface for managing releases. At the top, there's a navigation bar with links for Web App, Dashboards, Code, Work, Build and Release, Test, Wiki, and a gear icon. Below this is a secondary navigation bar with links for Builds, Releases (which is highlighted in blue), Packages, Library, Task Groups, and Deployment Groups. The main content area shows the title "All definitions > Web App-ASP.NET-CI - CD". Under this, there are two sections: "Artifacts" and "Environments". The "Artifacts" section contains a box for "Web App-ASP.NET-CI" with a download icon and a "Schedule not set" note. A red box highlights the "+ Add" button. The "Environments" section shows a single environment named "Dev" which is described as having "1 phase, 1 task".

3. Set the Source type to Git, the Project and Source (repository) to the Web App project, the Default branch to master and the default version to Latest from default branch. Then click

Add (you may need to scroll down).

Add artifact

Source type



Build



✓ Git



GitHub



Team Foundation ...

[4 more artifact types ▾](#)

Project * ⓘ

Web App

Source (repository) * ⓘ

Web App

Default branch * ⓘ

master

Default version * ⓘ

Latest from default branch

Checkout submodules ⓘ

Checkout files from LFS ⓘ

Shallow fetch depth ⓘ

4. In the Pipeline view hover over the Dev environment and select Clone.

The screenshot shows the Azure DevOps Pipeline view. On the left, there's an 'Artifacts' section with a 'Web App-ASP.NET-CI' item. On the right, there's an 'Environments' section containing a 'Dev' environment node. A red box highlights the 'Dev' node, and a 'Clone' button is visible below it, with a tooltip 'Clone environment'.

5. Select the cloned environment Copy of Dev and change the name to QA and close the Environment window.

The screenshot shows the Azure DevOps Pipeline view. The 'Environments' section now includes a 'QA' environment node, connected to the 'Dev' node. The 'QA' node is highlighted with a red box. The 'Environment' properties pane on the right shows the 'Environment name' field set to 'QA', which is also highlighted with a red box. A red box also highlights the close button in the top right corner of the environment window.

6. Select the QA environment (by clicking on the "1 phase, 1 task" link inside the QA environment) and in Tasks click the plus sign and then search for the Azure Resource Group

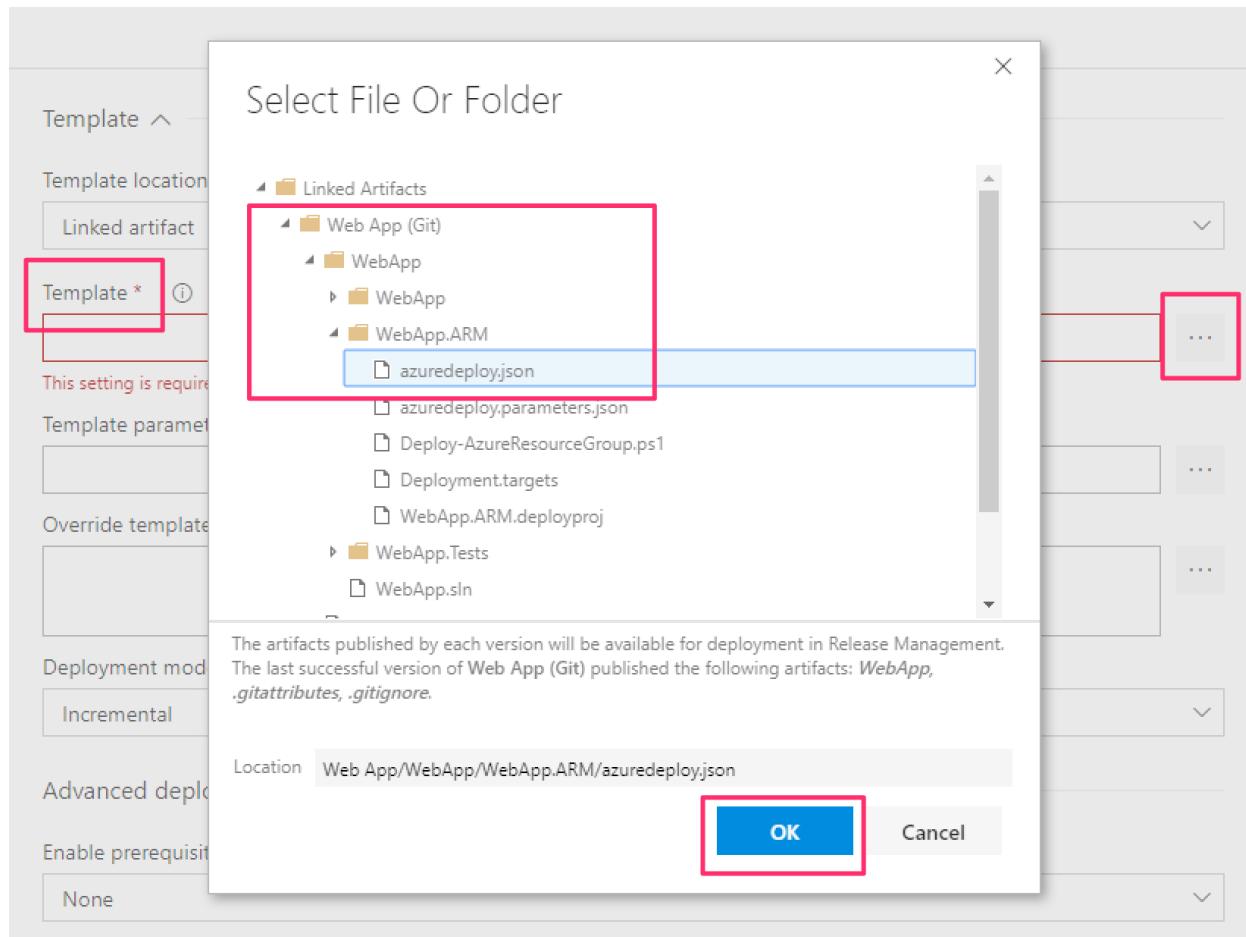
Deployment task. Select Add.

The screenshot shows the Azure DevOps interface for creating a deployment pipeline. In the center, there's a search bar labeled 'Add tasks' with the placeholder 'Don't see what you need? Check out our Marketplace.' Below it, a search result for 'Azure Resource Group Deployment' is shown, with its description 'Deploy, start, stop, delete Azure Resource Groups'. A red box highlights the search bar and the result card.

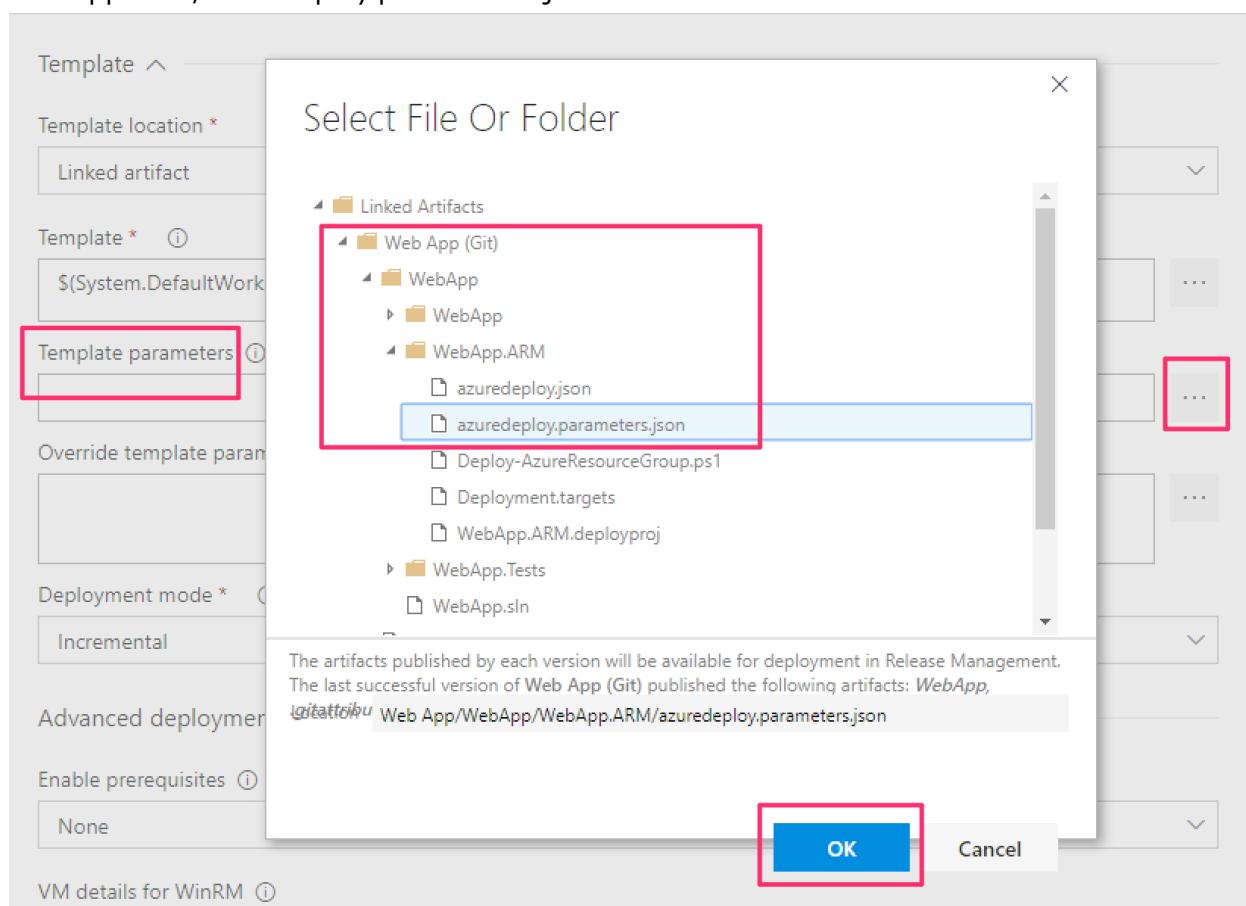
7. Set the Azure Details. Select the Azure subscription set up previously, Ensure that the Action is Create or update resource group, Enter a new resource group name for the QA environment e.g. WebAppQA-RG and set the location.

