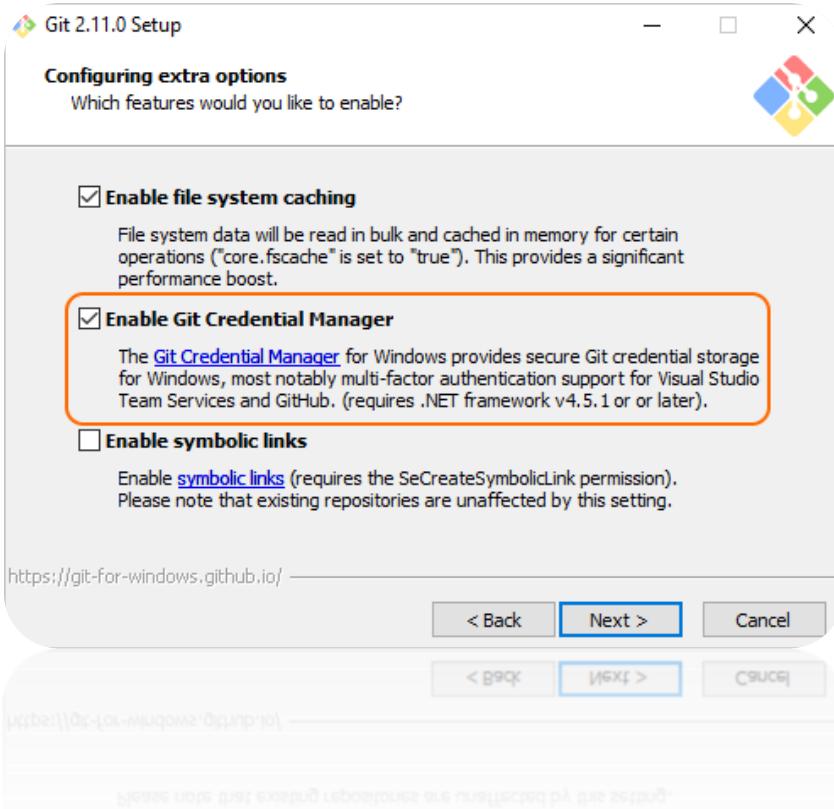


1. Pre-requisites

This lab assumes you have the Google Chrome browser installed and available for debugging. If you do not have Chrome installed, go to <https://www.google.com/chrome/browser/>

Download and run the latest [Git for Windows installer](#), which includes the Git Credential Manager for Windows. Make sure to leave the Git Credential Manager installation option enabled when prompted.



Note: When you connect to a VSTS Git repository from your Git client for the first time, the credential manager prompts for your Microsoft Account or Azure Active Directory credentials. If your account has multi-factor authentication enabled, you are prompted to go through that experience as well.

Download Azure CLI here: <https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?view=azure-cli-latest>

Install Azure CLI 2.0 on Windows

01/29/2018 • 2 minutes to read • Contributors

On Windows the Azure CLI binary is installed via an MSI, which gives you access to the CLI through the Windows Command Prompt (CMD) or PowerShell. If you are running Windows Subsystem for Linux (WSL), there are packages available for your Linux distribution. See the [main install page](#) for the list of supported package managers or how to install manually under WSL.

Install or update

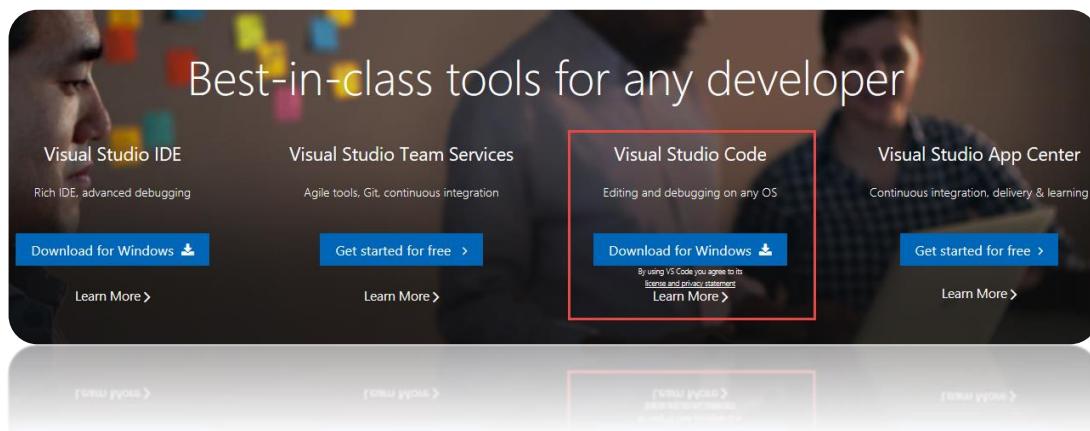
The MSI distributable is used for installing, updating, and uninstalling the `az` command on Windows.

[Download the MSI installer >](#)

When the installer asks if it can make changes to your computer, click the "Yes" box.

You can now run the Azure CLI with the `az` command from either Windows Command Prompt or PowerShell. PowerShell offers some tab completion features not available from CMD.

Download Visual Studio Code from <http://visualstudio.com>

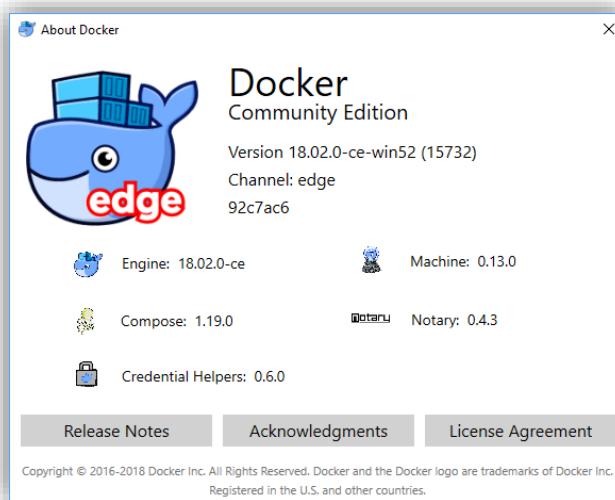


Install Docker from <https://docs.docker.com/install/>



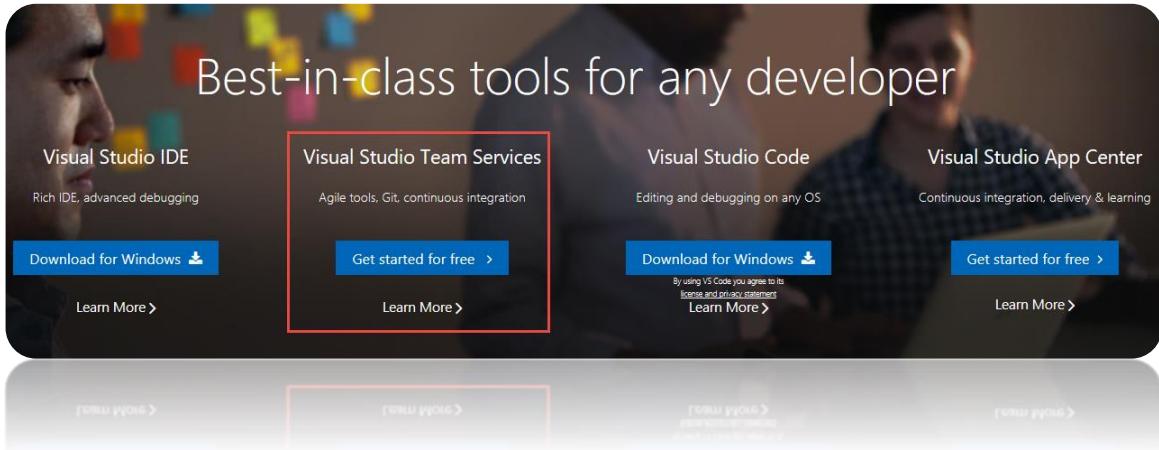
The screenshot shows the Docker Community Edition for Windows page on the dockerstore.com website. At the top, there's a navigation bar with links for Explore, Publish, Feedback, and Log In. Below the navigation is a search bar. The main content area features a large blue icon of a whale carrying shipping containers. To its right, the text "Docker Community Edition for Windows" is displayed, followed by "By Docker". A subtitle reads "The fastest and easiest way to get started with Docker on Windows". Below this, there are three small buttons: "Edition", "Windows", and "x86-64". On the right side of the main content, there's a section titled "Get Docker Community Edition for Windows" with a note that it's available for free. It also specifies compatibility with Microsoft Windows 10 Professional or Enterprise 64-bit. A link to the Docker Software End User License Agreement is provided. At the bottom of this section are two buttons: "Get Docker" and "Usage Instructions".

Note: Make sure you install Docker “Edge” for windows, not the “Stable” release. This guide has been verified against the following Docker version:

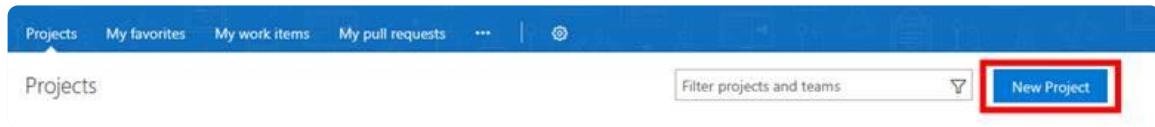


2. Create a Project in VSTS

1. Create a new instance of Visual Studio Team Services by navigating to <http://visualstudio.com>



2. Click on "**New Project**" in VSTS.



3. Enter Project Name, Description, Version control, and Work item process and click **Create**.

Create new project

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

Project name *

 ✓

Description

Version control

 ?

Work item process

 ?

Create

Cancel

Create

Cancel

4. Select “or initialize with a readme or gitignore”.
5. Add a .gitignore file by selecting “Node”,
6. Click Initialize.



Demo ☆

Briefly describe your project...

Add tags

Get started with your new project!

- ▽ Clone to your computer
- ▽ or push an existing repository from command line
- ▽ or import a repository
- △ or initialize with a README or gitignore

Add a README

Add a .gitignore: Node ▾

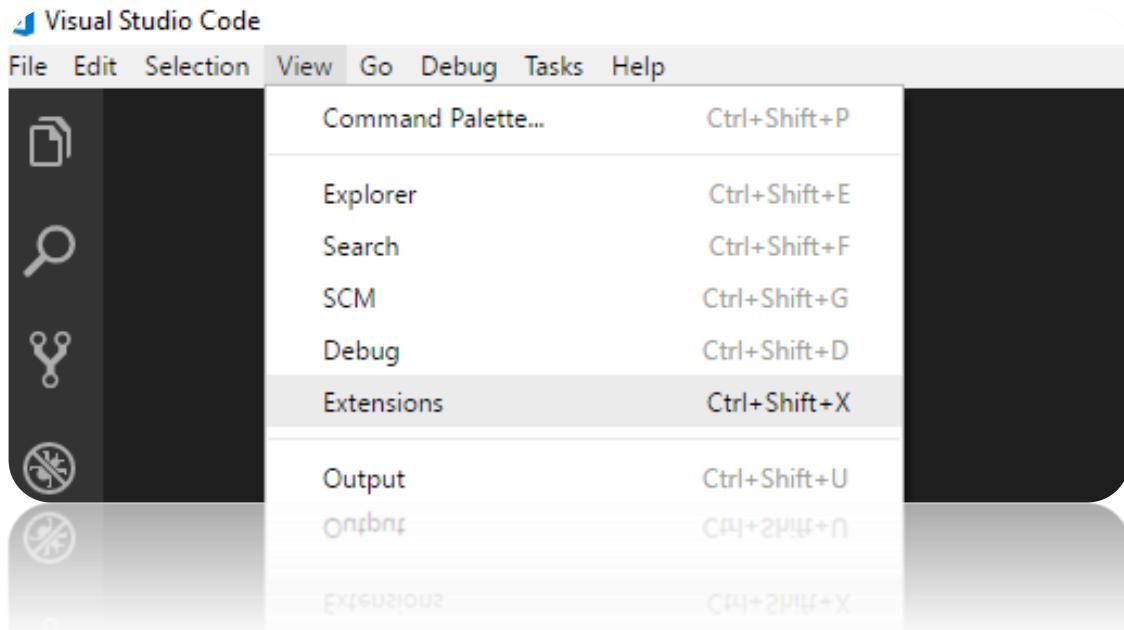
Initialize

-
- ▽ or build code from an external repository

Note: Readme file is used to give a brief introduction of the project and gitignore file is used to ignore tracking of files such as temp files and build results.

3. Open Visual Studio Code

1. Install Extensions by Selecting View → Extensions and typing "javascript"



Recommended extensions to install:

Angular 5 and TypeScript/HTML VS Code Snippets
Angular 5 Snippets - TypeScript, Html, Angular Material, ngRx, RxJS & Flex Layout
ESLint
JavaScript (ES6) code snippets
npm IntelliSense
Debugger for Chrome
Visual Studio Team Services
Docker
Docker Explorer
Nginx.Conf
Nginx.Conf Hint
Apache conf
Apache Conf Snippets

2. Launch Git Bash or use Windows Command line to execute the following commands to create our repository directory:

```
MINGW64:/c/shoppingcartdemo
codec@DESKTOP-GFGMI69 MINGW64 /c
$ cd /c
codec@DESKTOP-GFGMI69 MINGW64 /c
$ mkdir shoppingcartdemo
codec@DESKTOP-GFGMI69 MINGW64 /c
$ cd shoppingcartdemo/
codec@DESKTOP-GFGMI69 MINGW64 /c/shoppingcartdemo
$
```

3. Open your VSTS project in your browser
4. Click on Clone in the upper right-hand corner
5. Generate Git Credentials:

Clone repository

Clone Git repository using command line or IDE

Command line

HTTPS

SSH

https://mtctor.visualstudio.com/_git/Demo



[Generate Git credentials](#)

IDE

[Clone in Visual Studio](#)



Having problems authenticating in Git? Be sure to get the latest version of [Git for Windows](#) or our plugins for [IntelliJ](#), [Eclipse](#), [Android Studio](#) or [Windows command line](#).

6. Then enter a new password and click Save Git Credentials:

Clone repository

Clone Git repository using command line or IDE

Command line

HTTPS SSH

User name (primary)

Alias (optional)

Password *

Confirm Password *

Create a Personal access token

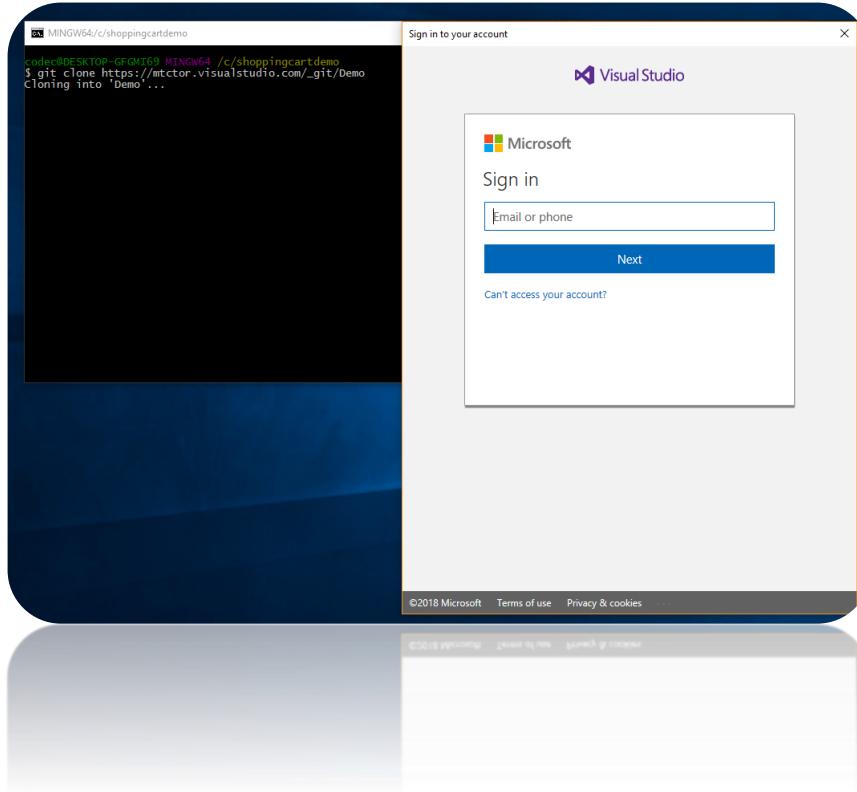
7. Copy the git repository url as follows:

The screenshot shows the Microsoft DevOps interface. The 'Code' tab is selected in the top navigation bar. In the main content area, there's a 'Clone repository' sidebar on the right. The 'HTTPS' URL field is highlighted with a red box and contains the URL https://mtctor.visualstudio.com/_git/Demo. The URL field has a small icon with a clipboard and a copy symbol.

