

CHAPTER 4



Build as Docker and Deploy to Azure

This chapter will take you through the steps required to build ASP.NET Core as a Docker container and upload the image to the Azure container registry (to learn more visit <https://docs.microsoft.com/en-us/azure/container-registry/>) using a Team Services build. You will also learn how to use the Docker container image in the Azure container registry to host an application in Azure App Service on Linux (more information available at <https://docs.microsoft.com/en-us/azure/app-service-web/app-service-linux-intro>).

Note The Docker tools for Team Services are still evolving. As of writing of this book, the Team Services tasks for Docker are in their early stages. The chapter summary describes a few more upcoming options regarding Azure's capabilities to handle Docker containers and VSTS Release Management's capabilities when combined with Visual Studio 2017.

Prerequisites: Ensure you have a 64-bit Windows 10 Pro, Enterprise, or Education (1511 November update, Build 10586 or later) machine installed with Visual Studio 2017 RC3 or later. You need a Team Services account and the extension for Docker (<https://marketplace.visualstudio.com/items?itemName=ms-vscs-rm.docker>) installed for that account. You are familiar working with Azure portal to create web apps etc.

Set Up the Environment to Develop Docker-enabled Application

The following steps will guide you in setting up VS 2017 and Docker for Windows, enabling you to create Docker-enabled .NET Core applications.

1. Make sure you have installed Visual Studio 2017 RC3 or later with ASP.NET and web development to enable web application development. See Figure 4-1.

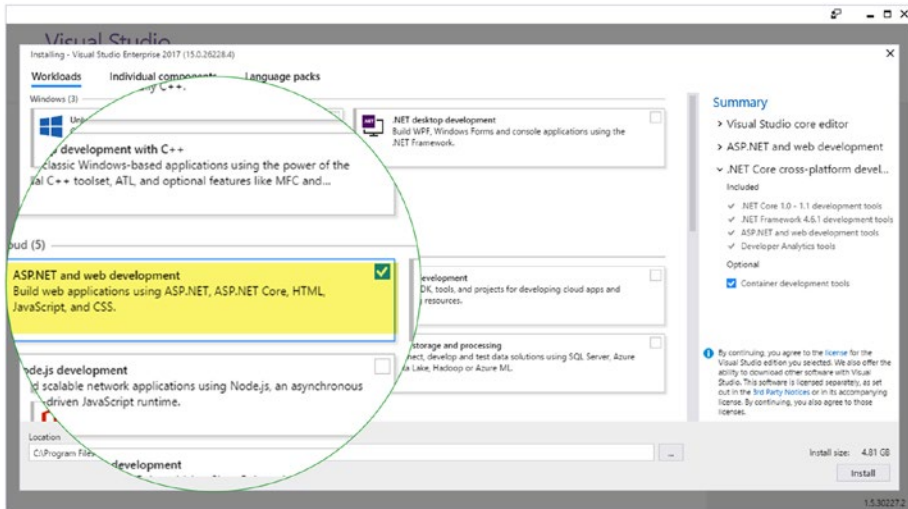


Figure 4-1. Install VS 2017 ASP.NET and web development

2. Install .NET Core cross-platform development, including container development tools. Container development tools allow you to create Docker-enabled ASP.NET core applications. See Figure 4-2.

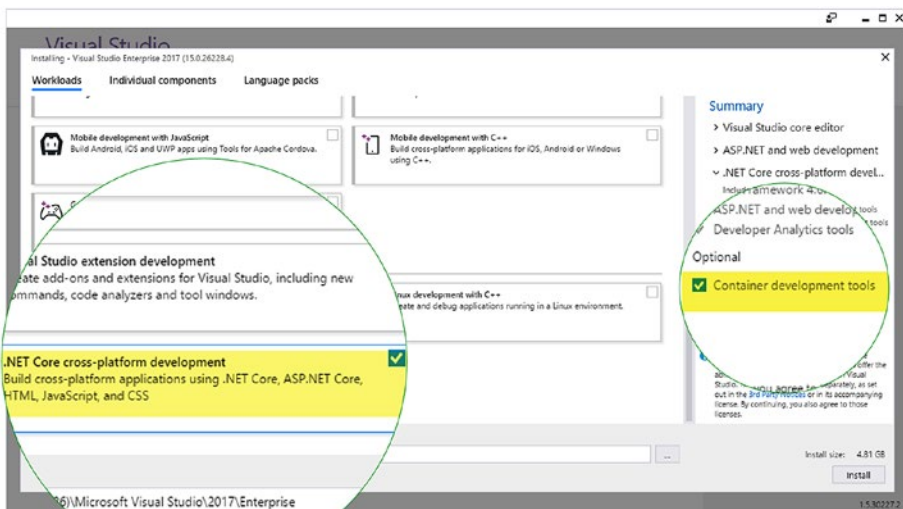


Figure 4-2. Install VS 2017 .NET Core cross-platform development

3. Download Docker for Windows from <https://docs.docker.com/docker-for-windows/install/#download-docker-for-windows>. Docker for windows allow you to develop Docker enabled applications in a Windows PC. For more information visit <https://www.docker.com/docker-windows>. Once completed, run downloaded MSI and click **Install**; follow the installation wizard steps. See Figure 4-3.

