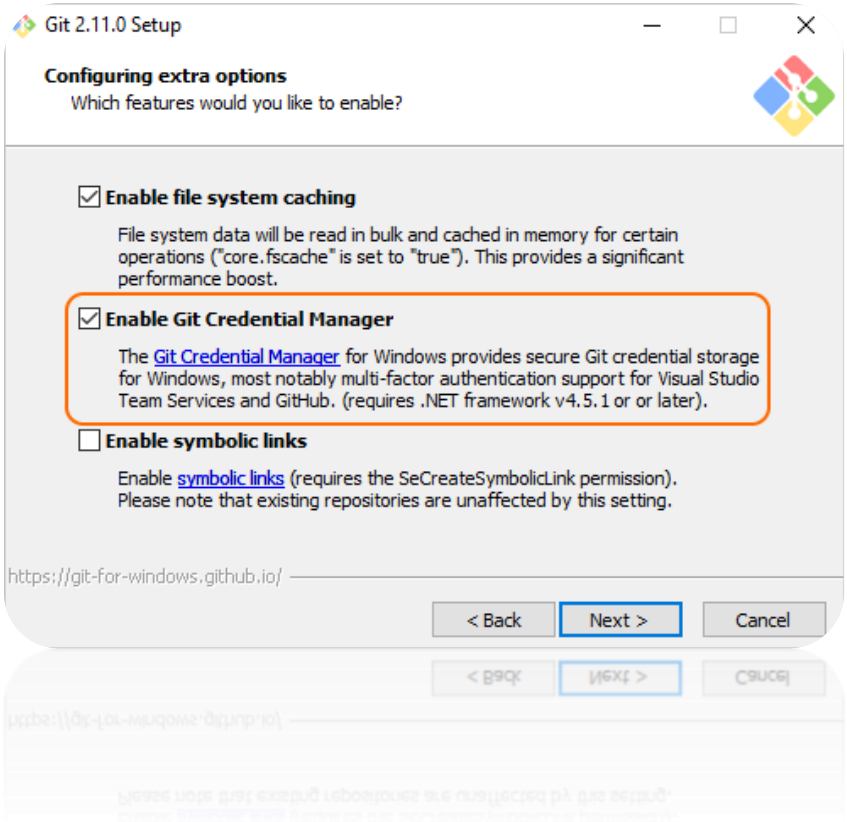
1. Pre-requisites

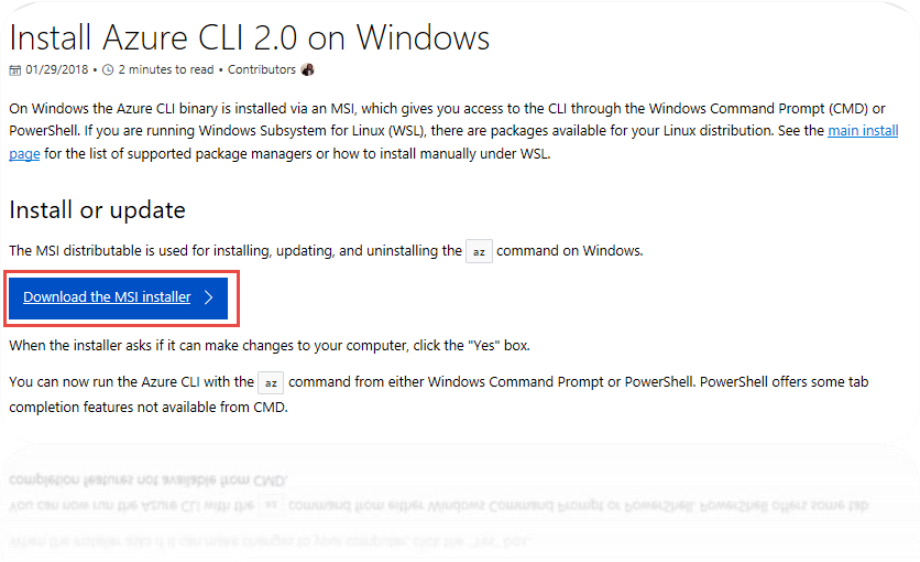
This lab assumes you have the Google Chrome browser installed and available for debugging. If you do not have Chrome installed, go to https://www.google.com/chrome/browser/

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

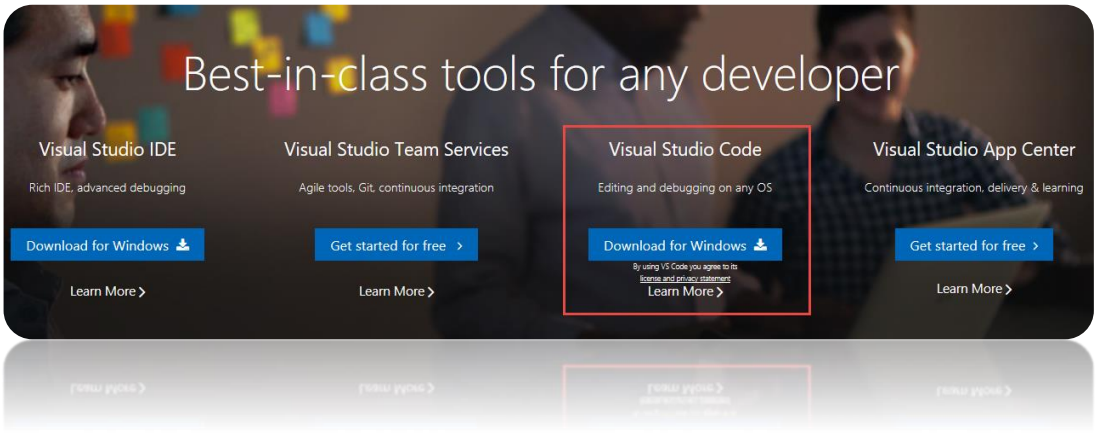
The MSI distributable is used for installing, updating, and uninstalling the `az` command on Windows.

Download the MSI installer

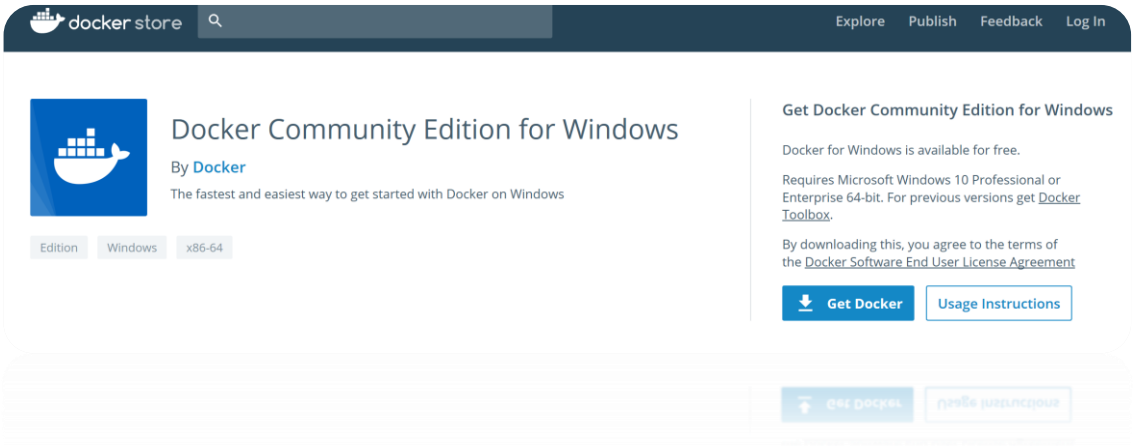
When the installer asks if it can make changes to your computer, click the "Yes" box.

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.

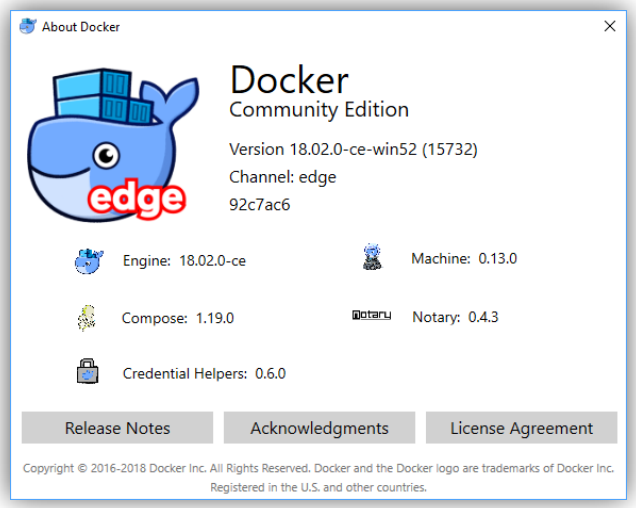
Download Visual Studio Code from http://visualstudio.com



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

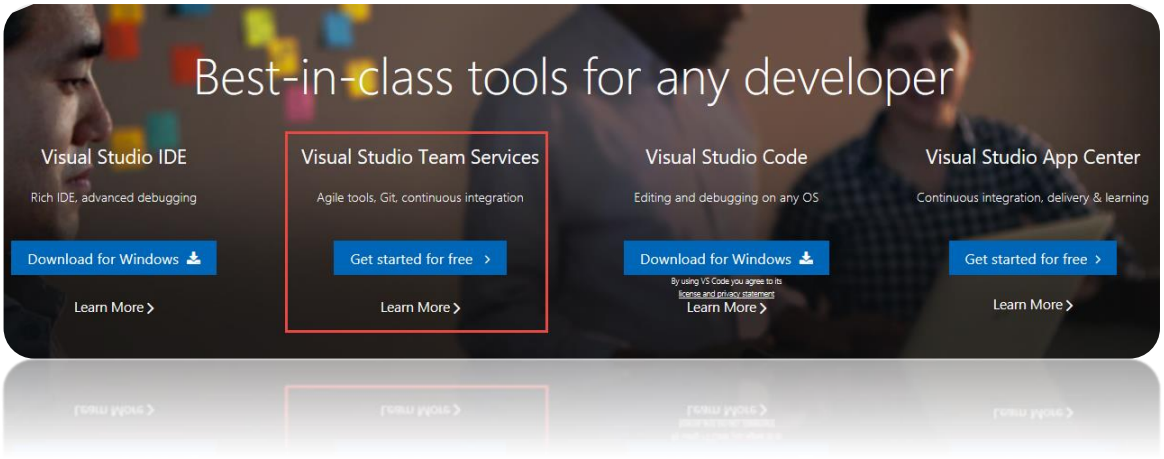


Note: Make sure you install Docker “Edge” for windows, not the “Stable” release. This guide has been verified against the following Docker version:

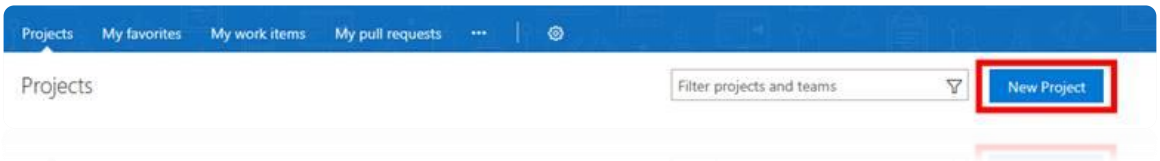


2. Create a Project in VSTS

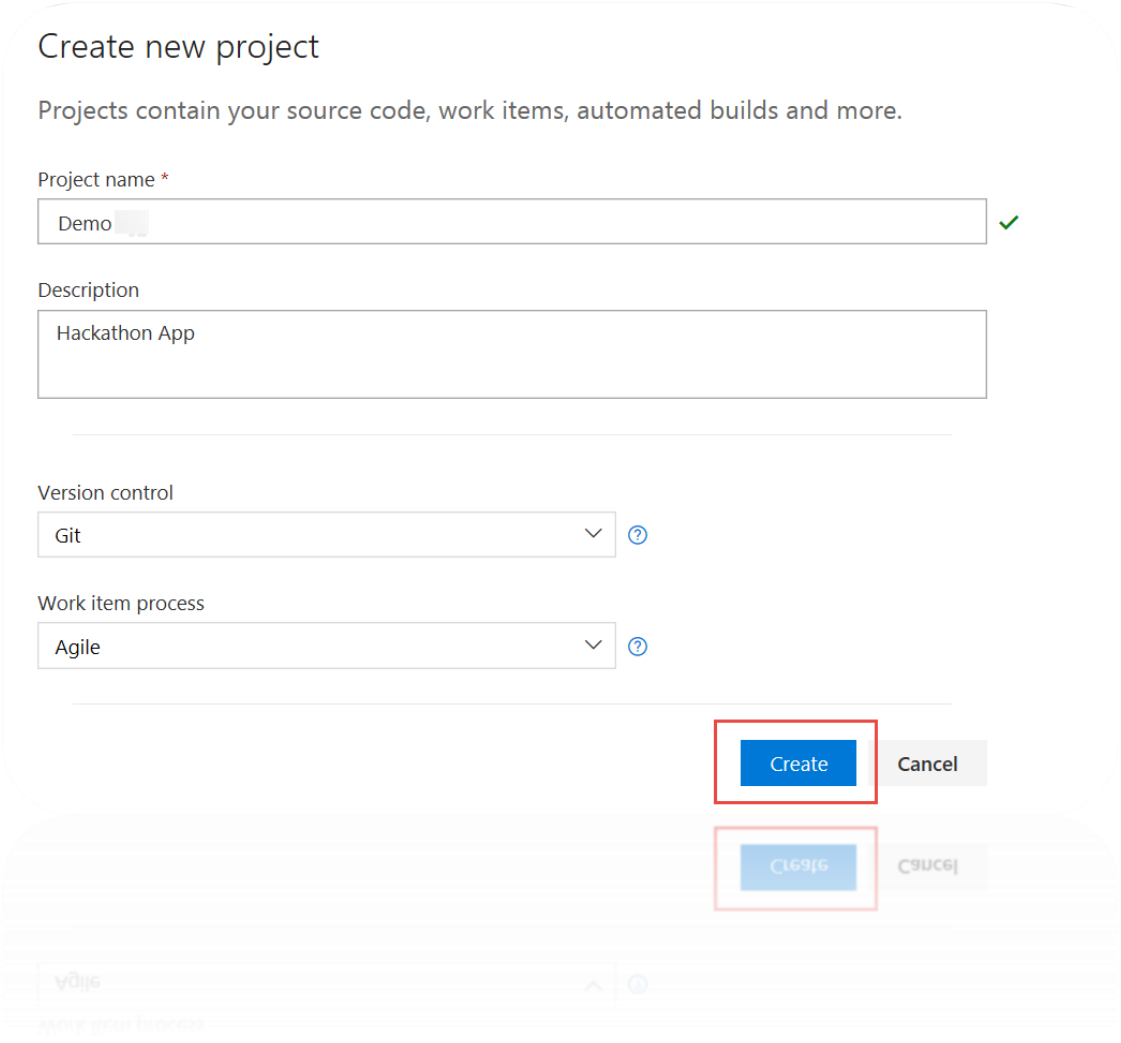
1. Create a new instance of Visual Studio Team Services by navigating to http://visualstudio.com



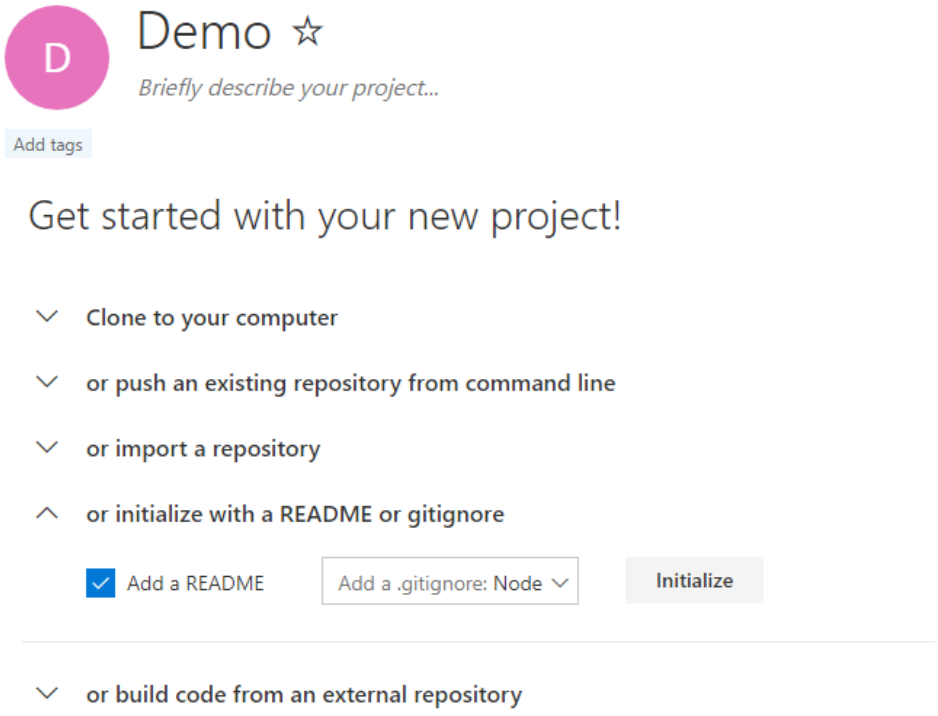
2. Click on "**New Project**" in VSTS.



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



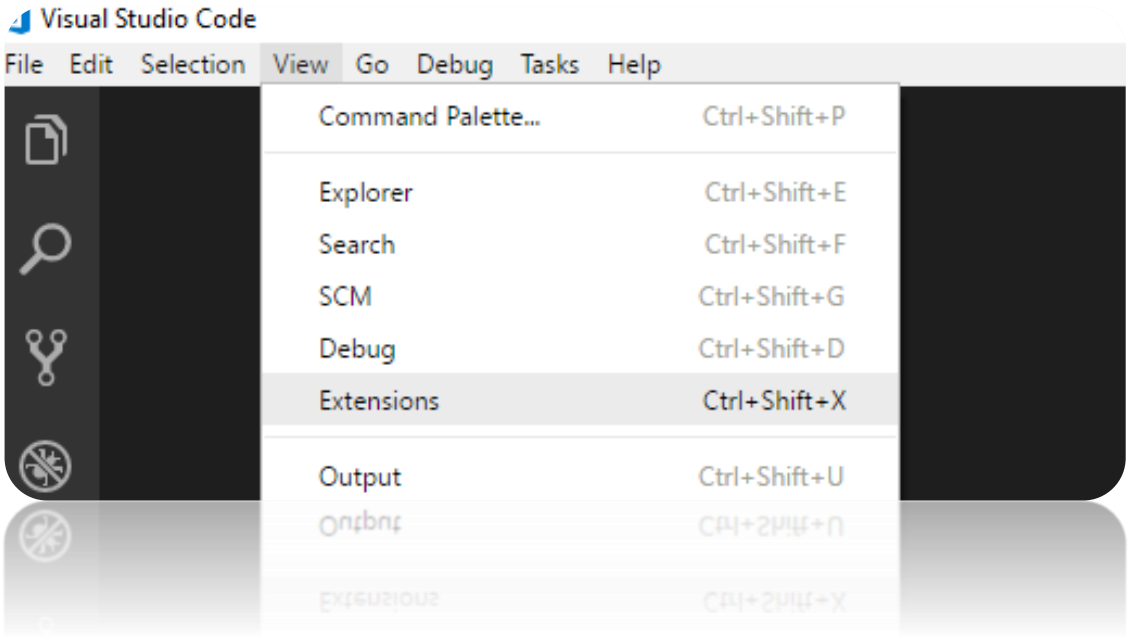
4. Select “or initialize with a readme or gitignore”.
5. Add a .gitignore file by selecting “Node”,
6. Click Initialize.