The screenshot shows the 'Azure Resource Group Deployment' task configuration screen. It includes fields for 'Display name' (set to 'Azure Deployment:Create Or Update Resource Gr...'), 'Azure Details' (with 'Azure subscription' set to 'Azure On Line Services (...' and 'Action' set to 'Create or update resource group'), 'Resource group' (set to 'WebAppQA-RG'), and 'Location' (set to 'North Europe'). Red boxes highlight the 'Action' dropdown, the 'Resource group' input field, and the 'Location' input field.

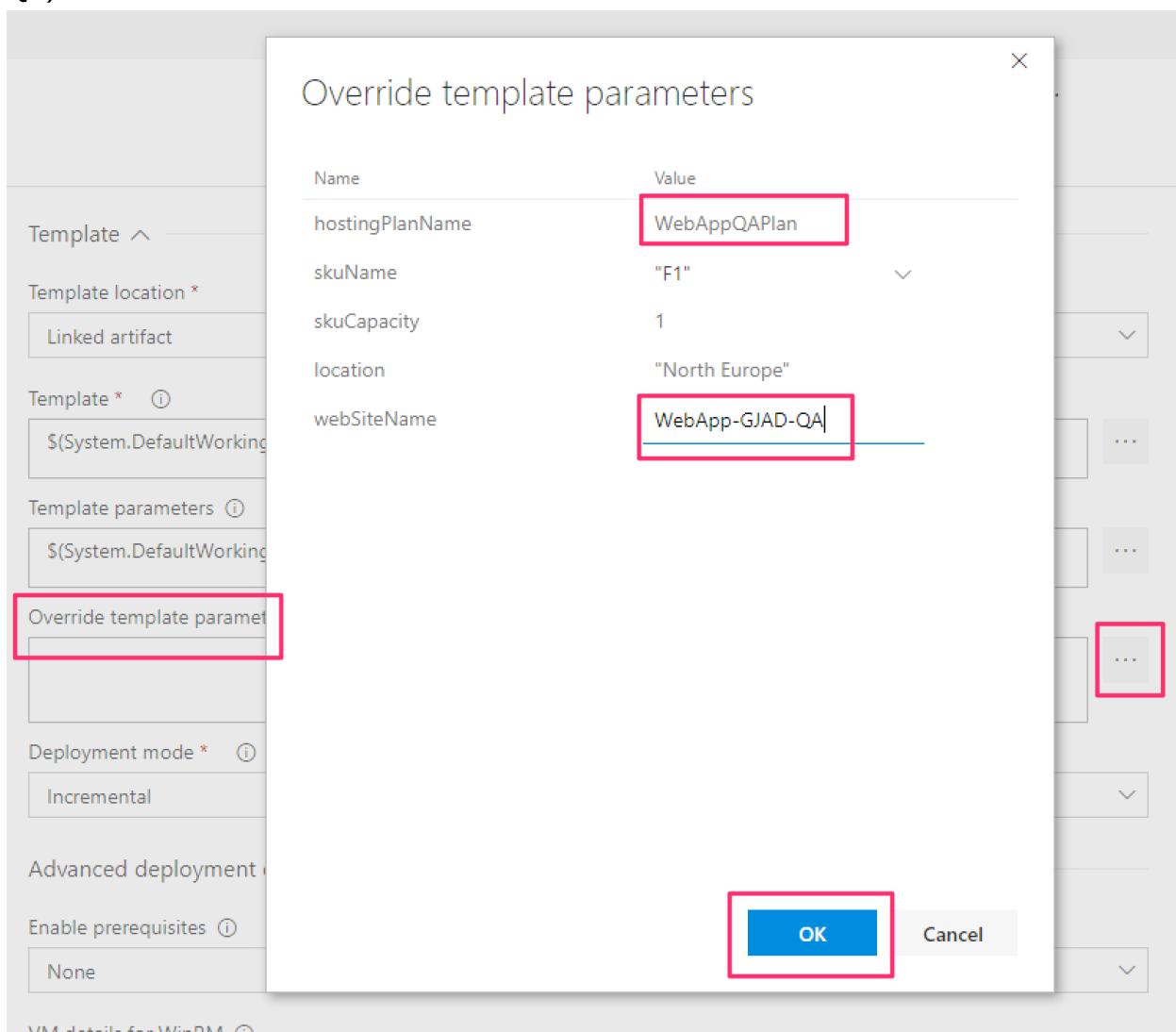
8. In the Template section set the Template (using the ... button) to Web App (Git) WebApp.ARM/azureddeploy.json and click OK.



9. Set the Template parameters field (using the ... button) to WebApp (Git) WebApp.ARM/azuredeploy.parameters.json and click OK.

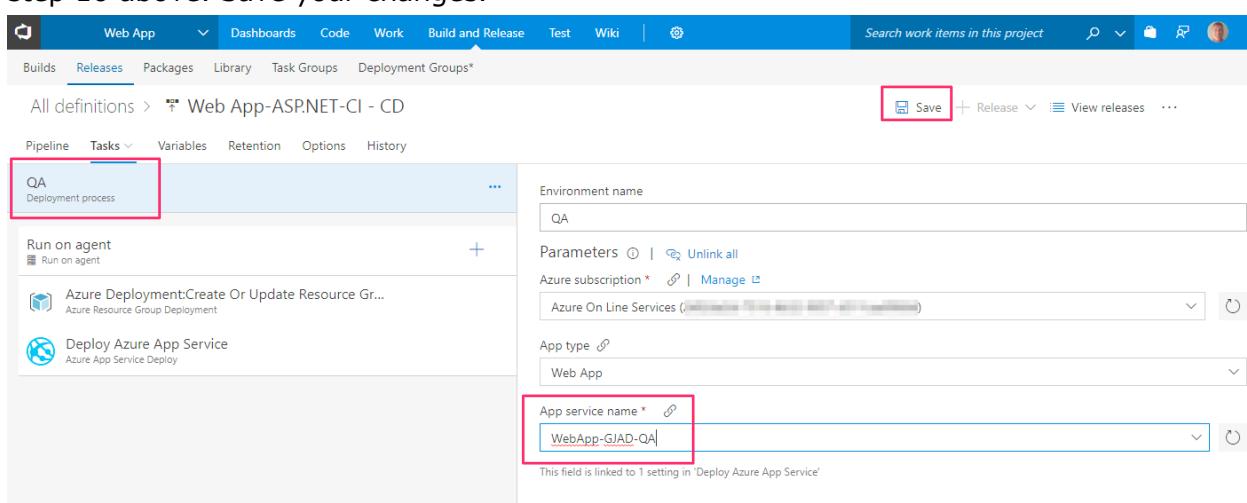


10. Set the Override Template parameters field (using the ... button) to WebAppQAPlan, the webSiteName to the same as the Dev website but with QA appended (e.g. WebApp-GJAD-QA)and click OK.



11. Move the Azure Deployment: Create or Update Resource Group task to be before the Deploy Azure App Service task by dragging and dropping the task.

12. In the QA deployment process settings set the App service name to the name you used in step 10 above. Save your changes.



13. Test the changes by making another code change (e.g. changing the heading again), committing and pushing, and observe the build and release. After a few minutes you should see that both the Dev and QA environments have been successfully deployed to.

The screenshot shows the Azure DevOps interface for managing releases. On the left, under 'Release Definitions', the 'Web App-ASP.NET-CI - ...' definition is selected. The main area displays the 'Web App-ASP.NET-CI - CD / Release-3' release. It consists of two environments: 'Dev' and 'QA'. Each environment contains several steps, each with a green checkmark indicating success. The steps are categorized into pre-deployment, deployment, and post-deployment phases. Three specific steps in the QA environment are highlighted with red boxes: 'Pre-deployment approval', 'Run on agent', and 'Deploy Azure App Service'. The 'Deploy Azure App Service' step is also highlighted with a red box.

14. Explore the [Azure portal](#) to find the resource group WebApp-QA-RG and the web app provisioned using ARM in the QA environment. Confirm that the App service has been deployed and open it using the URL in the App service overview.