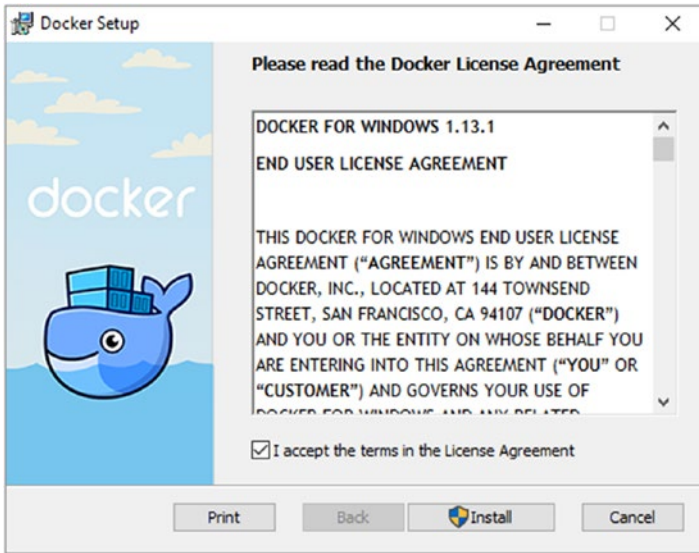


Figure 4-3. Installing Docker for Windows

4. Once the installation has completed, you will see an option available to “**Launch Docker**.” Select it and click on **Finish**. See Figure 4-4.



Figure 4-4. Select Launch Docker and Finish

■ **Note** If you have not set up Hyper-V, you will be asked to enable it. Click on OK for the message, install Hyper-V, and restart your computer after installation. Launch Docker again from the Start menu after restarting the machine.

5. Docker is shown as starting in the system tray. Once it has completed setting things up, you will be notified, as shown in Figure 4-5.

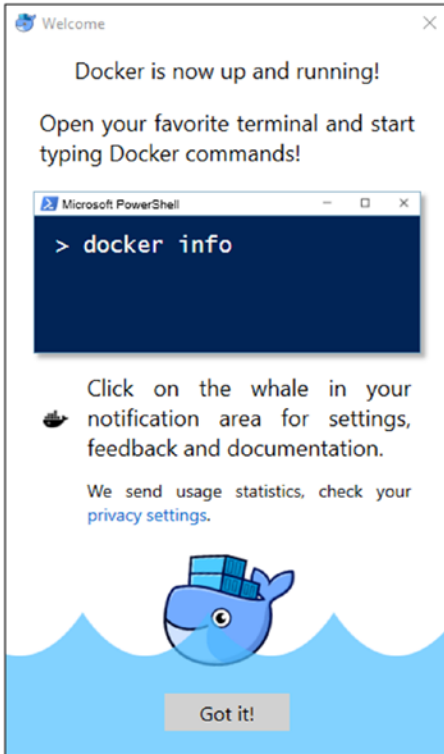


Figure 4-5. *Docker is up and running*

6. Open PowerShell and run a few Docker commands to confirm that Docker is running as expected on your machine.

`docker info` – This command will show the information about Docker for Windows. See Figure 4-6.

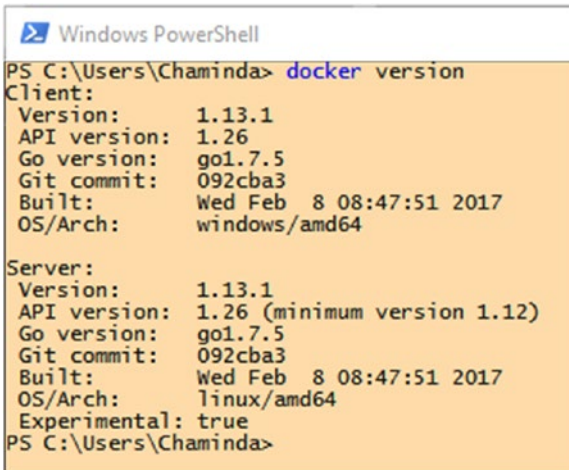


```

Windows PowerShell
PS C:\Users\Chaminda> docker info
Containers: 0
Running: 0
Paused: 0
Stopped: 0
Images: 0
Server Version: 1.13.1
Storage Driver: overlay2
Backing Filesystem: extfs
Supports d_type: true
Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
Volume: local
Network: bridge host ipvlan macvlan null overlay
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: aa8187dbd3b7ad67d8e5e3a15115d3eef43a7ed1
runc version: 9df8b306d01f59d3a8029be411de015b7304dd8f
init version: 949e6fa
Security Options:
seccomp
Profile: default
Kernel Version: 4.9.8-moby
Operating System: Alpine Linux v3.5
OSType: linux
Architecture: x86_64
CPUs: 2
Total Memory: 1.934 GiB
Name: moby
ID: FTDf:3UE6:8OW6:G5OZ:2Y55:XMDV:KZBR:ETH2:AYSX:STOX:CBZG:TFKJ
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): true
File Descriptors: 13
Goroutines: 21
System Time: 2017-02-11T14:40:05.7452444Z
EventListeners: 0
Registry: https://index.docker.io/v1/
Experimental: true
Insecure Registries:
127.0.0.0/8
Live Restore Enabled: false
PS C:\Users\Chaminda>
  