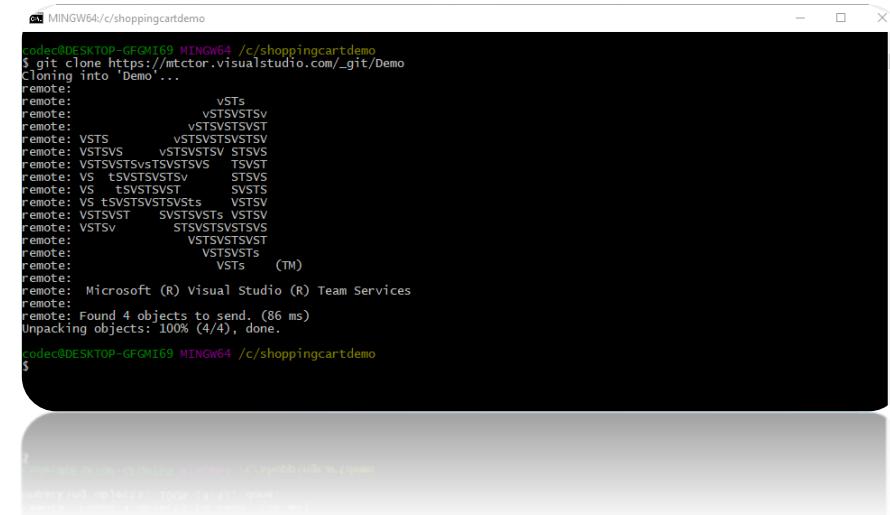
8. Clone the repository from the bash shell you opened earlier as follows:

```
Git clone <git Repository you copied in previous step>
```

9. Enter your credentials you setup in previous steps



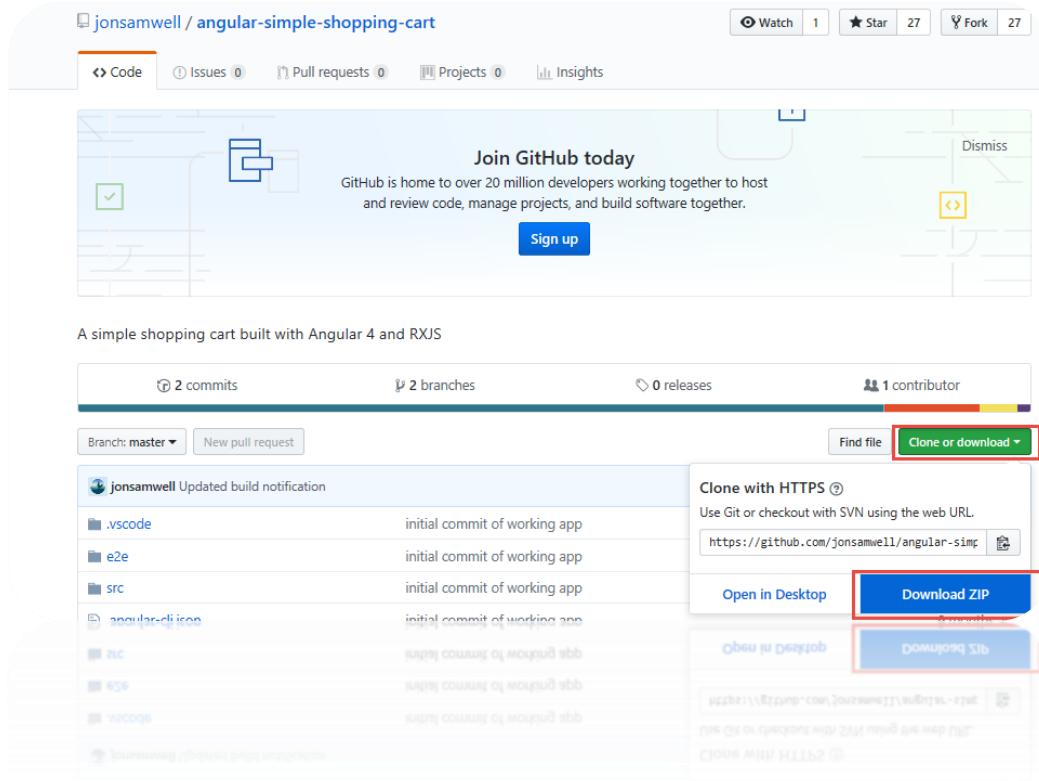
10. After successful login you should see:



4. Write some code...

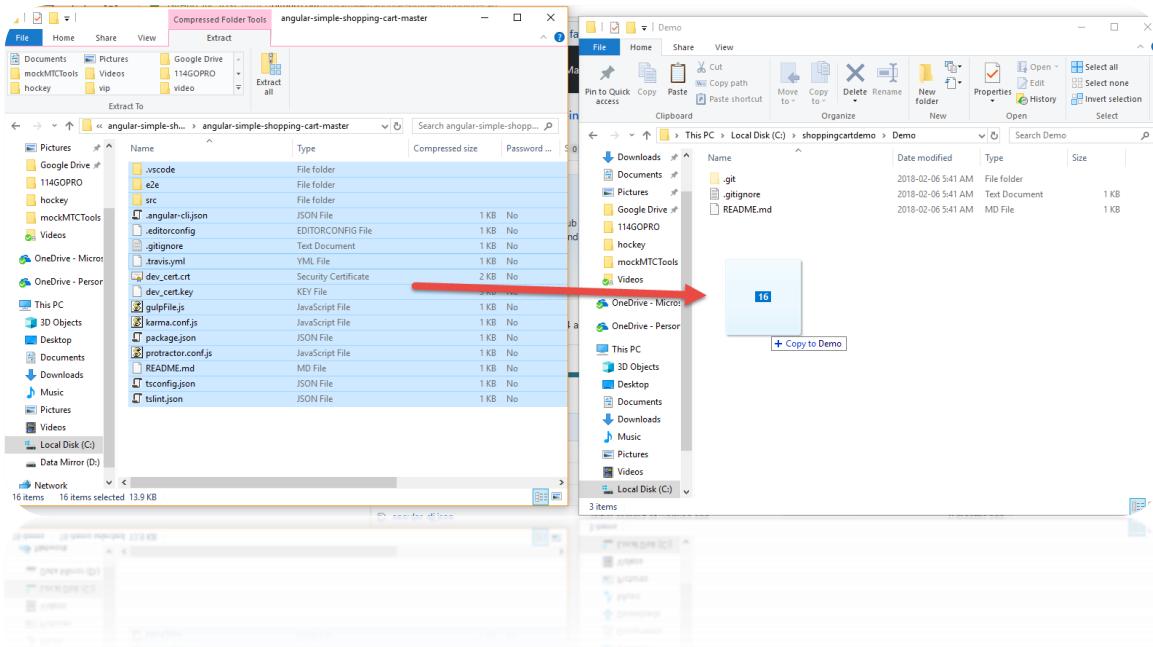
Not quite, we are just going to use an existing code base from GitHub and download the latest copy of the source to update our local repo.

1. Open the browser and navigate to <https://github.com/jonsamwell/angular-simple-shopping-cart>
2. Download code as follows:



- Extract the contents of the "angular-simple-shopping-cart-master" folder within the zip file to c:\shoppingcartdemo\demo

Note: answer "replace" when duplicate files found.



- Now we are going to add untracked files and commit our changes to our local repository, but before we can do that we have to tell Git who we are by issuing the two following commands:

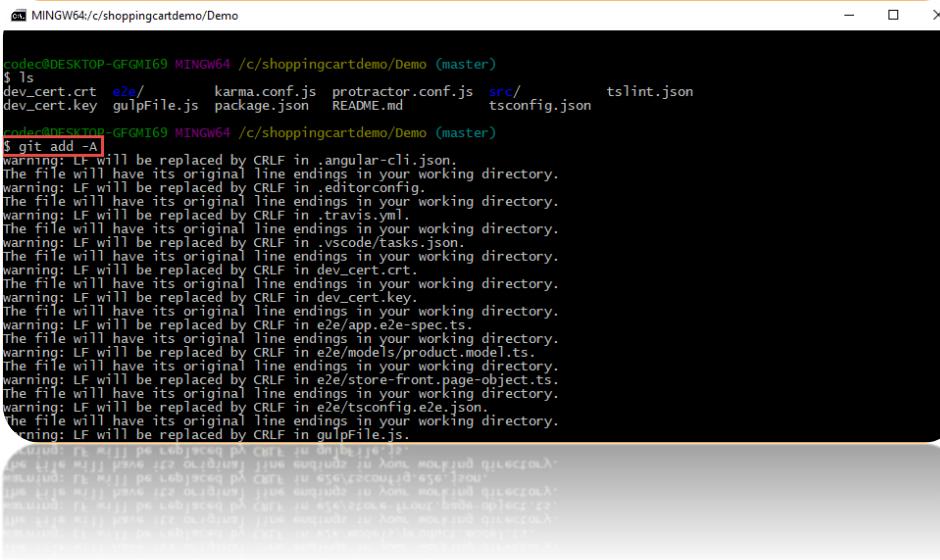
```
git config --global user.email "you@outlook.com"
```

```
git config --global user.name "Your Name"
```

```
codec@DESKTOP-GFGMI69 MINGW64 /c/shoppingcartdemo/Demo (master)
$ git config --global user.email "marfra@microsoft.com"
codec@DESKTOP-GFGMI69 MINGW64 /c/shoppingcartdemo/Demo (master)
$ git config --global user.name "Mark Franco"
codec@DESKTOP-GFGMI69 MINGW64 /c/shoppingcartdemo/Demo (master)
$
```

5. Add untracked files as follows:

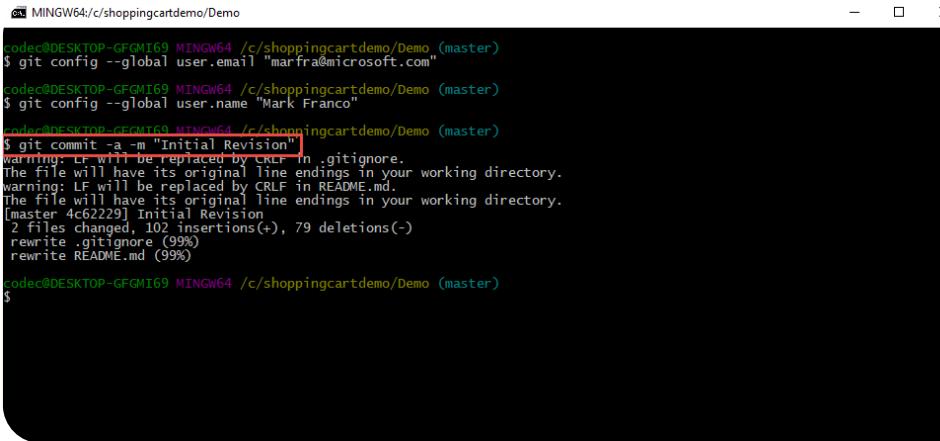
```
cd \Demo  
git add -A
```



```
codec@DESKTOP-GFGMI69 MINGW64 /c/shoppingcartdemo/Demo (master)  
$ ls  
dev_cert.crt  e2e/      karma.conf.js protractor.conf.js  src/      tsconfig.json  
dev_cert.key  gulpfile.js package.json  README.md    tslint.json  
codec@DESKTOP-GFGMI69 MINGW64 /c/shoppingcartdemo/Demo (master)  
$ git add -A  
warning: LF will be replaced by CRLF in .angular-cli.json.  
The file will have its original line endings in your working directory.  
warning: LF will be replaced by CRLF in .editorconfig.  
The file will have its original line endings in your working directory.  
warning: LF will be replaced by CRLF in .travis.yml.  
The file will have its original line endings in your working directory.  
warning: LF will be replaced by CRLF in .vscode/tasks.json.  
The file will have its original line endings in your working directory.  
warning: LF will be replaced by CRLF in dev_cert.key.  
The file will have its original line endings in your working directory.  
warning: LF will be replaced by CRLF in dev_cert.crt.  
The file will have its original line endings in your working directory.  
warning: LF will be replaced by CRLF in e2e/app.e2e-spec.ts.  
The file will have its original line endings in your working directory.  
warning: LF will be replaced by CRLF in e2e/models/product.model.ts.  
The file will have its original line endings in your working directory.  
warning: LF will be replaced by CRLF in e2e/store-front.page-object.ts.  
The file will have its original line endings in your working directory.  
warning: LF will be replaced by CRLF in e2e/tsconfig.e2e.json.  
The file will have its original line endings in your working directory.  
warning: LF will be replaced by CRLF in gulpfile.js.
```

6. Commit Changes:

```
git commit -a -m "Initial Revision"
```



```
codec@DESKTOP-GFGMI69 MINGW64 /c/shoppingcartdemo/Demo (master)  
$ git config --global user.email "marfra@microsoft.com"  
codec@DESKTOP-GFGMI69 MINGW64 /c/shoppingcartdemo/Demo (master)  
$ git config --global user.name "Mark Franco"  
codec@DESKTOP-GFGMI69 MINGW64 /c/shoppingcartdemo/Demo (master)  
$ git commit -a -m "Initial Revision"  
warning: LF will be replaced by CRLF in .gitignore.  
The file will have its original line endings in your working directory.  
warning: LF will be replaced by CRLF in README.md.  
The file will have its original line endings in your working directory.  
[master 4c62229] Initial Revision  
 2 files changed, 102 insertions(+), 79 deletions(-)  
 rewrite .gitignore (99%)  
 rewrite README.md (99%)  
codec@DESKTOP-GFGMI69 MINGW64 /c/shoppingcartdemo/Demo (master)  
$
```

- Push repository to VSTS into Master branch by executing the following command (no Screenshot):

```
Git push -repo <VSTS Git Repository url from previous steps>
```

i.e. `git push -repo https://mtctor.visualstudio.com/_git/Demo`

8. And Voila! You can now see your repository pushed up into VSTS:

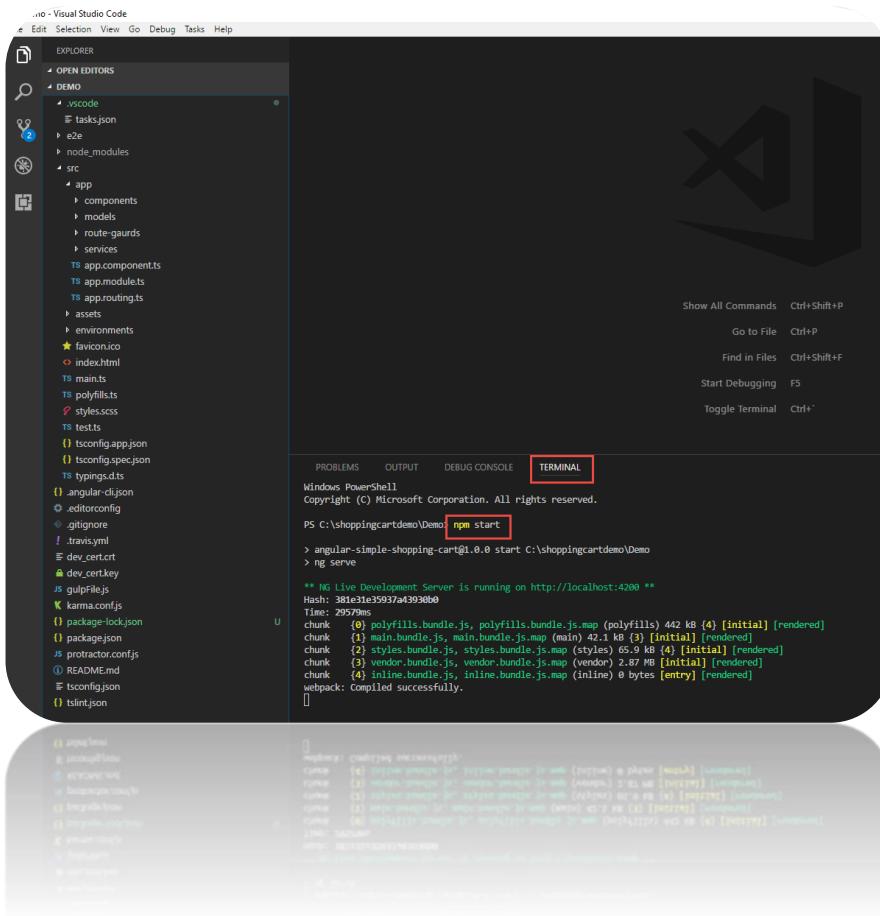
5. Setup VSTS integration using the new auth experience

1. Open VSCode and Select File->Open folder: "C:\shoppingcartdemo\Demo"
2. Watch this step by step video on how to setup the new Authentication experience.

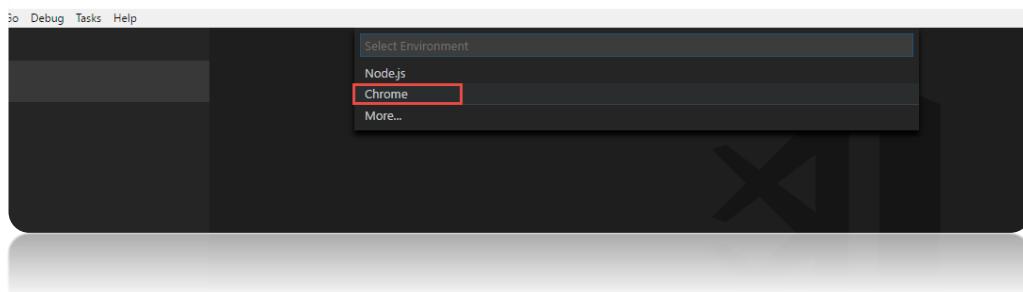
<https://youtu.be/HnDNDm1WCl0?t=2m55s>

6. Let's Build something...

1. Once you have Cached your credentials using the new authentication experience, ensure all dependencies are current by running "**npm install**" in the VS Code terminal window
2. Run a local instance of the app to see how it runs by running "**npm start**" in the vscode terminal:



3. Your app is compiled and running under a node web server, but we need to add a launch file so we can launch a debugger window using Chrome. We do so by creating a new configuration file by selecting the “Debug→Add Configuration” menu item and selecting “Chrome” from the drop down.



4. We need to ensure the new launch.json file is pointing to the correct url. Node will automatically assign a random port on your computer to host your angular application on and you can get this url from the previous step where you ran “NPM Start”:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\shoppingcartdemo\Demo> npm start

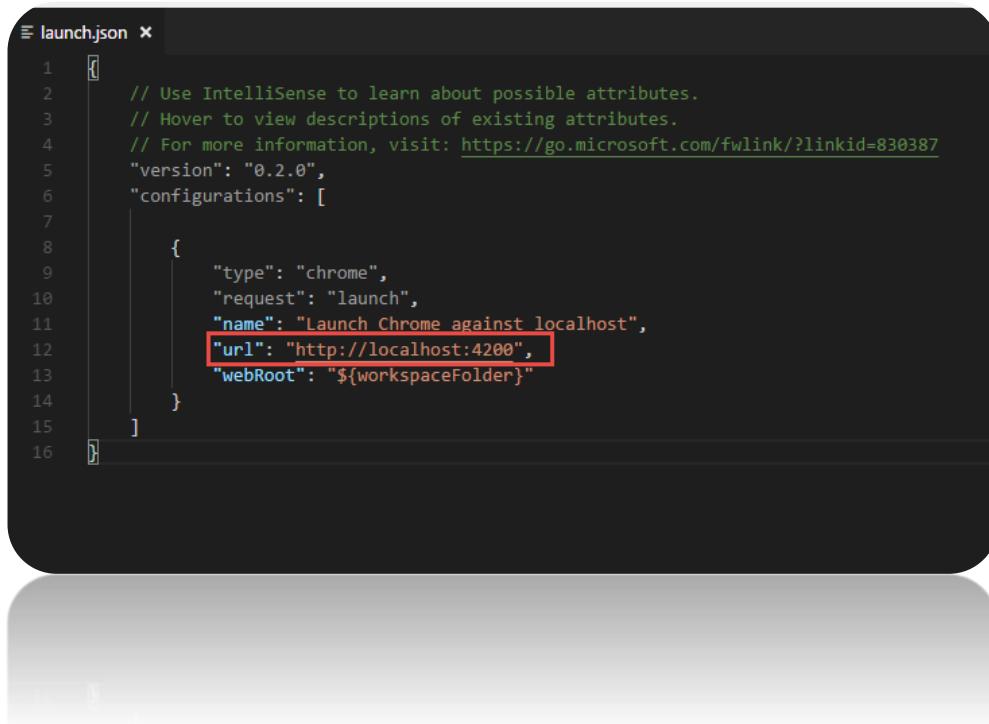
> angular-simple-shopping-cart@1.0.0 start C:\shoppingcartdemo\Demo
> ng serve

** NG Live Development Server is running on http://localhost:4200 **
Hash: 381e31e35937a43930b0
Time: 29579ms
chunk {0} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 442 kB {4} [initial] [rendered]
chunk {1} main.bundle.js, main.bundle.js.map (main) 42.1 kB {3} [initial] [rendered]
chunk {2} styles.bundle.js, styles.bundle.js.map (styles) 65.9 kB {4} [initial] [rendered]
chunk {3} vendor.bundle.js, vendor.bundle.js.map (vendor) 2.87 MB [initial] [rendered]
chunk {4} inline.bundle.js, inline.bundle.js.map (inline) 0 bytes [entry] [rendered]
webpack: Compiled successfully.