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

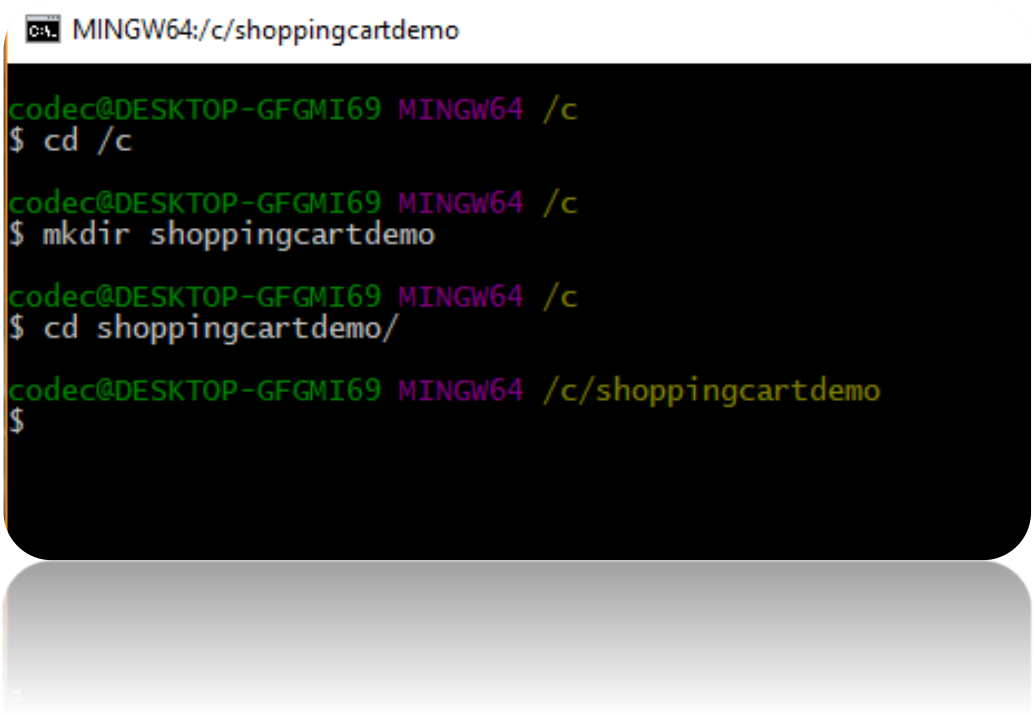
1. Install Extensions by Selecting View→ Extensions and typing "javascript"



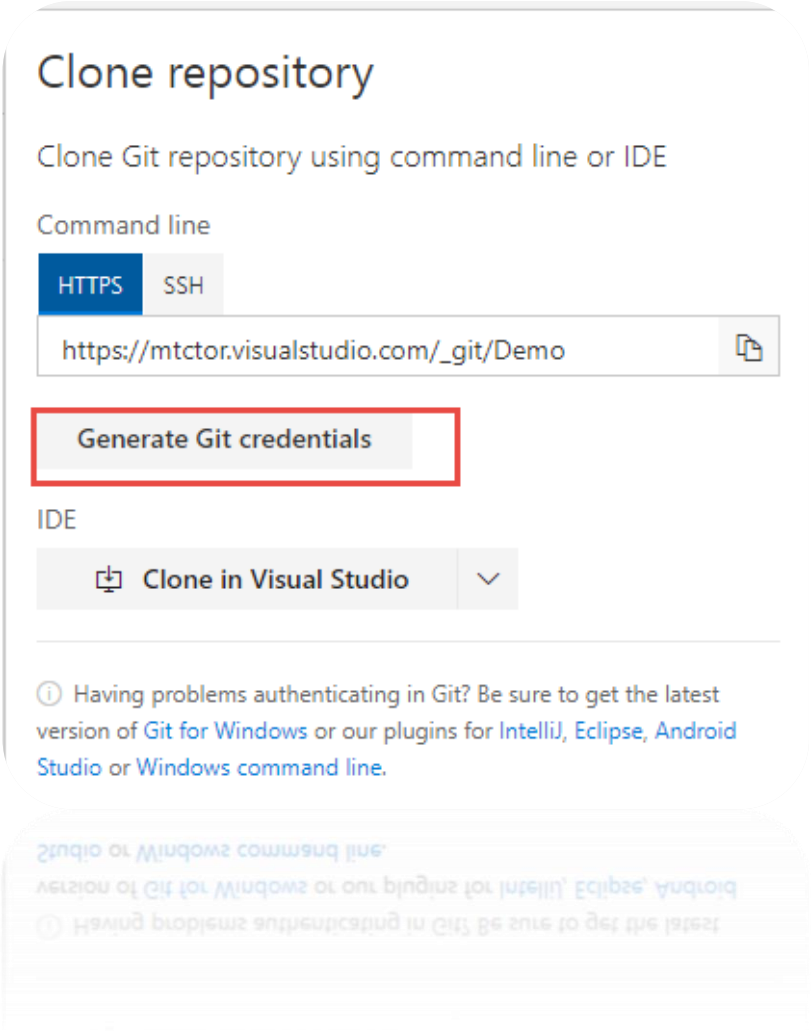
Recommended extensions to install:

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

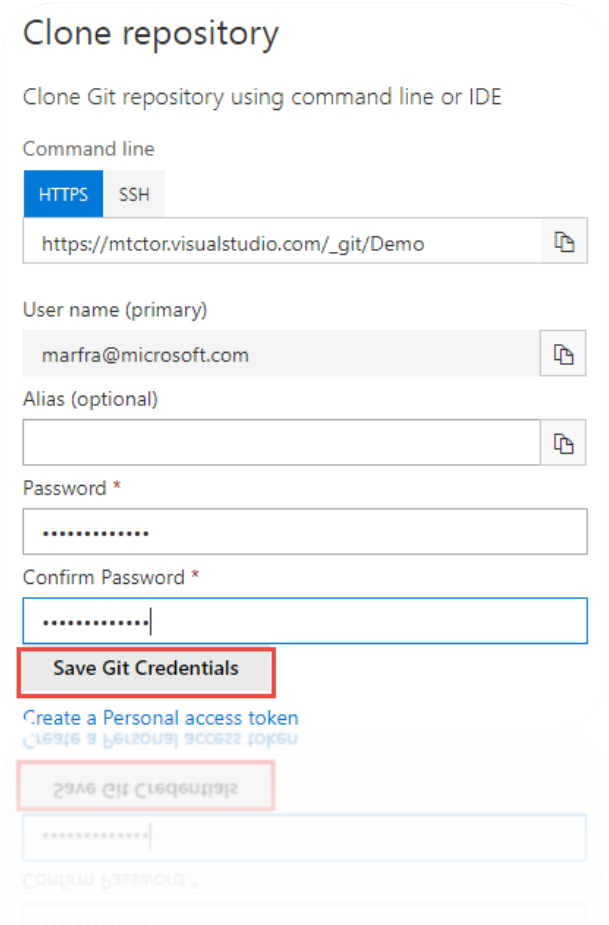
2. Launch Git Bash or use Windows Command line to execute the following commands to create our repository directory:



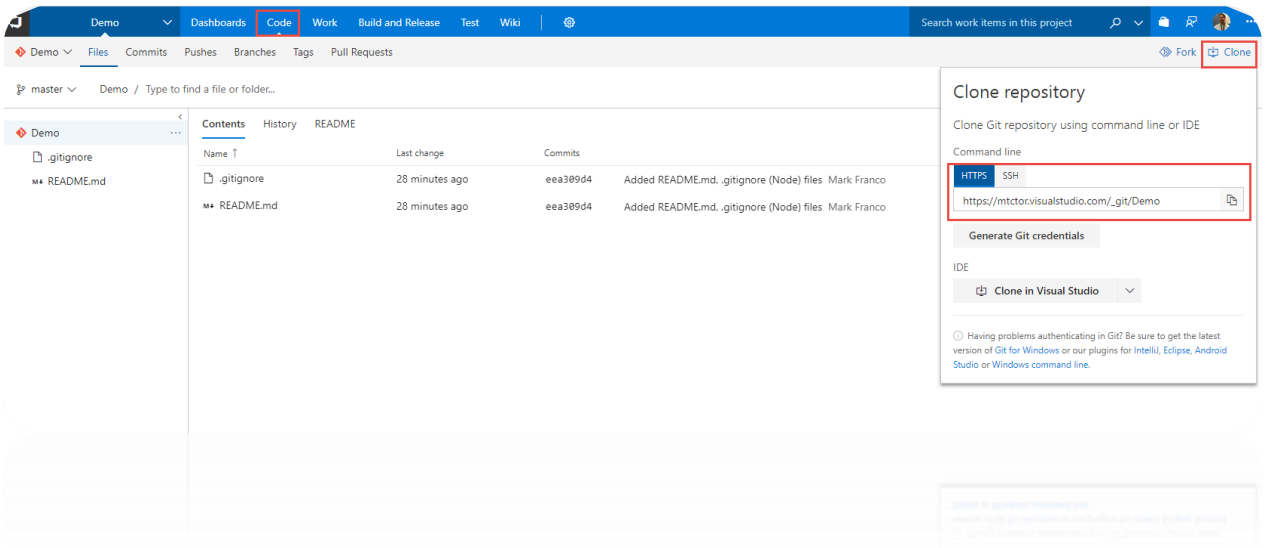
3. Open your VSTS project in your browser
4. Click on Clone in the upper right-hand corner
5. Generate Git Credentials:



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



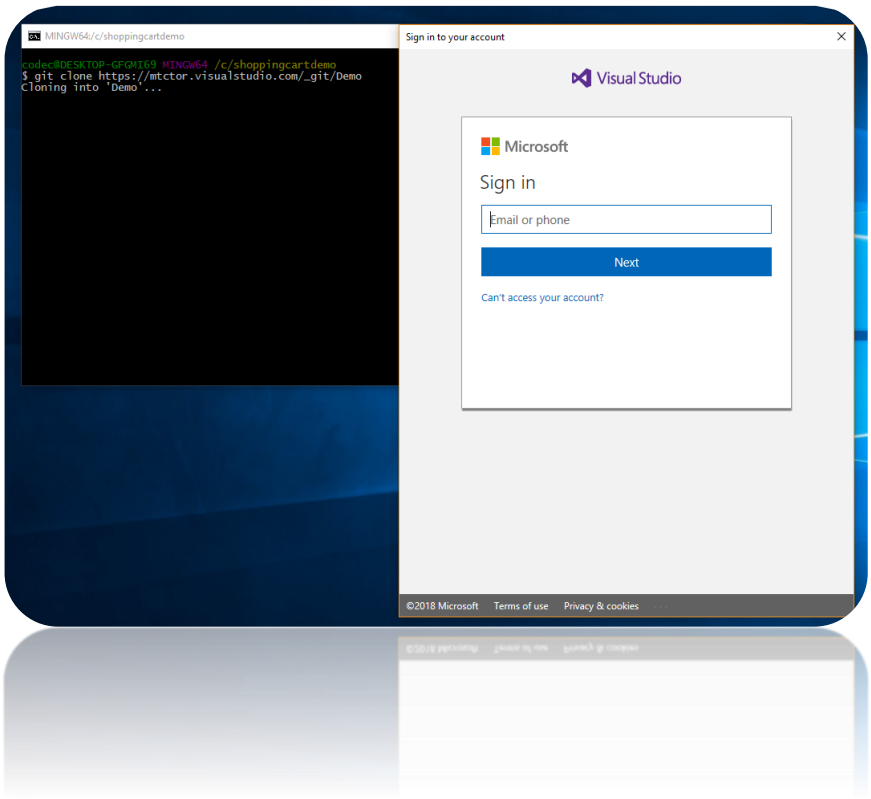
7. Copy the git repository url as follows:



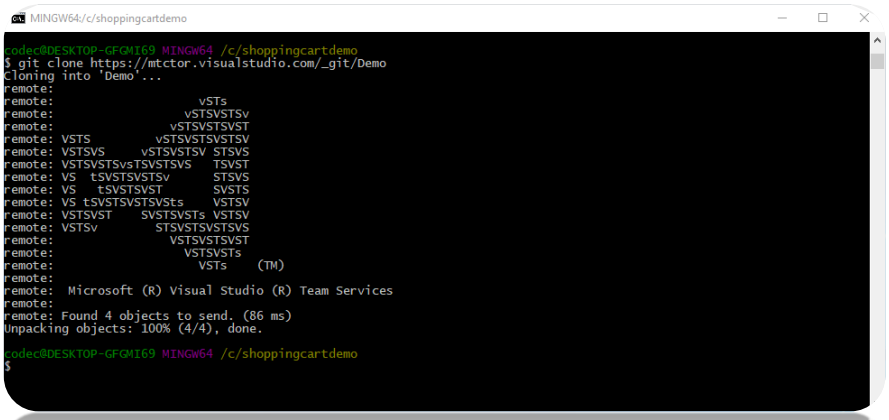
8. Clone the repository from the bash shell you opened earlier as follows:

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

9. Enter your credentials you setup in previous steps



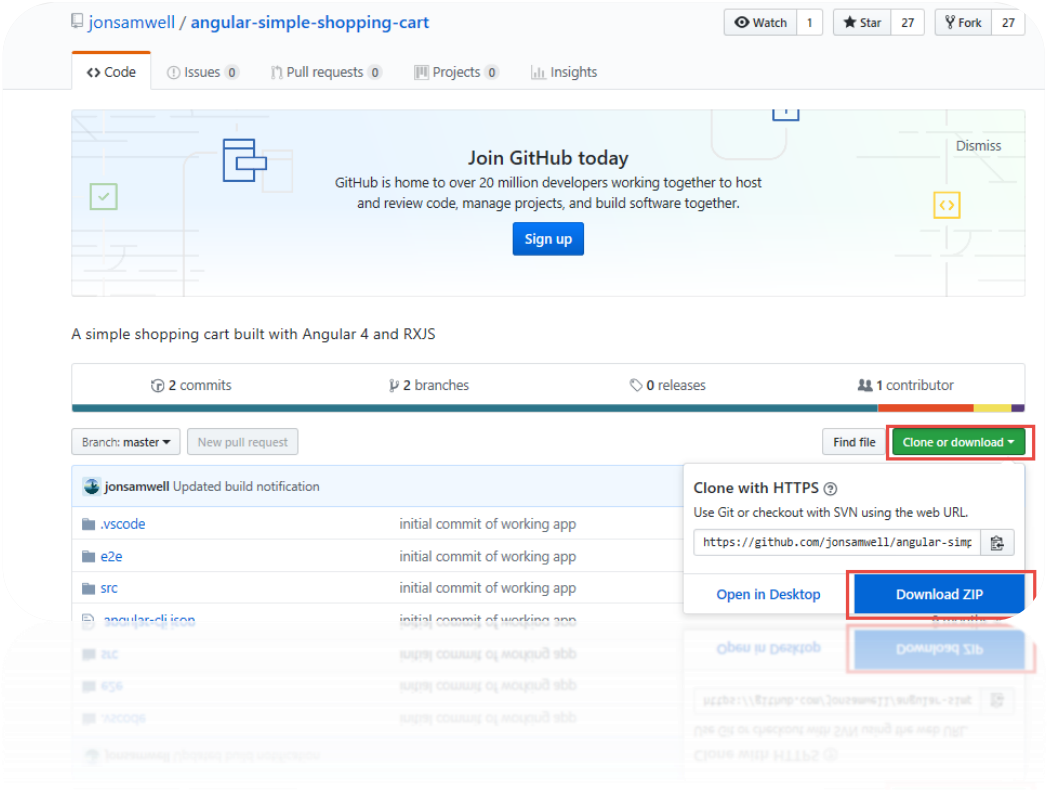
10. After successful login you should see:



4. Write some code…

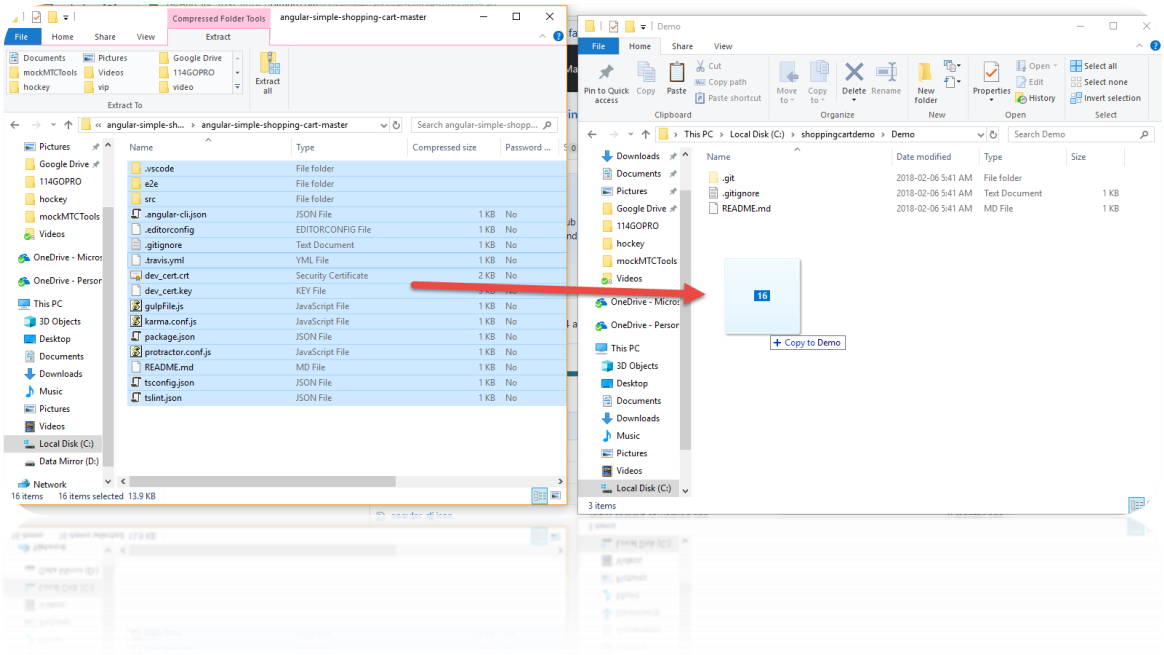
Not quite, we are just going to use an existing code base from GitHub and download the latest copy of the source to update our local repo.

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



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

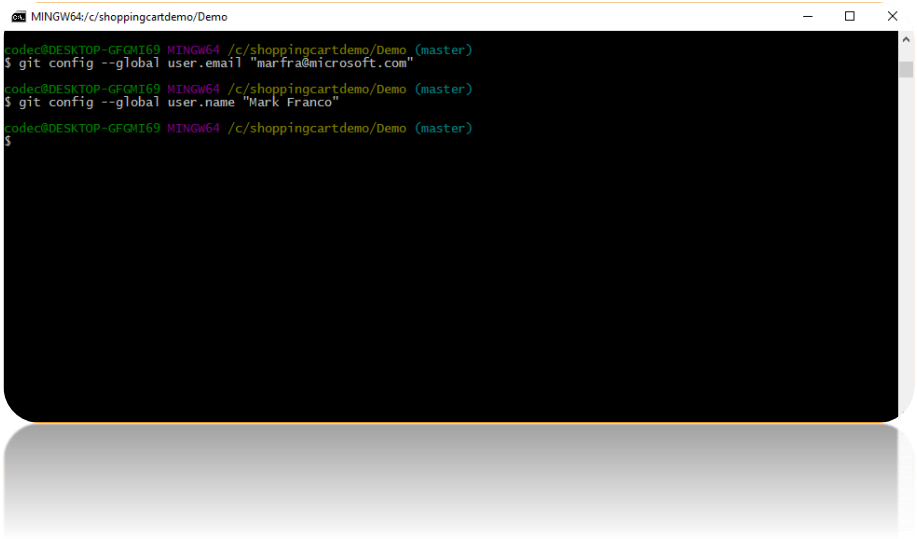
   Note: answer “replace” when duplicate files found.



4. 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:

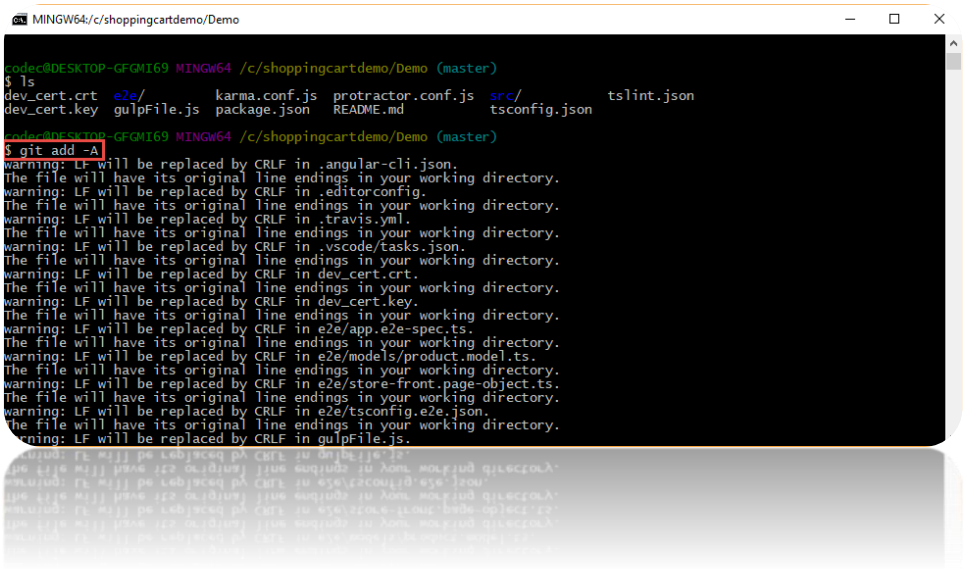
```
git config --global user.email "you@outlook.com"
```

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



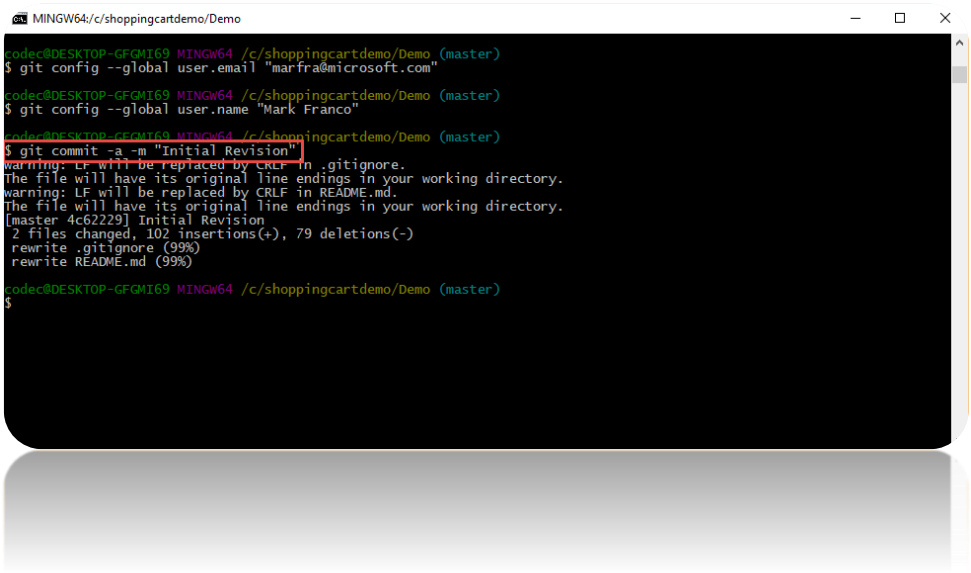
5. Add untracked files as follows:

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



6. Commit Changes:

```
git commit -a -m “Initial Revision”
```

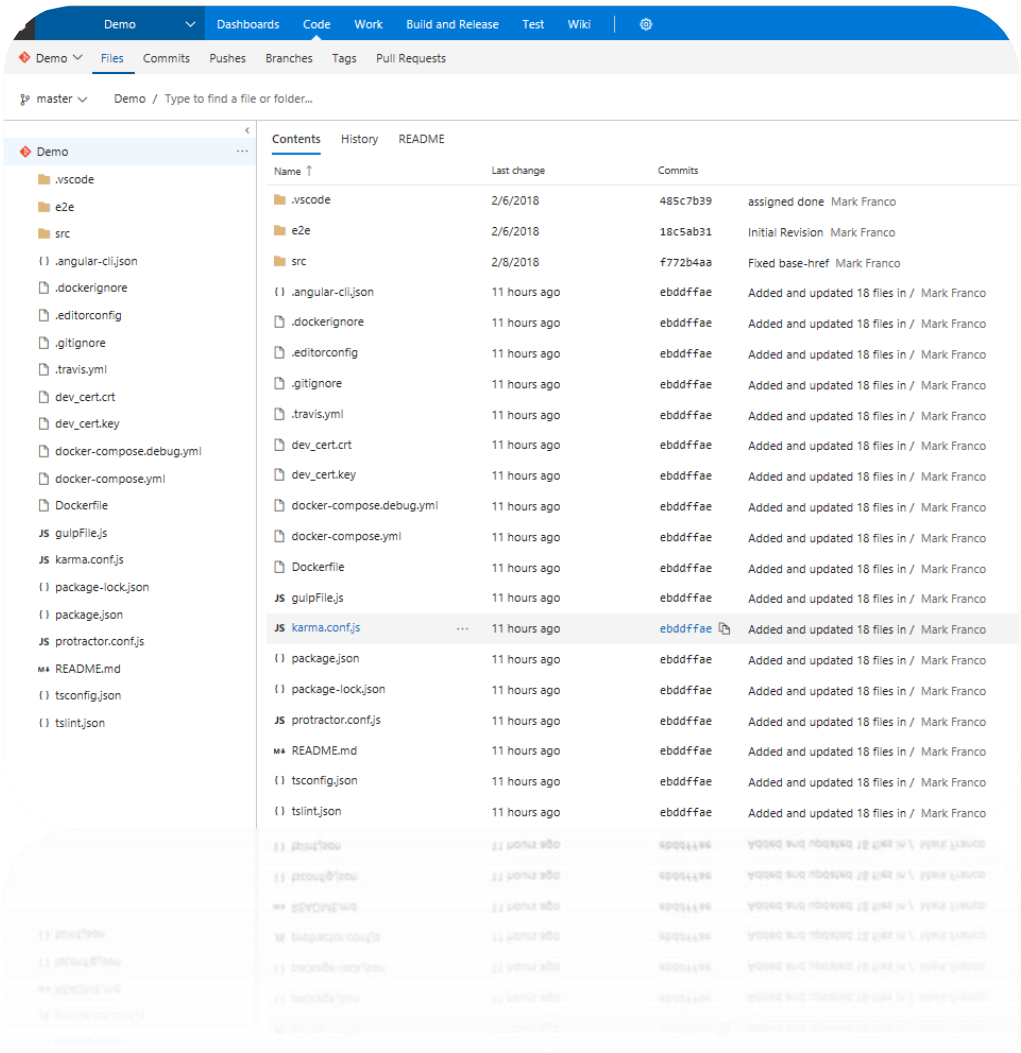


7. Push repository to VSTS into Master branch by executing the following command (no Screenshot):

   ```
   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:



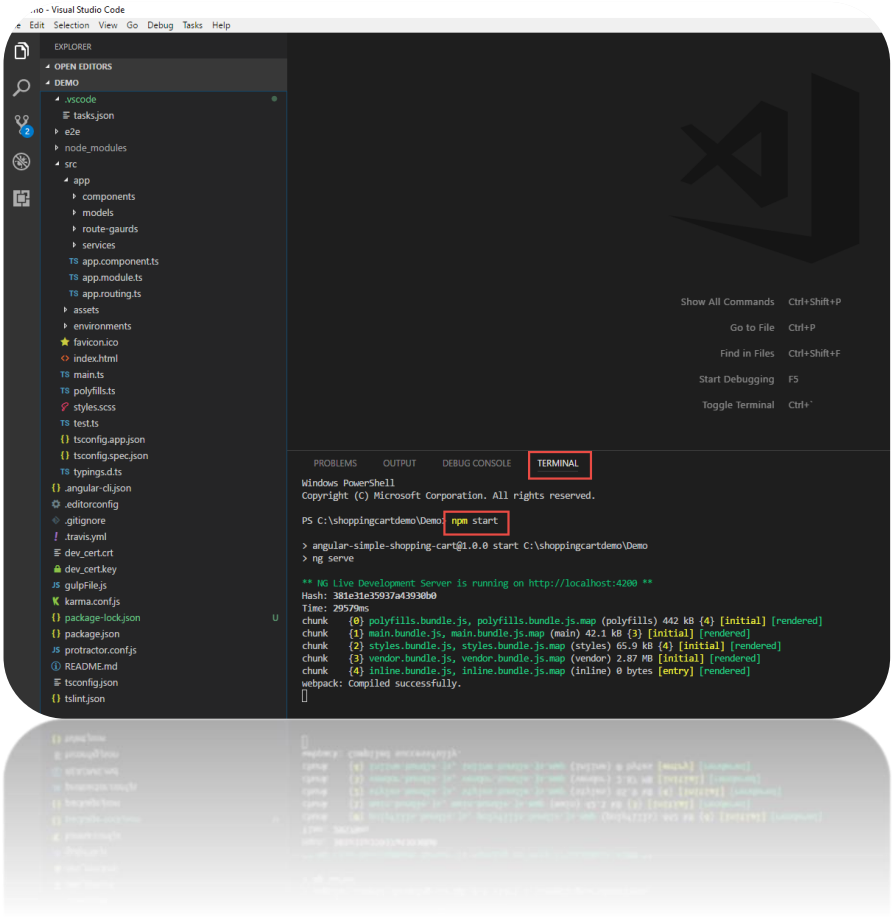
5. Setup VSTS integration using the new auth experience

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.

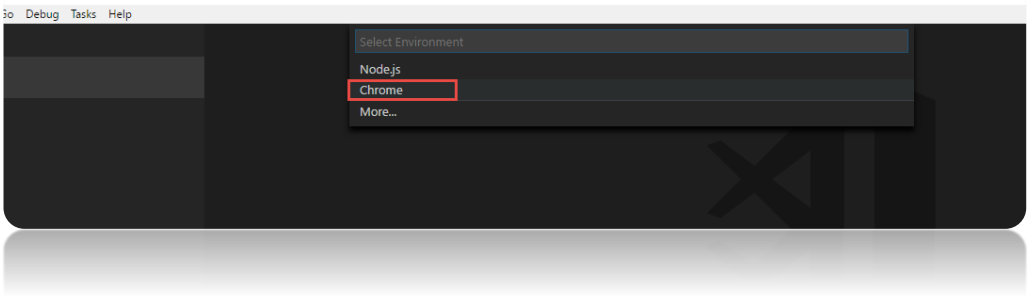
   https://youtu.be/HnDNdm1WCIo?t=2m55s

6. Let's Build something…

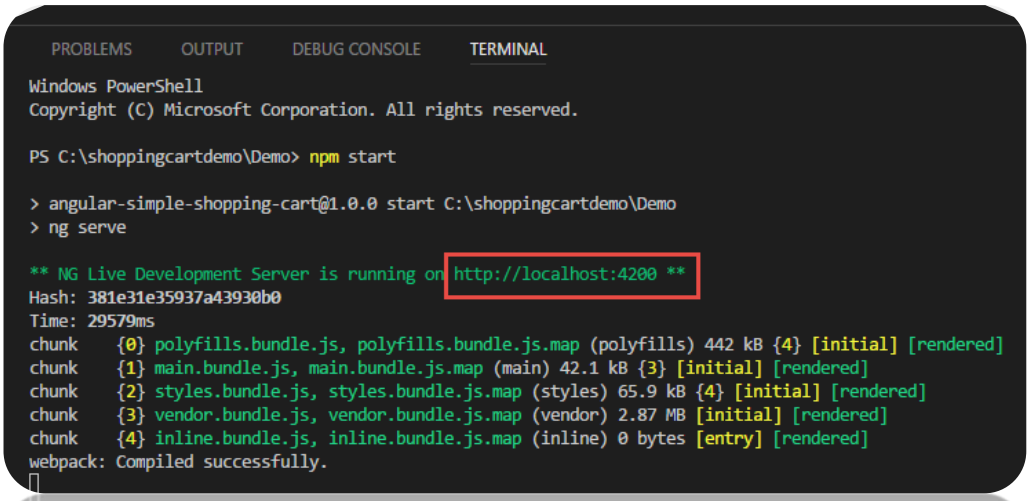
1. Once you have Cached your credentials using the new authentication experience, ensure all dependencies are current by running "npm install" in the VS Code terminal window
2. Run a local instance of the app to see how it runs by running "npm start" in the vscode terminal:



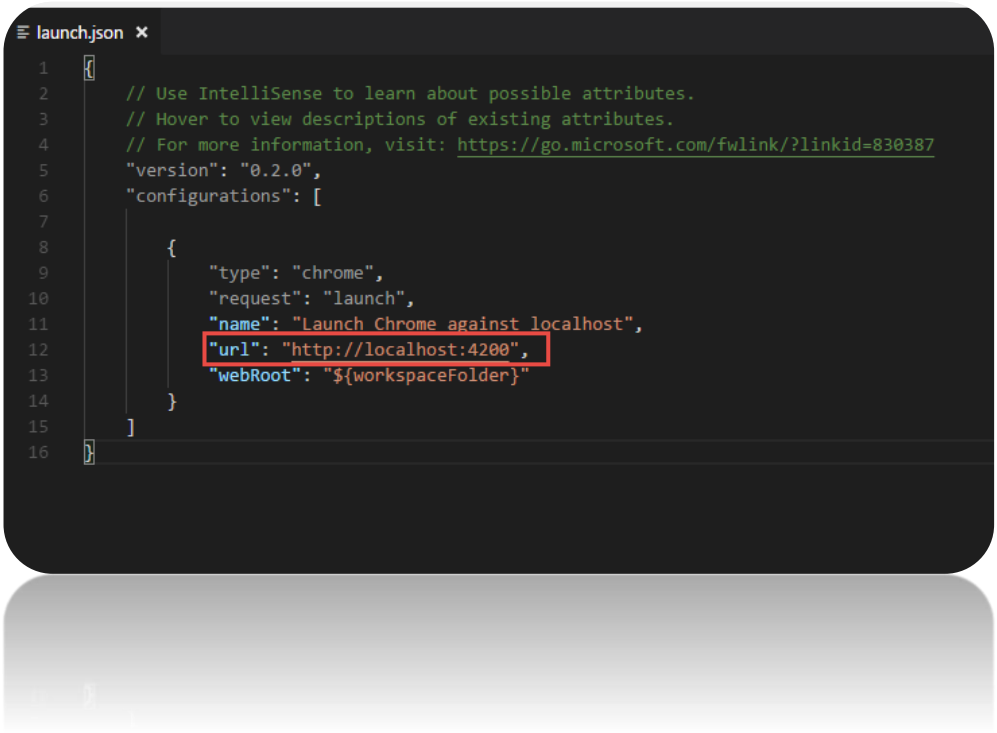
3. Your app is compiled and running under a node web server, but we need to add a launch file so we can launch a debugger window using Chrome. We do so by creating a new configuration file by selecting the "Debug→Add Configuration" menu item and selecting "Chrome" from the drop down.



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":



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:

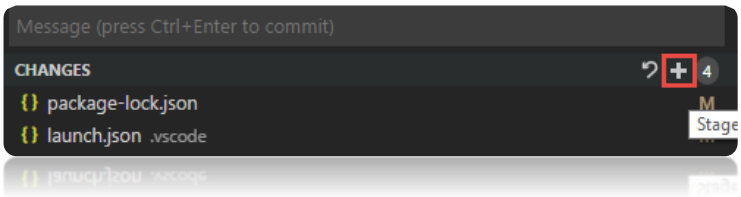


5. Now click on Debug→Start debugging

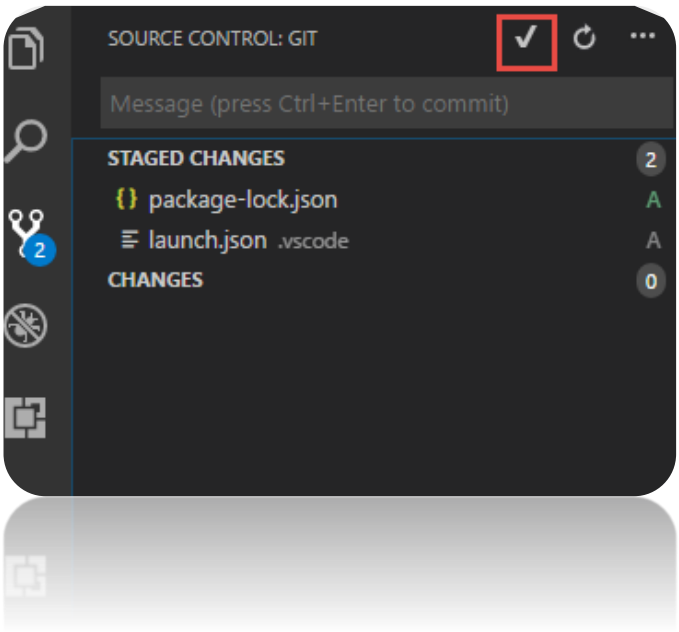
6. Try some breakpoints and debugging techniques…

7. Check in your additional file "**Launch.json**" using the VSCODE IDE now:

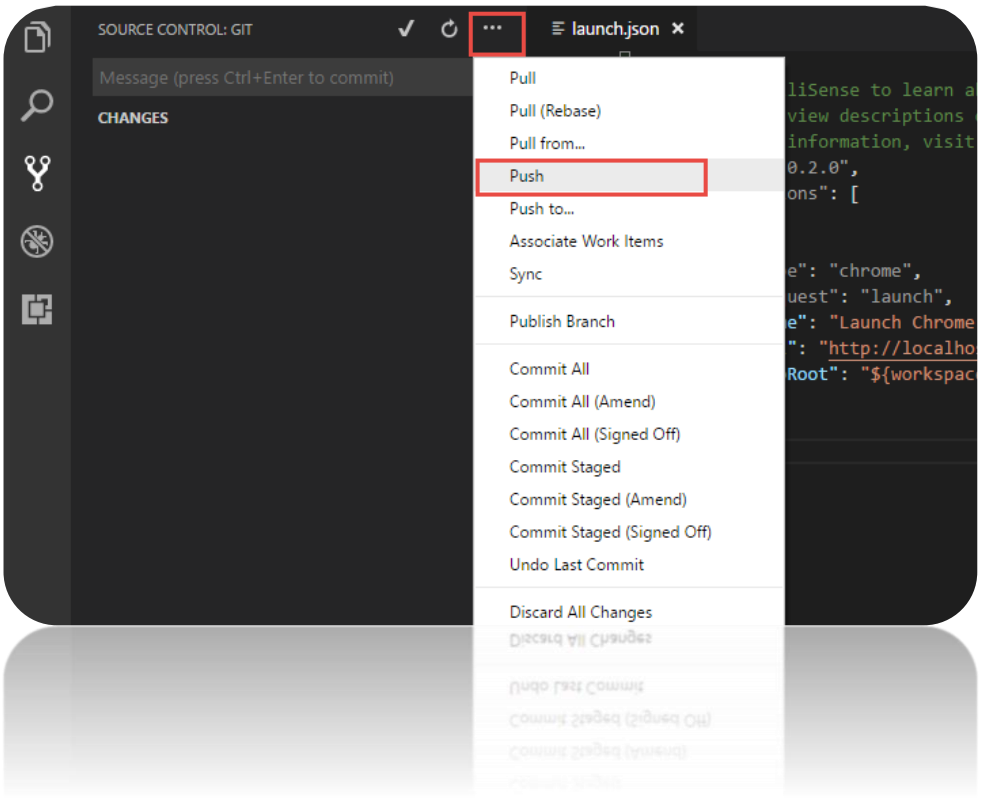
   Add Files to local repository (Stage)



8. Commit Changes to local Repository



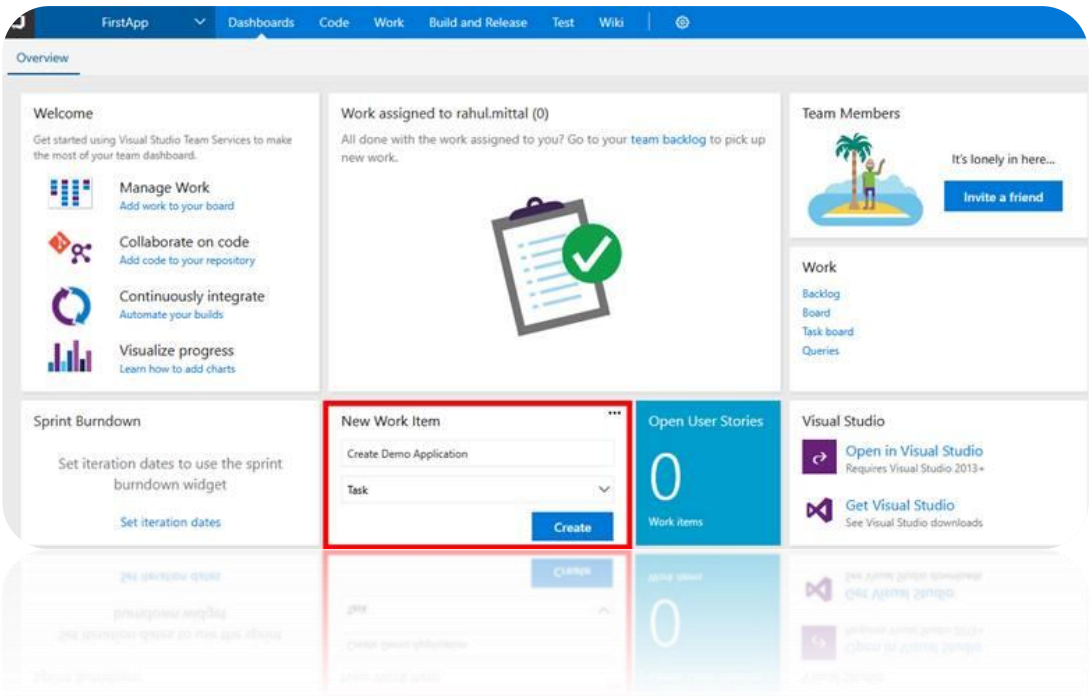
9. Push Changes from local repository to VSTS



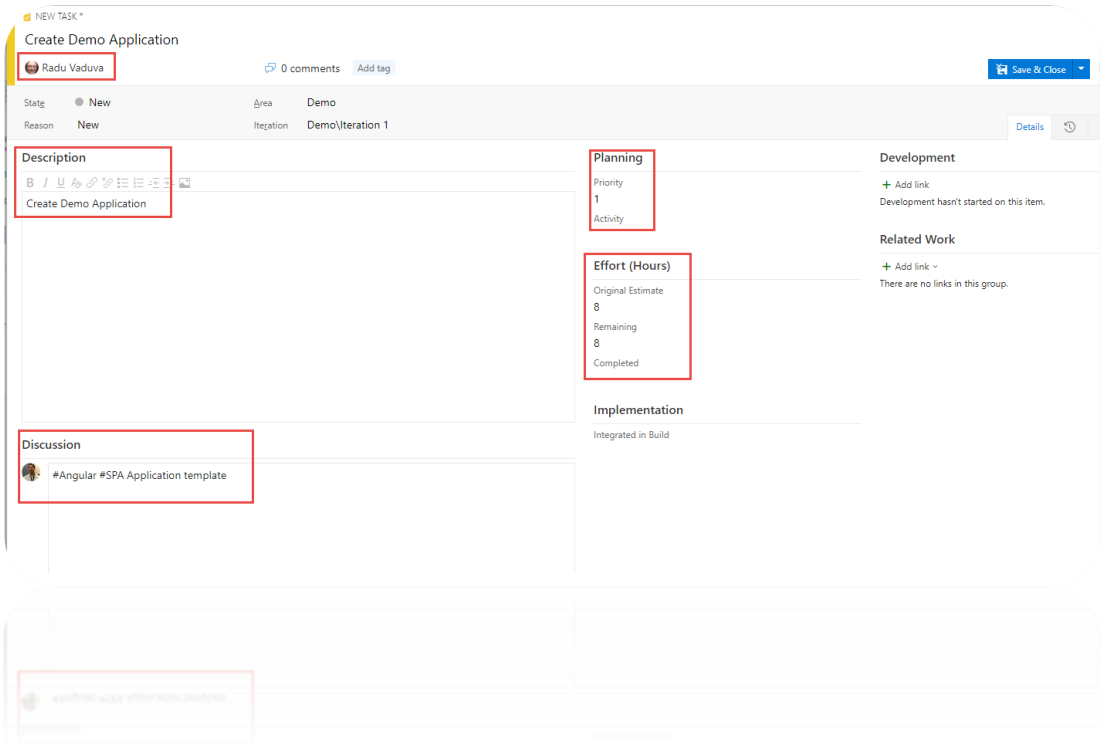
10. Build Complete…

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.



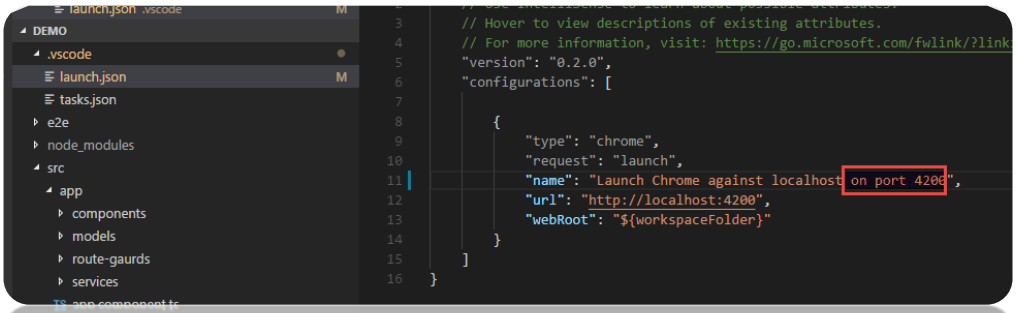
2. Assign a task to a resource (`Yourself` in this case), enter description, set priority, and specify effort. Click Save and Close.



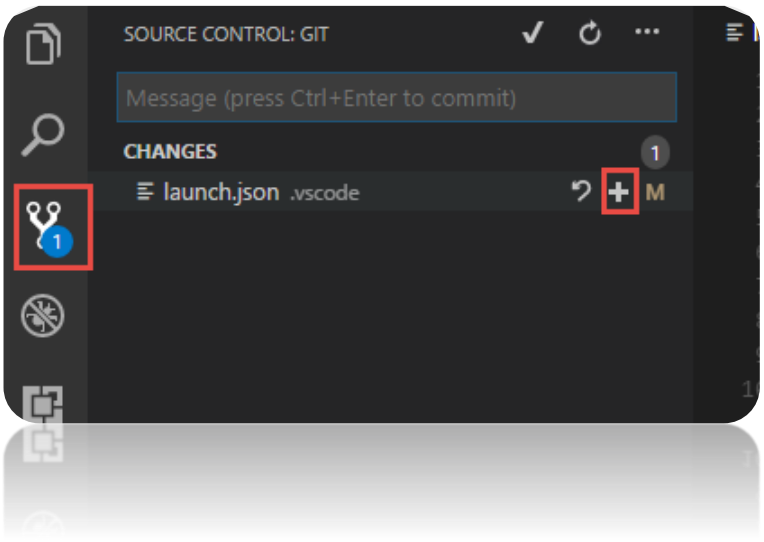
When user saves, unique task number is assigned to each task.

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

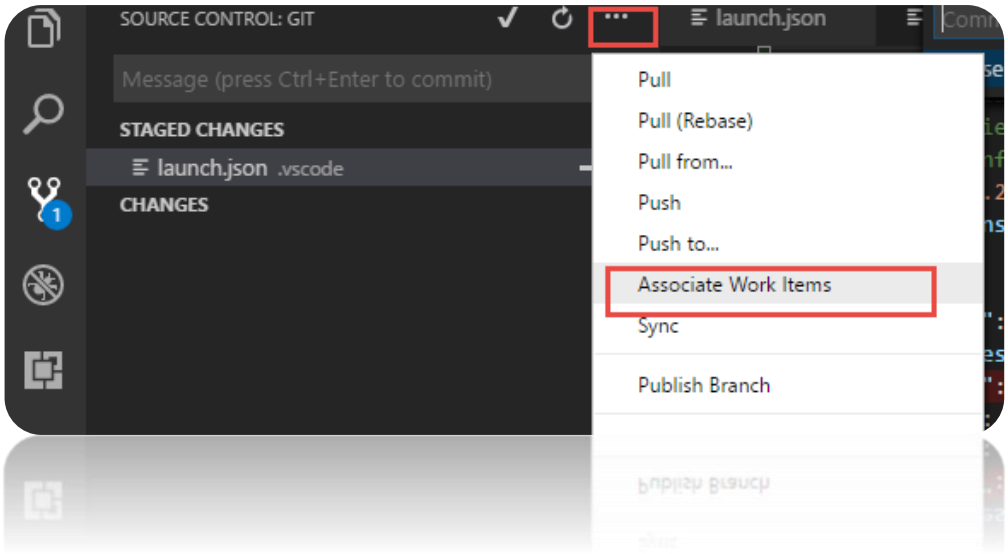
3. Make the code change by appending “on port 4200” as shown below:



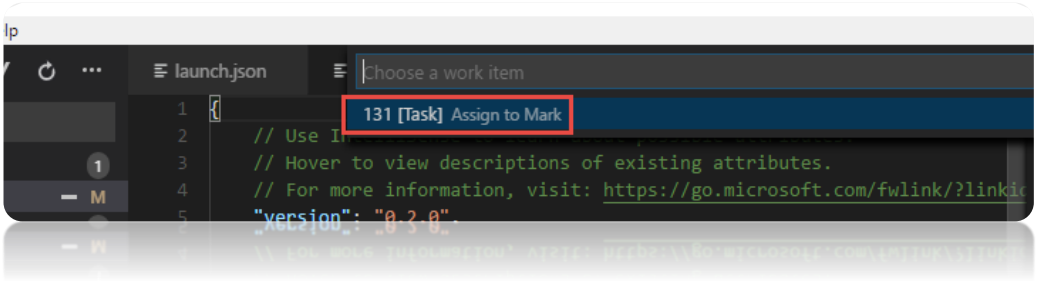
4. Add Change (Stage)



5. Commit Change by Associating work item:



6. Select Work Item task:

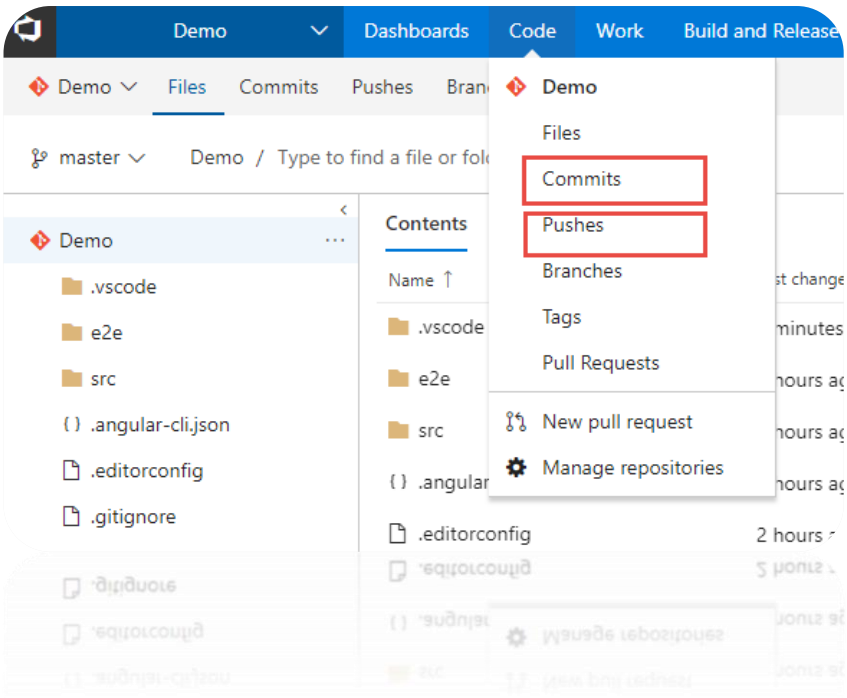


7. Commit Change with a message "Added port "

8. Push Change to VSTS.

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

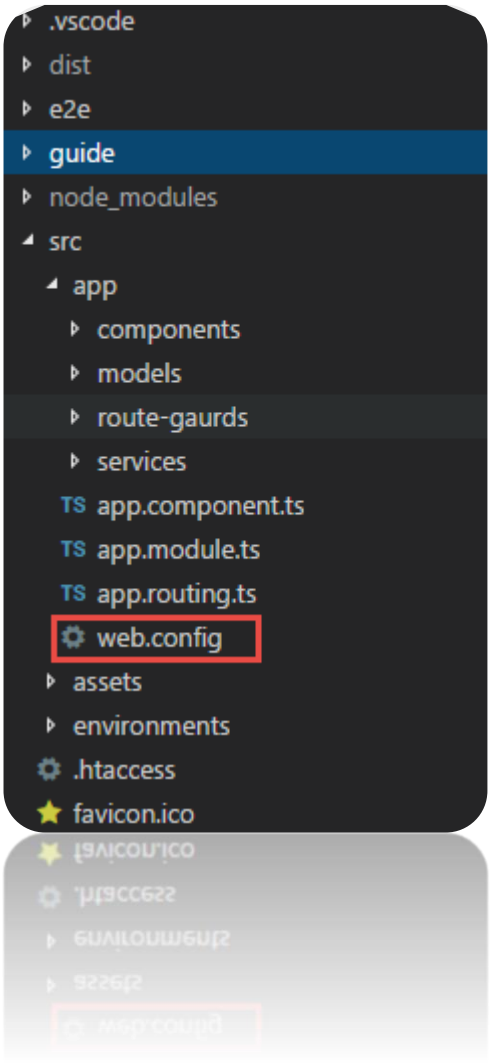
   Check out here:



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>
            <mimeMap 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');
```