```

Figure 4-6. Docker for Windows information

`docker version` – This command will show the version of Docker for Windows. See Figure 4-7.



```

Windows PowerShell
PS C:\Users\Chaminda> docker version
Client:
Version:      1.13.1
API version:  1.26
Go version:   go1.7.5
Git commit:   092cba3
Built:        Wed Feb  8 08:47:51 2017
OS/Arch:      windows/amd64

Server:
Version:      1.13.1
API version:  1.26 (minimum version 1.12)
Go version:   go1.7.5
Git commit:   092cba3
Built:        Wed Feb  8 08:47:51 2017
OS/Arch:      linux/amd64
Experimental: true
PS C:\Users\Chaminda>
  
```

Figure 4-7. Docker for Windows version

7. In your computer's system tray, right click on Docker and select **Settings**. See Figure 4-8.

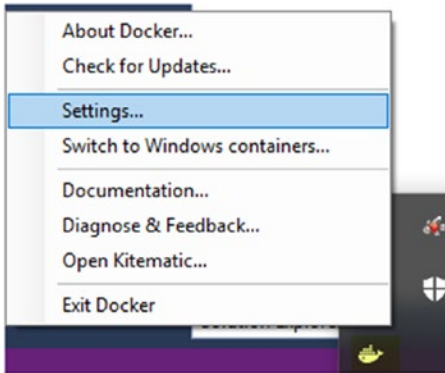


Figure 4-8. Launch Docker settings

8. Share the hard disk drives to Docker in the **Shared Drives** tab of the Docker Settings popup window. The system drive and the drive on which you plan to have your source code must be shared with Docker. Click **Apply** after selecting the desired drives to save the settings. See Figure 4-9.

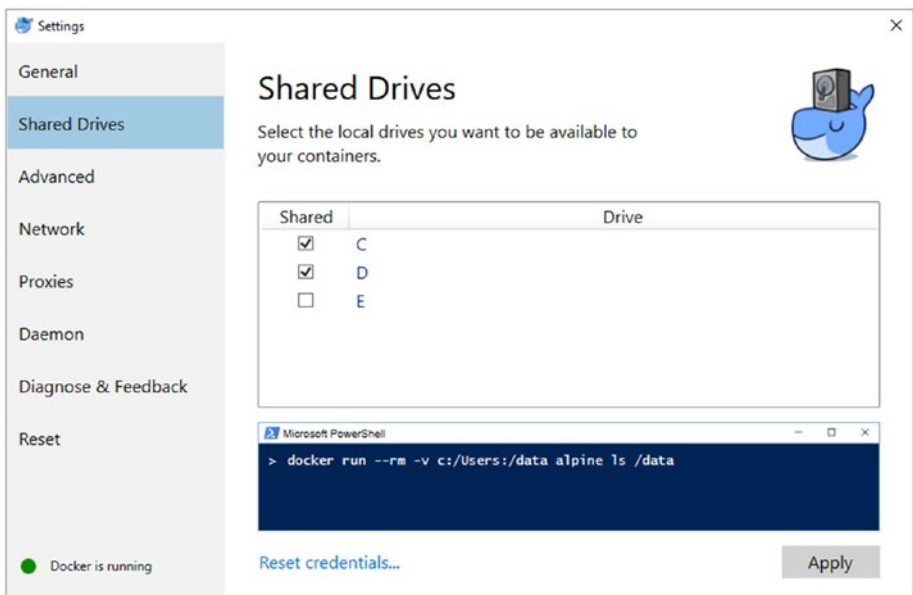


Figure 4-9. Sharing drives with Docker

9. Clicking **Apply** in the Shared Drives tab will open a popup window asking for authentication. Provide the credentials of the user you are logged on as to enable file sharing. See Figure 4-10.



Figure 4-10. Authorizing shared drives access for Docker

You have set up VS 2017, web and cross-platform development components, and Docker for Windows in the development environment. This will allow you to create and run Docker-enabled web applications on your machine. Docker for Windows runs a Linux virtual machine in Hyper-V to allow the running of Docker containers in Windows 10.

Lesson 4.01 – Create a Docker-Enabled ASP.NET Core Application

Let's create a Docker-enabled ASP.NET Core Web API in a new team project using the following steps.

1. Follow the instructions at <https://www.visualstudio.com/en-us/docs/setup-admin/create-team-project> and create a team project (described at the same link) in your VSTS account, with Git as the repository. We have to use Git repository since we are going to build the application on the Linux platform. The new team project will be created when you click on the Create project button. You will be navigated to the new team project automatically, once it is created. See Figure 4-11.