Memory: Compiled successfully.
CPU: {4} [idle] [idle] [idle] [idle]
CPU: {3} [idle] [idle] [idle] [idle]
CPU: {2} [idle] [idle] [idle] [idle]
CPU: {1} [idle] [idle] [idle] [idle]
```

The screenshot shows the Visual Studio Code interface with the 'TERMINAL' tab selected. The terminal window displays the command 'npm start' being run in a Windows PowerShell environment. The output shows the Angular Live Development Server starting up on port 4200. A red box highlights the URL 'http://localhost:4200' in the output. Below the terminal, there are status bars for memory and CPU usage, which are mostly idle.

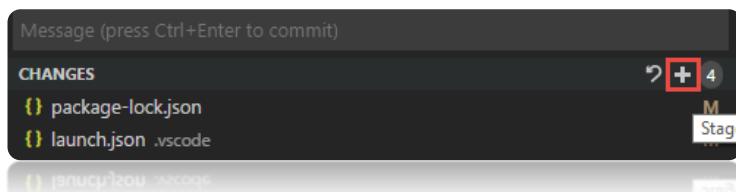
Note: With the above url , you are going to update the **launch.json** file and specifically update the "url" property of the Chrome configuration as such:



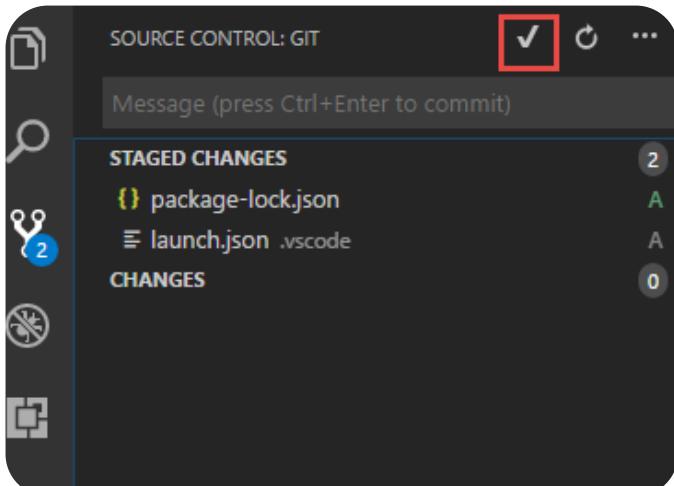
```
1  {
2      // Use IntelliSense to learn about possible attributes.
3      // Hover to view descriptions of existing attributes.
4      // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
5      "version": "0.2.0",
6      "configurations": [
7          {
8              "type": "chrome",
9              "request": "launch",
10             "name": "Launch Chrome against localhost",
11             "url": "http://localhost:4200",
12             "webRoot": "${workspaceFolder}"
13         }
14     ]
15 }
16 }
```

5. Now click on Debug→Start debugging
6. Try some breakpoints and debugging techniques...
7. Check in your additional file "**Launch.json**" using the VS CODE IDE now:

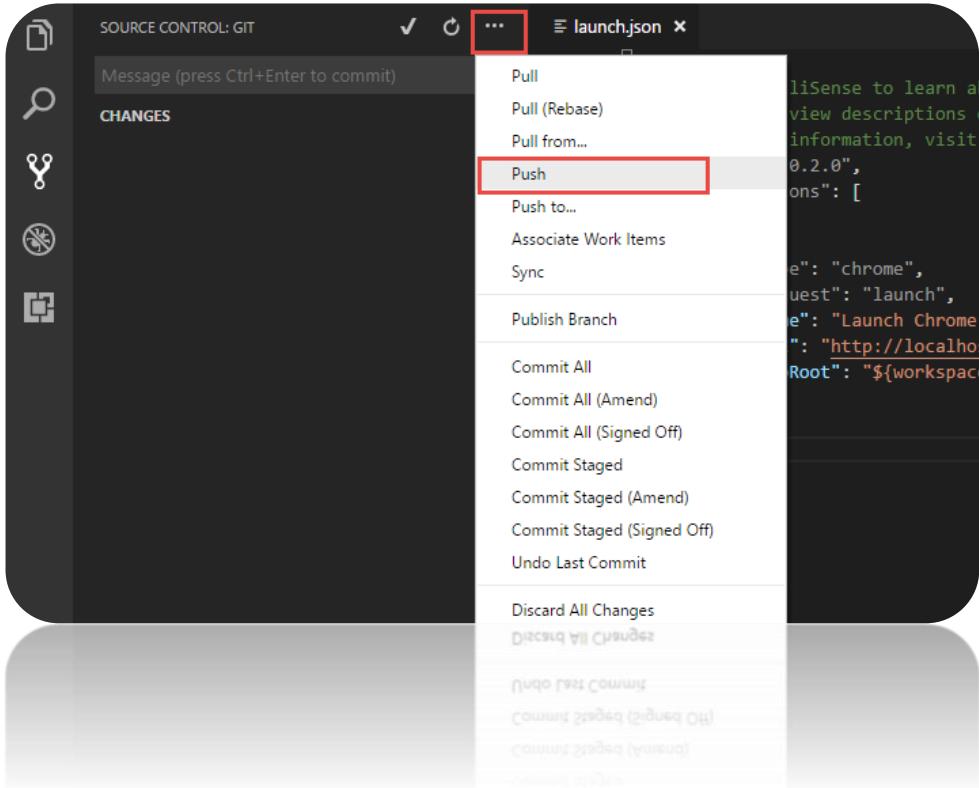
Add Files to local repository (Stage)



8. Commit Changes to local Repository



9. Push Changes from local repository to VSTS



10. Build Complete...

7. Setting Up Work Item Check-in and Build Configuration

1. Go to VSTS dashboard and create a task. We will associate this task with check-in.

The screenshot shows the VSTS (Visual Studio Team Services) dashboard for a project named "FirstApp". The top navigation bar includes links for Overview, Dashboards, Code, Work, Build and Release, Test, Wiki, and a settings gear icon. The main content area is titled "Overview". On the left, there's a "Welcome" section with links to Manage Work, Collaborate on code, Continuously integrate, and Visualize progress. Below it is a "Sprint Burndown" section. In the center, a large box is titled "Work assigned to rahul.mittal (0)" with a sub-instruction: "All done with the work assigned to you? Go to your team backlog to pick up new work." To the right of this box is a "Team Members" section featuring a placeholder image of a person on a beach and a button to "Invite a friend". Further down the center is a "New Work Item" dialog box with a red border around its title and input fields. It contains a text input field with "Create Demo Application" and a dropdown menu set to "Task". At the bottom of this dialog is a blue "Create" button. To the right of the "New Work Item" box is a "Work items" card showing "0" work items. On the far right, there's a "Visual Studio" section with links to "Open in Visual Studio" (requires VS 2013+) and "Get Visual Studio". Below these sections are two smaller cards related to "Build and Release".

- Assign a task to a resource (**Youself** in this case), enter description, set priority, and specify effort. Click Save and Close.

When user saves, unique task number is assigned to each task.

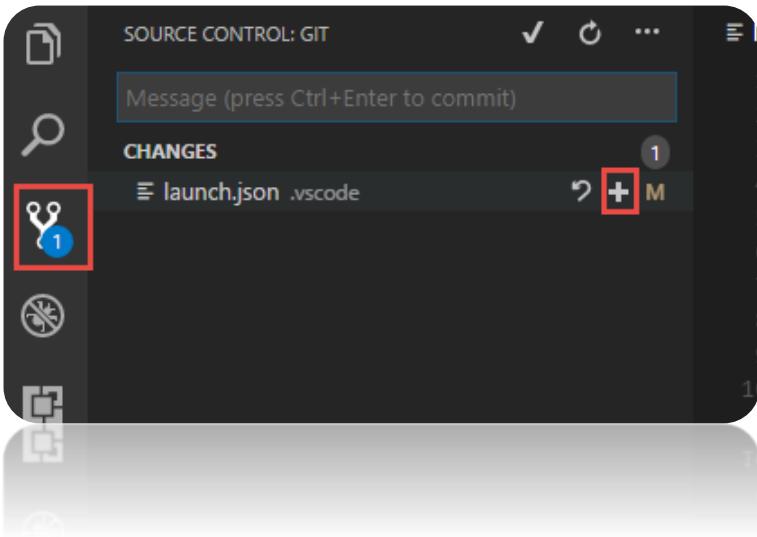
Go back to VS CODE, make changes to the launch.json file and associate the work item while committing the code.

- Make the code change by appending "on port 4200" as shown below:

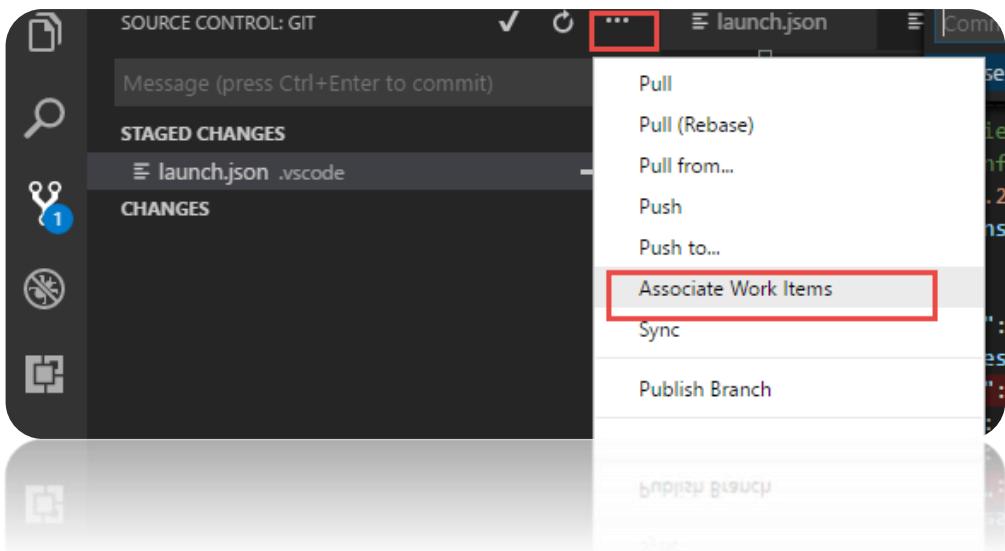
```

{
  // See https://go.microsoft.com/fwlink/?LinkId=733558
  // for more information about launching web browsers.
  // Hover over descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830302
  "version": "0.2.0",
  "configurations": [
    {
      "type": "chrome",
      "request": "launch",
      "name": "Launch Chrome against localhost on port 4200",
      "url": "http://localhost:4200",
      "webRoot": "${workspaceFolder}"
    }
  ]
}
  
```

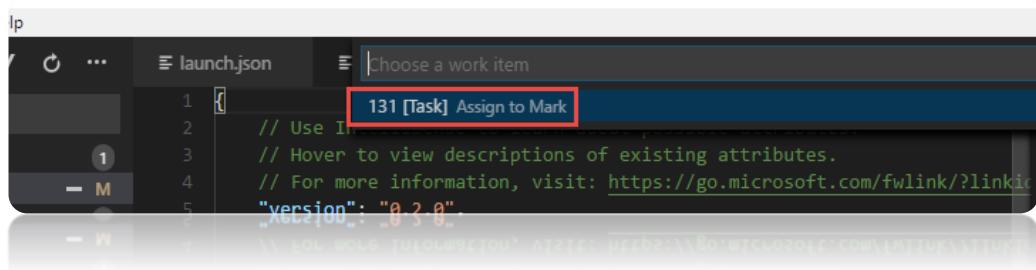
4. Add Change (Stage)



5. Commit Change by Associating work item:



6. Select Work Item task:



7. Commit Change with a message "Added port "

8. Push Change to VSTS.

9. When we go to task board in VSTS, we can see development history associated with this item.

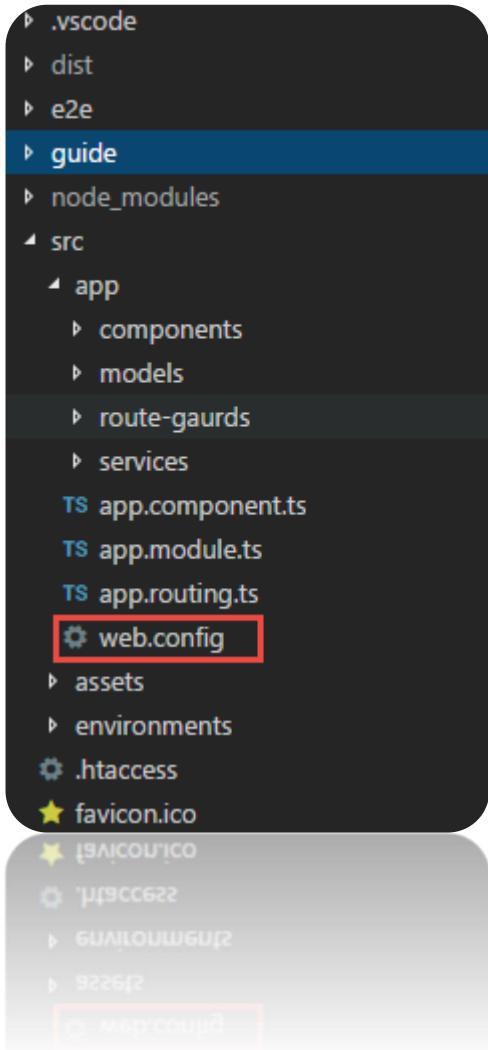
Check out here:

A screenshot of the VSTS repository interface. The top navigation bar includes 'Dashboards', 'Code', 'Work', and 'Build and Release'. Below the navigation, there's a dropdown for 'Demo' and tabs for 'Files', 'Commits', 'Pushes', and 'Branches'. The 'Files' tab is selected. On the left, there's a tree view of the repository structure under 'Demo'. A context menu is open over the 'Demo' repository name, listing options like 'Files', 'Commits' (which is highlighted with a red box), 'Pushes' (also highlighted with a red box), 'Branches', 'Tags', and 'Pull Requests'. There are also links for 'New pull request' and 'Manage repositories'.

8. Deploy to Azure App Services

We will need to add a web.config file to instruct our underlying web server on Azure to rewrite all incoming request to serve our *index.html* file.

1. Create a new file named web.config in `src\app\` by right-clicking on `src\app` folder and selecting 'New File'



2. Add the following contents to the **web.config**:

```
<configuration>

  <system.webServer>
    <staticContent>
      <mimeTypeMap fileExtension=".json" mimeType="application/json" />
    </staticContent>

    <rewrite>
      <rules>
        <clear />

        <!-- ignore static files -->
        <rule name="AngularJS Conditions" stopProcessing="true">
          <match url="(assets/.*|.js|.css)" />
          <conditions logicalGrouping="MatchAll" trackAllCaptures="false" />
          <action type="None" />
        </rule>

        <!-- check if its root url and navigate to default page -->
        <rule name="Index Request" enabled="true" stopProcessing="true">
          <match url="^$" />
          <action type="Redirect" url="/home" logRewrittenUrl="true" />
        </rule>

        <!--remaining all other url's point to index.html file -->
        <rule name="AngularJS Wildcard" enabled="true">
          <match url="(.*)" />
          <conditions logicalGrouping="MatchAll" trackAllCaptures="false" />
          <action type="Rewrite" url="index.html" />
        </rule>

      </rules>
    </rewrite>
  </system.webServer>
</configuration>
```

3. Modify the **/gulpfile.js** as follows to remove the code that modifies the index.html `<base href="/">` element.

Note: The original developer added this code, but it is no longer needed as you can leverage angular CLI to modify this directly. Also, we have added a copy process to deploy the **web.config** to the distribution folder:

```
var gulp = require('gulp');

var replace = require('gulp-replace');
var htmlmin = require('gulp-htmlmin');

gulp.task('js:minify', function () {
  gulp.src(['./dist/main.*.js', './dist/polyfills.*.js',
  './dist/inline.*.js'])
    .pipe(replace(/\/*([\s\S]*?)\*/g, ""))
    .pipe(gulp.dest("./dist"));
});

gulp.task('web:config', function () {
  gulp.src ['./src/app/web.config"])
    .pipe(gulp.dest("./dist"));
});