The screenshot shows the Azure portal's App Service overview for 'WebApp-GJAD-QA'. The left sidebar lists various tabs: Overview (highlighted in blue), Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quickstart, Deployment credentials, Deployment slots, Deployment options, and Continuous Delivery (Preview). The main content area shows the 'Resource group (change) WebAppQA-RG'. Key details include: Status (Running), Location (North Europe), Subscription (change) Azure On Line Services, and Subscription ID (redacted). The 'URL' field shows 'https://webapp-gjad-qc.azurewebsites.net', which is highlighted with a red box. To the right, there are two charts: 'Http 5xx' and 'Data In'. The 'Data In' chart shows a significant spike from 0 to 24.28 MB between 13:00 and 13:30. Other details shown include App Service plan/pricing tier (WebAppQAPlan (Free) 0 Small), Continuous delivery status (Edit continuous delivery), OS name (Windows Server 2016), and HTTP SERVER ERRORS (0).

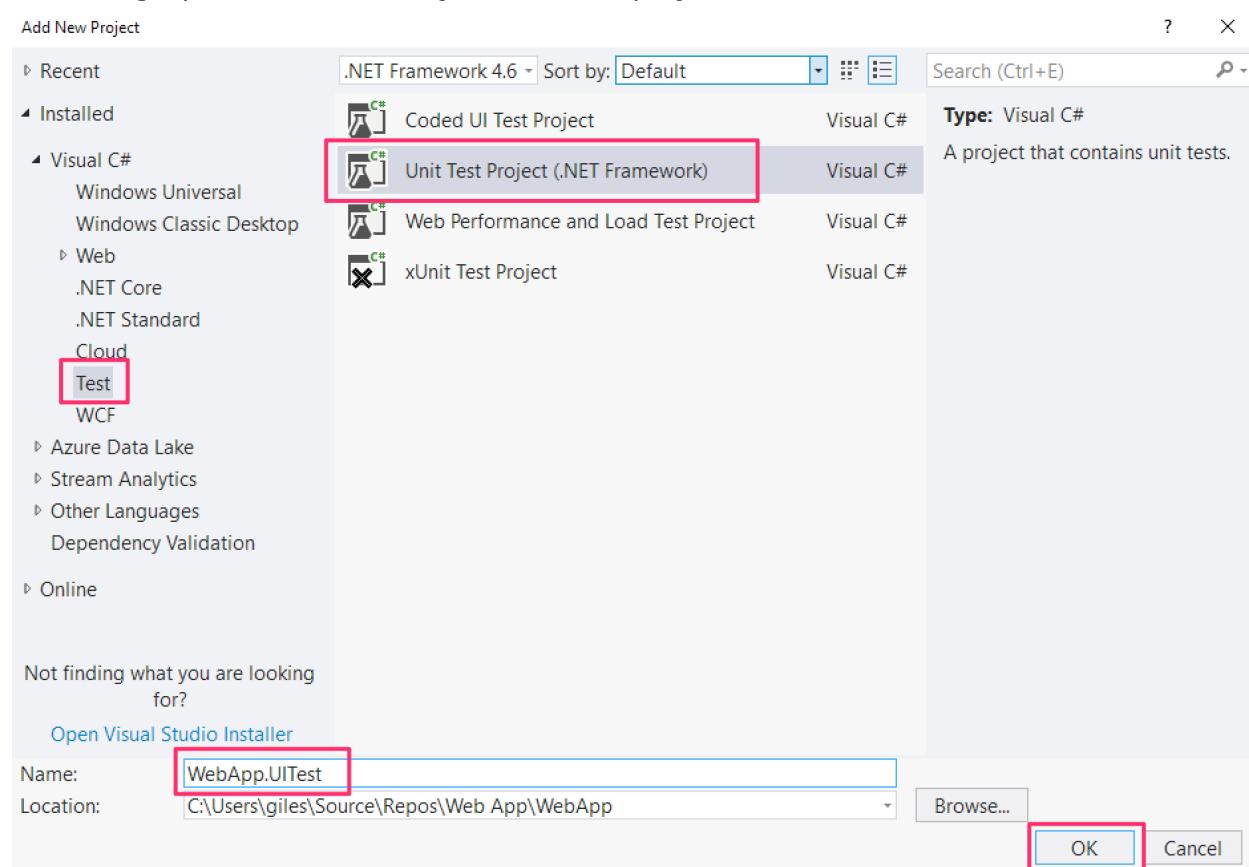
You have now created a DevOps pipeline that deploys to multiple environments, and provisions the QA environment on demand without a manual process.

Lab 6: Automated Testing with Selenium

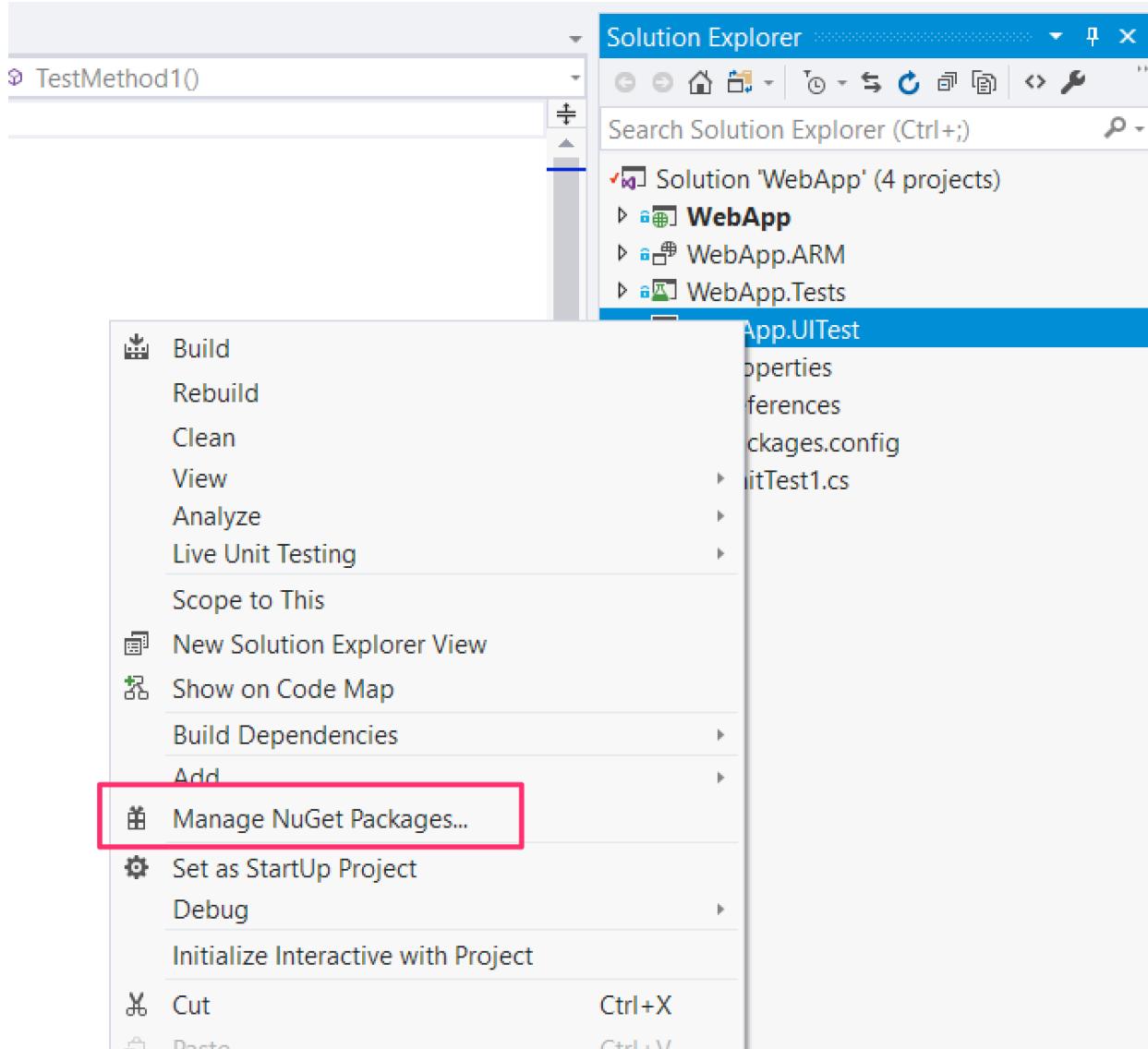
Integrating automated tests into your DevOps pipeline can help drive quality whilst deploying more frequently. This lab integrates [Selenium](#), a popular testing framework, into your pipeline. Selenium allows you to automate the testing of your web application using all the main browsers, reducing the manual testing required.

Task 1: Create the Selenium tests

1. In Visual Studio, Solution Explorer, select the solution and Add | New Project. Then select the Test category and Unit Test Project. Give the project a name and click OK.



2. The Selenium libraries need to be added to the project. Right-click on the test project and select Manage NuGet Packages.