×

Create team project

Project name

Description

Process template

Agile ▾

This template is flexible and will work great for most teams using Agile planning methods, including those practicing Scrum.

Version control

Git ▾

Git is a Distributed Version Control System (DVCS) that uses a local repository to track and version files. Changes are shared with other developers by pushing and pulling changes through a remote, shared repository.

Create project Cancel

Figure 4-11. Creating the team project

2. Open Visual Studio 2017 and, in Team Explorer, click on **Manage Connections ► Connect to Project**. In the resulting popup window, select your Microsoft account. If you are not signed in to your Microsoft account in VS 2017, sign in or add your Microsoft account using the **Showing hosted repositories for** dropdown. Select the Git repository of the team project created in Step 1 of this lesson, provide a **Path** for the local repository, and click the **Clone** button to clone the repository to your machine. See Figure 4-12.

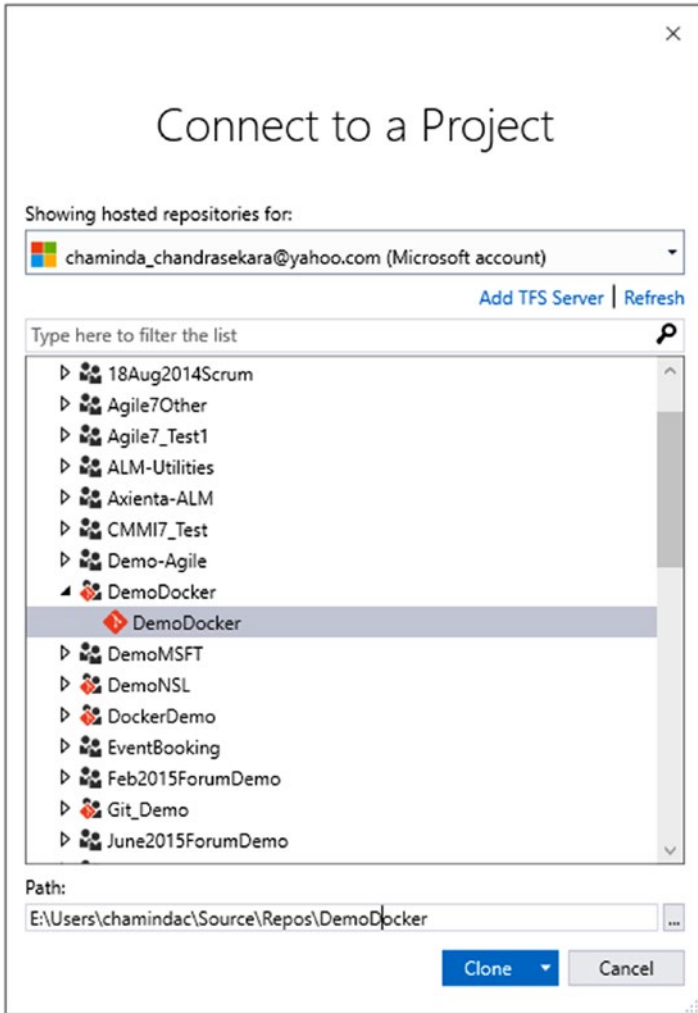


Figure 4-12. Cloning the Git repository

3. Create a new VS solution by clicking **New** under the **Solutions** tab in the Team Explorer window. See Figure 4-13.