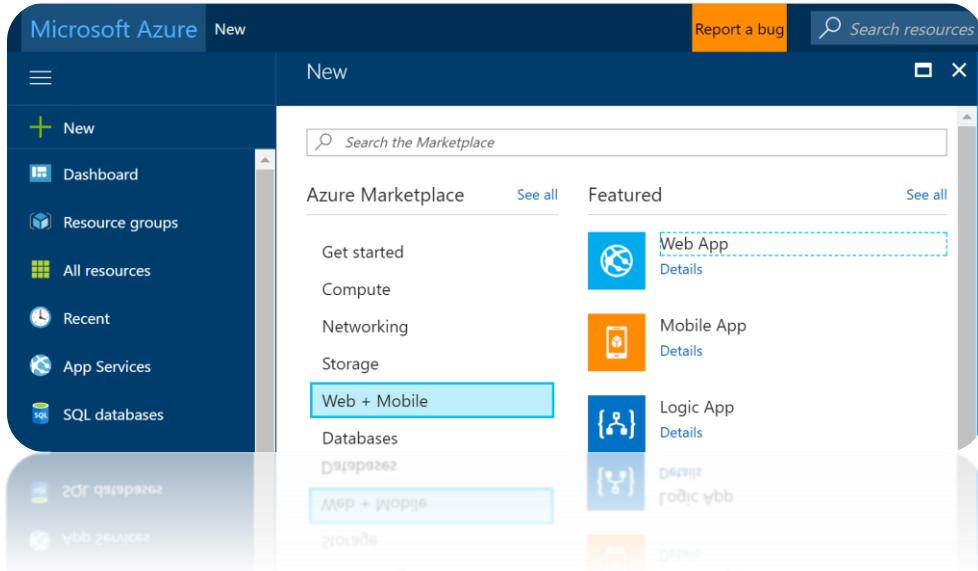
gulp.task("html:minify", function () {
  return gulp.src('dist/*.html')
    .pipe(htmlmin({ collapseWhitespace: true }))
    .pipe(gulp.dest('./dist'));
});

gulp.task("default", ["js:minify", "html:minify", "web:config"]);
```

9. Create the Azure App Service

The next step is to create an Azure Web App which will host our Angular application. You can [sign up](#) for a free or paid account and log in the [Azure portal](#).

1. New -> Web and Mobile -> Web App



2. Fill in the web app details as such:

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with a 'New' button and a list of services: Dashboard, Resource groups, All resources, Recent, App Services, SQL databases, Virtual machines (classic), Virtual machines, Cloud services (classic), Subscriptions, App Service plans, Application Insights, Azure Active Directory, Monitor, Security Center, Help + support, Advisor, and Billing. Below this sidebar, there's a list of recent items: Billing, Totriba, Help + support, Security Center, and Advisor.

The main area is titled 'Web App Create'. It contains the following fields:

- App name:** A text input field with placeholder text "Enter a name for your App" and ".azurewebsites.net" suffix.
- Subscription:** A dropdown menu set to "Microsoft Azure Internal Consumption".
- Resource Group:** A section with radio buttons for "Create new" and "Use existing", and a text input field for the group name.
- OS:** A radio button group with "Windows" selected and "Linux" as an option.
- App Service plan/Location:** A dropdown menu showing "VRSAPIPlan(East US)".
- Application Insights:** A toggle switch set to "Off".

At the bottom of the dialog, there are two buttons: "Create" (highlighted in blue) and "Automation options".

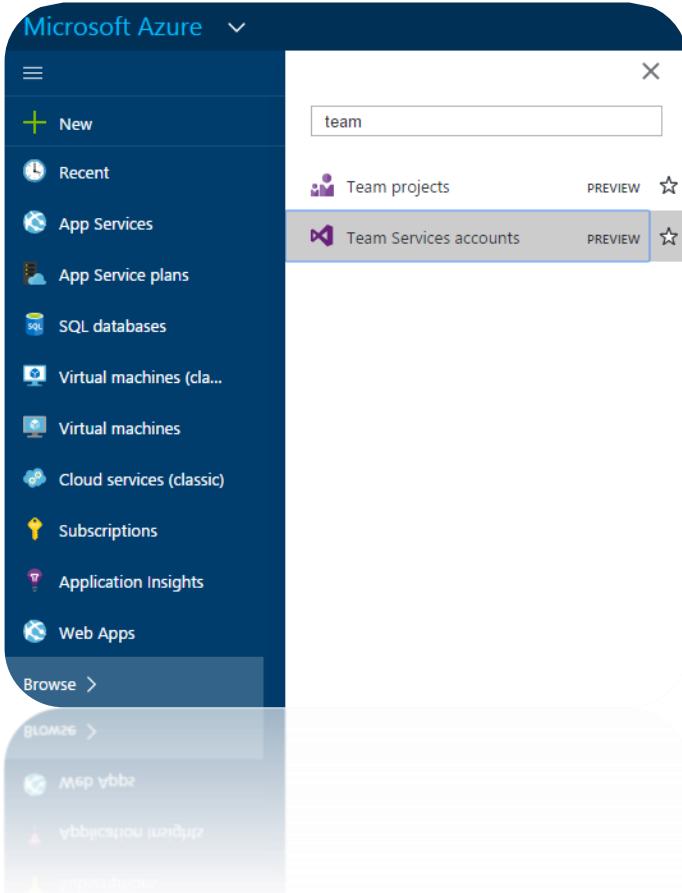
3. Then Click "Create".

10. Linking your VSTS account to your Azure subscription

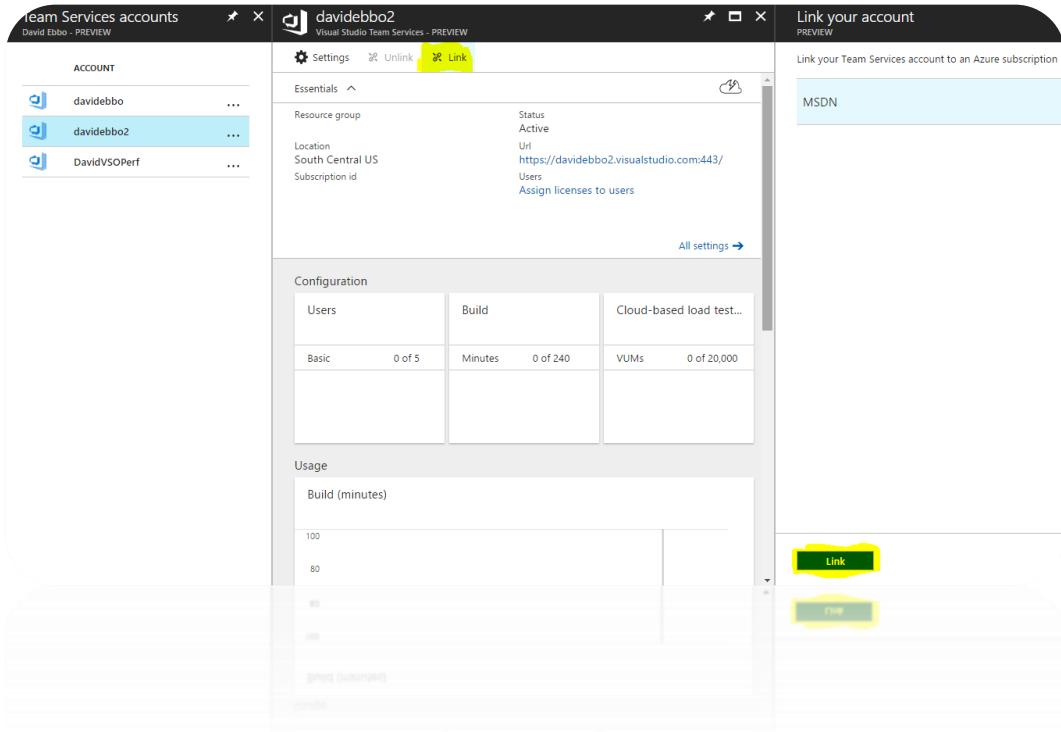
Next, you need to link your VSTS account to your Azure subscription (see also [this post](#) on this topic).

To do this, go to the Azure Portal...

1. Click More Services (image says 'Browse' but that was the old name) and search for 'Team':



- Now select the relevant Team Services account, click Link button at the top, and then the Link button in the other blade:



And you're done! You will now be able to set up continuous deploying to your git repos hosted in VSTS.

11. Setting Up CI Pipeline With VSTS

In the next steps we will set up our VSTS CI/CD pipeline to push the Angular application to the newly created Azure Web App. Start by creating a new build definition under VSTS:

1. *Build and Release -> Builds -> New*
2. Add an npm task to install the npm packages required by the Angular application
 ... > AngularDeploymentToAzure-CI

The screenshot shows the 'Tasks' tab of a VSTS build definition named 'AngularDeploymentToAzure-CI'. The tasks listed are: 'Get sources' (using 'AngularDeploymentToAzure' repository and 'master' branch), 'npm install' (using 'npm'), 'npm run build' (using 'npm'), and 'Publish Artifact: dist' (Publish Build Artifacts). The 'npm install' task is currently selected.

3. Add another npm task to build the application and create the dist folder:

The screenshot shows the 'Tasks' tab of the same build definition. A new 'npm run build' task has been added below the existing one. The task configuration is as follows:
 - Display name: 'npm run build'
 - Command: 'custom'
 - Working folder with package.json: (empty)
 - Command and arguments: 'run-script build'
 - Custom registries and authentication: (dropdown menu)
 - Control Options: (dropdown menu)

4. Add a publish artifact task that generates the dist artifact which will be provided later on as an input to our release definition:

The screenshot shows the Azure DevOps interface for managing a build pipeline. At the top, there are navigation links: Builds, Releases, Library, Task Groups, and Deployment Groups*. Below these, the pipeline name is displayed as AngularDeploymentToAzure-Cl. The main area is divided into two sections: 'Process' on the left and 'Tasks' on the right.

Process: Shows the build process steps:

- Get sources (AngularDeploymentToAzure branch, master)
- npm install
- npm run build
- Publish Artifact: dist** (selected, highlighted with a blue border)

Tasks: A ribbon-style menu with tabs: Tasks, Variables, Triggers, Options, Retention, and History. The 'Tasks' tab is currently active.

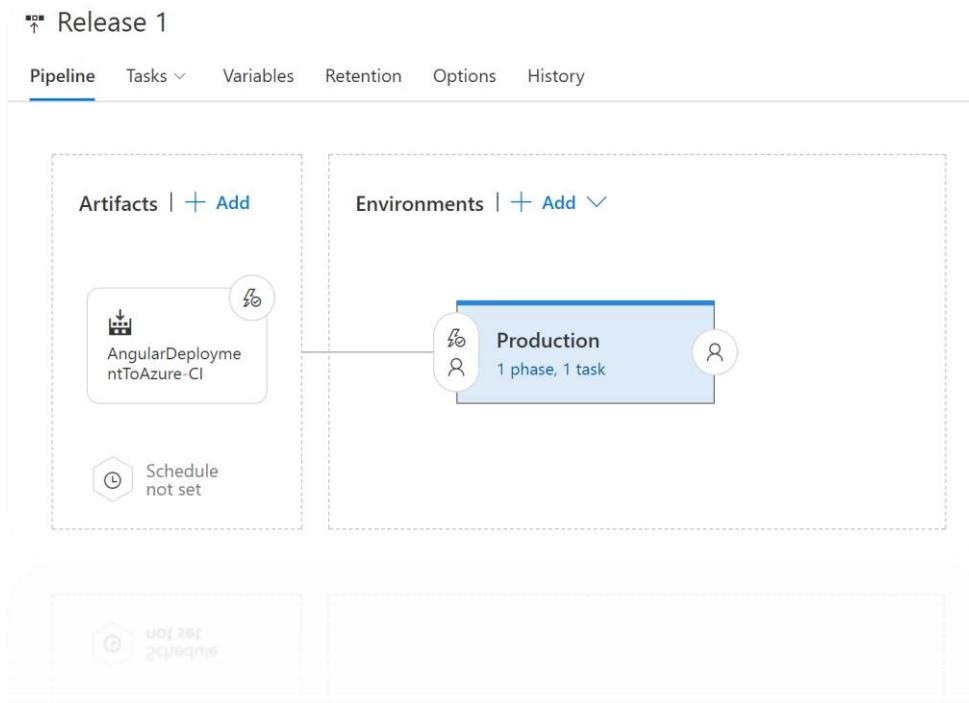
Task Configuration (Publish Build Artifacts):

- Version:** 1.*
- Display name:** Publish Artifact: dist
- Path to Publish:** dist
- Artifact Name:** dist
- Artifact Type:** Server

Control Options: Includes 'Cancel' and 'Apply changes' buttons.

12. Setting Up CD Pipeline With VSTS

The last step is to add a CD pipeline which will deploy the artifacts created by the build to the Azure Web App. In this demo I am keeping the release pipeline simple by deploying the artifacts directly to production. In a real life application you will probably create multiple environments before releasing to production (Development, QA, Staging, etc.):



The production environment includes a single task that deploys the Angular application to an Azure Web App:

A screenshot of the Azure DevOps Pipeline Tasks page. The top navigation bar includes 'Builds', 'Releases', 'Library', 'Task Groups', and 'Deployment Groups'. Below the navigation, the pipeline name is 'Release 1' and the tab selected is 'Tasks'. The pipeline consists of one step: 'Production Deployment process' which runs on an agent and contains a task named 'Deploy Angular App To Azure Web App' (Azure App Service Deploy). The task configuration shows the following details:

- Display name:** Deploy Angular App To Azure Web App
- Azure subscription:** [Manage] (Subscription ID obscured)
- App Service name:** AngularDeploymentToAzure
- Virtual application:** (empty field)
- Package or folder:** \$(System.DefaultWorkingDirectory)/AngularDeploymentToAzure CI/dist
- Deployment triggers:** (empty field)
- Deployment validation:** (empty field)

That's it!

You now have a fully functional CI/CD pipeline that will deploy your Angular application to an Azure Web App (Windows based) the next time you check in your code.

Up Next: Retrieve Products from a Cosmos DB Backend

13. Cosmos DB - Replacing Static Products Back End

1. Create Cosmos DB Account, Database, and Collection

- a. In the Azure Portal click on “New Resource” then type Cosmos DB in the search bar. Press “Enter”. Click the search result titled “Cosmos DB”. Click “Create” on the next page.
- b. For ID, choose an ID for the Cosmos DB account. Choose “SQL” as the account type. Resource group and Location depend on existing deployment configuration. Choose whatever settings make sense. Click create.
- c. Click the “Add Collection” button. Choose a database title and specify “Products” as the collection name. Choose “Fixed” for storage capacity. Choose “/id” as the partition key. Set the throughput to 1000 (the minimum) RU. Click OK.

2. Import Data

- a. Download and locally extract the contents of:
<https://cosmosdbportalstorage.blob.core.windows.net/datamigrationtool/2018.02.28-1.8.1/dt-1.8.1.zip>
- b. Download and store locally:
https://github.com/Microsoft/MTC_EnterpriseSPADev/blob/master/src/assets/products.json
- c. Run the dtui executable in the extracted archive.
- d. Click next in the tool
- e. Click add files, then select the file above step “products.json”. Click next.
- f. In the Azure Portal, select your cosmos DB. It can be found in the resource group in which it was created. Click on it. Then click “Keys” under settings.

- g. Copy and paste the “Primary Connection String” value into Notepad. Add a “;” to the end if one isn’t present. Add “Database=<DB NAME YOU Chose>” to the connection string in Notepad. Copy the whole string.
- h. Paste the complete connection string into the connection string field in dtui.
- i. Put Products into the collection field in dtui. The partition key and id field should both be “/id”. Click next.
- j. Click next until the import completes.
- k. In the Azure Portal, click on your cosmos db (per 13.2.6), then “Data Explorer”, then select your collection, then click “Documents”. You should see 5 documents with GUID id’s.

3. Create Azure Function App

- a. Follow the instructions in the section titled “Build a function in Visual Studio 2017” from

<https://docs.microsoft.com/en-us/azure/cosmos-db/tutorial-functions-http-trigger#publish-the-azure-function>

Deviations from the preceding guide are as follows:

- Use “*CosmosFunctions*” as the project name in step 1.
- For step 2b, use “*Microsoft.Azure.DocumentDB*” instead of the default of “*Microsoft.Azure.Graphs*”.
- For step 3, use “*ProductFunction*” as the function title.

- b. Replace the code in “ProductFunction.cs” with the code below. Change <>YOUR URI>> to the URI of your Cosmos DB from the Cosmos DB account overview page. Change <>YOUR KEY>> to the primary key from the Keys tab.
- c. **NOTE:** In a production setting, be sure to externalize these values.