3. Select Browse, search for Selenium and install Selenium.WebDriver

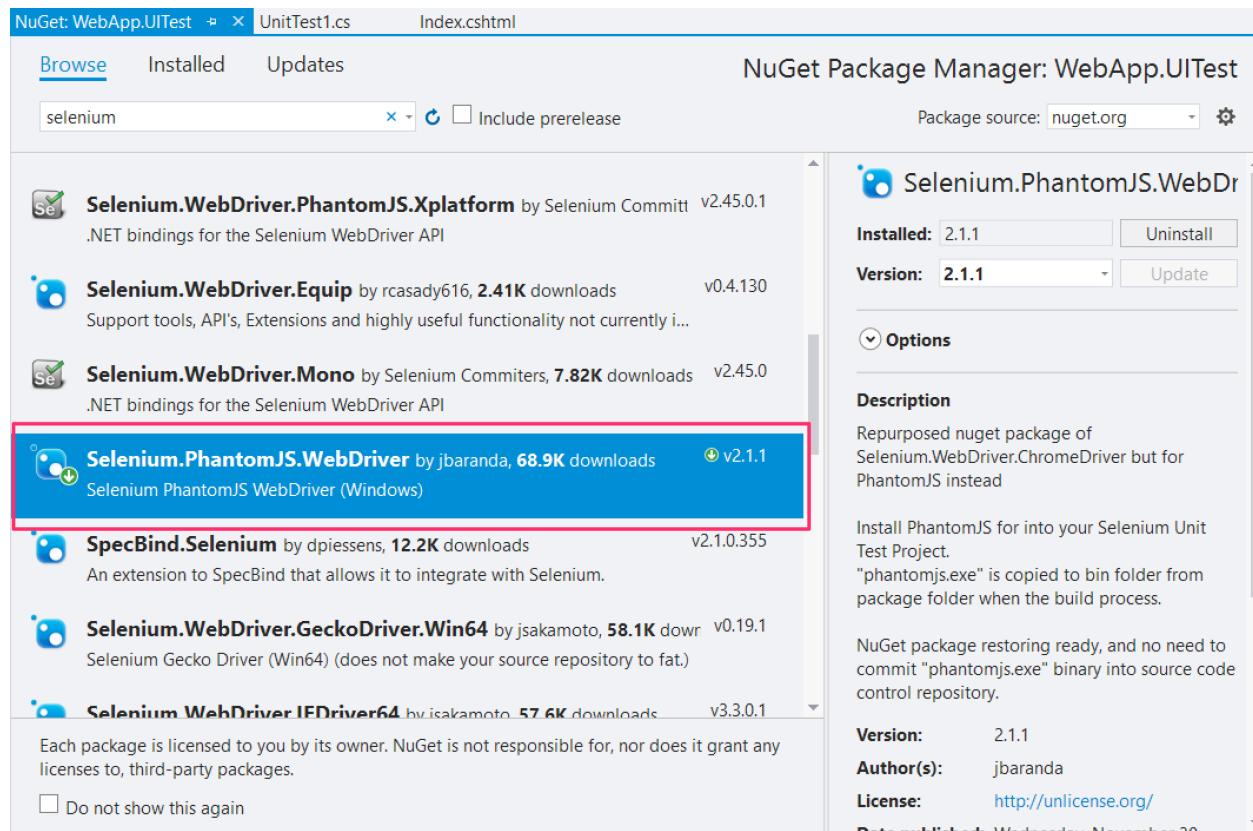
The screenshot shows the NuGet Package Manager interface for the 'WebApp.UITest' project. The 'Browse' tab is selected, and the search bar contains the text 'selenium'. The search results list several Selenium packages:

- Selenium.WebDriver** by Selenium Committers, 4.27M downloads. .NET bindings for the Selenium WebDriver API. Version v3.8.0. This item is highlighted with a red box.
- Selenium.Support** by Selenium Committers, 3.44M downloads. Provides support classes for Selenium WebDriver.
- Selenium.Chrome.WebDriver** by jbaranda, 205K downloads. Selenium Chrome WebDriver (Win32).
- Selenium.WebDriver.ChromeDriver** by jsakamoto, 1.5M download. Selenium Google Chrome Driver (Win32, macOS, and Linux64) (does not...)
- Selenium.Firefox.WebDriver** by jbaranda, 228K downloads. Selenium Firefox WebDriver Marionette (Win64).
- Selenium.WebDriver.IEDriver** by jsakamoto, 679K downloads. Selenium Internet Explorer Driver (Win32) (this package does not make you...)
- Selenium.WebDriverRanorexSelenium** by Selenium Committers, 45K. v3.8.0.

On the right side, the details for the selected 'Selenium.WebDriver' package are shown. The 'Version' is set to 'Latest stable 3.8.0'. The 'Install' button is highlighted with a red box.

and then install Selenium.PhantomJS.WebDriver (you'll probably need to scroll down the

results to find it)



4. Close the NuGet page.

5. Replace the contents of UnitTest1.cs with the following:

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium.Remote;
using OpenQA.Selenium.PhantomJS;

namespace WebApp.UITest
{
    [TestClass]
    public class UITests
    {
        private static RemoteWebDriver _webDriver = null;
        private static string _webAppBaseURL;

        [ClassInitialize()]
        public static void ClassInit(TestContext context)
        {
            _webDriver = new PhantomJSDriver();

            // Allow for web app compilation and startup post deployment
            _webDriver.Manage().Timeouts().ImplicitWait =
TimeSpan.FromSeconds(15);

            _webAppBaseURL = "https://<yourwebapp>.azurewebsites.net/";
        }
    }
}
```

```
[ClassCleanup()]
public static void Cleanup()
{
    if (_webDriver != null)
    {
        _webDriver.Quit();
    }
}

[TestMethod]
[TestCategory("UI")]
[TestCategory("Selenium")]
[TestCategory("PhantomJS")]
public void HomePageFoundTest()
{
    _webDriver.Url = _webAppBaseURL;

    string actualPageTitle = _webDriver.Title;
    string expectedPageTitle = "Home Page – My ASP.NET
Application";

    Assert.AreEqual(expectedPageTitle, actualPageTitle);
}

[TestMethod]
[TestCategory("UI")]
[TestCategory("Selenium")]
[TestCategory("PhantomJS")]
public void AboutPageFoundTest()
{
    _webDriver.Url = _webAppBaseURL + "/Home/About";

    string actualPageTitle = _webDriver.Title;
    string expectedPageTitle = "About – My ASP.NET Application";

    Assert.AreEqual(expectedPageTitle, actualPageTitle);
}

[TestMethod]
[TestCategory("UI")]
[TestCategory("Selenium")]
[TestCategory("PhantomJS")]
public void ContactPageFoundTest()
{
    _webDriver.Url = _webAppBaseURL + "/Home/Contact";

    string actualPageTitle = _webDriver.Title;
    string expectedPageTitle = "Contact – My ASP.NET Application";

    Assert.AreEqual(expectedPageTitle, actualPageTitle);
}

[TestMethod]
[TestCategory("UI")]
```

```
[TestCategory("Selenium")]
[TestCategory("PhantomJS")]
public void SupportEmailAddressChromeTest()
{
    string supportEmailAddress = "Support@example.com";

    _webDriver.Url = _webAppBaseURL + "/Home/Contact";
    RemoteWebElement supportEmailElement =
(RemoteWebElement)_webDriver.FindElementByLinkText(supportEmailAddress);

    Assert.AreEqual(supportEmailAddress,
supportEmailElement.Text);
}

[TestMethod]
[TestCategory("UI")]
[TestCategory("Selenium")]
[TestCategory("PhantomJS")]
public void MarketingEmailAddressTest()
{
    string marketingEmailAddress = "Marketing@example.com";

    _webDriver.Url = _webAppBaseURL + "/Home/Contact";
    RemoteWebElement marketingEmailElement =
(RemoteWebElement)_webDriver.FindElementByLinkText(marketingEmailAddress);

    Assert.AreEqual(marketingEmailAddress,
marketingEmailElement.Text);
}

[TestMethod]
[TestCategory("UI")]
[TestCategory("Selenium")]
[TestCategory("PhantomJS")]
public void IndexTitleTest()
{
    string expectedTitle = "Another New Title";

    _webDriver.Url = _webAppBaseURL + "/Home/Index";
    RemoteWebElement titleElement =
(RemoteWebElement)_webDriver.FindElementByName("H1");

    Assert.AreEqual(expectedTitle, titleElement.Text);
}
}
```

6. This code contains six tests that will be executed against the web application using the Selenium PhantomJS driver. The tests check that pages and content on those pages exist.
Note:

- You need to add your QA web app url into line 22 (the one you can find as per the last step in Lab 5 above):

```
_webAppBaseUrl = "https://<yourwebapp>.azurewebsites.net/";
```

- The final test (IndexTitleTest) checks the heading on the home page that you have been changing in the previous labs. Make sure that the expected value is correct for your web app (line 110).