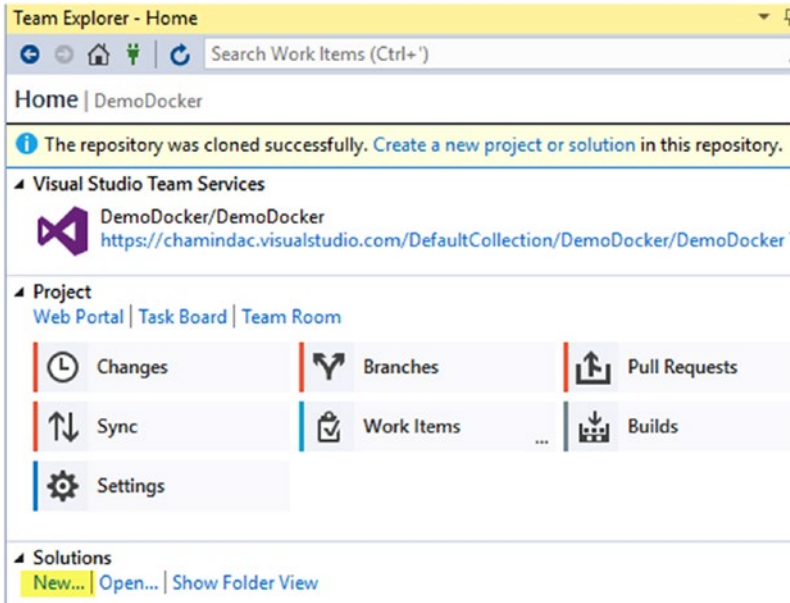


Figure 4-13. Creating a new VS solution

4. Select the Blank Solution template to create a new empty VS solution and name it “DemoCoreDocker.” See Figure 4-14.

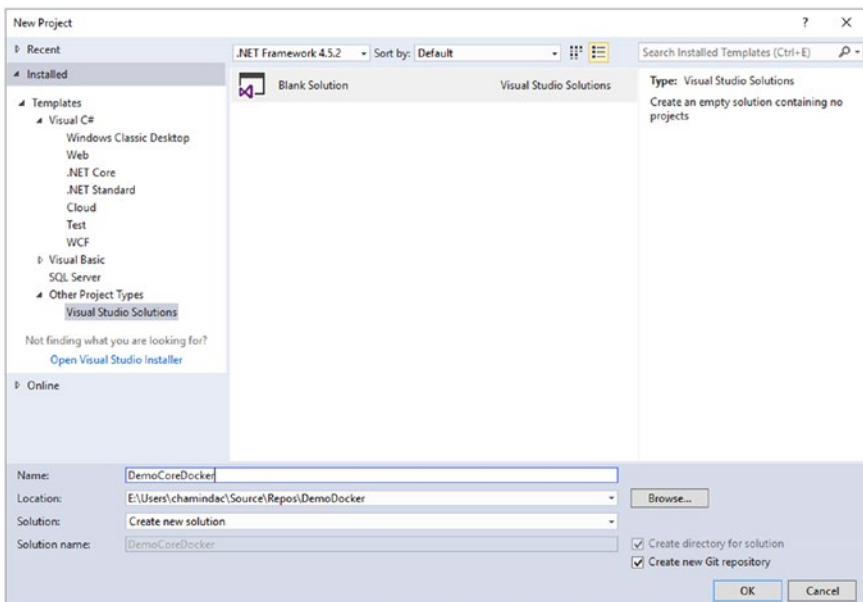


Figure 4-14. Creating an empty solution in Visual Studio

5. Right click on the solution named “DemoCoreDocker” in VS Solution Explorer and go to Add ► New Project. In the Add New Project popup window, select the ASP.NET Core Web Application (.NET Core) project template, fill Name field with CoreDockerAPI, and click OK. See Figure 4-15.

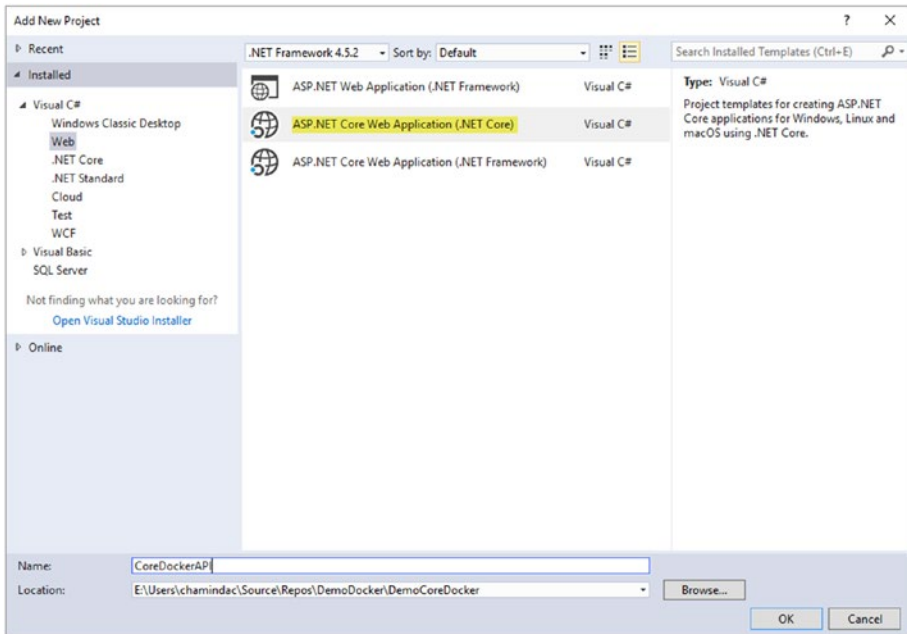


Figure 4-15. Selecting ASP.NET Core Web Application (.NET Core)

6. In the next popup window, select Web API from the ASP.NET Core template options. Check whether it is set as **No Authentication** and keep the “**Enable Docker Support**” box unchecked. We are going to enable Docker support after testing the application. See Figure 4-16.