```

using System.Linq;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.Azure.WebJobs.Host;
using Microsoft.Azure.Documents.Client;
using System;
using System.Collections.Generic;
using Newtonsoft.Json;
using System.Text;

namespace CosmosFunction
{
    public static class ProductFunction
    {
        [FunctionName("ProductFunction")]
        public static async Task<HttpResponseMessage> Run([HttpTrigger(AuthorizationLevel.Anonymous,
            "get", "post", Route = null)]
            HttpRequestMessage req, TraceWriter log)
        {
            log.Info("C# HTTP trigger function processed a request.");

            DocumentClient client = new DocumentClient(new Uri("<>YOUR URI>>"), "<>YOUR DOC DB Auth Key>>");

            IQueryables query = client.CreateDocumentQuery(UriFactory.CreateDocumentCollectionUri("<>YOUR DB NAME>>",
                "Products"),
                "select * from c");

            List<object> output = new List<object>();
            foreach (var v in query)
            {
                output.Add(v);
                Console.WriteLine(v);
            }
        }
    }
}

```

4. Test the Function

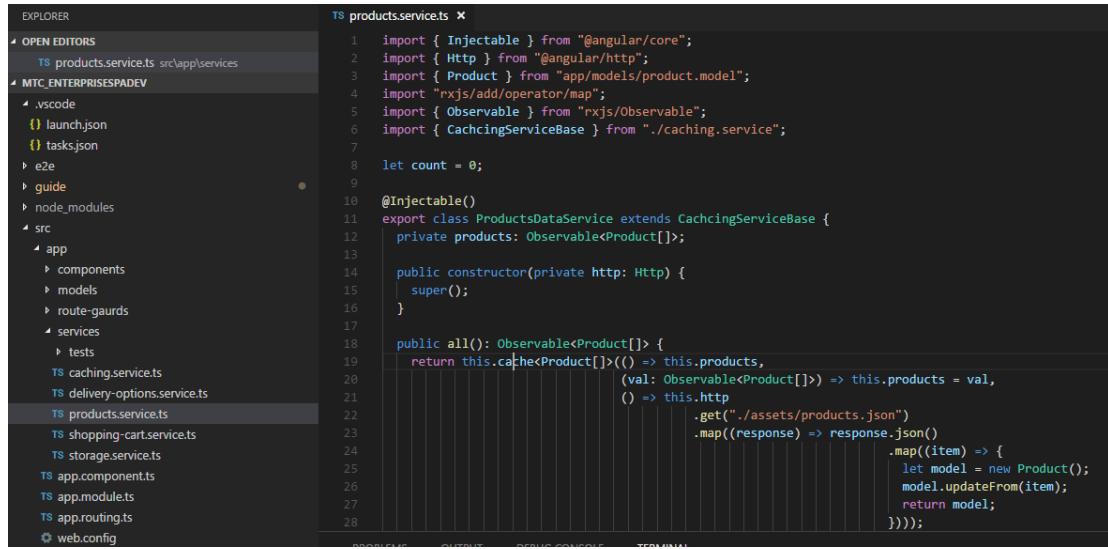
- a. Click F5 in Visual Studio 2017. If prompted, click yes when asked to install the Azure Functions Tools. This will take a minute or so.
- b. A command prompt will launch and run the functions runtime. Copy and paste the URL at the bottom of the output into a browser to test the function. You can also use a tool like Postman.

5. Deploy Azure Function App

- a. Right click on the CosmosFunctions project and click publish.
- b. When prompted to pick a publish target, Select Azure Function App, and the Create New radio button.
- c. Choose a meaningful function app name.
- d. Select your subscription.
- e. Select your resource group.
- f. Click new beside app service plan. Create a new plan in the same region as Cosmos DB.
- g. Click create at the bottom of the dialog.

6. Integrate Azure Function into SPA

- a. Navigate to the Azure Functions deployment in the Azure Portal.
- b. Click on the Platform Features tab, then select CORS.
- c. Add "<http://localhost:4200>" to the list. Ignore any formatting errors and hit "Save".
- d. Open Visual Studio Code and open the ".\src\app\services\products.services.ts" file. The contents of the file should look like this:



```
products.service.ts
1 import { Injectable } from '@angular/core';
2 import { Http } from '@angular/http';
3 import { Product } from './models/product.model';
4 import 'rxjs/add/operator/map';
5 import { Observable } from 'rxjs/Observable';
6 import { CachingServiceBase } from './caching.service';
7
8 let count = 0;
9
10 @Injectable()
11 export class ProductsDataService extends CachingServiceBase {
12   private products: Observable<Product[]>;
13
14   public constructor(private http: Http) {
15     super();
16   }
17
18   public all(): Observable<Product[]> {
19     return this.cache.getProduct[](() => this.products,
20       (val: Observable<Product[]>) => this.products = val,
21       () => this.http
22         .get("./assets/products.json")
23         .map((response) => response.json())
24         .map((item) => {
25           let model = new Product();
26           model.updateFrom(item);
27           return model;
28         }));
29 }
```

- e. Modify HTTP Get call on line 22 so that we are no longer grabbing the products from the filesystem, but instead, getting the products from our Cosmos DB Enabled Azure function that we created above. We will simply need to grab the “Products” URI from the azure function overview section within the azure portal (portal.azure.com).
- f. Modify As such:

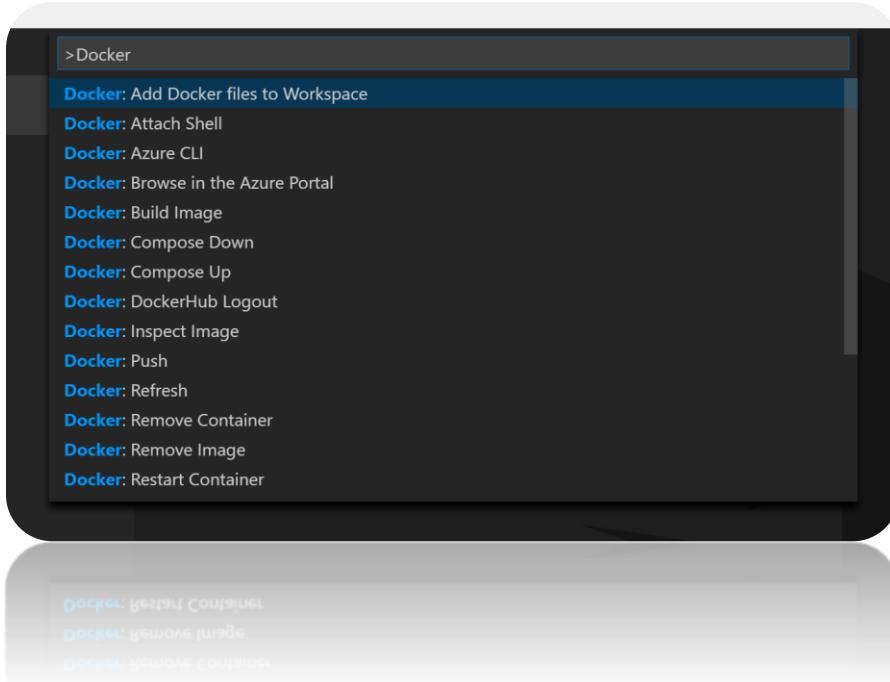
```
17  public all(): Observable<Product[]> {
18    return this.cache<Product[]>(() => this.products,
19      (val: Observable<Product[]>) => this.products = val,
20      () => this.http
21        .get(`https://<<YOUR APP NAME>>.azurewebsites.net/api/ProductFunction`)
22        .map((response) => response.json())
23        .map((item) => {
24          let model = new Product();
25          model.updateFrom(item);
26          return model;
27        }));
28
```

- g. Done! Rebuild and Test out your Angular App again and see your products being pulled from Cosmos DB.

Up Next: Deploy your SPA to a Linux Docker Image & Deploy to Azure Web App (Linux based) using CI/CD

14. Deploy your SPA to a Linux Docker Image

1. Add Docker Files to workspace like so:

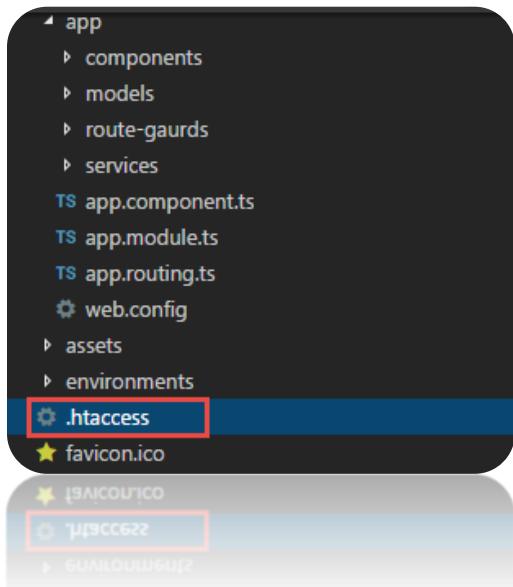


When Prompted Select: **node.js** and then set the Port to **4200**. This is just to create the base implementation of the Docker image files, but we will replace the contents with our own commands for our SPA to work in a simple Apache Web Server image provided by the image library on the Docker public registry.

2. Once the DockerFile is added to your Angular application its time to add the necessary commands to assemble a docker image which will be used to create docker containers that will run on both the development machine as well as on the production server. We will assume that Apache 2.4 will be used as the web server.
3. Modify the contents of the DockerFile so that it builds an image based on the httpd:2.4 Docker Public image and copies the dist folder that is generated by the angular build process into the specified directory inside the image. Overwrite the DockerFile with the code below:

```
FROM httpd:2.4
COPY dist /usr/local/apache2/htdocs/
```

1. Add a new file under **/app** named **".htaccess"**. This file is required to instruct Apache how to route your angular application.



2. Add the entire contents below into the ".htaccess" file:

```
RewriteEngine On
# If an existing asset or directory is requested go to it as it is
RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -f [OR]
RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -d
RewriteRule ^ - [L]

# If the requested pattern is file and file doesn't exist, send 404
RewriteCond %{REQUEST_URI} ^(/[a-z_-\s0-9\.]+)\.[a-zA-Z]{2,4}$
RewriteRule ^ - [L,R=404]

# otherwise use history router
RewriteRule ^ /index.html
```

3. Modify the **/gulpfile.js** once again as follows to include the .htaccess file.

```
var gulp = require('gulp');
var replace = require('gulp-replace');
var htmlmin = require('gulp-htmlmin');

gulp.task('js:minify', function () {
  gulp.src(["./dist/main.*.js", "./dist/polyfills.*.js",
  "./dist/inline.*.js"])
    .pipe(replace(/\/*([\s\S]*?)\*/[ \s\S]?/g, ""))
    .pipe(gulp.dest("./dist"));
});

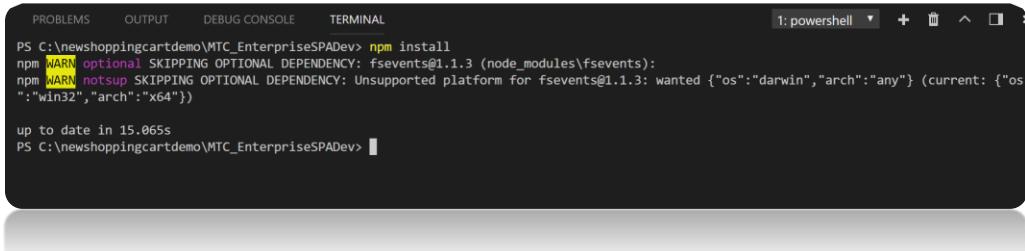
gulp.task('web:config', function () {
  gulp.src(["./src/app/web.config"])
    .pipe(gulp.dest("./dist"));
});

gulp.task('apache:htaccess', function () {
  gulp.src(["./src/app/.htaccess"])
    .pipe(gulp.dest("./dist"));
});

gulp.task("html:minify", function () {
  return gulp.src('dist/*.html')
    .pipe(htmlmin({ collapseWhitespace: true }))
    .pipe(gulp.dest('./dist'));
});

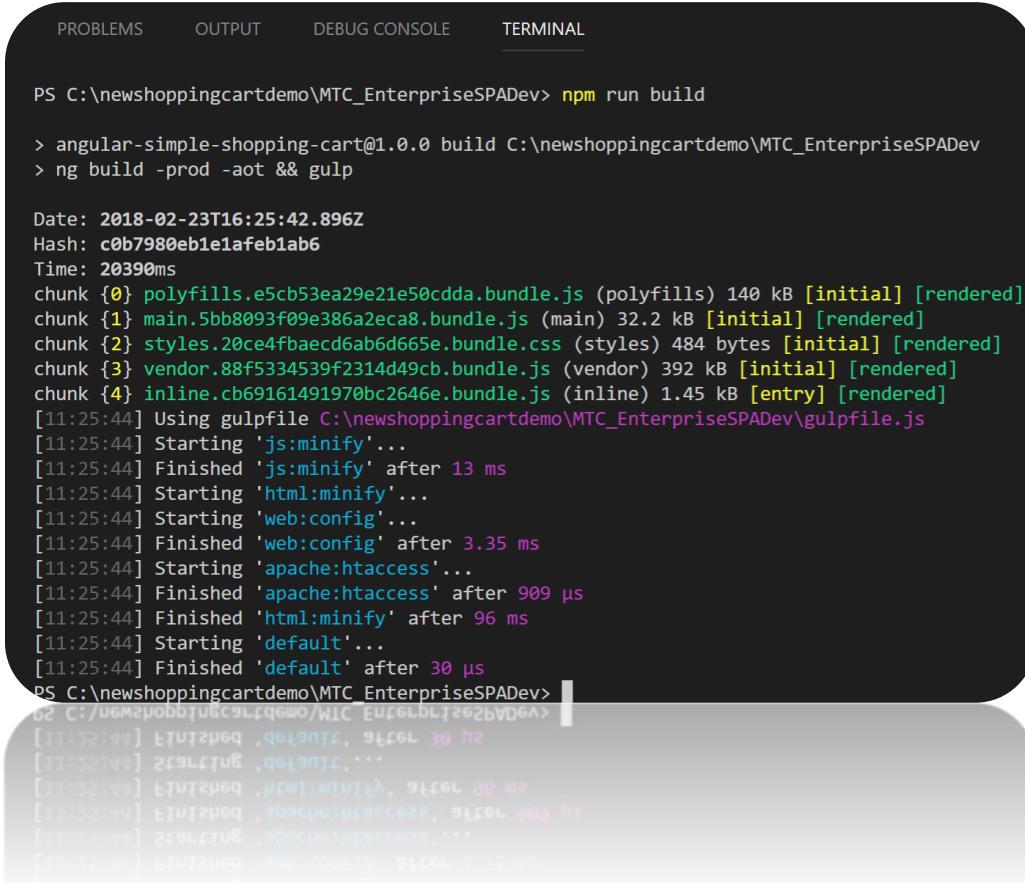
gulp.task("default", ["js:minify", "html:minify", "web:config",
"apache:htaccess"]);
```

4. In the terminal Run: “**npm install**”



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\newshoppingcartdemo\MTC_EnterpriseSPADev> npm install
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.1.3 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
up to date in 15.065s
PS C:\newshoppingcartdemo\MTC_EnterpriseSPADev>
```

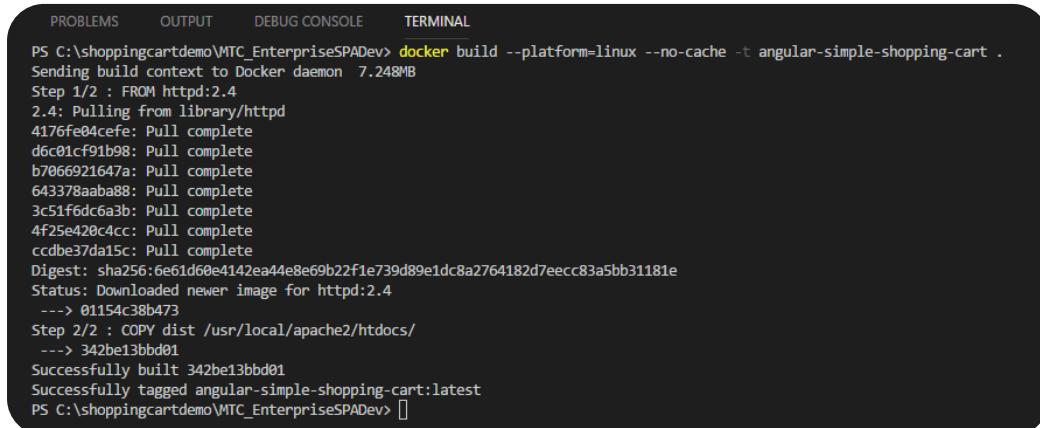
5. In the terminal Run: “**npm run build**”



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\newshoppingcartdemo\MTC_EnterpriseSPADev> npm run build
> angular-simple-shopping-cart@1.0.0 build C:\newshoppingcartdemo\MTC_EnterpriseSPADev
> ng build -prod -aot && gulp

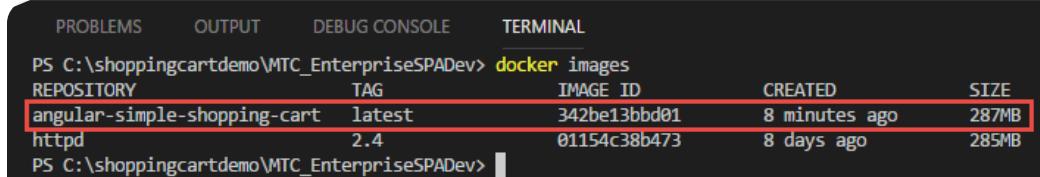
Date: 2018-02-23T16:25:42.896Z
Hash: c0b7980eb1e1afeb1ab6
Time: 20390ms
chunk {0} polyfills.e5cb53ea29e21e50cdda.bundle.js (polyfills) 140 kB [initial] [rendered]
chunk {1} main.5bb8093f09e386a2eca8.bundle.js (main) 32.2 kB [initial] [rendered]
chunk {2} styles.20ce4fbaecd6ab6d665e.bundle.css (styles) 484 bytes [initial] [rendered]
chunk {3} vendor.88f5334539f2314d49cb.bundle.js (vendor) 392 kB [initial] [rendered]
chunk {4} inline.cb69161491970bc2646e.bundle.js (inline) 1.45 kB [entry] [rendered]
[11:25:44] Using gulpfile C:\newshoppingcartdemo\MTC_EnterpriseSPADev\gulpfile.js
[11:25:44] Starting 'js:minify'...
[11:25:44] Finished 'js:minify' after 13 ms
[11:25:44] Starting 'html:minify'...
[11:25:44] Starting 'web:config'...
[11:25:44] Finished 'web:config' after 3.35 ms
[11:25:44] Starting 'apache:htaccess'...
[11:25:44] Finished 'apache:htaccess' after 909 µs
[11:25:44] Finished 'html:minify' after 96 ms
[11:25:44] Starting 'default'...
[11:25:44] Finished 'default' after 30 µs
PS C:\newshoppingcartdemo\MTC_EnterpriseSPADev>
```

6. In the terminal Run: “`docker build --platform=linux --no-cache -t angular-simple-shopping-cart`.”



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\shoppingcartdemo\MTC_EnterpriseSPADev> docker build --platform=linux --no-cache -t angular-simple-shopping-cart .
Sending build context to Docker daemon 7.248MB
Step 1/2 : FROM httpd:2.4
2.4: Pulling from library/httpd
4176fe04cefe: Pull complete
d6c01cf91b98: Pull complete
b7066921647a: Pull complete
643378aab88: Pull complete
3c51f6dc6a3b: Pull complete
4f25e420c4cc: Pull complete
ccdbe37da15c: Pull complete
Digest: sha256:6e1d60e4142ea4e8e69b22f1e739d89e1dc8a2764182d7eecc83a5bb31181e
Status: Downloaded newer image for httpd:2.4
--> 01154c38b473
Step 2/2 : COPY dist /usr/local/apache2/htdocs/
--> 342be13bbd01
Successfully built 342be13bbd01
Successfully tagged angular-simple-shopping-cart:latest
PS C:\shoppingcartdemo\MTC_EnterpriseSPADev>
```

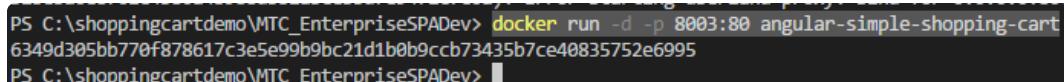
7. In the Terminal Run: “`docker images`”. You will see two images. The HTTPD which is your base image that was downloaded from the Docker registry and the angular-simple-shopping-cart image you just built.