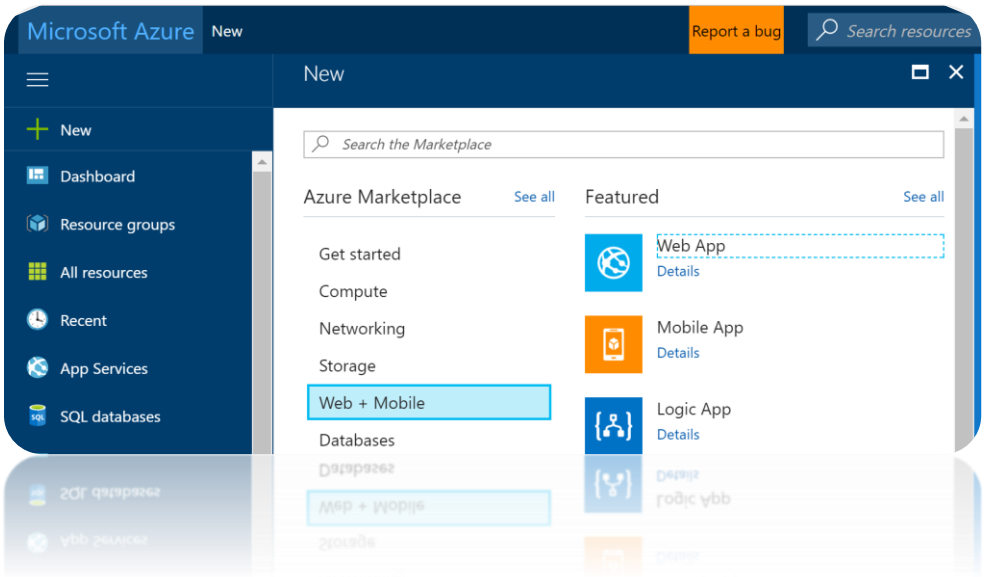
```
var replace = require('gulp-replace');
var htmlmin = require('gulp-htmlmin');

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

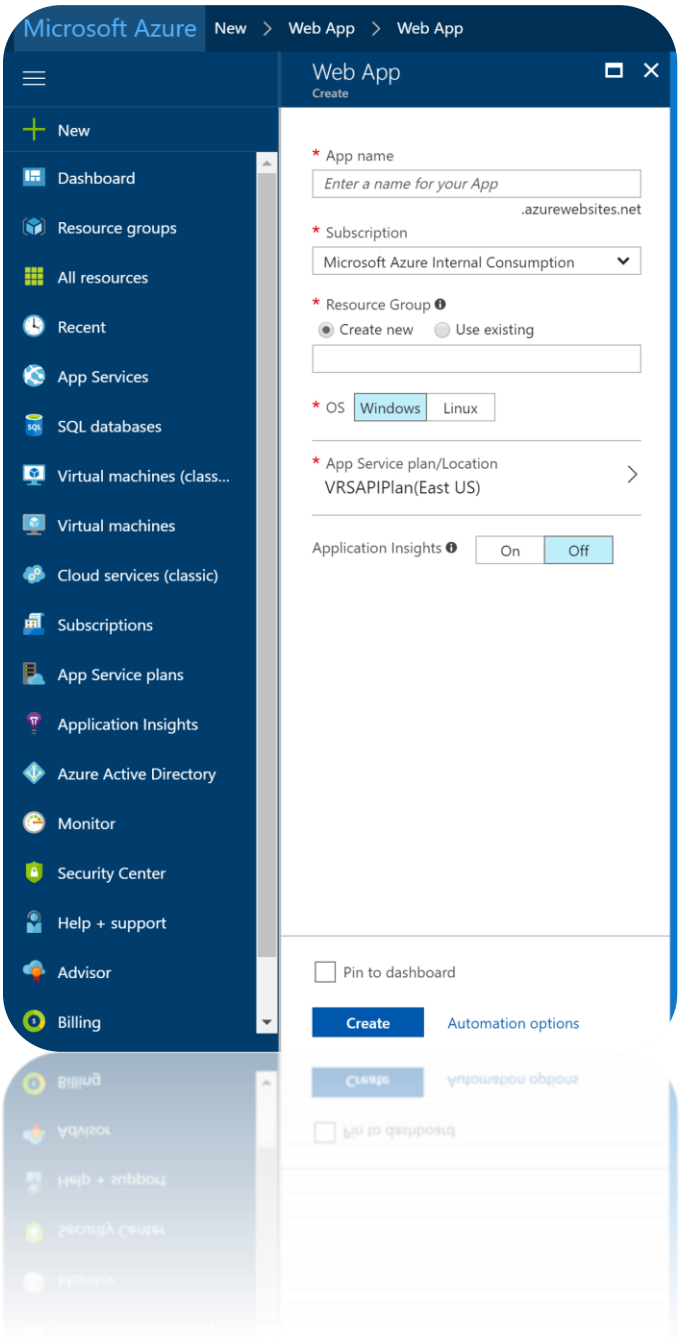
9. Create the Azure App Service

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:



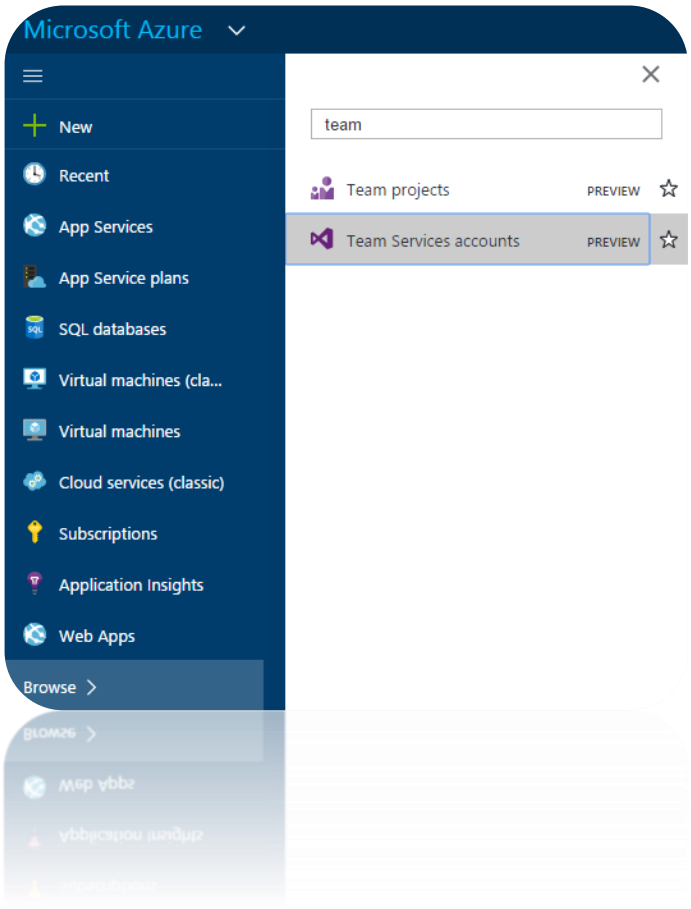
3. Then Click "Create".

10. Linking your VSTS account to your Azure subscription

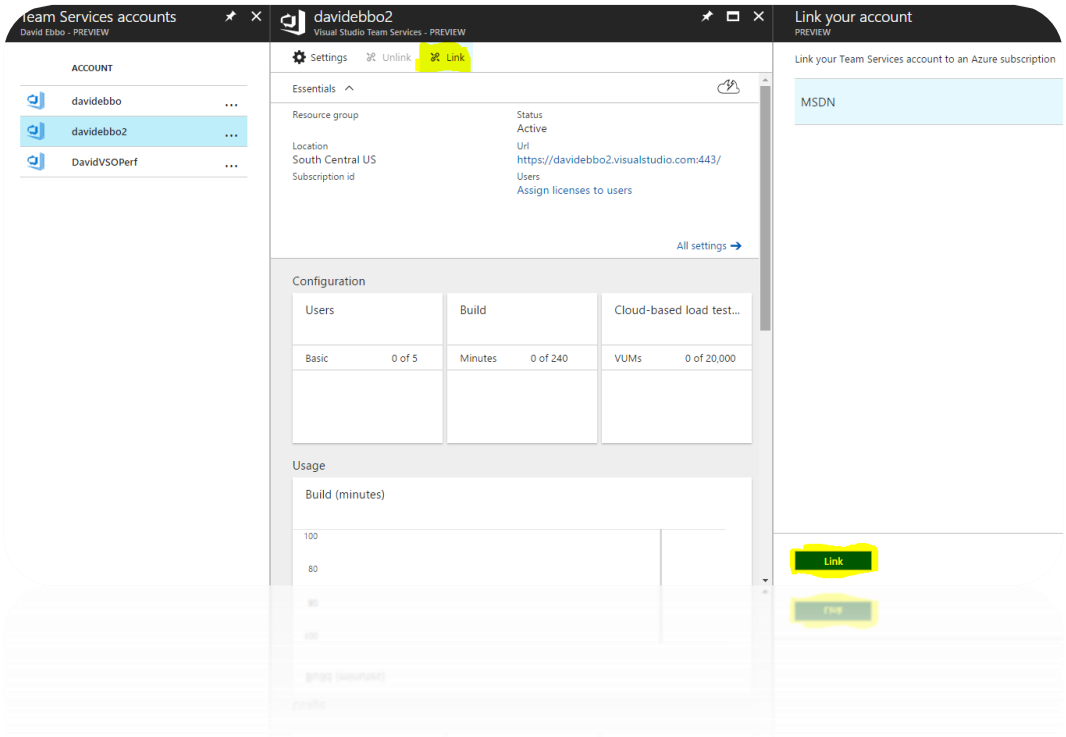
Next, you need to link your VSTS account to your Azure subscription (see also this post on this topic).

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':



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

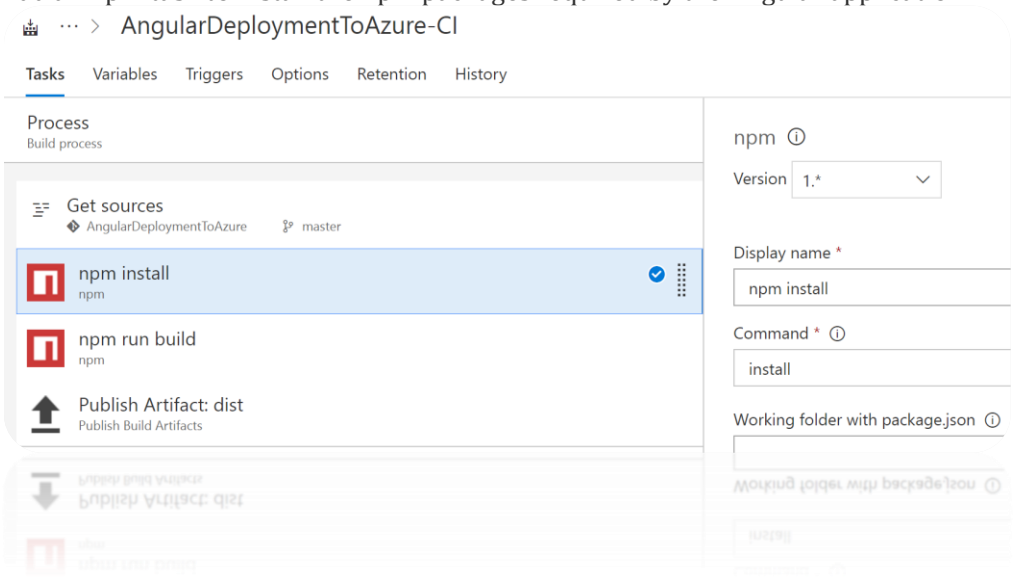


And you're done! You will now be able to set up continuous deploying to your git repos hosted in VSTS.

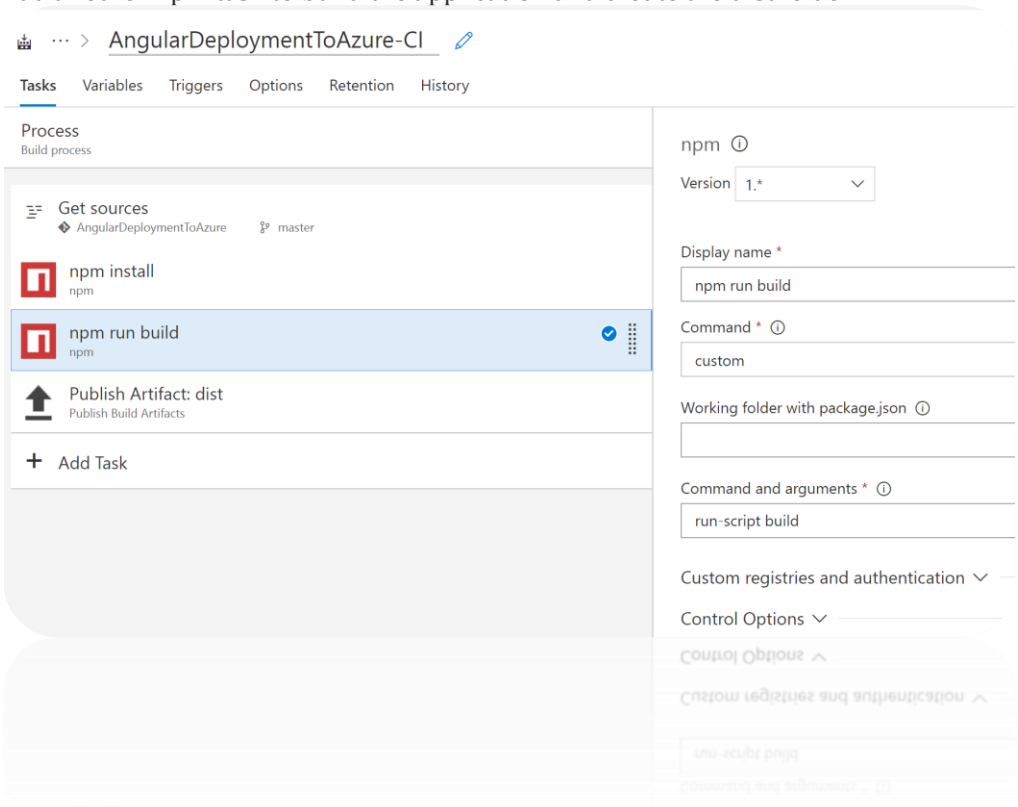
11. Setting Up CI Pipeline With VSTS

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:

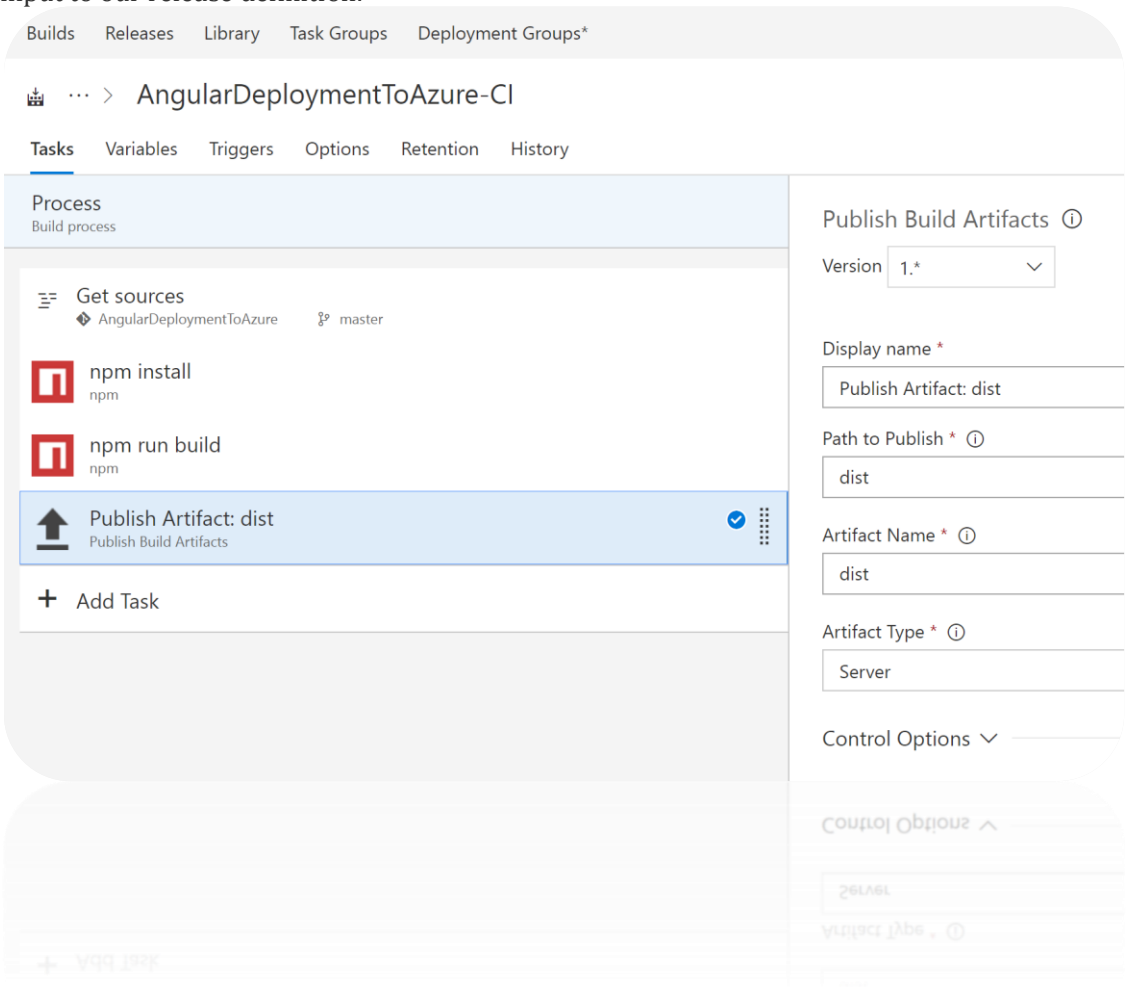
1. *Build and Release -> Builds -> New*
2. Add an npm task to install the npm packages required by the Angular application



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

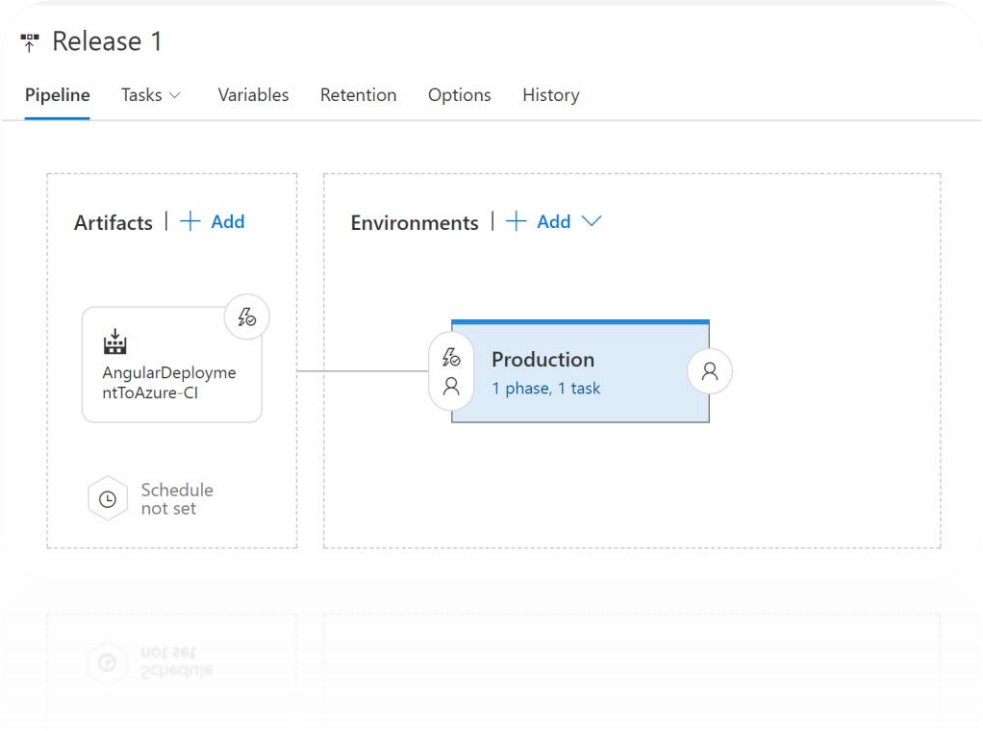


4. Add a publish artifact task that generates the dist artifact which will be provided later on as an input to our release definition:

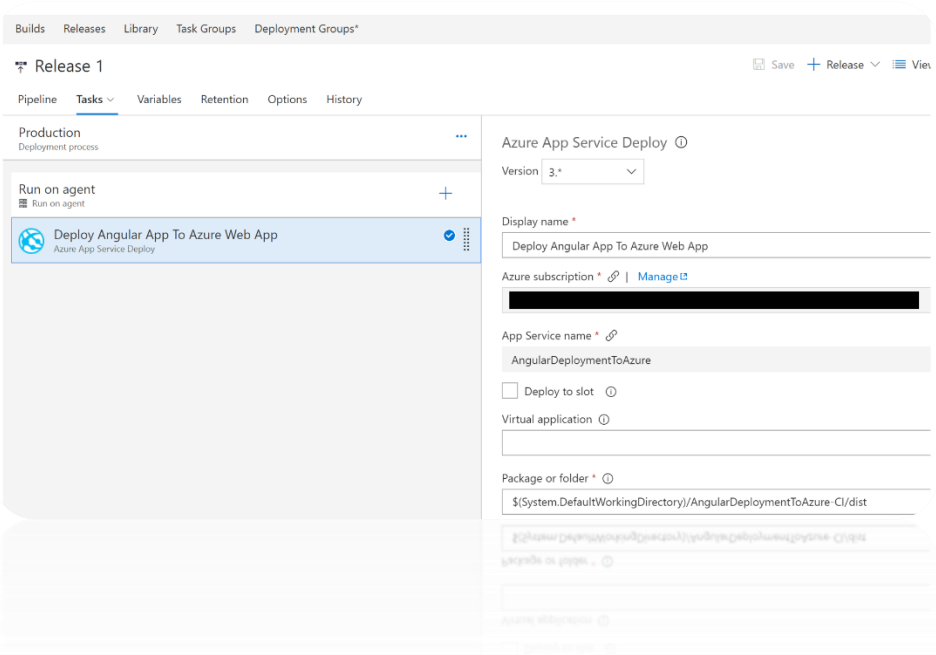


12. Setting Up CD Pipeline With VSTS

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:



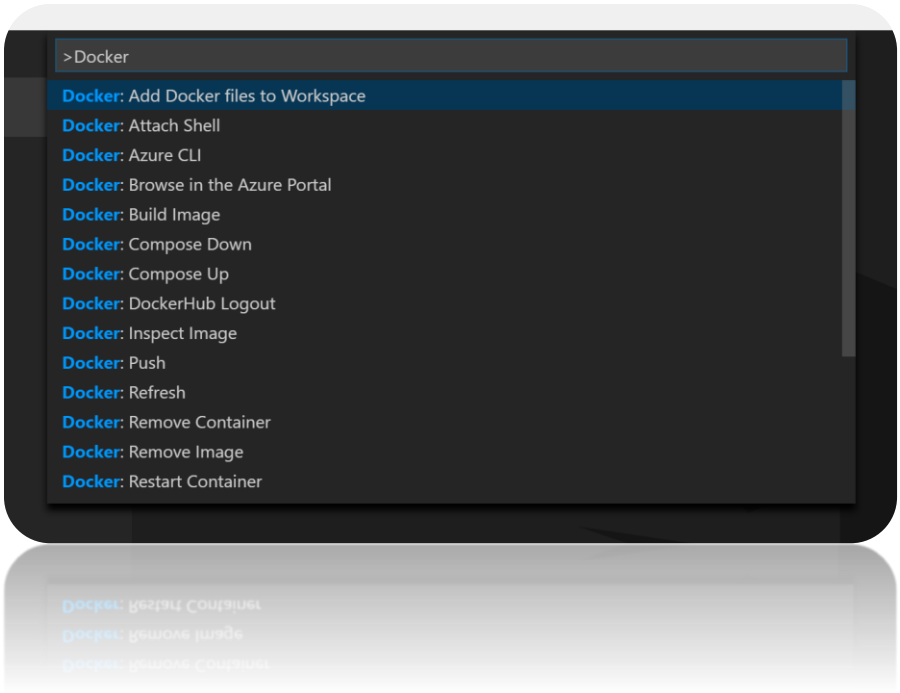
That's it!

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

13. Deploy your SPA to a Linux Docker Image

1. Add Docker Files to workspace like so:

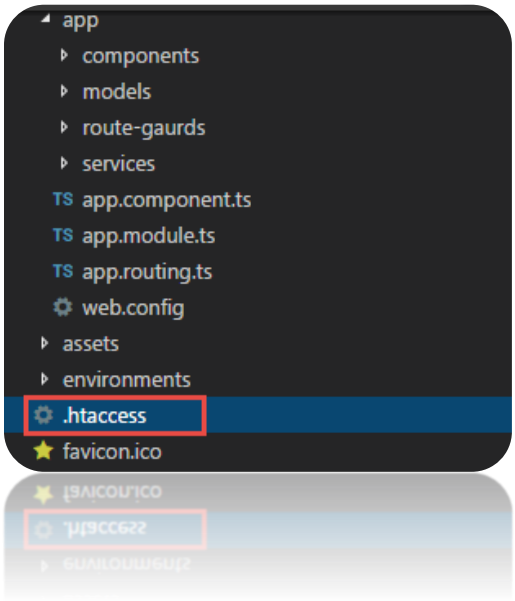


When Prompted Select: node.js and then set the Port to 4200. This is just to create the base implementation of the Docker image files, but we will replace the contents with our own commands for our SPA to work in a simple Apache Web Server image provided by the image library on the Docker public registry.

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.



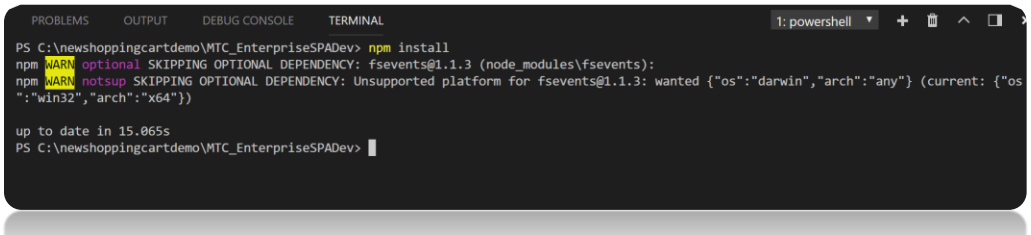
2. Add the entire contents below into the "**.htaccess**" file:

```
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
```

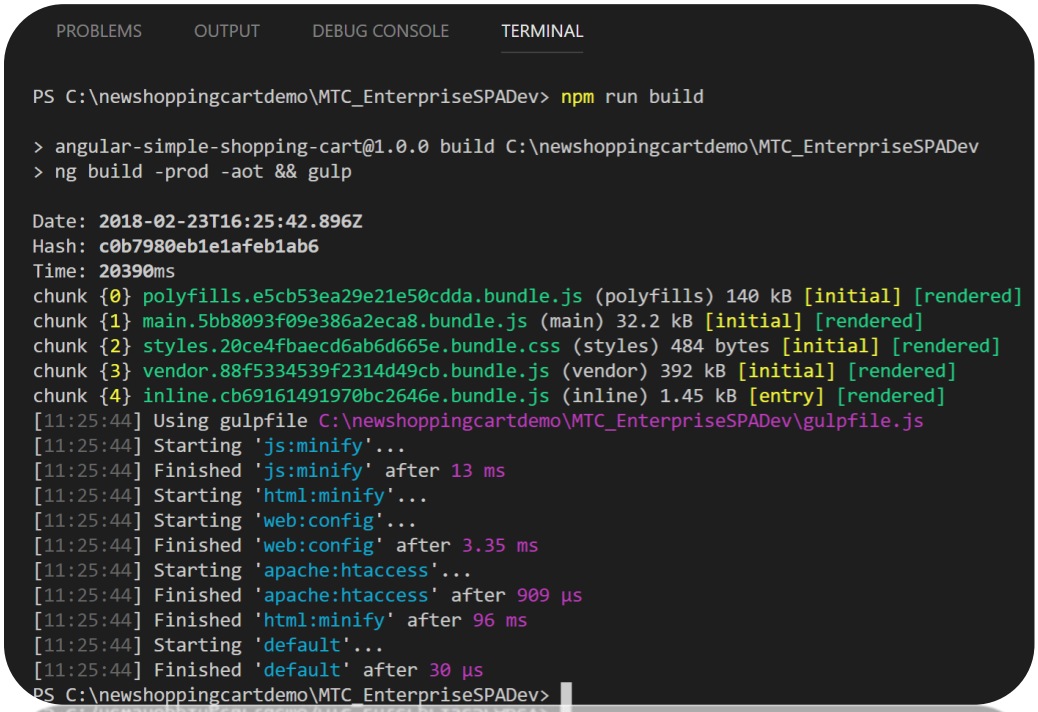
3. Modify the **/gulpfile.js** once again as follows to include the .htaccess file.