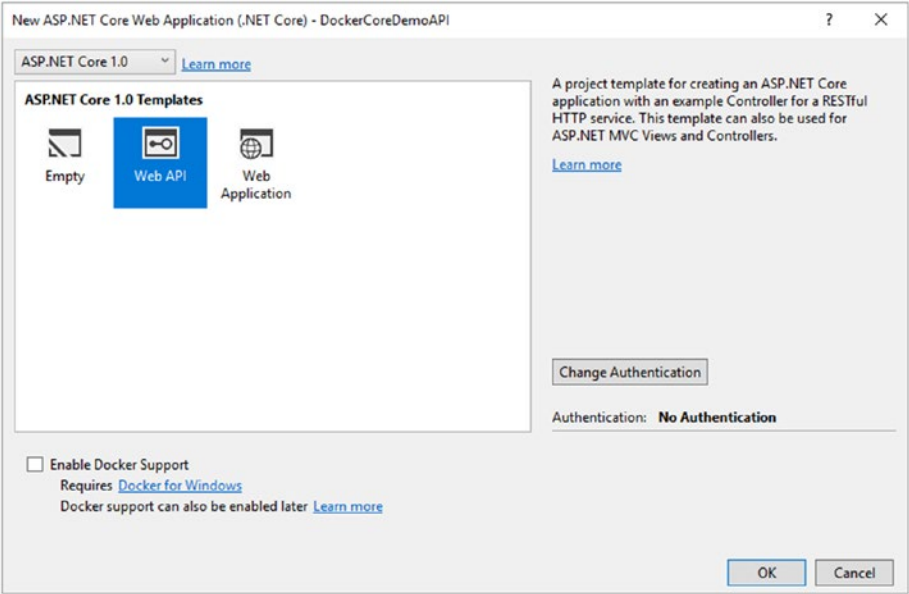


Figure 4-16. Creating ASP.NET Core Web API

If any other Authentication option is shown, click **Change Authentication** and set it to **No Authentication**. Click OK. See Figure 4-17.

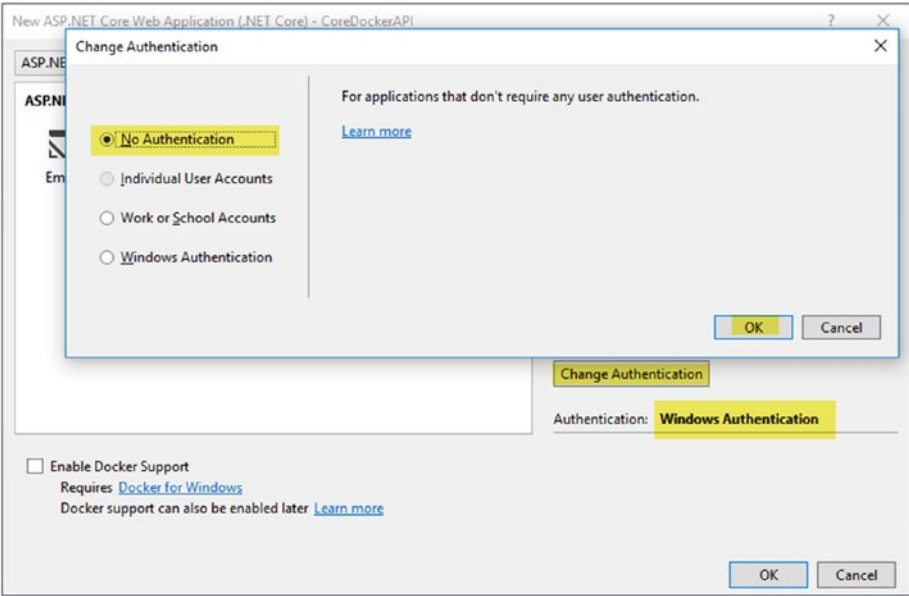


Figure 4-17. No authentication required

7. Clicking OK after selecting Web API template in the window shown in Figure 4-16 will add the Web API project to the solution. It should be available in the Solution Explorer, as shown in Figure 4-18.

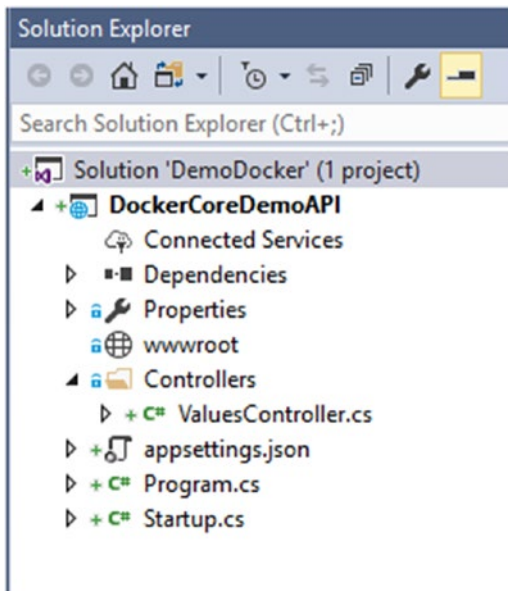


Figure 4-18. API project added to solution

8. Open the ValuesController.cs file and replace the following line in the Get method

```
return new string[] { "value1", "value2" };
```

with the following:

```
return new string[] { "value1", System.Runtime.InteropServices.  
RuntimeInformation.OSDescription };
```

This will allow the default /api/values to display the operating system description of the host of the CoreDockerAPI. See Figure 4-19.