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
angular-simple-shopping-cart	latest	342be13bbd01	8 minutes ago	287MB
httpd	2.4	01154c38b473	8 days ago	285MB

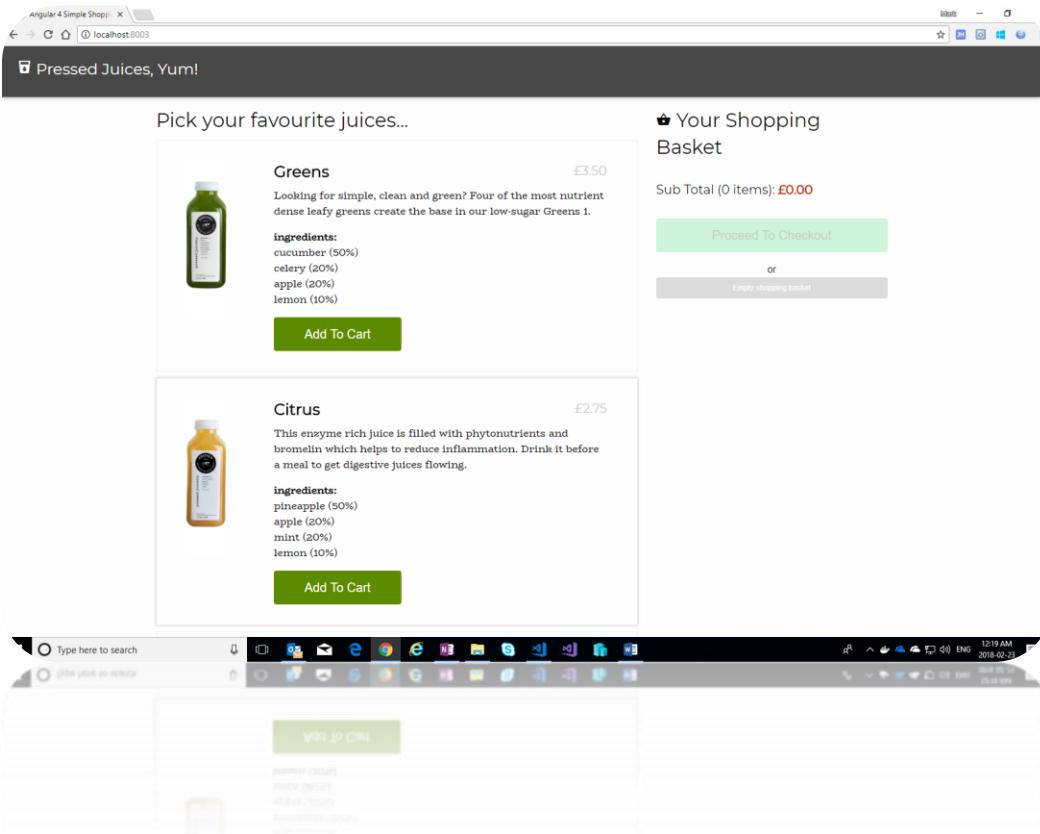
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\shoppingcartdemo\MTC_EnterpriseSPADev> docker images
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
angular-simple-shopping-cart   latest   342be13bbd01   8 minutes ago  287MB
httpd                2.4     01154c38b473   8 days ago    285MB
PS C:\shoppingcartdemo\MTC_EnterpriseSPADev>
```

8. In the Terminal Run: “`docker run -d -p 8003:80 angular-simple-shopping-cart`”. Where 8003 is the port that will be used outside of the container and 80 is the port being exposed inside the container.



```
PS C:\shoppingcartdemo\MTC_EnterpriseSPADev> docker run -d -p 8003:80 angular-simple-shopping-cart
6349d305bb770f878617c3e5e99b9bc21d1b0b9ccb73435b7ce40835752e6995
PS C:\shoppingcartdemo\MTC_EnterpriseSPADev>
```

9. Open a web browser and navigate to: "`localhost:8003`". If successful, you should see our Angular SPA running locally as such:

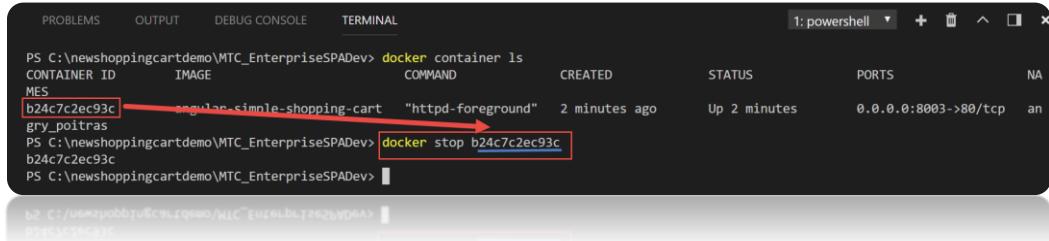


10. In terminal Run: “`docker container ls`”. This will show you the container for which your image is running, and the details of that particular instance. Sample output would look like the following:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: powershell ▾ + 🗑 ⌂ X

PS C:\newshoppingcartdemo\MTC_EnterpriseSPADev> docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NA
MES
b24c7c2ec93c angular-simple-shopping-cart "httpd-foreground" 2 minutes ago Up 2 minutes 0.0.0.0:8003->80/tcp an
gry_poitras
PS C:\newshoppingcartdemo\MTC_EnterpriseSPADev> █
```

11. In Terminal Run: “`docker stop b24c7c2ec93c`” Where `b24c7c2ec93c` is the container ID from the previous step. This will be unique to each container. The following command will stop the container running your image. Sample output would like the following:



A screenshot of a Windows PowerShell window titled "1: powershell". The terminal shows the following command sequence:

```
PS C:\newshoppingcartdemo\MTC_EnterpriseSPADev> docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
b24c7c2ec93c        angular-simple-shopping-cart   "httpd-foreground"   2 minutes ago     Up 2 minutes      0.0.0.0:8003->80/tcp
gry_poitras
PS C:\newshoppingcartdemo\MTC_EnterpriseSPADev> docker stop b24c7c2ec93c
b24c7c2ec93c
PS C:\newshoppingcartdemo\MTC_EnterpriseSPADev>
```

The container ID "b24c7c2ec93c" and the command "docker stop b24c7c2ec93c" are highlighted with red boxes.

12. Dockerizing your Angular application is now complete!

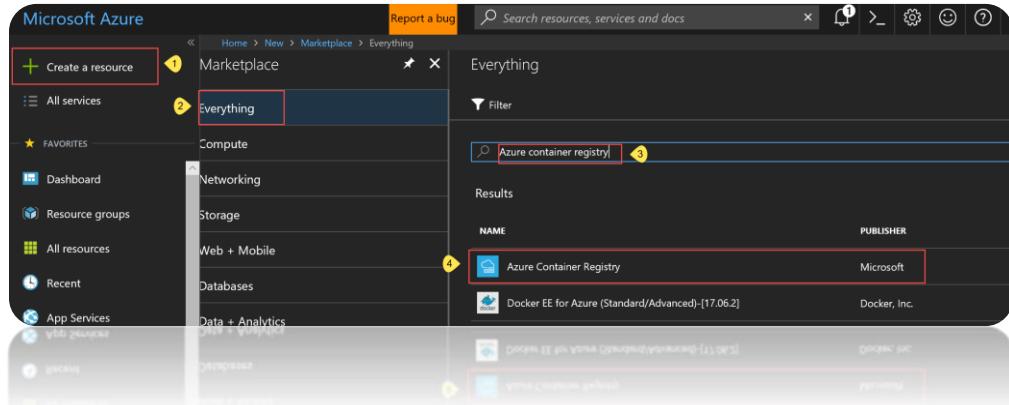
Deploy your Docker image to Azure PAAS Services using CI/CD

1. We are now going to deploy our Angular SPA to an App Service (Linux Based). We will use Docker to create a Linux image with Apache running. The steps are as follows:
 - a. Create an Azure Container Registry for our Docker Images.

Note: We could have easily used any other registry, but we will use Azure's Container Registry as setting up a CI /CD pipeline is straightforward for the purposes of this hackathon.

- b. Configure VSTS Build and deployment tasks to Build and Update the Azure Container Registry with the newly built image, mentioned in the previous section, and then deploy the image from the Registry to the Azure App Service(Linux) Deployment slot using the CI / CD integration features within VSTS.
 - c. Create a "Azure App Service(Linux)" instance.

2. First off, let's create the Azure Container Registry by creating a resource in the Azure portal (portal.azure.com):



Note: The registry name is also the username to be used for authenticating against the registry for access:

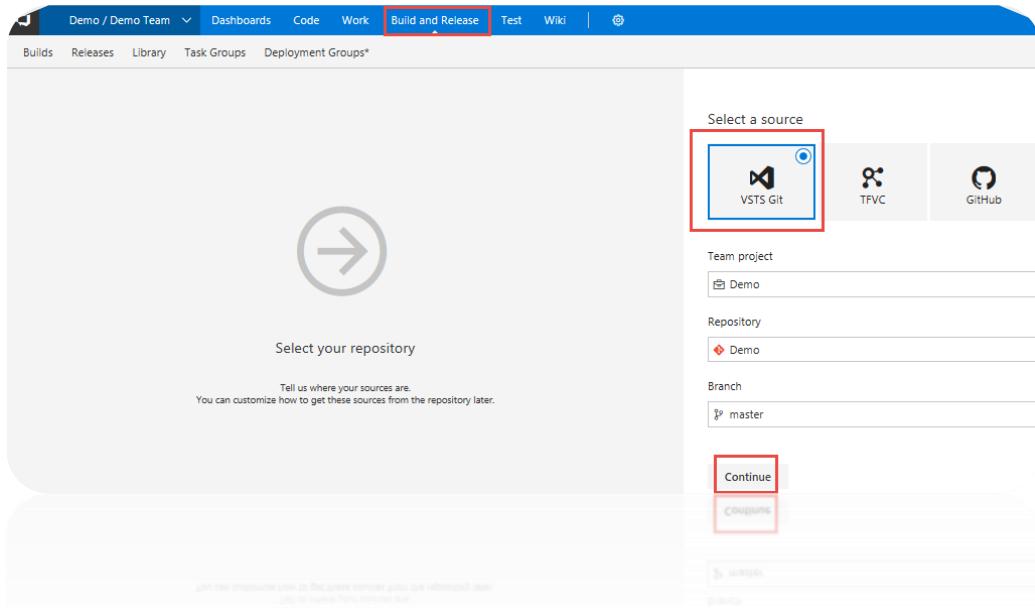
3. Provide the following settings for the Azure Container Registry as such:

The dialog box is titled 'Create container registry'. It contains the following fields:

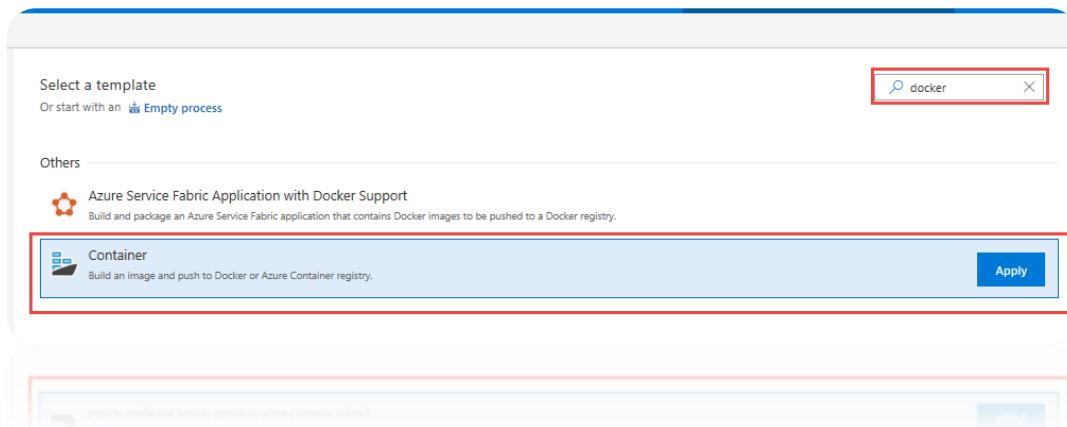
- * Registry name: `hackathondemo` (highlighted with a red box)
- * Subscription: `TORMTC`
- * Resource group:
 - Create new
 - Use existing`hackathondemo` (highlighted with a red box)
- * Location: `Canada Central`
- * Admin user:
 - Enable
 - Disable
- * SKU:
 - `Standard`

At the bottom, there is a checkbox for 'Pin to dashboard' and two buttons: 'Create' (highlighted with a blue box) and 'Automation options'.

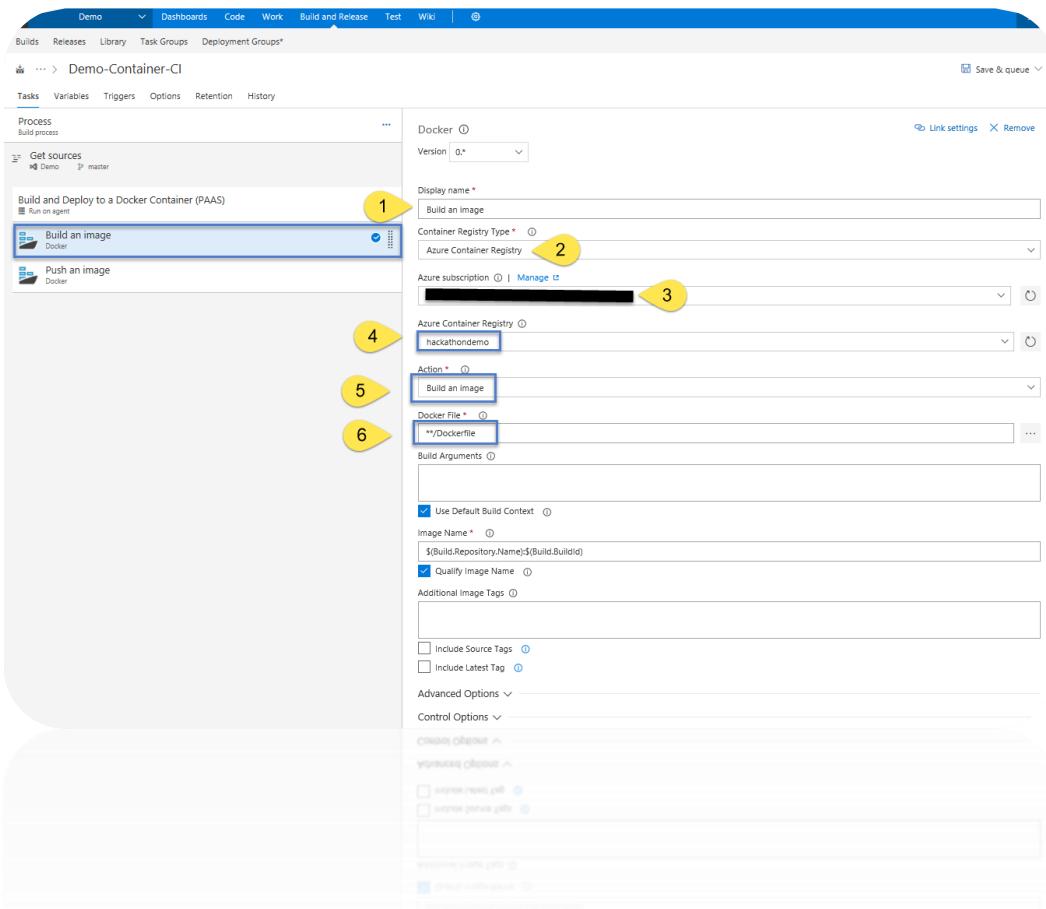
4. Configure a new Build and Deployment Definition for CI / CD. Use your Demo VSTS instance you have created earlier within this document:



5. Select the Container template:



6. Modify the “Build an image” step as such:



7. Modify the “Push an image” step as such:

The screenshot shows the Azure DevOps pipeline editor interface. At the top, there's a navigation bar with links like 'Builds', 'Releases', 'Dashboards', 'Code', 'Work', 'Build and Release', 'Test', 'Wiki', and 'Search'. Below the navigation bar, the pipeline name 'Demo-Container-CI' is displayed. The pipeline consists of several steps: 'Get sources' (repository 'Demo' branch 'master'), 'Build and Deploy to a Docker Container (PaaS)' (with 'Run on agent' selected), 'Build an image' (using Docker), and finally, the 'Push an image' task which is highlighted with a blue selection bar. This task is configured to use the 'Docker' task type, version 0.*. It has a 'Display name' of 'Push an image'. Under 'Container Registry Type', 'Azure Container Registry' is selected. An 'Azure Subscription' dropdown is set to a specific subscription. The 'Azure Container Registry' dropdown shows 'hackathondemo' selected. The 'Action' dropdown is set to 'Push an image'. The 'Image Name' field contains the expression '\$(Build.Repository.Name)\$(Build.BuildId)'. The 'Quality Image Name' checkbox is checked. There are also options for 'Include Source Tags' and 'Include Latest Tag'. The 'Image Digest File' field is empty. Below the main configuration area, there are sections for 'Advanced Options' and 'Control Options'.

8. Add Node Package manager “Install” & “Build” steps to build the Angular Application:

Adding the install step...

The screenshot shows the Azure DevOps pipeline editor for a 'Demo-Container-Cl' pipeline. On the left, there's a list of tasks: 'Get sources', 'Build and Deploy to a Docker Container (PAAS)', 'Build an image', and 'Push an image'. A yellow circle labeled '1' is on the '+' button to add a new task. To the right, a search bar says 'Add tasks' and 'Refresh'. Below it is a list of available tasks. A blue box highlights the 'npm' task, which is described as 'Install and publish npm packages, or run an npm command. Supports npmjs.com and authenticated registries like Package Management.' A yellow circle labeled '2' is on the task name, and another labeled '3' is on the 'Add' button.

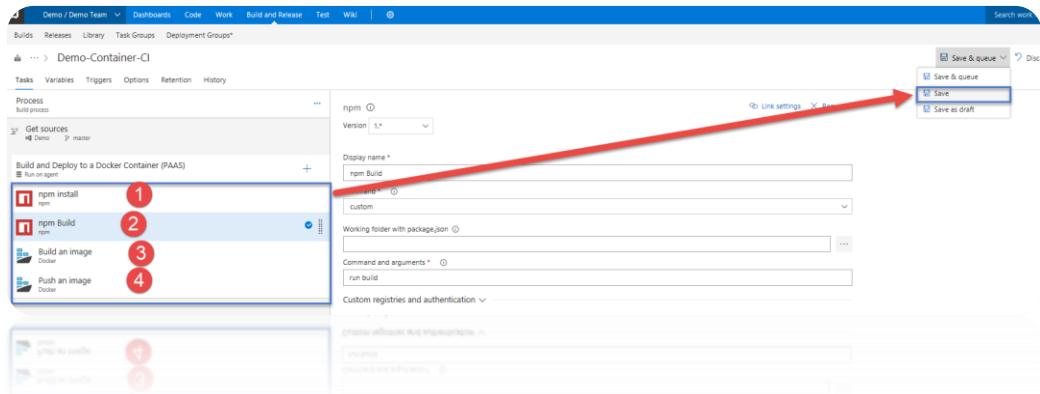
Adding the Build Step...