```
string expectedTitle = "Another New Title";
```

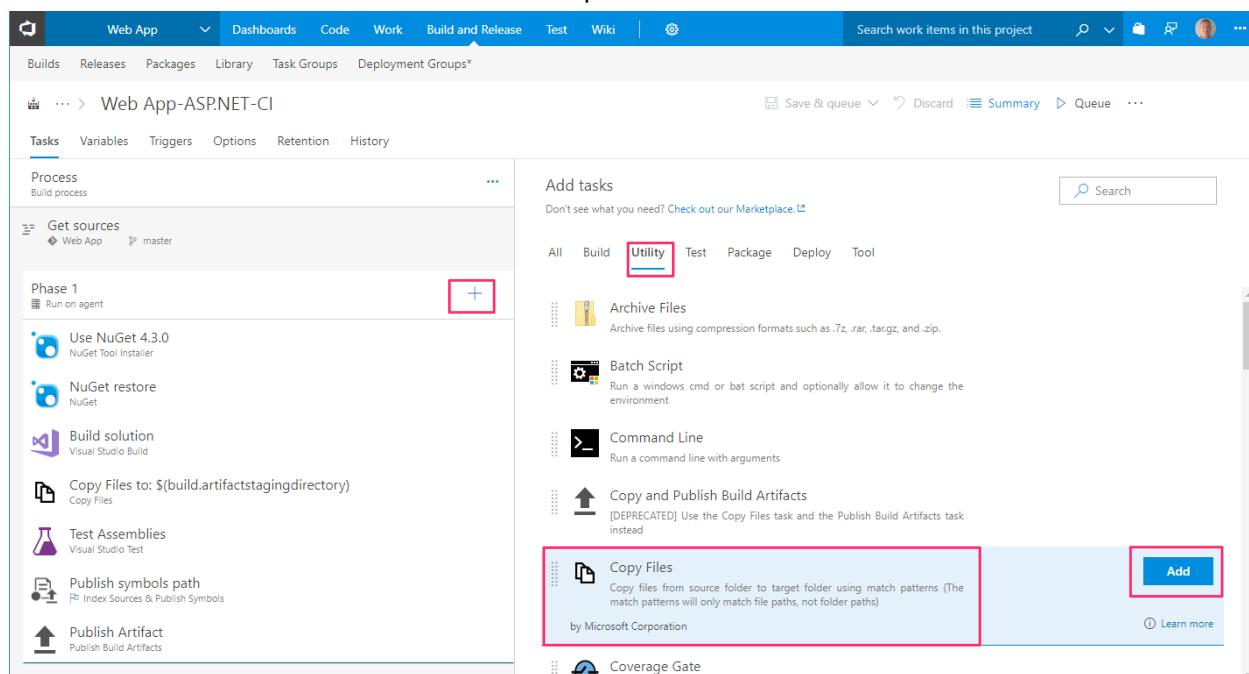
- The [TestCategory] properties which will be used later.

7. Save all files.

You now have some Selenium tests in the project. Before committing these to source control the next step is to amend the VSTS CI build and CD release to incorporate the tests into the flow.

Task 2: Add the tests into the pipeline

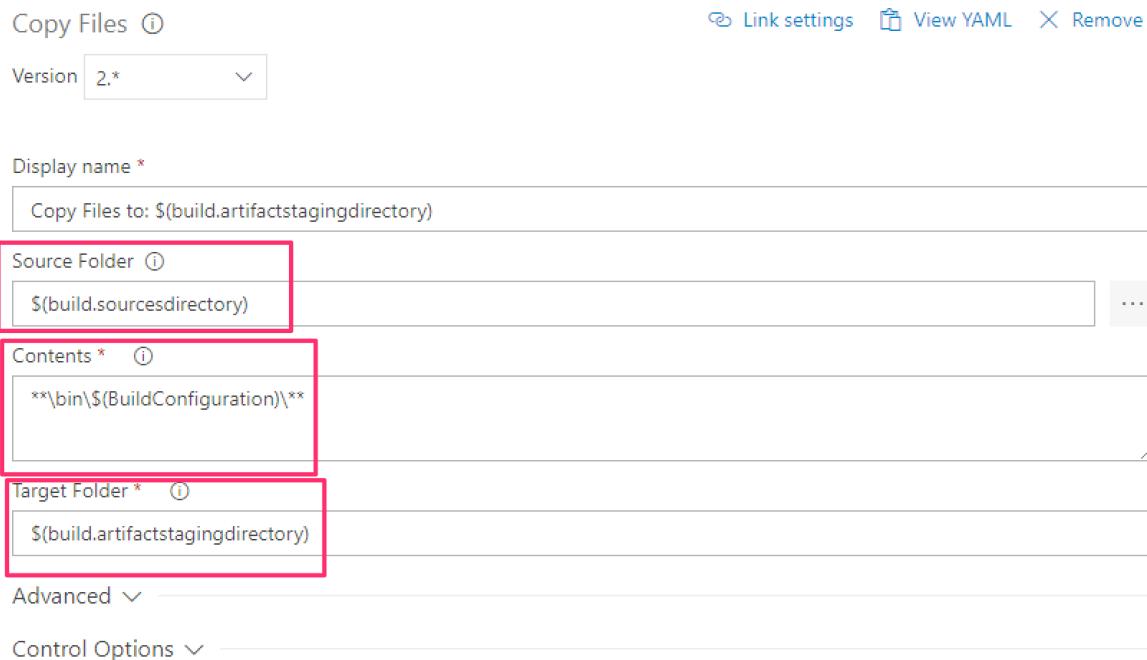
1. First the build definition needs to ensure that the tests are available in the build output, so that they can be run in the release. Open the build definition (Build and Release | Builds | Edit).
2. In Phase 1, click + select the Utility tab and select the Copy Files task. This task will ensure that the tests are available in the build output to be used in the release.



3. Set the following values in the Copy Files task:

- Source Folder: \$(build.sourcesdirectory)

- Contents: **bin\$(BuildConfiguration)**
- Target Folder: \$(build.artifactstagingdirectory)
- The Display name will update as you make changes.



4. Move the Copy Files task to be after the Build solution task. The tests will now be available in the build output the next time the build runs.
5. In this example we want the CI build to continue to only run unit tests and not the new Selenium tests. Therefore update the Test Assemblies task and add `TestCategory!=PhantomJS` in the Test filter criteria field. This uses the TestCategories in the test code to filter the tests to be run.

The screenshot shows the CI build pipeline. The 'Test Assemblies' task is selected and has a red box around its 'Test filter criteria' field, which contains 'TestCategory!=PhantomJS'. Other fields shown include 'Run only impacted tests' and 'Test mix contains UI tests'.

6. Save (but not queue) the build.

7. The next step is to execute the tests as part of the release. Open the Release definition (Build and Release | Releases | Edit) and click on the tasks for the QA environment. Click the + button, select the Test tab and select the Visual Studio Test task. The Selenium tests are within a unit test so this task can execute them.

The screenshot shows the Azure DevOps interface for a release definition named "Web App-ASP.NET-CI - CD". The "Tasks" tab is selected. In the "Add tasks" dialog, the "Test" tab is highlighted. A specific task, "Visual Studio Test", is highlighted with a red box. An "Add" button is also highlighted with a red box.

8. Select the test task. In the Test filter criteria field add TestCategory=PhantomJS. This will only execute tests that have the matching [TestCategory] property in the code and provides control over which tests to run. In this example we have decided not to run the Selenium tests in the CI build but want to run them in the QA environment.

The screenshot shows the "Visual Studio Test" task configuration in the release definition. The "Test filter criteria" field contains the value "TestCategory=PhantomJS".

9. Ensure the test task is the last task and save the release.
10. Now return to Visual Studio and the Team Explorer. You may need to click the back arrow or home button in Team Explorer. Then commit and push the changes. This will trigger the CI build, which will now include the tests in the output, and then trigger the CD release, which will execute the tests found in the build output. Wait for the build and release into QA to complete and then open the release summary. You should see that there are tests in the QA environment and that they have passed.

Web App-ASP.NET-CI - CD / Release-10

Summary Environments Artifacts Variables General Commits Work items Tests Logs History

⟳ | ⚡ Deploy Save Abandon Send Email

Details		Work items	
Triggered by Web App-ASP.NET-CI 20180123.12. Edit		No associated work items found.	
Continuous deployment requested for Giles Davies 4 minutes ago		Tags	
Web App-ASP.NET-CI / 20180123.12 (Build) ↗ master		Add...	
Web App / 12e8e2e7 (Added Selenium tests) (Git) ↗ master			
Environments			
Environment	Actions	Deployment status	Triggered
Dev	...	SUCCEEDED	4 minutes ago
QA	...	SUCCEEDED	3 minutes ago
			Completed just now 100%
Tests			

11. Click on the 100% pass link for the QA environment and you will see the individual test results. By default the view is filtered to only show failed tests, select All outcomes to see the