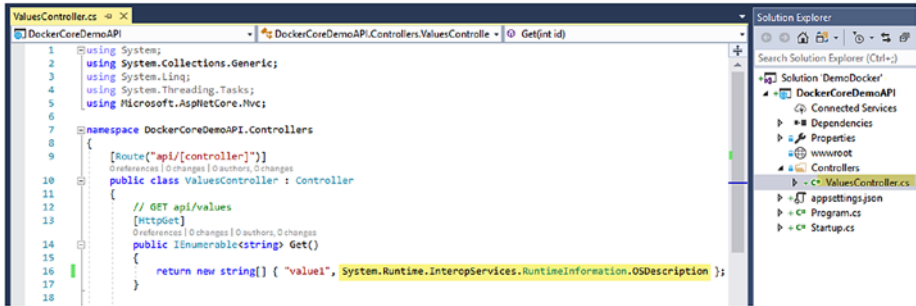


Figure 4-19. Allow web API to show host OS description

9. Run the application by hitting F5 or use IIS Express, as shown in Figure 4-20.

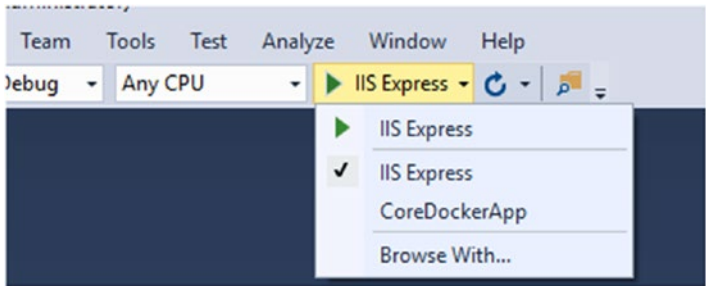


Figure 4-20. Running the web API

A browser window will launch, and you will see that the API is returning the current OS description as “Microsoft Windows” with the version number. This confirms the web API you have created is not yet enabled with Docker and is still running on the Windows platform. See Figure 4-21.

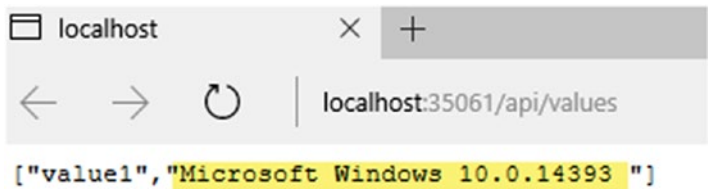


Figure 4-21. Web API running on Windows OS

10. Since we have a working web API project, before making any additional changes we should commit it to the repository. For this, click on **Changes** in the Team Explorer **Home** and commit the solution to the local Git repository. Then, you can push the changes to the Git repository of the team project by clicking on Push. We must push to the repository, since this is the first set of code that is committed. To learn more about working with Team Foundation Git with Visual Studio, refer to the article at <https://www.visualstudio.com/en-us/docs/git/gitquickstart>. See Figure 4-22.

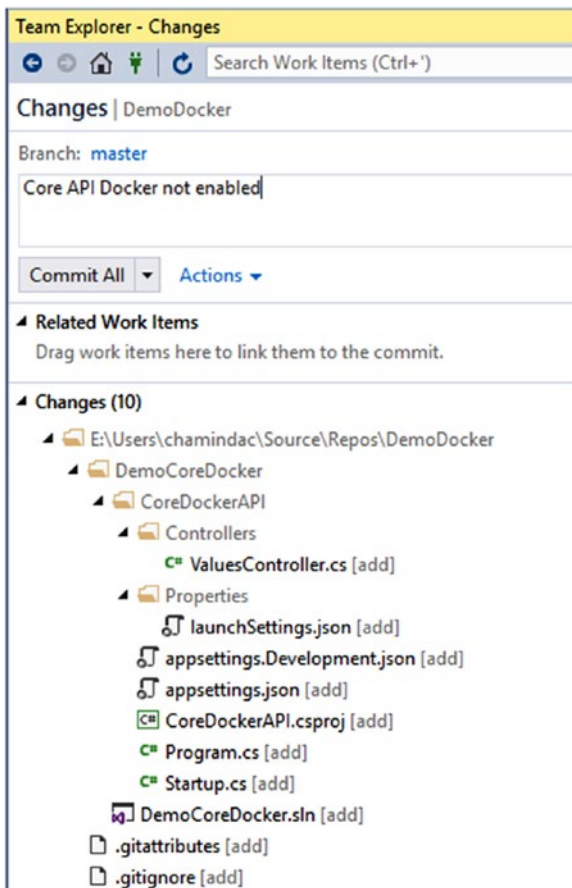


Figure 4-22. Committing the solution to Git repository

11. Let's add Docker support to the web API to enable it to be deployed as a Docker container. Right click on DockerCoreDemoApp in the Solution Explorer page and go to Add ► Docker Support. See Figure 4-23.