This screenshot is similar to the previous one but shows a different stage of the pipeline. It includes the same tasks: 'Get sources', 'Build and Deploy to a Docker Container (PAAS)', 'Build an image', and 'Push an image'. A yellow circle labeled '1' is on the '+' button. To the right, the 'Add tasks' list shows the 'npm' task again, with its description and a note about not using it if you're also using the 'npm' task. A yellow circle labeled '2' is on the task name, and another labeled '3' is on the 'Add' button.

We will modify this npm step so that it will build the application instead of running “install” again. We will do so by modifying the following settings:

This screenshot shows the configuration of the 'npm' task. The task is highlighted with a blue box and a yellow circle labeled '1'. To the right, the configuration pane shows the 'Display name' set to 'npm Build' (labeled '2'), the 'Command' set to 'custom' (labeled '3'), and the 'Command and arguments' field containing 'run build' (labeled '4'). Other settings like 'Version', 'Working folder with package.json', and 'Custom registries and authentication' are also visible.

9. Next, we will move our NPM tasks to the top and save the definition so that the final ordering looks like the following:



10. Finally, we need to specify that the build will happen on hosted VM managed by Microsoft, but we must make sure it is a Linux VM, not a Windows VM which is the default. The reasoning for this, is we need to ensure Docker can Build the image we specified within the “DockerFile”. We are targeting Apache server HTTPD which is a more robust web server than “nginx” and native to all Linux distributions. We pull this base image from the Docker registry as stated within the Docker File within our source code repository.

The line associated with defining the base image within the “DockerFile” is as follows:

```
FROM httpd:2.4
```

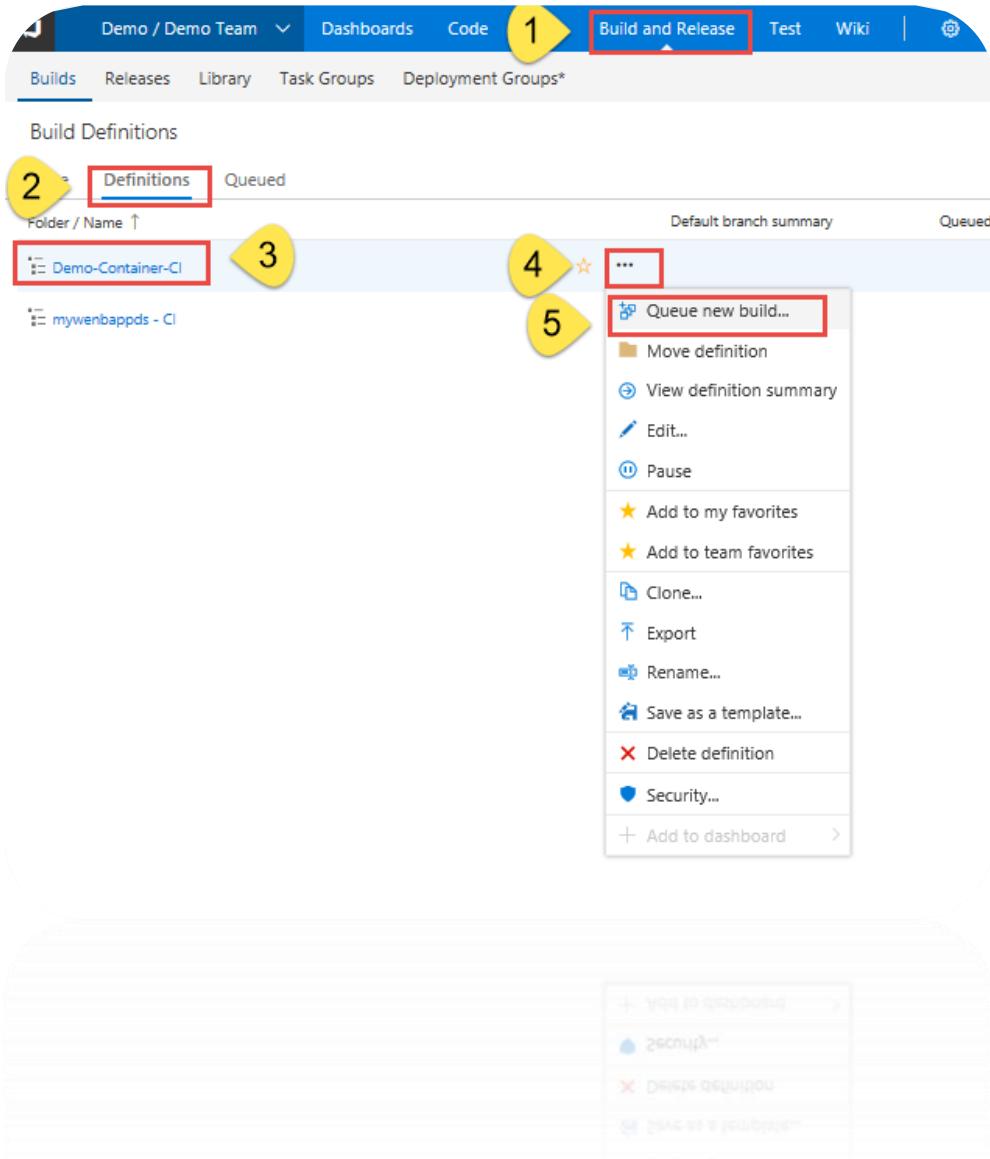
Read more on this image here: https://hub.docker.com/_/httpd/

11. So, with that, let's switch to a Linux hosted build server as such:

The screenshot shows the 'Build Definitions' page in the Azure DevOps interface. A context menu is open over the 'Demo-Container-CI' definition, with the 'Edit...' option highlighted by a red box.

The screenshot shows the 'Edit' dialog for the 'Demo-Container-CI' build definition. The 'Agent queue' dropdown is open, showing options like 'Hosted Linux Preview', 'Hosted macOS Preview', 'Hosted VS2017', and 'Default (No agents)'. The 'Hosted Linux Preview' option is highlighted by a red box.

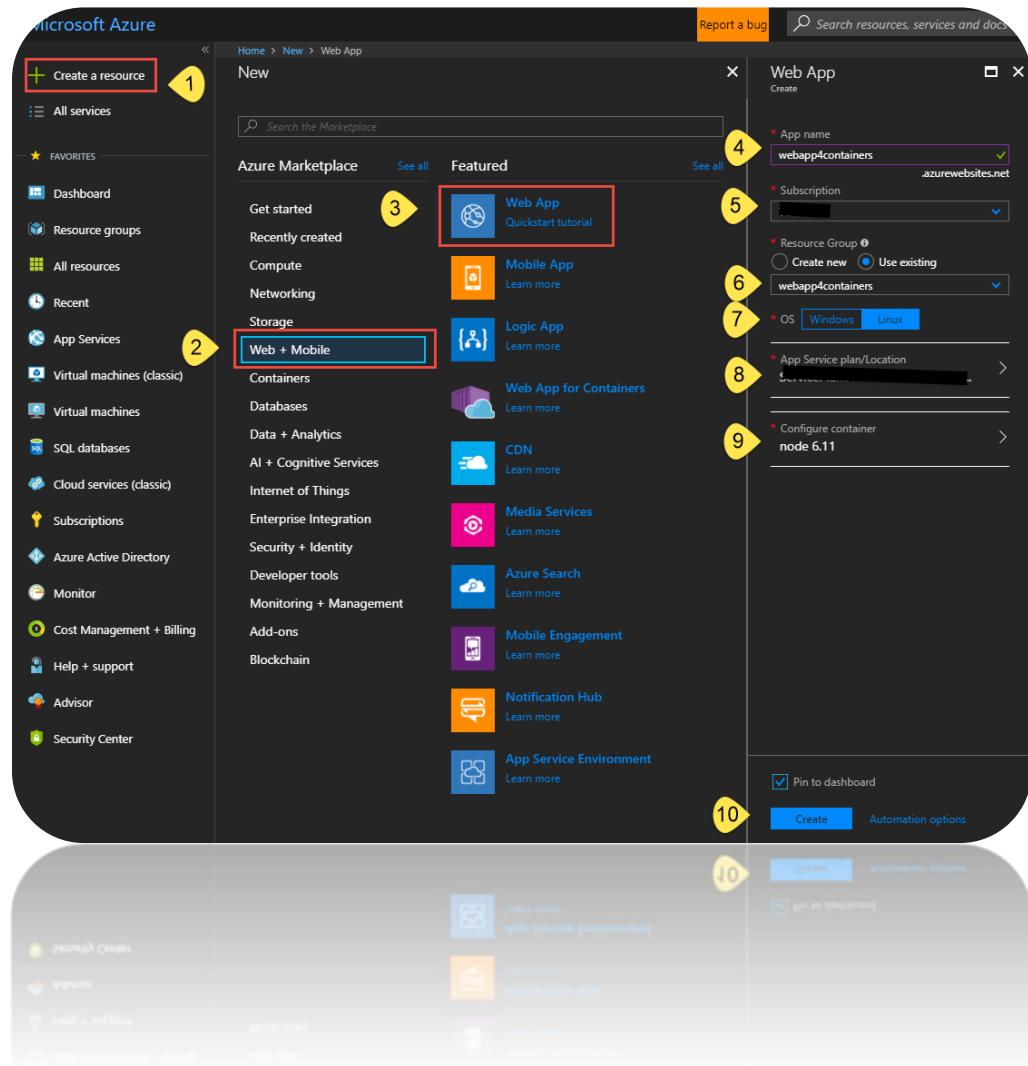
12. Queue a new build as such:



13. At this point the CI/CD pipeline will create a new docker image every time the code is checked in. Notice that I am using the build number as part of the image name to differentiate the different images that are resulting from different builds. The image below shows the ACR repository.

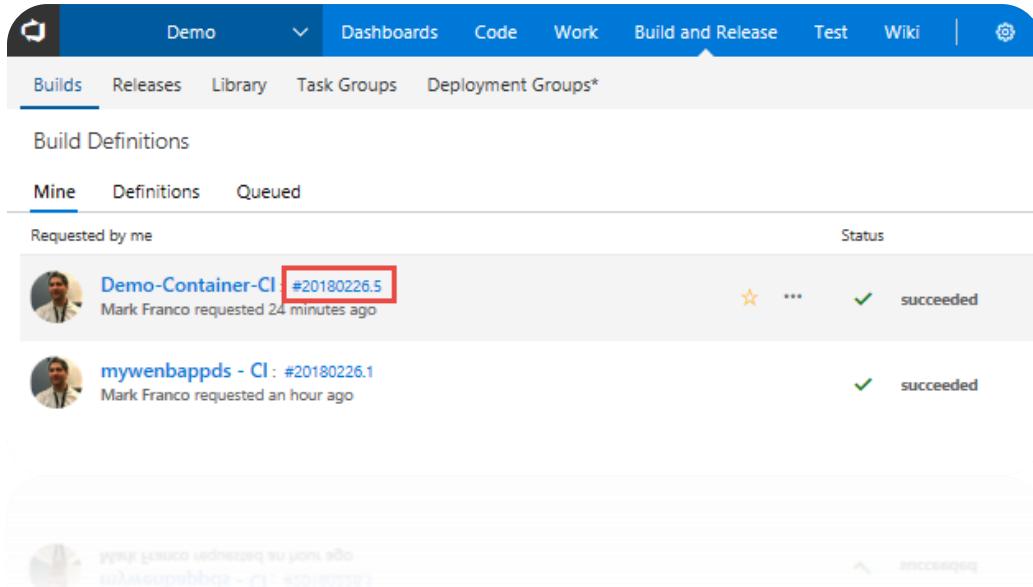
The screenshot shows the Microsoft Azure portal interface. On the left, the navigation menu is visible with various service icons. The main content area is titled "hackathondemo - Repositories" under "Container registry". The left sidebar contains links for Overview, Activity log, Access control (IAM), Tags, Quick start, Settings (Access keys, Locks, Automation script), Services (Repositories, Webhooks, Replications (Preview)), and Support + Troubleshooting (New support request). The main pane displays a list of repositories. One repository, "demo", is selected and highlighted with a red box. To the right of the repository list is a "Tags" section. Within the "Tags" section, there is a list of tags, with the tag "23" highlighted with a red box. A callout bubble points to the "23" tag with the text "Our newly built Docker Image with build number 23".

14. Now that the image is stored inside the ACR we will need to set up continuous deployment of our Docker-enabled app to an Azure web app(Linux). Start by creating an Azure web app to host the container. This can be achieved in Azure by creating a "Web App" (Linux Based) as shown below. Notice that I am pointing my web app to the ACR repository that I created in the previous step. At this point you may be thinking that I am hard coding my app to utilize an image with a specific tag number. Don't worry about it as I will override that when I push the docker container from within VSTS.



15. Next, we create a release definition on VSTS that will deploy to the Web App we had created above. Follow these steps to create a release definition:

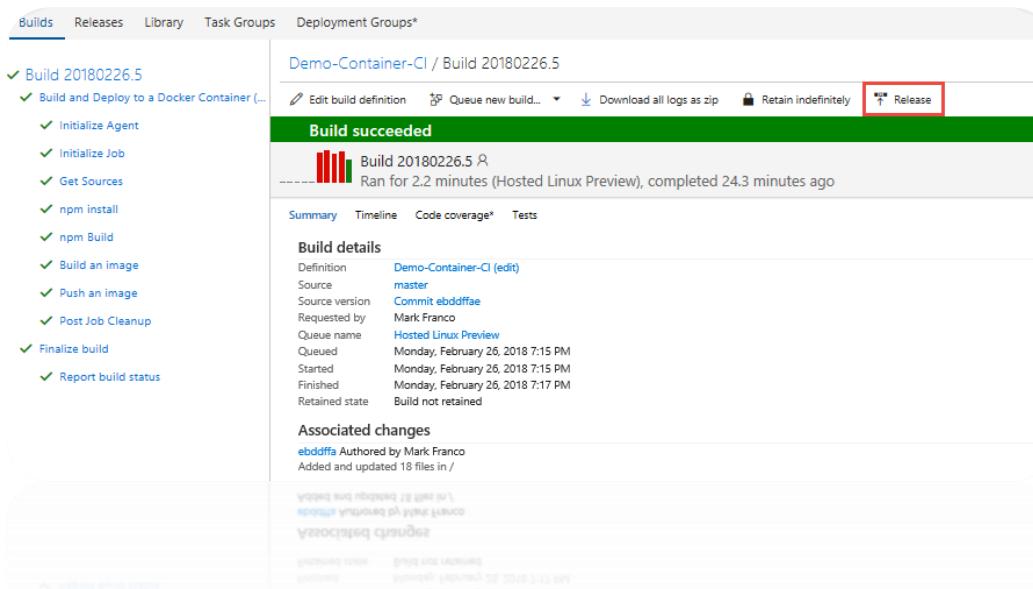
In the **Build & Release** hub, open the build summary for your build.



The screenshot shows the VSTS Build & Release hub with the 'Builds' tab selected. There are two builds listed under 'Requested by me':

- Demo-Container-CI #20180226.5**: Mark Franco requested 24 minutes ago. Status: succeeded.
- mywenbappds - CI : #20180226.1**: Mark Franco requested an hour ago. Status: succeeded.

16. In the build summary page, choose the **Release** icon to start a new release definition.



The screenshot shows the build summary for 'Demo-Container-CI / Build 20180226.5'. The 'Release' button is highlighted with a red box. The summary section shows the build succeeded, running for 2.2 minutes and completed 24.3 minutes ago.