tests.

Web App-ASP.NET-CI - CD / Release-10

Summary Environments Artifacts Variables General Commits Work items **Tests** Logs History

Deploy Save Abandon Send Email

Total tests	Pass percentage	Run duration
 6 Passed (6) Failed (0) Others (0)	100%	38s 20ms

Group by Test run Outcome All

Test	Failing since	Failing release	Duration
6/6 Passed - VSTest Test Run			0:00:19.260
AboutPageFoundTest			0:00:00.223
ContactPageFoundTest			0:00:00.237
HomePageFoundTest			0:00:17.984
IndexTitleTest			0:00:00.247
MarketingEmailAddressTest			0:00:00.273
SupportEmailAddressChromeTest			0:00:00.297

You now have automated UI tests being executed everytime a new version of your application is deployed.

Optional: You might want to see what happens when a test fails. You could change the heading for the web app again, but not update the tests, and commit the change. You should then see a successful CI build (as the UI Tests aren't run in the build), a successful release to Dev but a failed release to QA (as that's the environment we chose to run the UI Tests in).

Optional: Add a Deployment status widget showing test results to the Lab Progress dashboard by:

- Searching for and adding the Deployment status widget:

The screenshot shows the 'Lab Progress' dashboard in Azure DevOps. On the left, there are cards for 'Web App Project' and 'Web App-ASP.NET-CI'. The 'Web App-ASP.NET-CI' card displays a green bar chart. In the center, there's a card for 'Another New Title' with text about ASP.NET. On the right, the 'Add Widget' modal is open. A red box highlights the 'Deployment status' card, which includes a description: 'Shows the deployment and test status of a branch across the environments in your release definitions.' Below the card is a 'Learn more' button. At the bottom right of the modal are 'Add' and 'Close' buttons.

- Set the Build definition and Linked release definition including all environments:

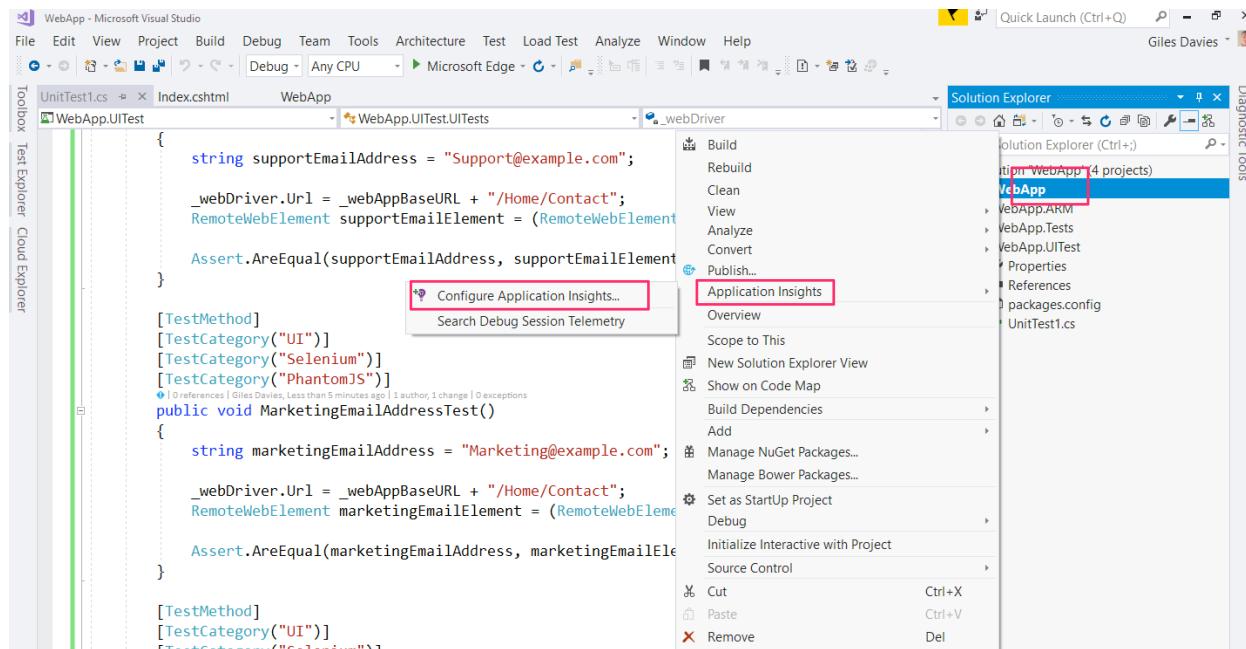
The screenshot shows the 'Lab Progress' dashboard with the 'Deployment status' configuration modal open. The 'Build definition' dropdown is set to 'Web App-ASP.NET-CI'. The 'Linked release definitions' dropdown shows 'Web App-ASP.NET-CI - CD: Web App-ASP.NET-CI - CD' with 'Dev' and 'QA' environments selected. A red box highlights the 'Save' button at the bottom right. Other sections like 'Configuration' and 'Branch' are also visible.

- Save the changes, close the widget gallery and save the dashboard by clicking on the blue edit button in the bottom right hand corner.

Lab 7: Monitoring with Application Insights

DevOps doesn't stop at deployment into production. Monitoring and understanding your application provides valuable insight.

1. In Visual Studio right-click on the WebApp project | Application Insights | Configure Application Insights



2. Update the SDK, if needed, and click Start Free.

The screenshot shows the Microsoft Application Insights landing page. At the top, there's a navigation bar with tabs: 'Application Insights Configuration' (selected), 'UnitTest1.cs', 'Index.cshtml', and 'WebApp'. Below the navigation, there's a large heading 'Application Insights' with a lightbulb icon. A section titled 'SDK' contains a message: 'A new version of Microsoft.ApplicationInsights.Web (v2.4.1) is available. Your project is using v2.2' and a blue 'Update SDK' button, which is highlighted with a red box. Another section titled 'Gain insights through telemetry, analytics and smart detection' lists three features: 'Detect' (with a lightbulb icon), 'Monitor' (with a chart icon), and 'Integrate' (with a gear icon). At the bottom, there's a large blue 'Start Free' button, which is also highlighted with a red box.

3. Set the correct account, subscription and resource - for resource that will probably be a New resource. Also decide whether you want Application Insights to be capped at the free tier or

allow it to exceed the free tier limits. If in doubt select to halt collection at the free limits.

The screenshot shows the Azure Application Insights Pricing page. At the top, there are three dropdown menus: 'Account' (set to Microsoft), 'Subscription' (set to Azure On Line Services), and 'Resource' (set to WebApp (New resource)). Below these, there is a link 'Configure settings...'. The page then displays the following pricing information:

Base Monthly Price	Free
Included Data	1 GB / Month
Additional Data	\$2.30 per GB*
Data retention (raw and aggregated data)	90 days

*Pricing is subject to change. Visit our [pricing page](#) for most recent pricing details.

Below this, there are two radio button options:

- Allow Application Insights to collect data beyond 1GB/Month.
- Application Insights will remain free and halt data collection after 1GB/Month.

A large blue 'Register' button is centered at the bottom of the page.

4. Commit and push the changes. This will trigger another build and release.

The screenshot shows Microsoft Visual Studio with the 'WebApp' project open. In the center, the 'Application Insights Configuration' tab is selected, showing the 'Resource Settings' section. It includes fields for 'Sending telemetry to' (WebApp in Default-ApplicationInsights-EastUS) and 'CodeLens and Diagnostic Tools are reading telemetry from' (WebApp as Giles Davies). Below this, there's a progress bar for 'Configured' at 80%, with status items: 'SDK added', 'App registered with Application Insights', and 'Exception collection enabled'.

To the right, the 'Team Explorer - Changes' pane is visible, showing a commit history for the 'master' branch. The latest commit is highlighted with a red box and labeled 'Added Application Insights'. A context menu is open over this commit, with the 'Commit All and Push' option also highlighted with a red box.

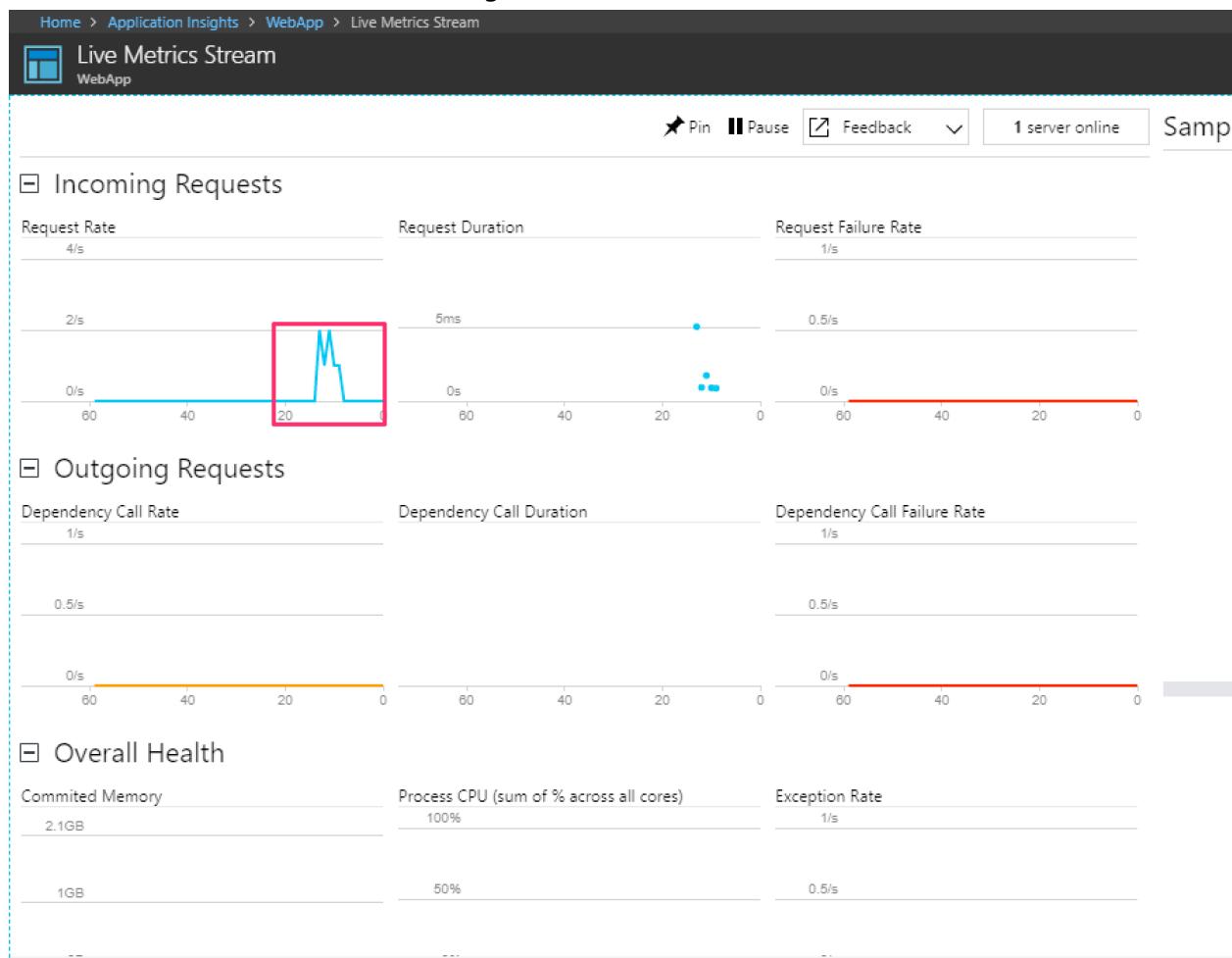
5. When the release to QA has completed go to the [Azure portal](#) select the Application Insights service for the Web App (it may take a little while for data to show). Click on the Live Stream button.

The screenshot shows the Azure Application Insights Overview page for a resource named 'WebApp'. The 'Live Stream' button in the top navigation bar is highlighted with a red box. The main dashboard displays key metrics: 0 Alerts, 1 Server, 0 Users, 0 Detections, and 0 Availability. A detailed timeline chart for 'WEBAPP' shows a single data point with a server response time of 883.3 ms. Below the timeline, a bar chart indicates 6 failed requests. The left sidebar contains links for Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems.

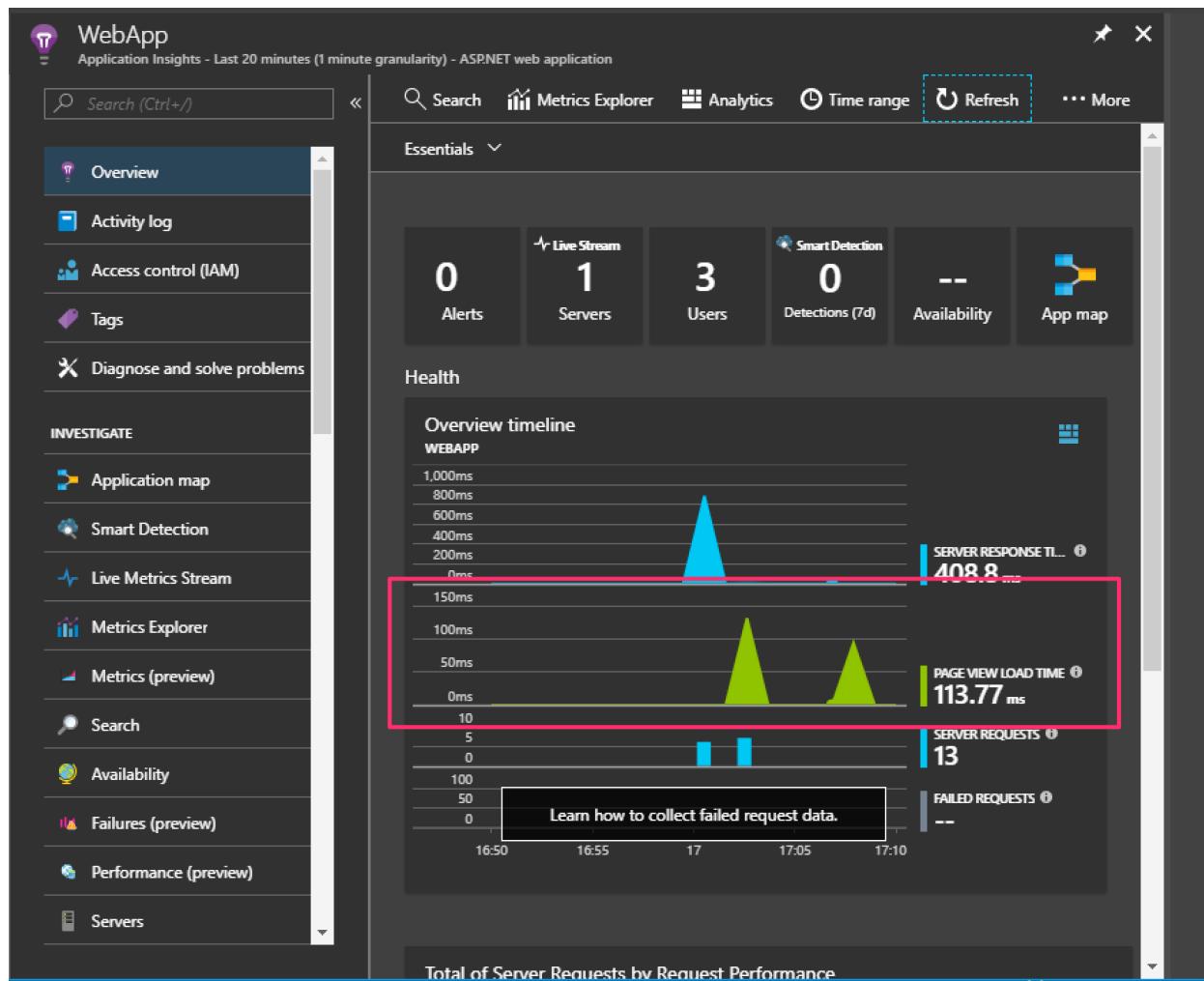
6. Give it a few seconds for some baseline data to show.

The screenshot shows the 'Live Metrics Stream' page for the 'WebApp' resource. It displays three main sections: 'Incoming Requests', 'Outgoing Requests', and 'Overall Health'. Each section contains three metrics represented by graphs. In the 'Incoming Requests' section, the 'Request Duration' graph is highlighted with a red box. In the 'Outgoing Requests' section, the 'Dependency Call Duration' graph is highlighted with a red box. In the 'Overall Health' section, the 'Process CPU (sum of % across all cores)' graph is highlighted with a red box. The graphs show various performance metrics over time, such as request rates, failure rates, and dependency call durations.

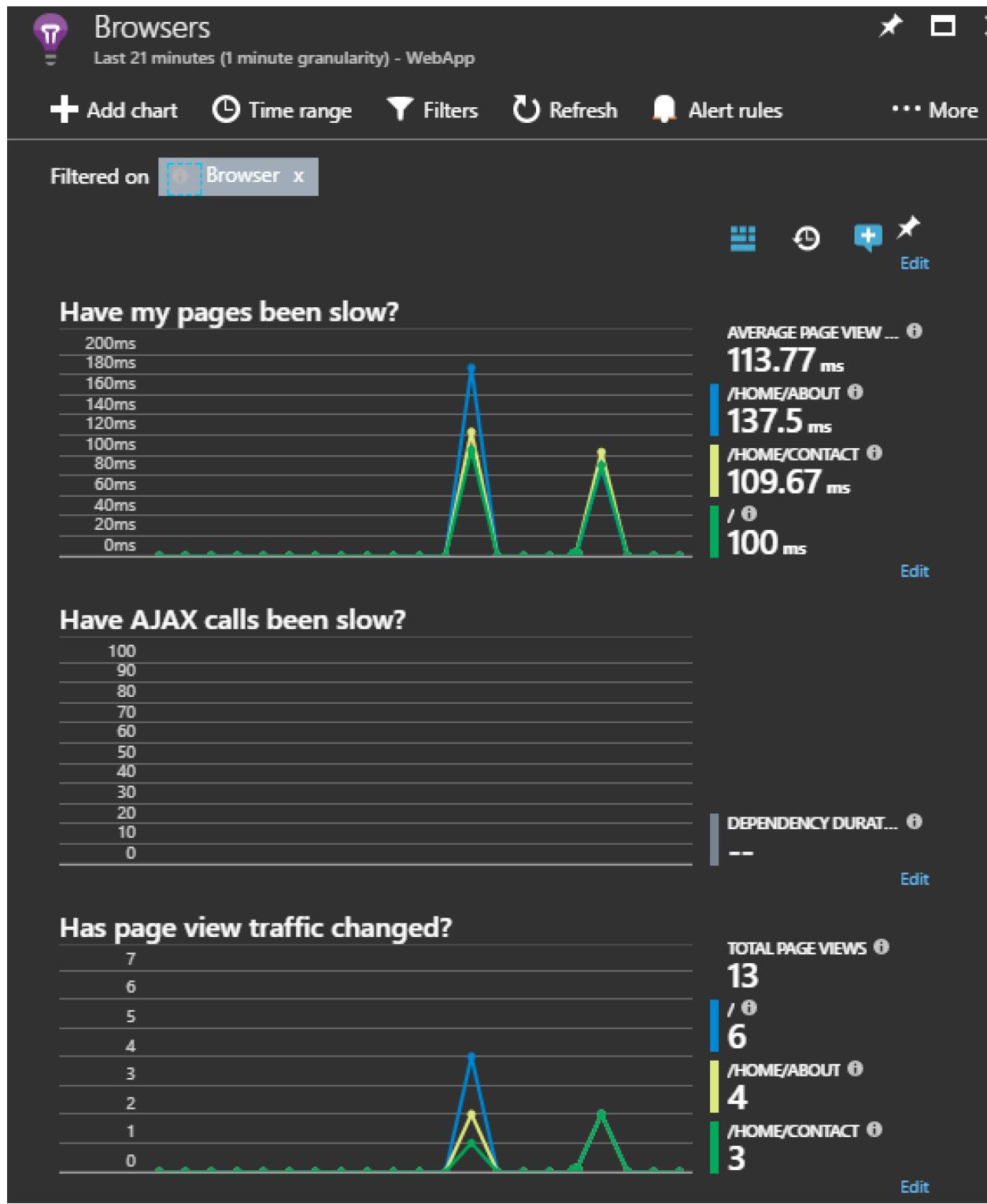
7. Go to your deployed app and click on the Home, About and Contact pages, then return to the Live Stream to see the data showing.



8. Close the Live Stream and click on the Page View Load Time graph.



9. Take a look at data on page response time

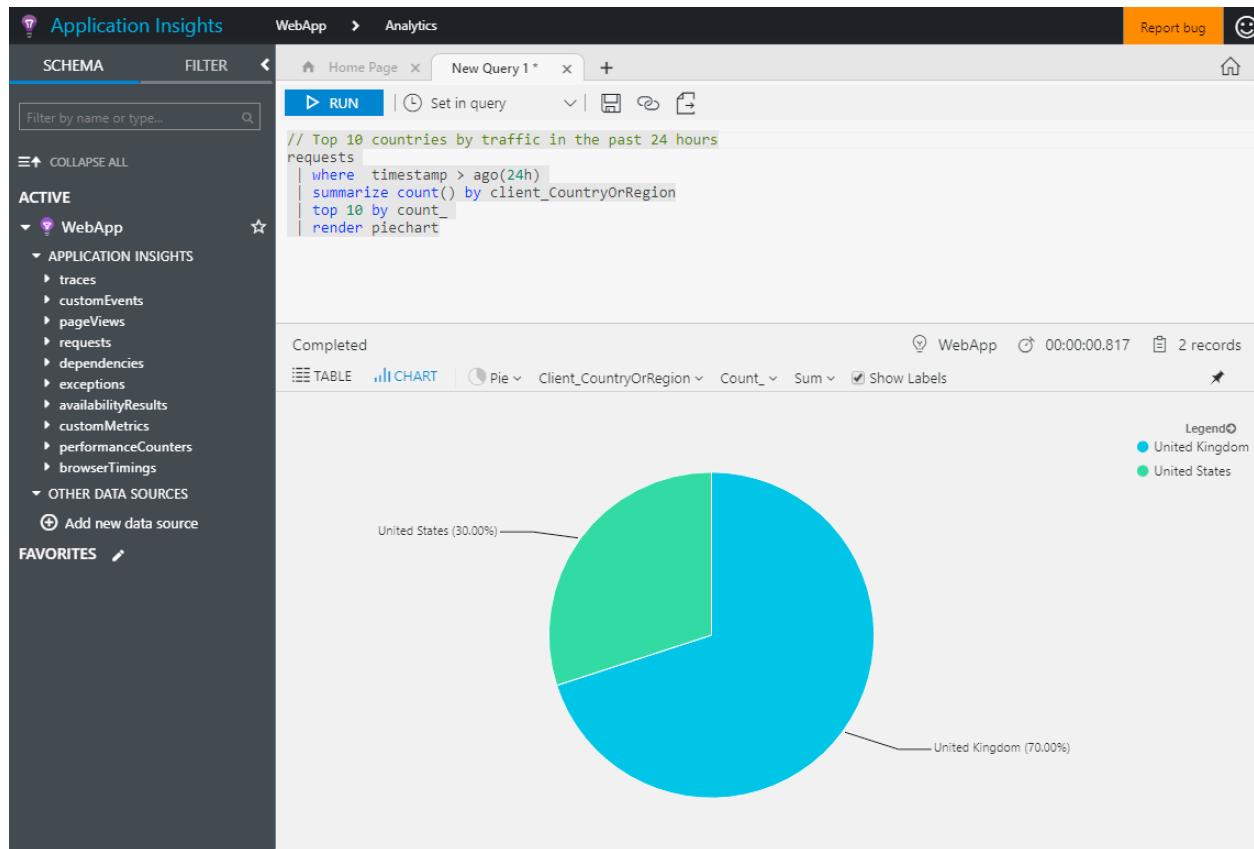


10. Close the Page response time blade and select Analytics.

The screenshot shows the Application Insights Analytics blade for a WebApp application. The top navigation bar includes 'Search (Ctrl+)', 'Metrics Explorer' (highlighted with a red box), 'Analytics' (highlighted with a red box), 'Time range', 'Refresh', and 'More'. On the left, a sidebar lists 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'INVESTIGATE', 'Application map', 'Smart Detection', 'Live Metrics Stream', and 'Metrics Explorer'. The main area displays 'Essentials' metrics: 0 Alerts, 1 Live Stream, 3 Servers, 0 Users, 0 Detections (7d), Availability (--), and App map. Below this is a 'Health' section with an 'Overview timeline' for 'WEBAPP' showing a server response time of 408.8 ms. A large blue spike is visible on the timeline graph.

11. Try some of the queries such as the Users query