```
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"]);
```

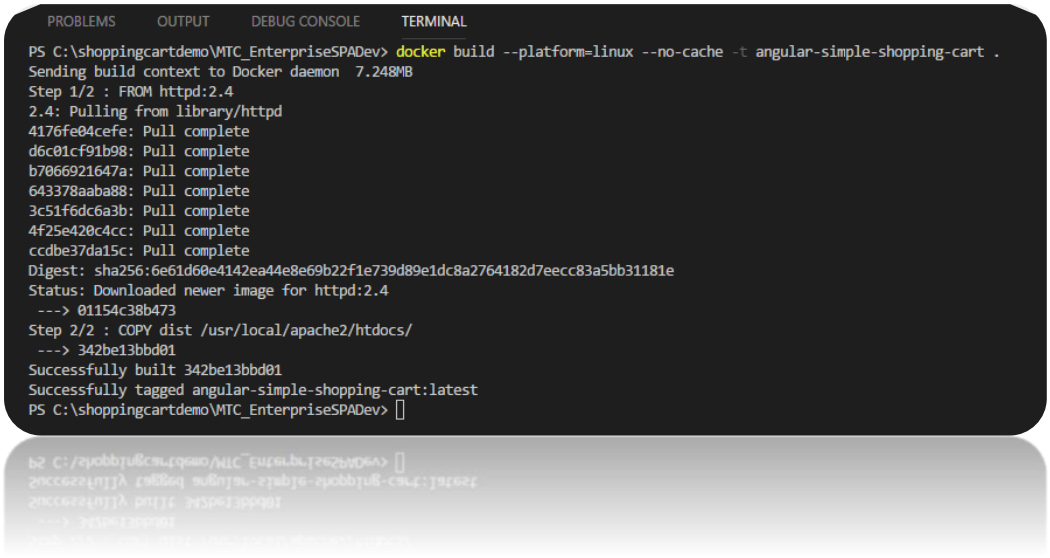
4. In the terminal Run: “npm install”



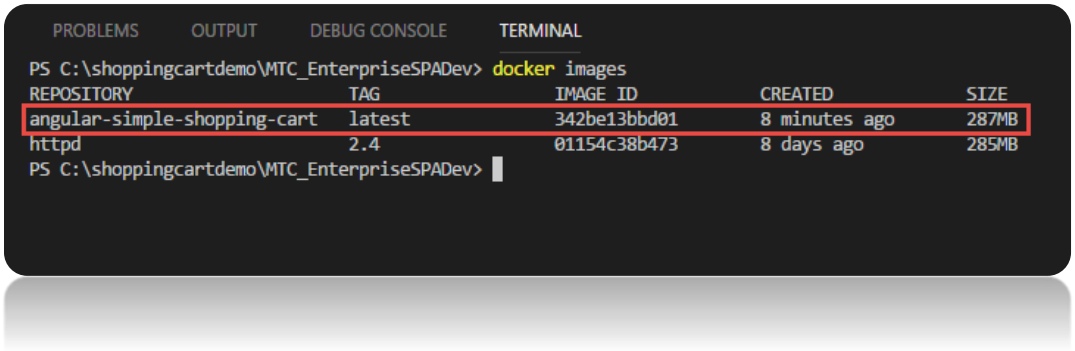
5. In the terminal Run: “npm run build”



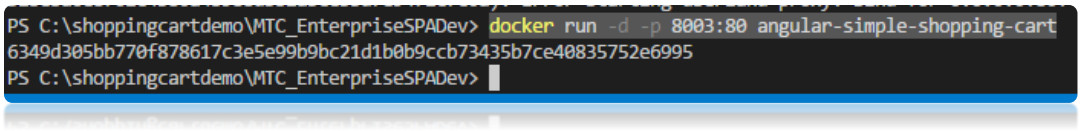
6. In the terminal Run: “docker build --platform=linux --no-cache -t angular-simple-shopping-cart .”



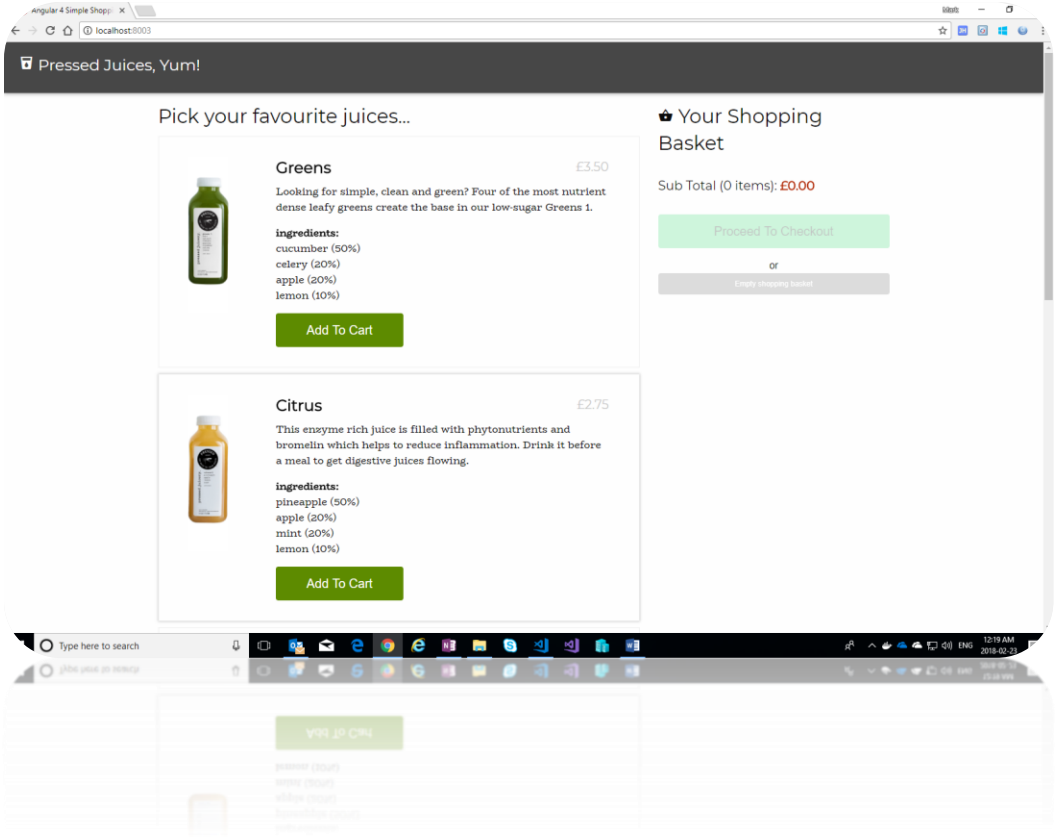
7. In the Terminal Run: “docker images”. You will see two images. The HTTPD which is your base image that was downloaded from the Docker registry and the angular-simple-shopping-cart image you just built.



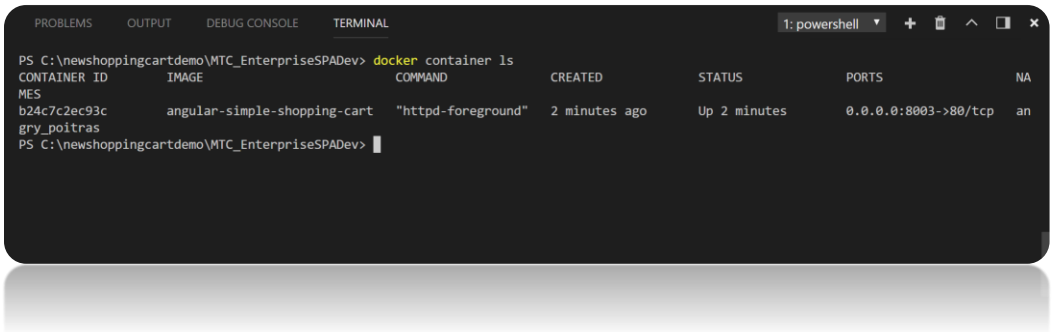
8. In the Terminal Run: “docker run -d -p 8003:80 angular-simple-shopping-cart”. Where 8003 is the port that will be used outside of the container and 80 is the port being exposed inside the container.



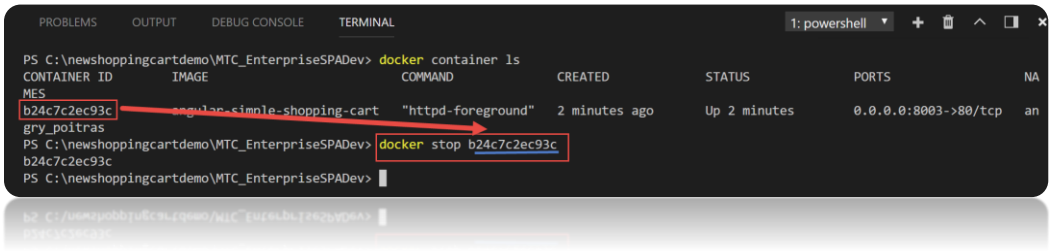
9. Open a web browser and navigate to: “localhost:8003”. If successful, you should see our Angular SPA running locally as such:



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:



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:



12. Dockerizing your Angular application is now complete!

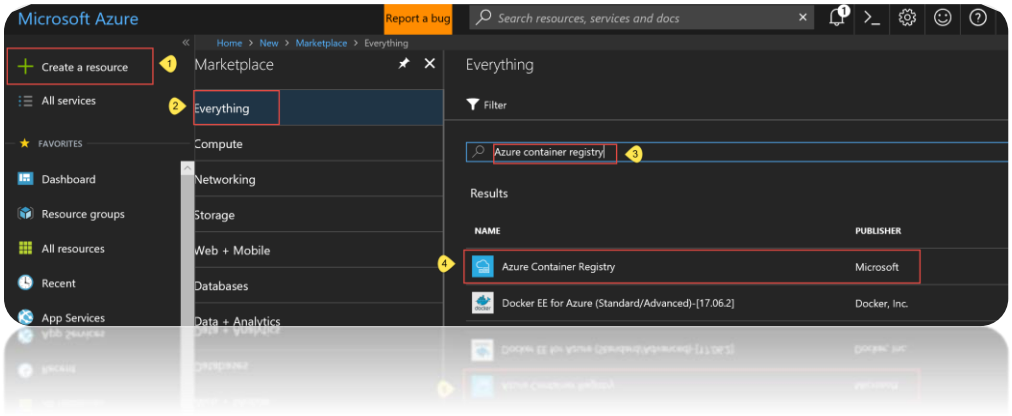
Deploy your Docker image to Azure PAAS Services using CI/CD

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.

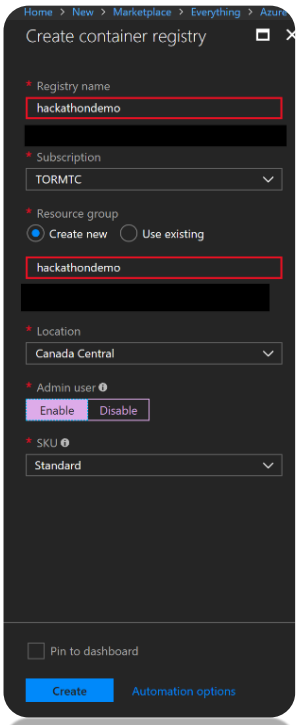
    b. Configure VSTS Build and deployment tasks to Build and Update the Azure Container Registry with the newly built image, mentioned in the previous section, and then deploy the image from the Registry to the Azure App Service(Linux) Deployment slot using the CI / CD integration features within VSTS.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

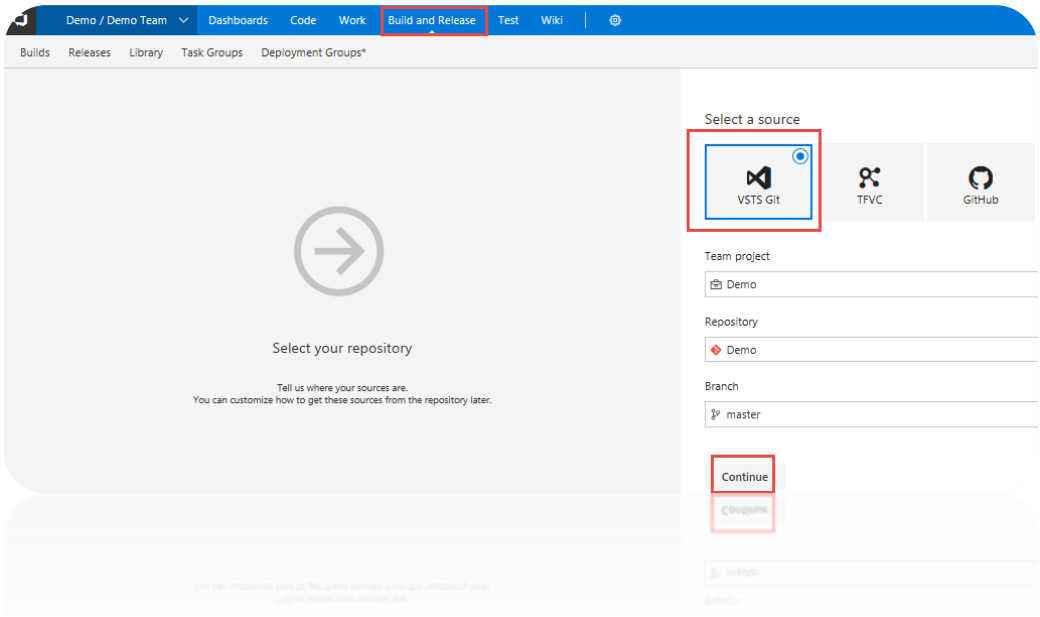


Note: The registry name is also the username to be used for authenticating against the registry for access:

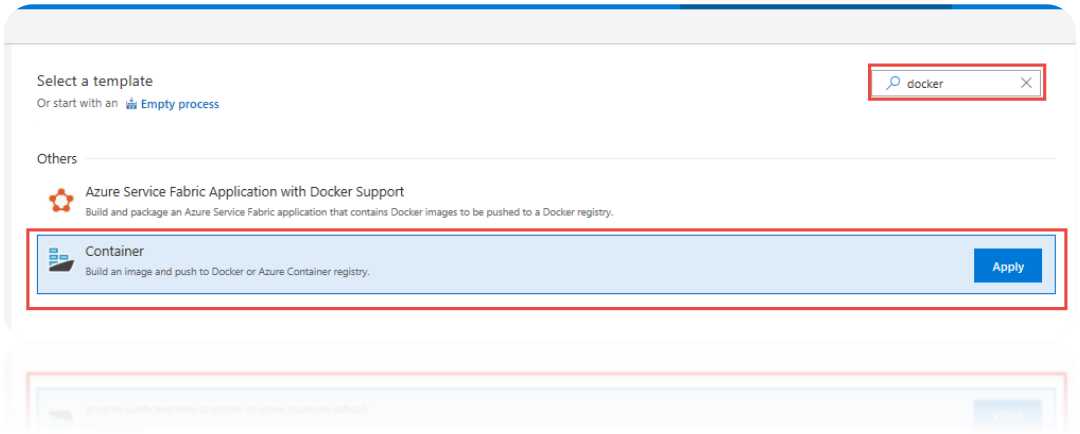
3. Provide the following settings for the Azure Container Registry as such:



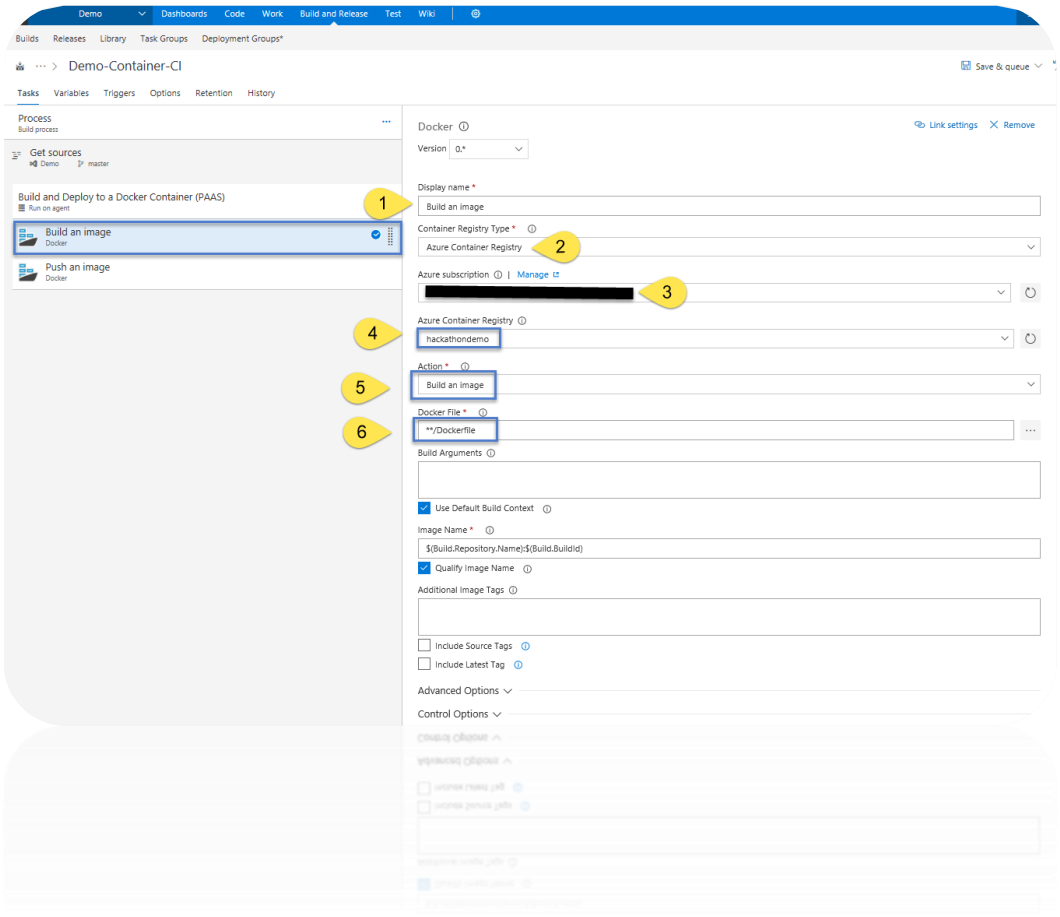
4. Configure a new Build and Deployment Definition for CI / CD. Use your Demo VSTS instance you have created earlier within this document:



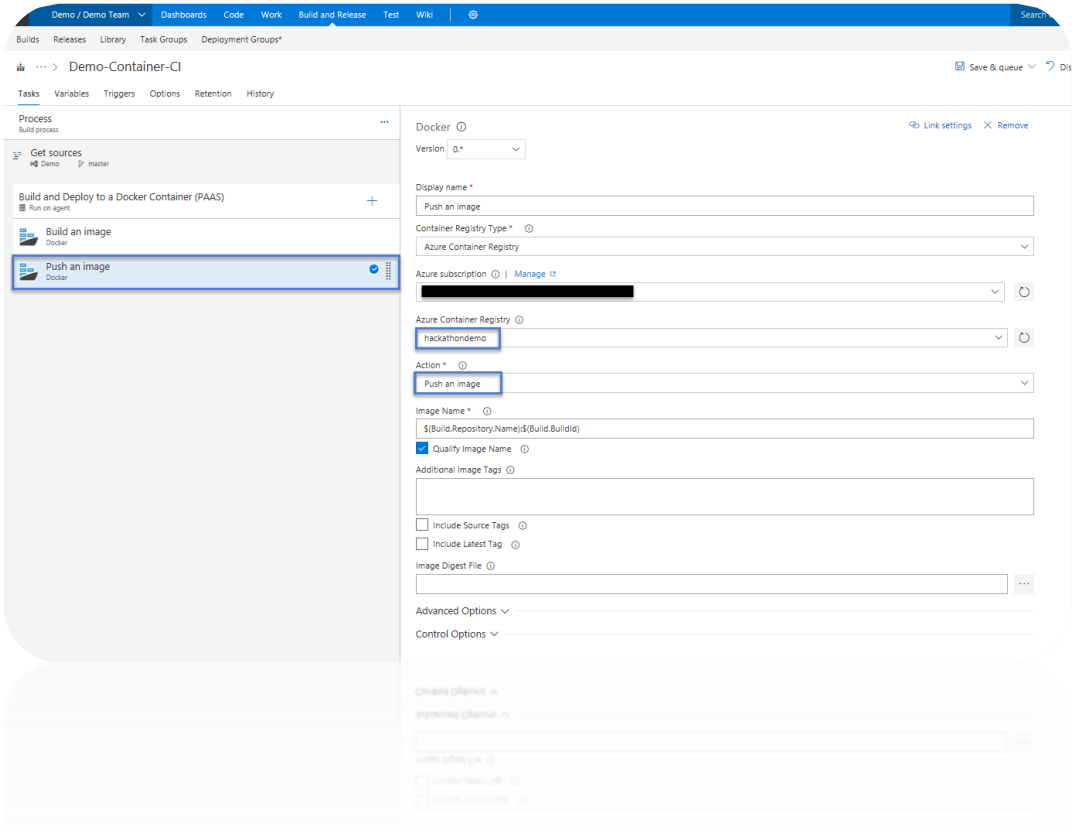
5. Select the Container template:



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

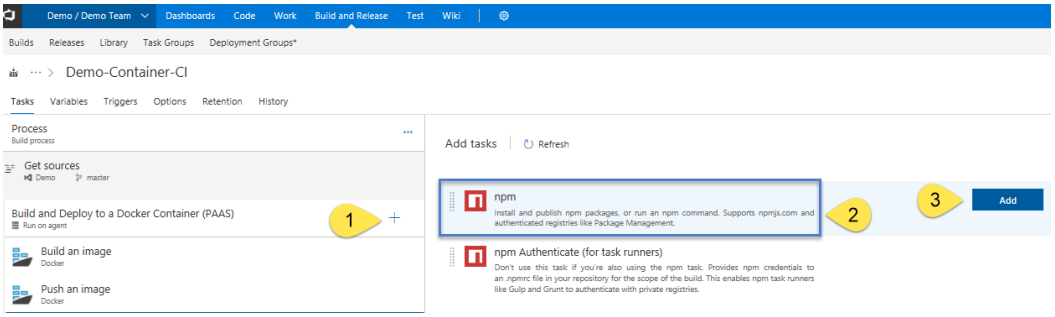


7. Modify the “Push an image” step as such:

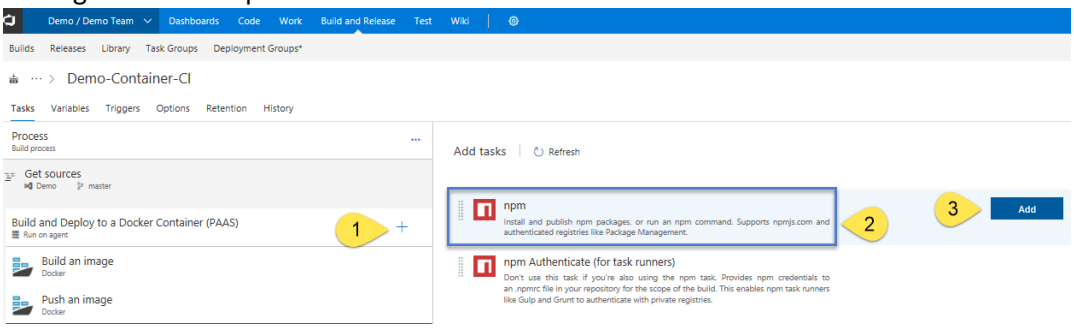


8. Add Node Package manager “Install” & “Build” steps to build the Angular Application:

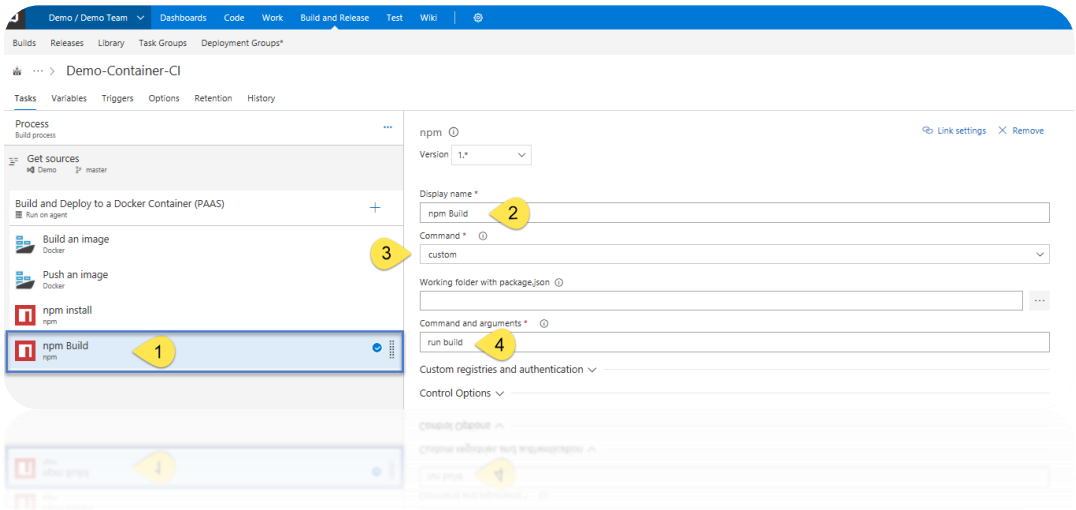
Adding the install step…



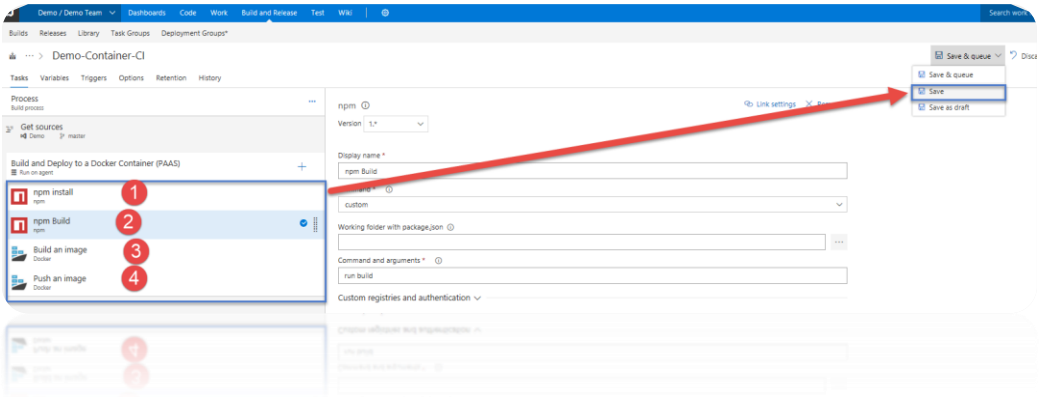
Adding the Build Step…



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:



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



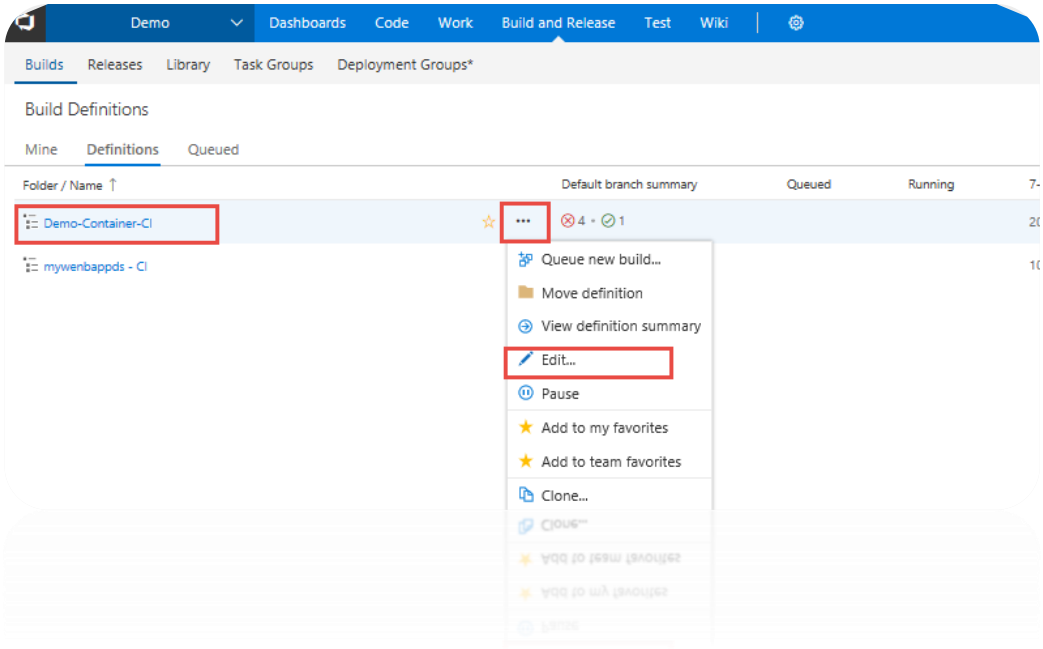
10. Finally, we need to specify that the build will happen on hosted VM managed by Microsoft, but we must make sure it is a Linux VM, not a Windows VM which is the default. The reasoning for this, is we need to ensure Docker can Build the image we specified within the “DockerFile”. We are targeting Apache 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.

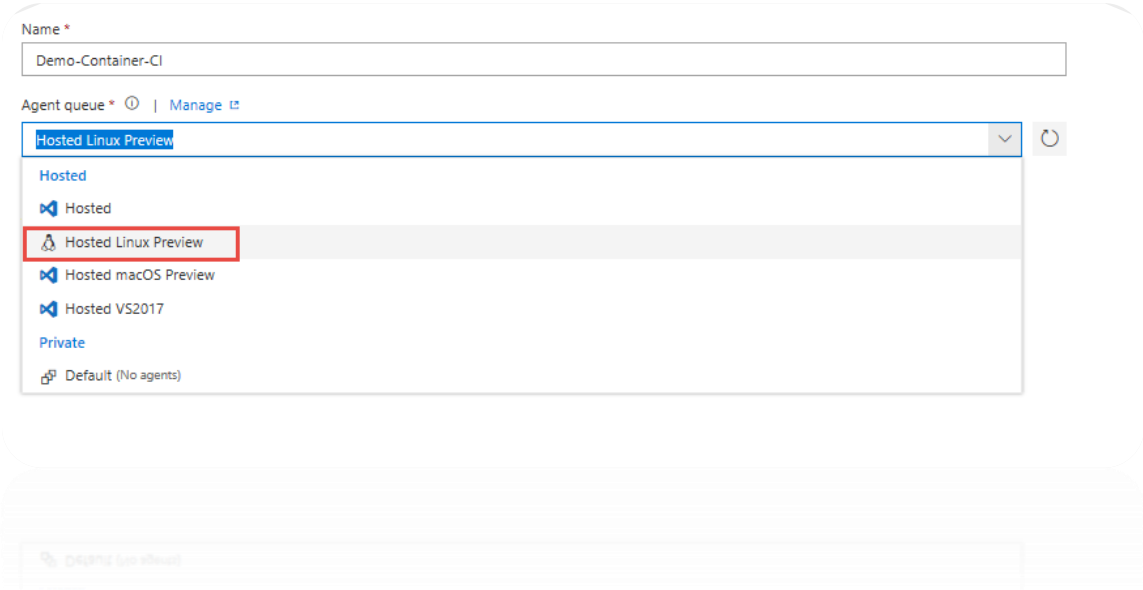
    The line associated with defining the base image within the “DockerFile” is as follows:

```
FROM httpd:2.4
```

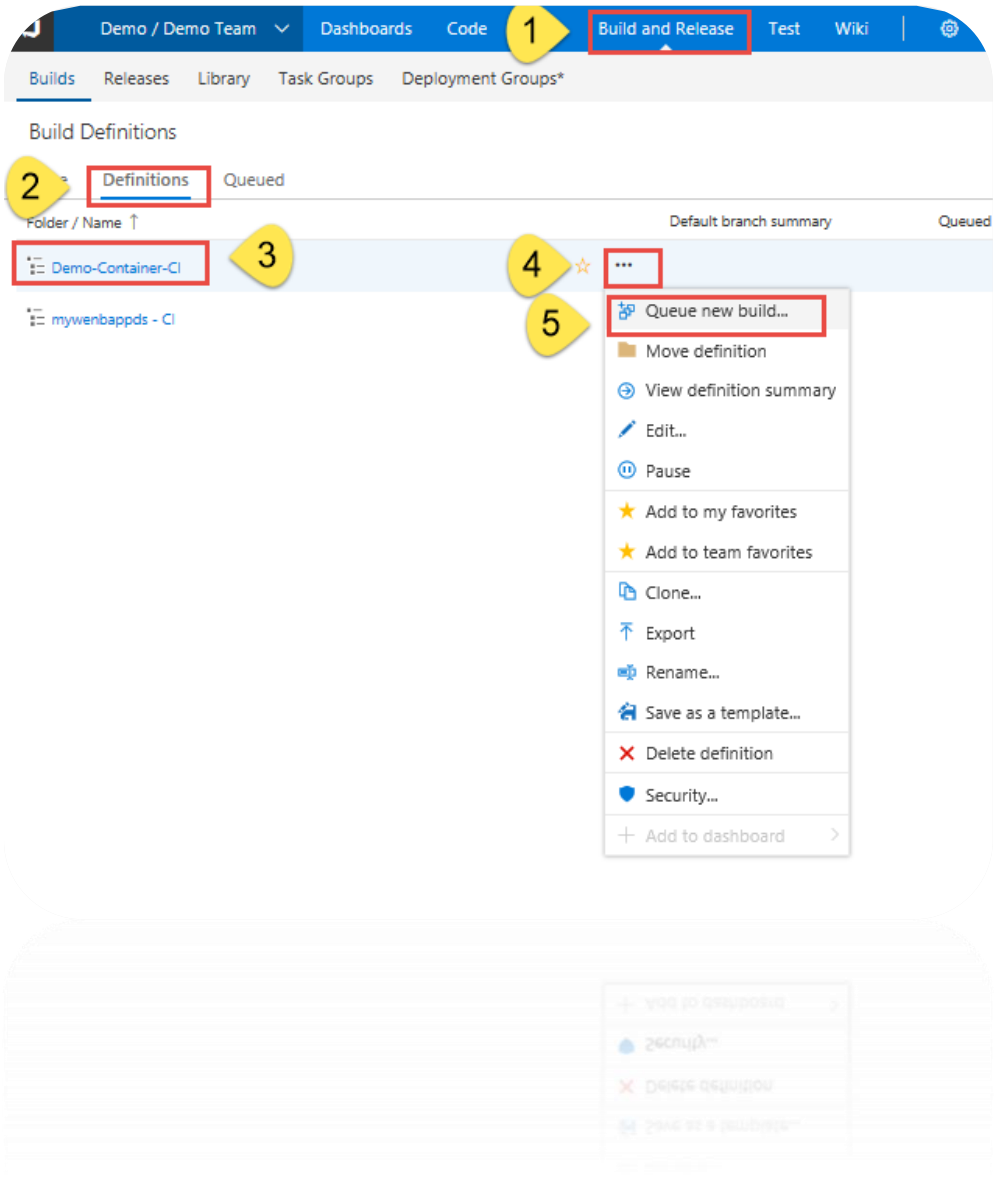
    Read more on this image here: https://hub.docker.com/_/httpd/

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

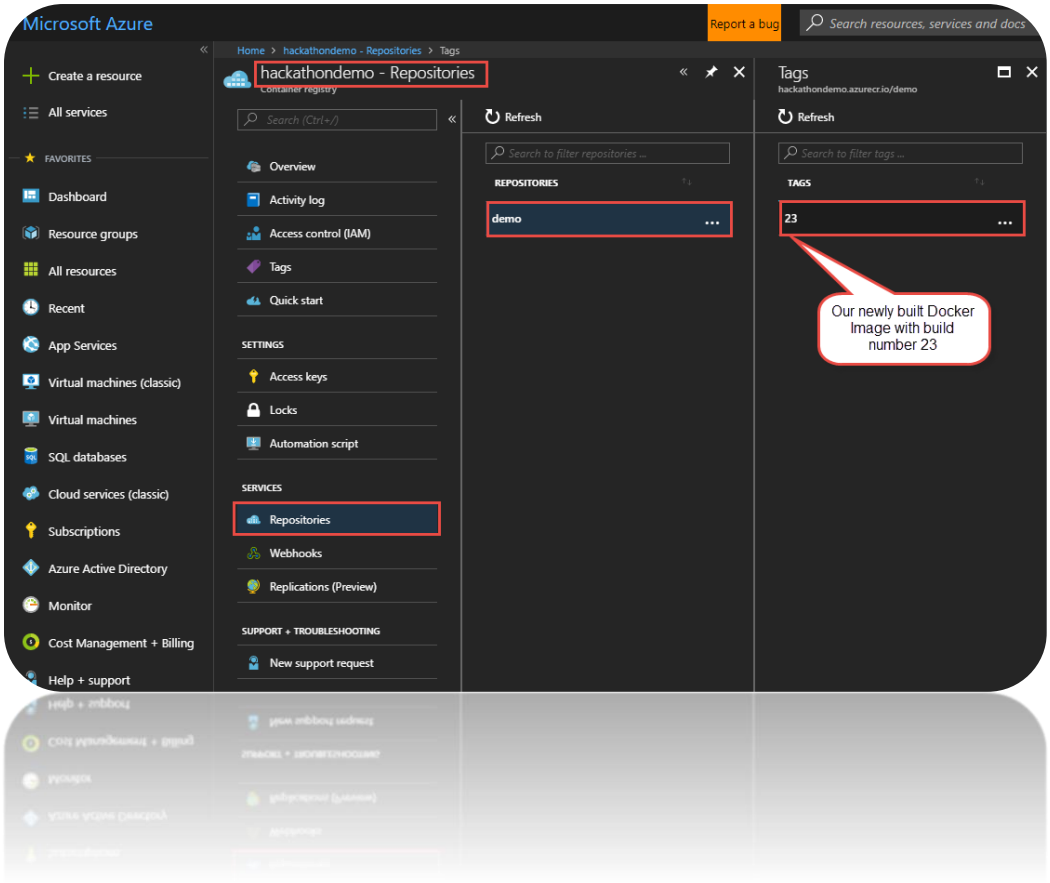




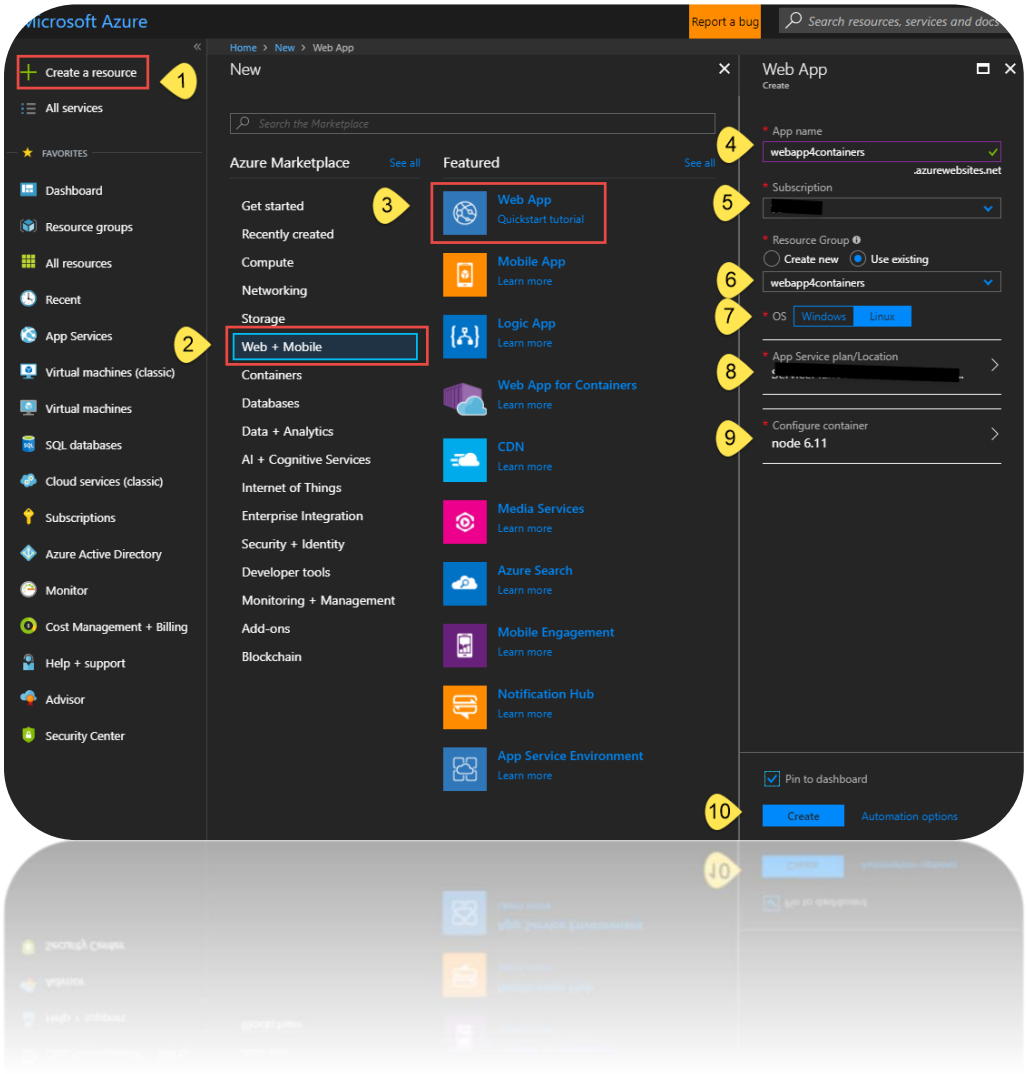
12. Queue a new build as such:



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.

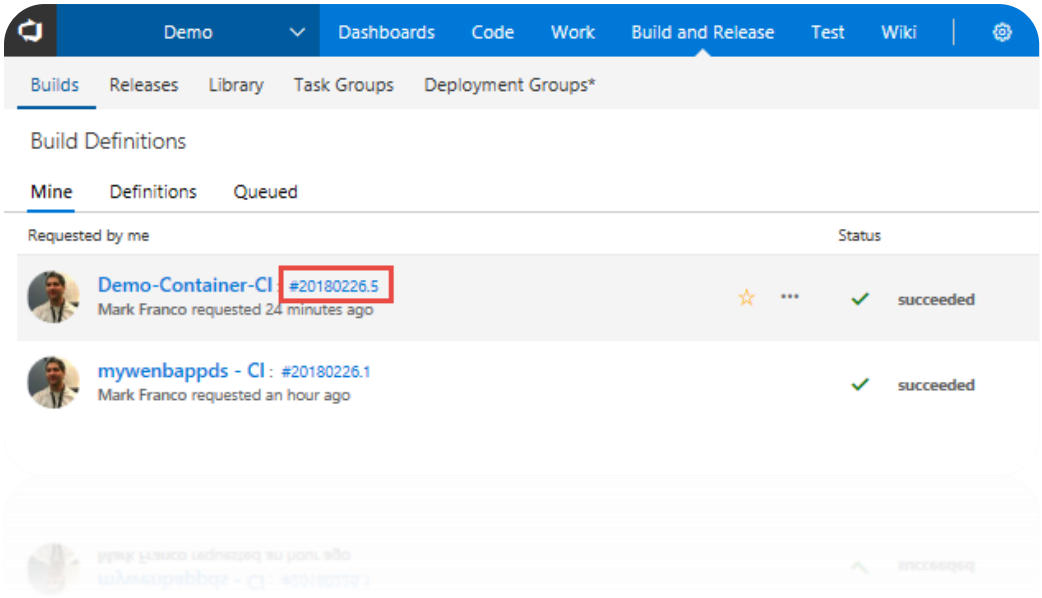


14. Now that the image is stored inside the ACR we will need to set up continuous deployment of our Docker-enabled app to an Azure web app(linux). Start by creating an Azure web app to host the container. This can be achieved in Azure by creating a "Web App for Containers" as shown below. Notice that 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.

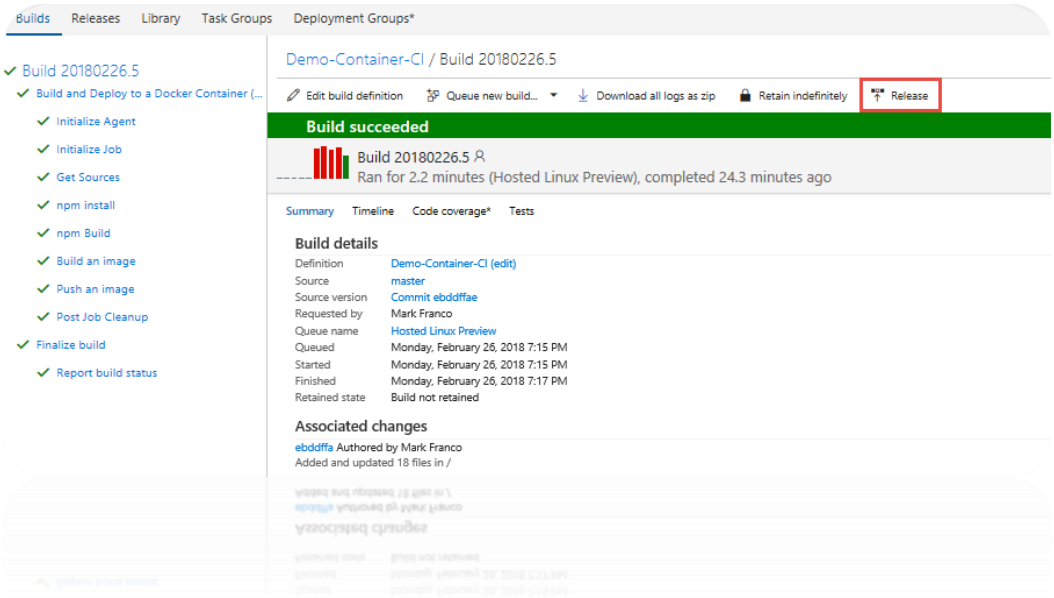


15. Next, we create a release definition on VSTS that will deploy to the Azure App Service we created above. Follow these steps to create a release definition:

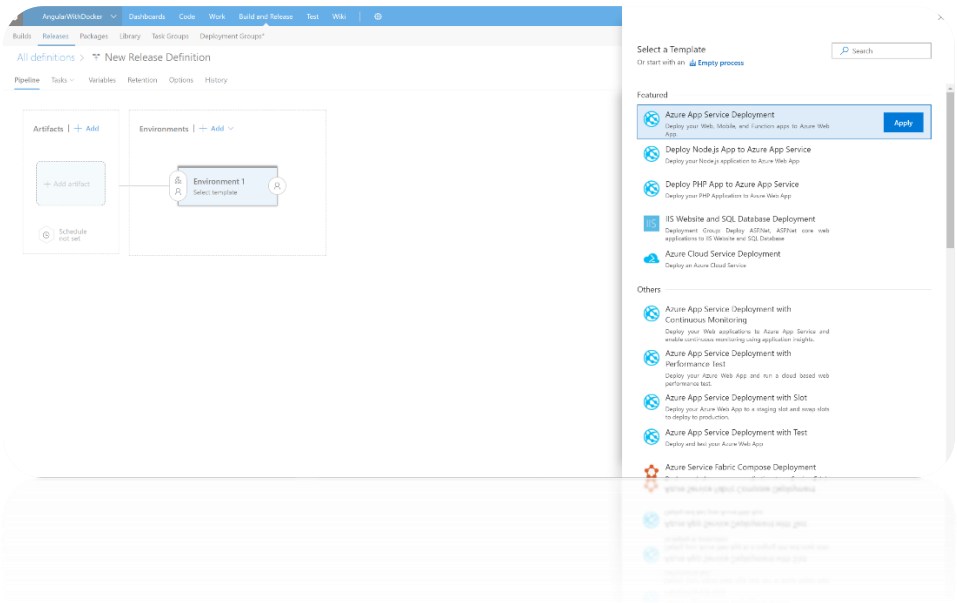
    In the **Build & Release** hub, open the build summary for your build.



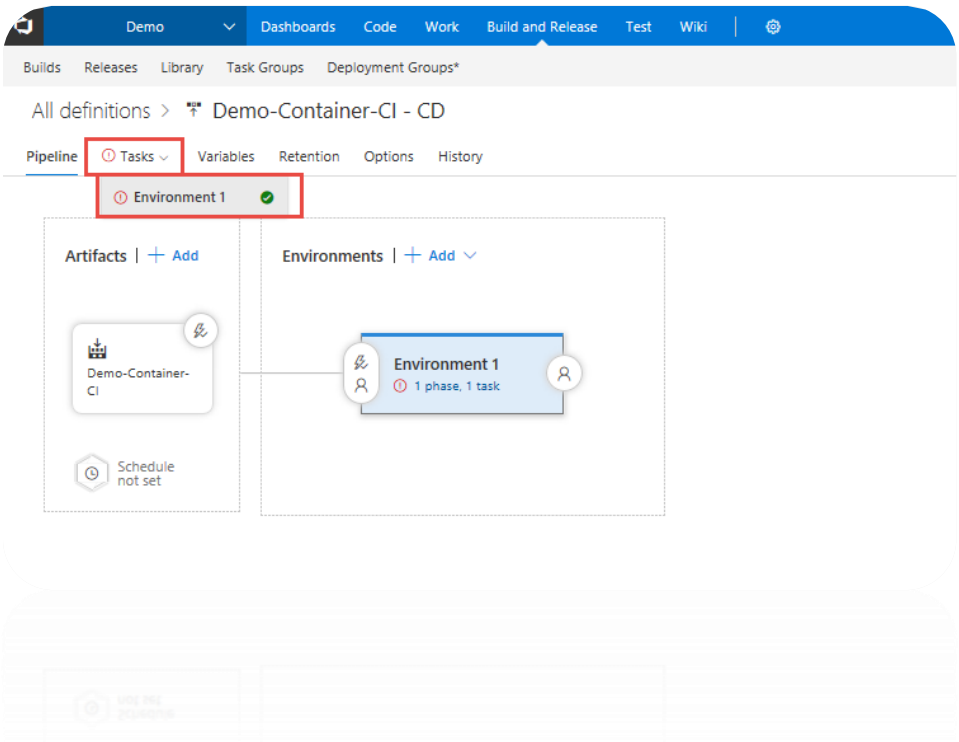
16. In the build summary page, choose the **Release** icon to start a new release definition.