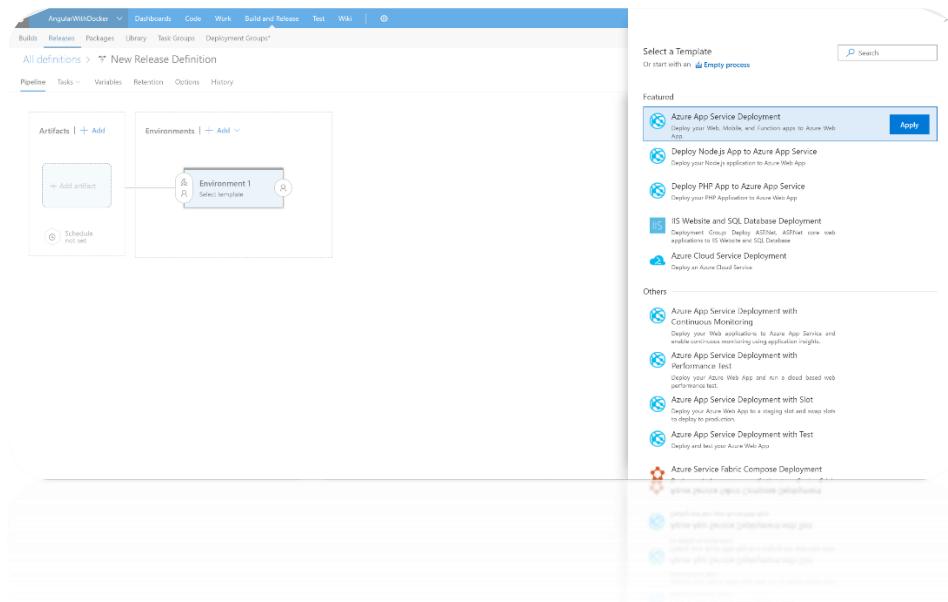
Build details

Definition	Demo-Container-CI (edit)
Source	master
Source version	Commit ebddffea
Requested by	Mark Franco
Queue name	Hosted Linux Preview
Queued	Monday, February 26, 2018 7:15 PM
Started	Monday, February 26, 2018 7:15 PM
Finished	Monday, February 26, 2018 7:17 PM
Retained state	Build not retained

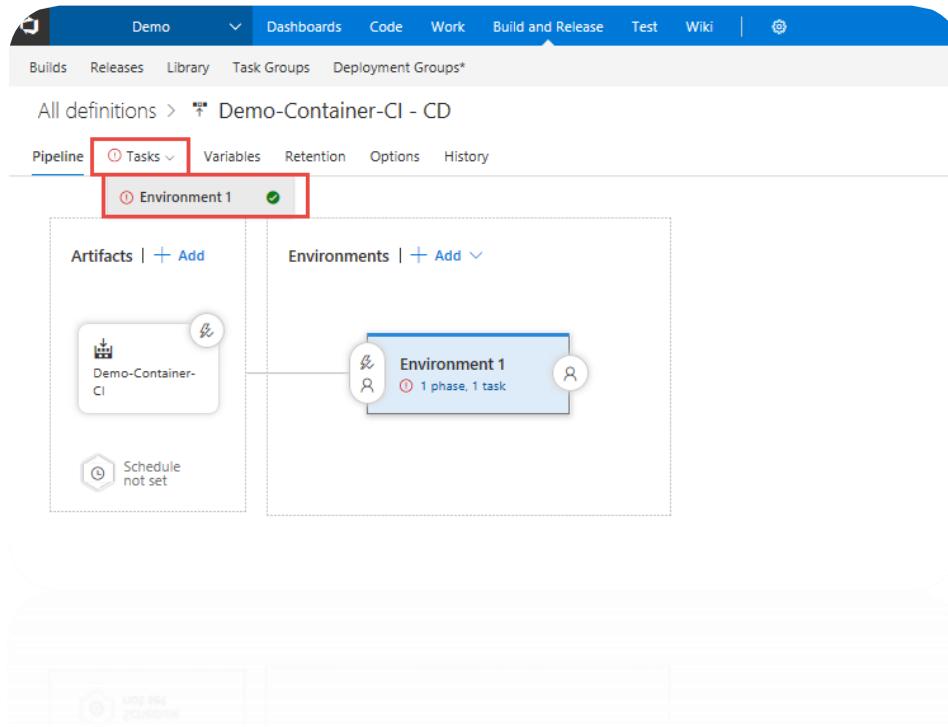
Associated changes

ebddffea Authored by Mark Franco
Added and updated 18 files in /

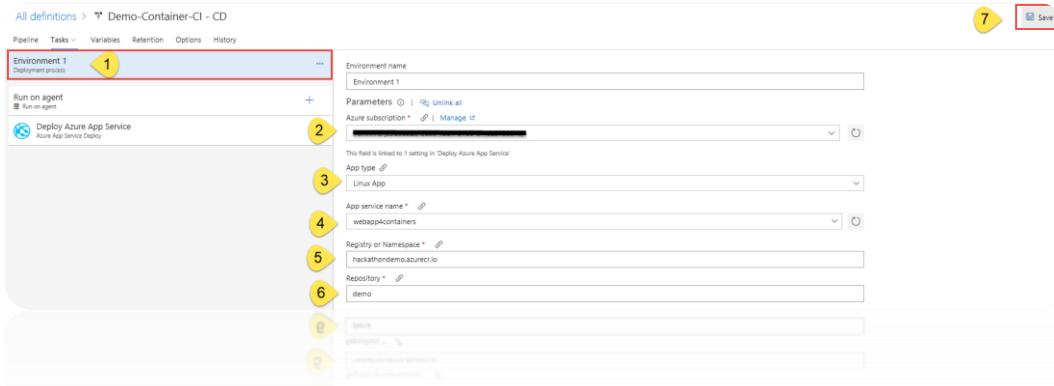
17. Select the Azure App Service Deployment task and choose Apply.



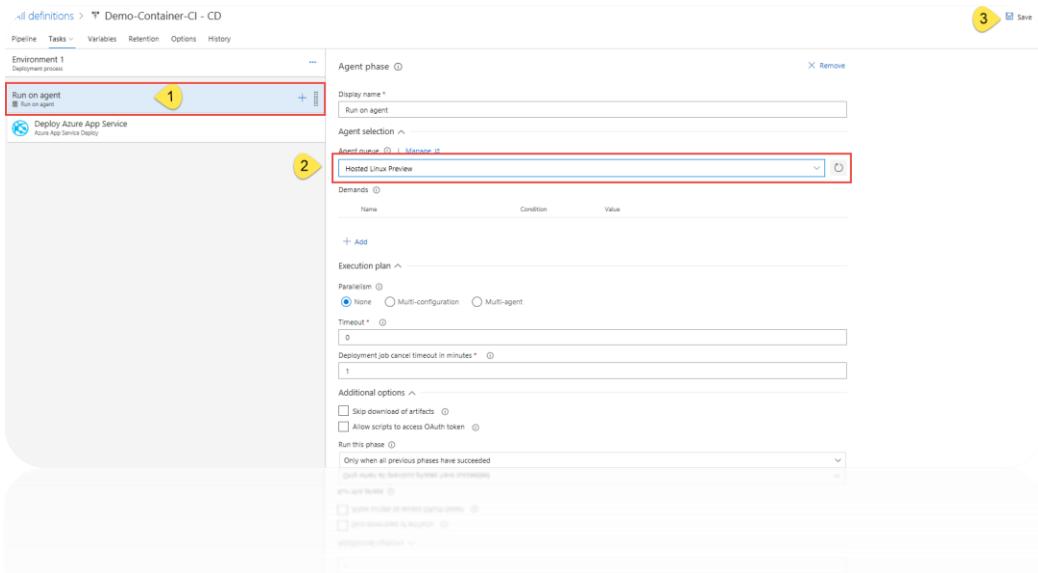
18. Select Tasks then Environment 1



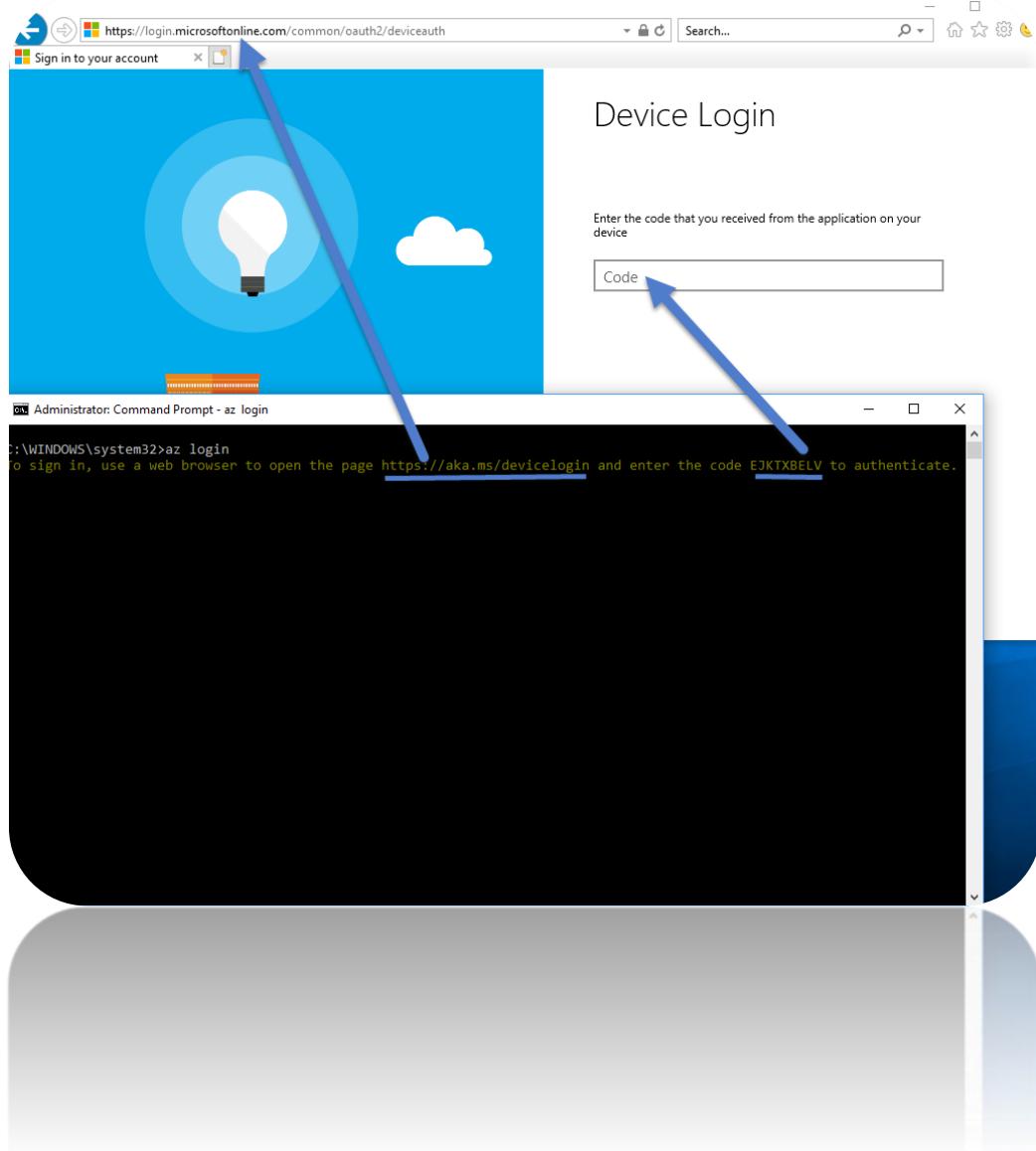
19. Configure the properties as follows:



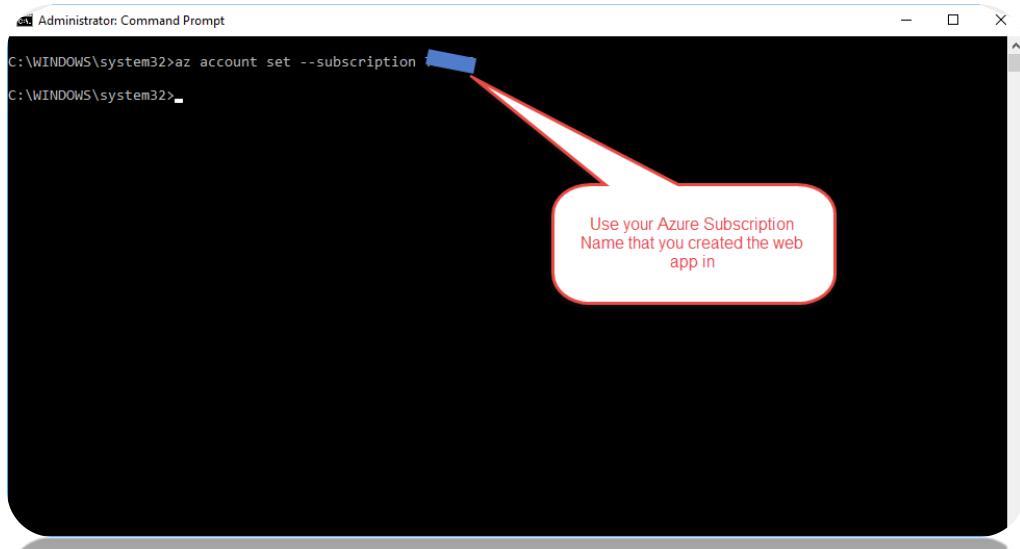
20. Set the “Run on Agent” so that the “Agent Queue” is set to “Hosted Linux Preview” as such:



21. Next, Open a windows CMD prompt in Administrator mode and execute the below commands:
22. **"az login"**. And follow the login process which will give you a device code that needs to be copied and pasted into the browser as such:

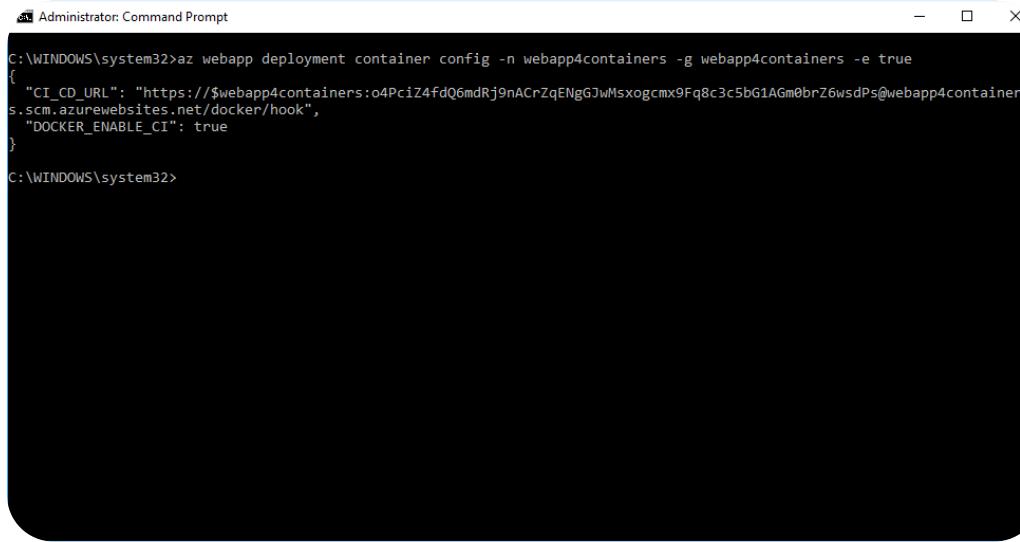


23. "az account set --subscription <YOUR SUBSCRIPTION NAME>".



A screenshot of an Administrator Command Prompt window. The window title is "Administrator: Command Prompt". The command entered is "C:\WINDOWS\system32>az account set --subscription". A red callout bubble points from the word "subscription" to a note: "Use your Azure Subscription Name that you created the web app in".

24. "az webapp deployment container config -n webapp4containers -g webapp4containers -e true".



A screenshot of an Administrator Command Prompt window. The command entered is "C:\WINDOWS\system32>az webapp deployment container config -n webapp4containers -g webapp4containers -e true". The output shows a JSON configuration block:

```
{  
  "CI_CD_URL": "https://$webapp4containers:o4Pciz4fdQ6mdRj9nACrZqENgGJwMsxogcmx9Fq8c3c5bG1AGm0brZ6wsdPs@webapp4containers.scm.azurewebsites.net/docker/hook",  
  "DOCKER_ENABLE_CI": true  
}
```

Note: the above command will open the Web App to allow for a custom container image from a public or private Container Registry. In our case, we are using Azure Container Registry.

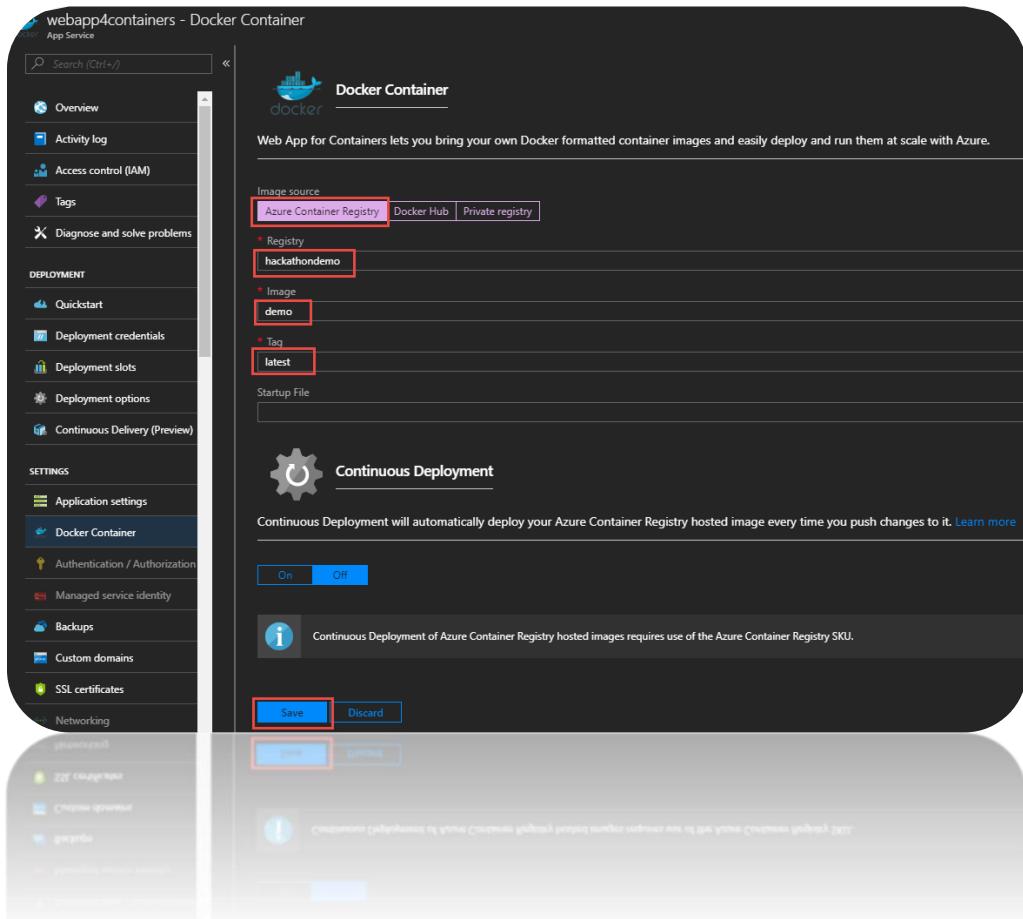
25. Now validate that you can see the Azure Container Registry option within the Docker Container settings area of our Web App. Note: if the page does not look like this, press CTRL→F5 in the browser window and you should see the updated version as such:

The screenshot shows the Azure Portal interface for a web application named "webapp4containers". The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Deployment (Quickstart, Deployment credentials, Deployment slots, Deployment options, Continuous Delivery (Preview)), and Settings (Application settings, Docker Container, Authentication / Authorization). The "Docker Container" link is highlighted with a red box. The main content area is titled "Docker Container" and displays the "Web App for Containers lets you bring your own Docker formatted container images and easily deploy and run them at scale with Azure." Below this, there is a form with the following fields:

- Image source: A dropdown menu with three options: Azure Container Registry (selected), Docker Hub, and Private registry.
- Image and optional tag (e.g. 'imagetag'): An input field.
- Server URL: An input field.
- Login username: An input field containing "marfia".
- Password: An input field with a redacted password.
- Startup File: An input field.

At the bottom of the form are "Save" and "Discard" buttons. A large red box highlights the entire configuration form.

26. Next Configure the Azure web app so that it is configured to your Azure Container Registry like so:



Note: This step is needed as there is a discrepancy in the VSTS Release “Publish to Azure Web App” action that does not configure this automatically for you during a release.

27. That's it! You have Create a full CI /CD using Docker Containers within a PAAS service in Azure known as Azure Web Apps! You can initiate a build in VSTS and the build will trigger a release upon successful build.

BONUS:

Find the setting to enable automatic build upon code check-in/Push.

Happy coding 😊.