The screenshot shows the Application Insights Analytics blade for a WebApp application. The left sidebar shows 'ACTIVE' sections for 'WebApp' and 'APPLICATION INSIGHTS' (traces, customEvents, pageViews, requests, dependencies, exceptions, availabilityResults, customMetrics, performanceCounters, browserTimings). The main area features a video player with the title 'Application Insights Analytics' and a subtitle 'Gain deep insights to your applications and users.' Below the video are four 'COMMON QUERIES': 'USERS' (Top 10 countries by traffic in the past 24 hours, highlighted with a red box), 'PERFORMANCE' (What are the 50th, 90th, and 95th percentiles of request duration in the past 24 hours), 'ERRORS' (Find what exceptions led to failed requests in the past 24 hours), and 'USAGE' (What are the top 10 custom ev...). Each query has a 'RUN' button.



This is just a tiny sample of what can be done with Application Insights, you can [read more here](#)

Optional: Add an Application Insights widget showing test results to the Lab Progress dashboard by:

- Find and install the Application Insights widget in the [marketplace](#)

The screenshot shows the Visual Studio Marketplace page for the 'Azure Application Insights Widgets' extension. The extension is developed by Microsoft and has 3,821 installs with a rating of 4.5 stars from 7 reviews. It is described as a free tool to show Application Insights charts and metrics on VSTS dashboards. A prominent green 'Get it free' button is highlighted with a red box.

Below the main description, there are tabs for 'Overview', 'Q & A', and 'Rating & Review'. The 'Overview' tab contains a section titled 'Be proactive' which explains how to monitor and detect issues in apps and services without leaving VSTS. It also shows two preview cards: one for 'Failed requests (sum)' with a value of 927 for the last hour, and another for 'ASP.NET request execution time (avg)' with a value of 293 ms for the last 30 days. To the right, there are configuration fields for 'Title' (set to 'My App'), 'Application Id' (set to '12345678-1234'), and 'API Key' (a placeholder field).

- Searching for and adding the Application Insights Chart (and/or Metrics) widget:

The screenshot shows the VSTS dashboard interface. On the left, there are several cards: 'Web App Project' (Reference project for the DevOps Lab. Feedback and testing from: Mark Harrison, Tamara Walther), 'Web App-ASP.NET-CI' (with a green bar chart), 'Release Definition Overview' (Dev, QA, Release-12, Release-11, Release-10, Release-9, Release-8), and 'Deployment status' (Environments: Dev, QA; Builds: ...123.10, ...123.11, ...123.12, ...123.13, ...123.14). On the right, a modal window titled 'Add Widget' is open. It contains a search bar with 'insight' typed into it. Below the search bar, a card for 'Azure Application Insights Cha...' is shown, featuring a purple icon with a lightbulb and a line graph, and text about monitoring app performance. This card is highlighted with a red box. At the bottom of the modal, there is an 'Add' button, which is also highlighted with a red box.

- Configure the widget following these instructions

The screenshot shows the VSTS dashboard 'Lab Progress'. On the left, the 'Release Definition Overview' and 'Deployment status' cards are visible. On the right, the 'Azure Application Insights Chart Widget' is displayed, showing a chart with the average browser page load time as 1.36 seconds over the last 12 hours. A lightbulb icon is present on the chart, indicating it can be interacted with. The entire chart area is highlighted with a red box.

- Save the changes, close the widget gallery and save the dashboard by clicking on the blue edit button in the bottom right hand corner.

Other tasks

This lab outlines the key practices in implementing a DevOps pipeline but there are many other tasks that could be incorporated into the flow including:

- Setting pre and post approvals on release environments
- Using variables in releases across environment
- Adding load testing to the flow
- Linking changes to user stories and other work items to understand what has been built and released.

- Using Git branches and merging via Pull Requests

Other labs

If you'd like to continue exploring DevOps with VSTS then related labs include:

[A range of VSTS Labs from Agile to Testing](#)

[Node.js and Express continuous deployment with Visual Studio Team Services and Azure App Service](#)

[DevOps with Visual Studio Team Services for Java](#)