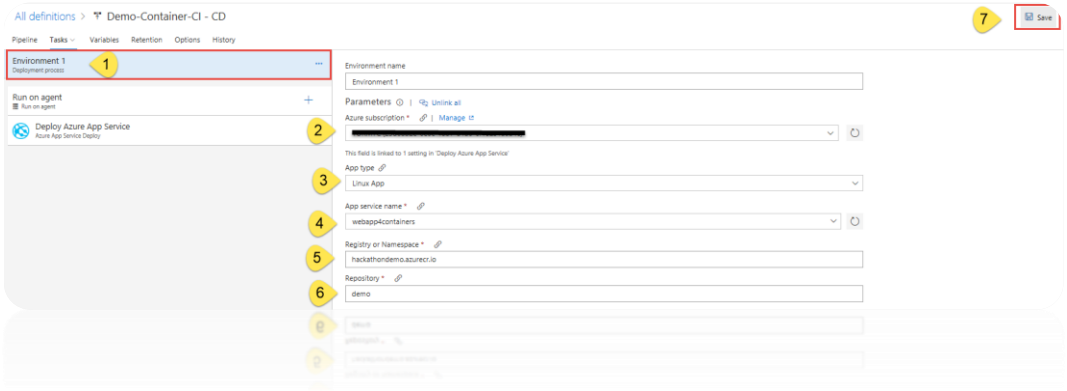
17. Select the **Azure App Service Deployment** task and choose **Apply**.



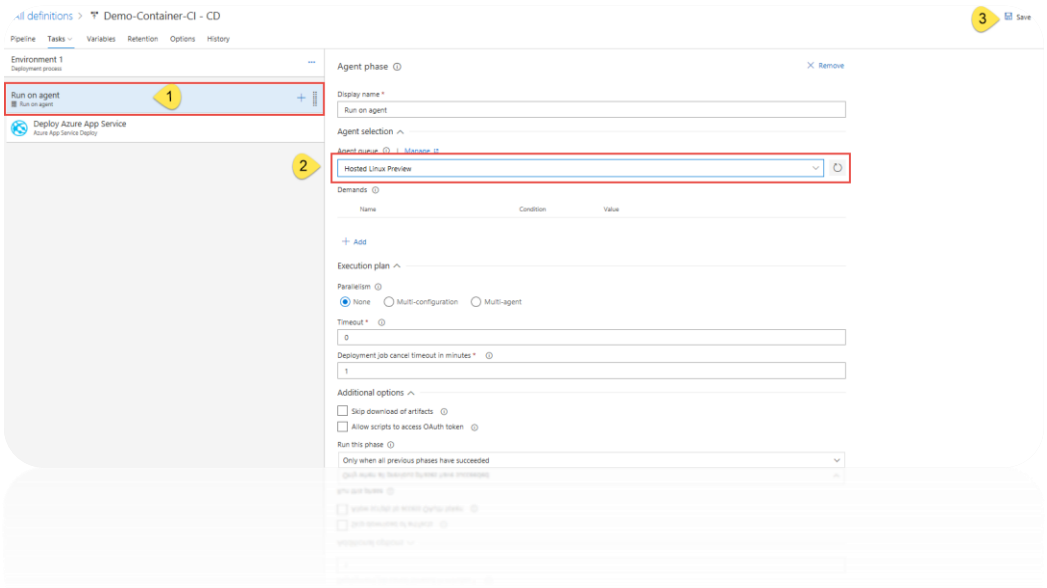
18. Select **Tasks** then **Environment 1**



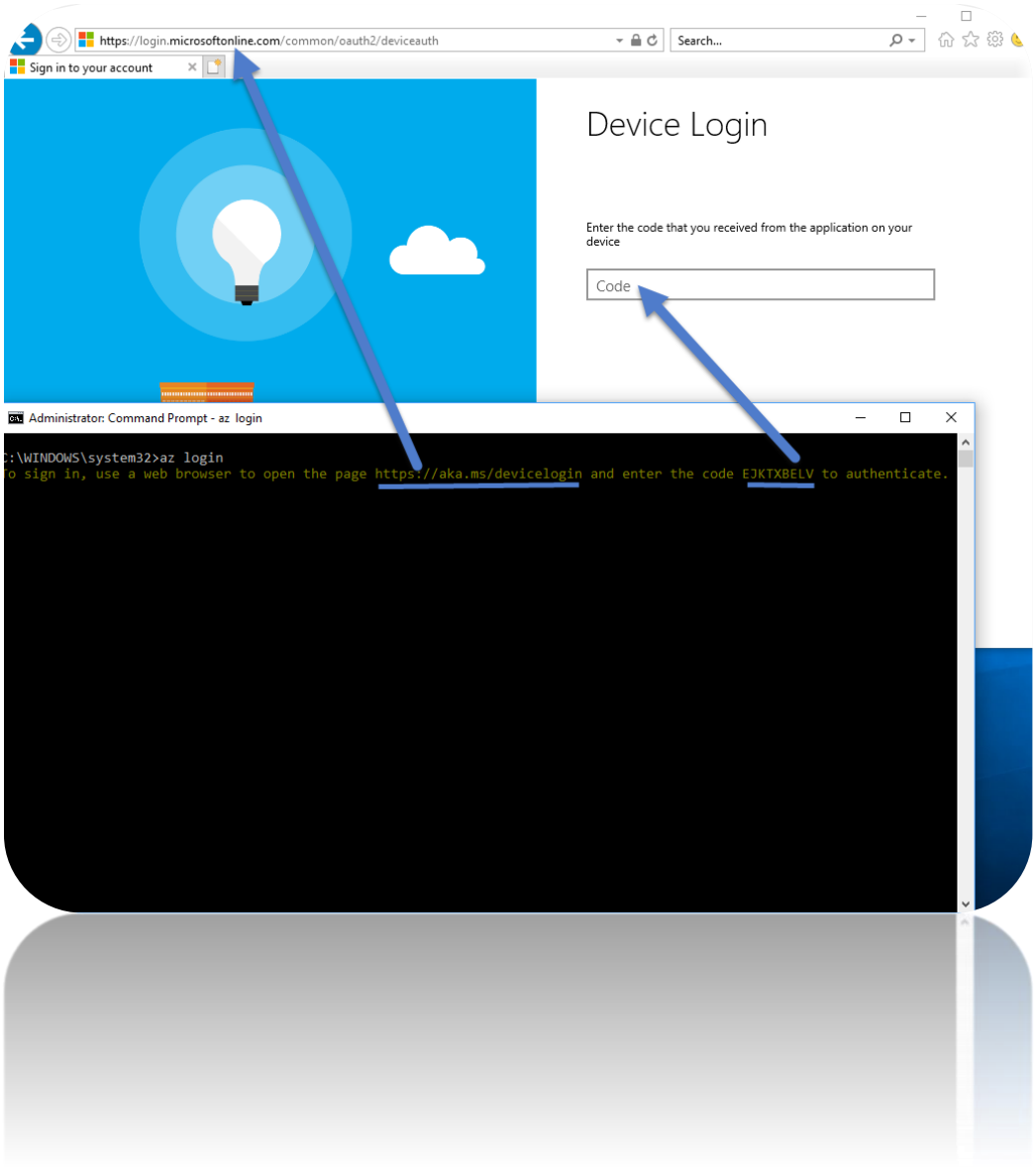
19. Configure the properties as follows:



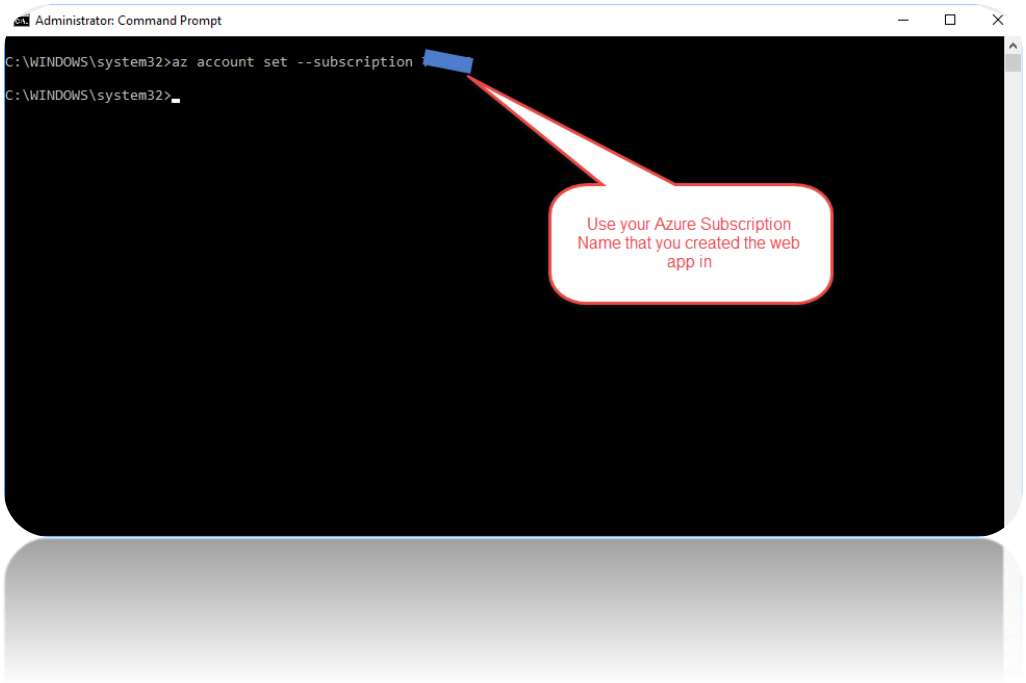
20. Set the “Run on Agent” so that the “Agent Queue” is set to “Hosted Linux Preview” as such:



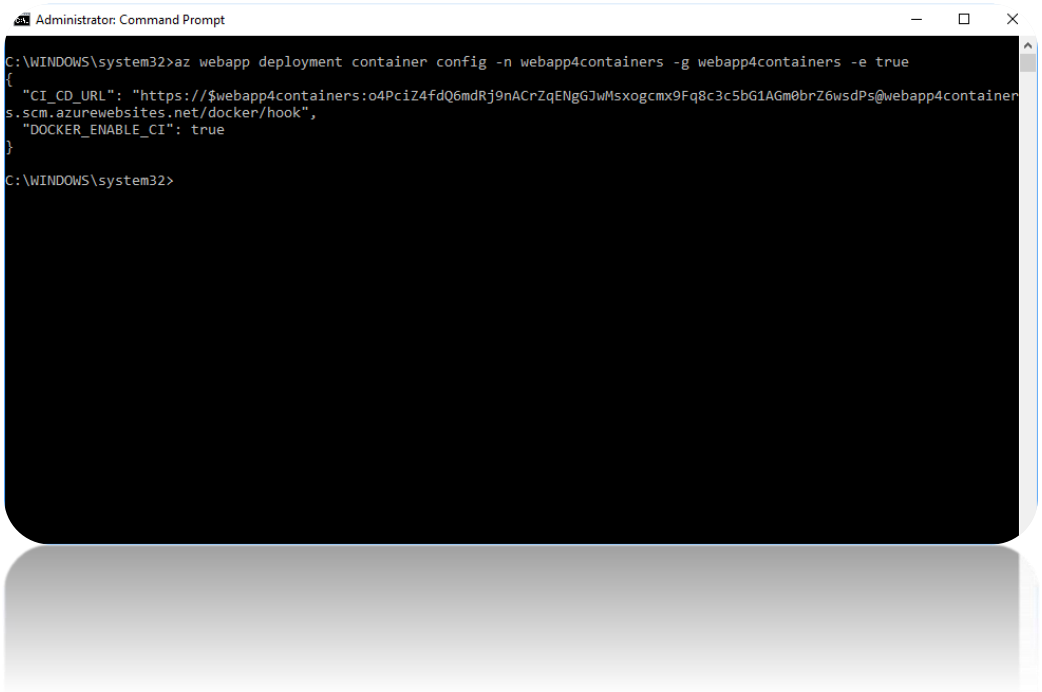
21. Next, Open a windows CMD prompt in Administrator mode and execute the below commands:
22. "az login". And follow the login process which will give you a device code that needs to be copied and pasted into the browser as such:



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

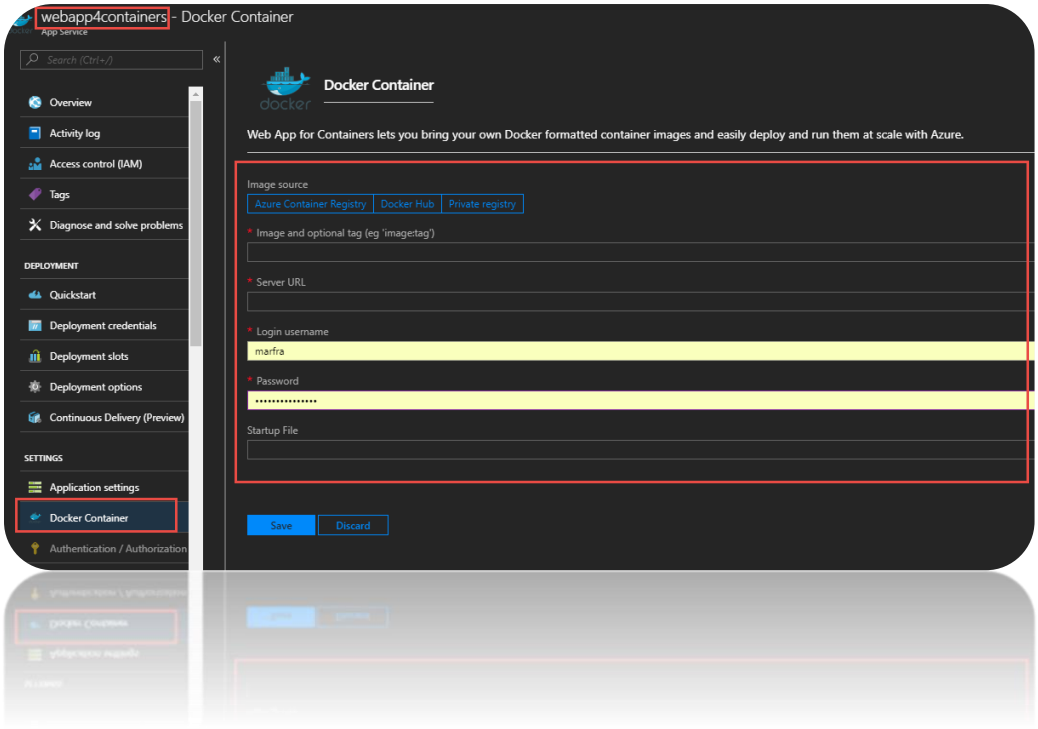


24. “az webapp deployment container config -n webapp4containers -g webapp4containers -e true”.

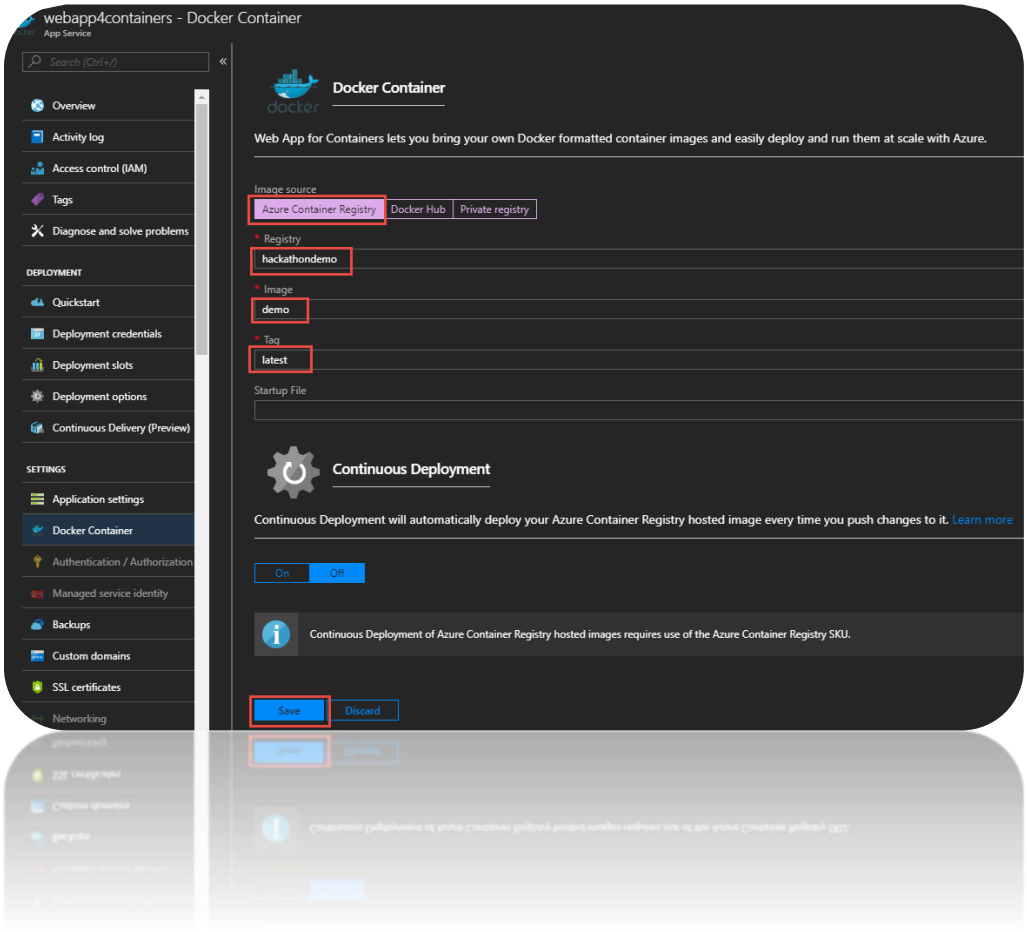


Note: the above command will open the Web App to allow for a custom container image from a public or private Container Registry. In our case, we are using Azure Container Registry.

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:



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 ☺.