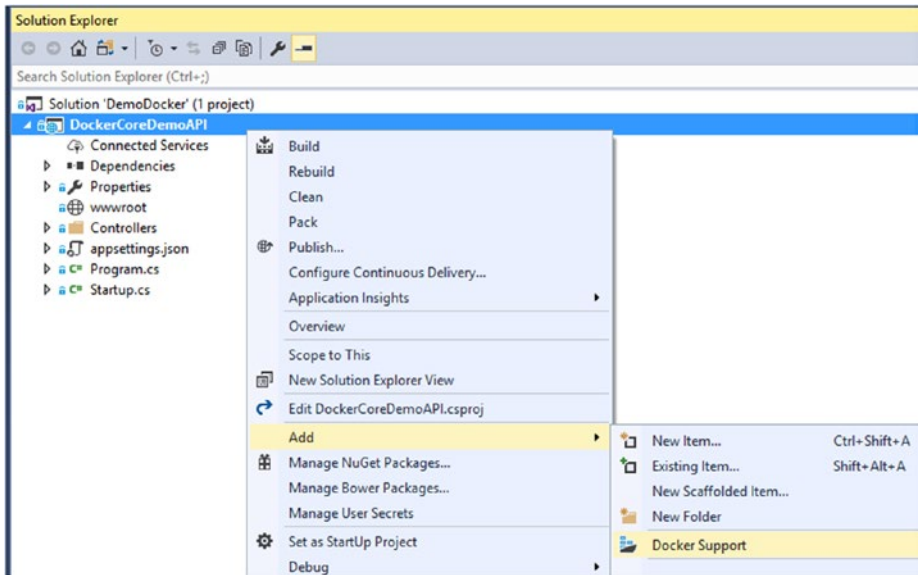


Figure 4-23. Adding Docker support to web API

12. New docker-compose.dcproj (it is the project highlighted in blue in Figure 4-24) and .yaml files are added at the solution level. A Dockerfile is added to the CoreDockerAPI project, along with a .dockerignore file. Dockerfiles and .yaml files contain the information required to compose a Docker container, while the .dockerignore file has the information on which content to ignore when packaging. You do not have to change anything for this lesson. You can get more information about these files in the following articles:

<https://docs.docker.com/compose/compose-file/>
<https://docs.docker.com/engine/reference/builder/>

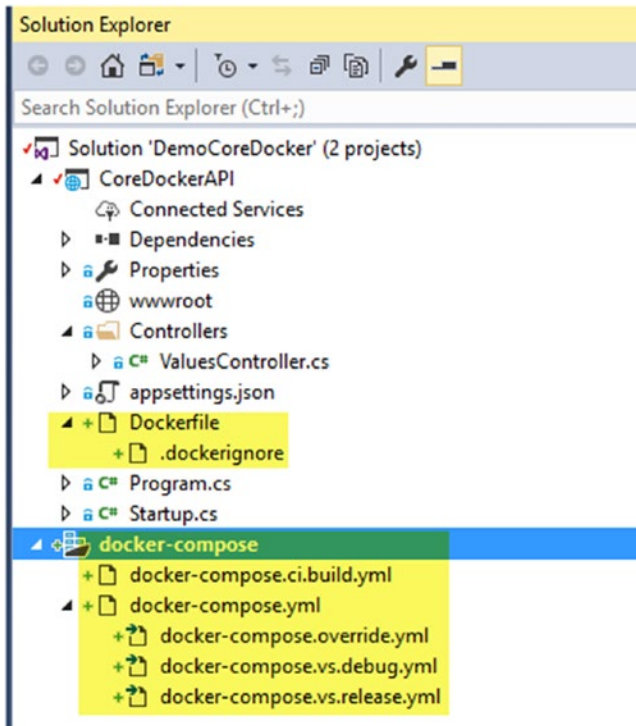


Figure 4-24. Docker-enabled web API

13. docker-compose should be set as the startup project to enable running it with Visual Studio. If not available, right click on docker-compose and choose Set as Startup Project. Then, hit F5 or click on Docker, as shown in Figure 4-25, to run the web API in Docker. This will take few minutes to run, as it is building a Docker image and deploying it to a Docker container (Docker on Windows is enabled by running a Linux Docker container in Hyper-V).

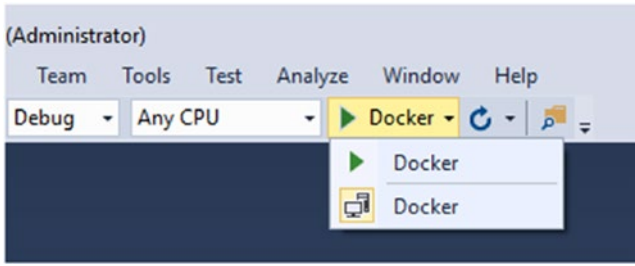


Figure 4-25. Running web API with Docker

A browser window will be launched, and you can see the running OS description shown as Linux. This confirms the web API is running in a Docker container on Linux. See Figure 4-26.



Figure 4-26. Web API running on Linux Docker container

14. Commit the new files to the local Git repository and sync changes with the team project Git repository to enable them to be built with a Team Services build in a future lesson in this chapter (<https://www.visualstudio.com/en-us/docs/git/gitquickstart>). See Figure 4-27.

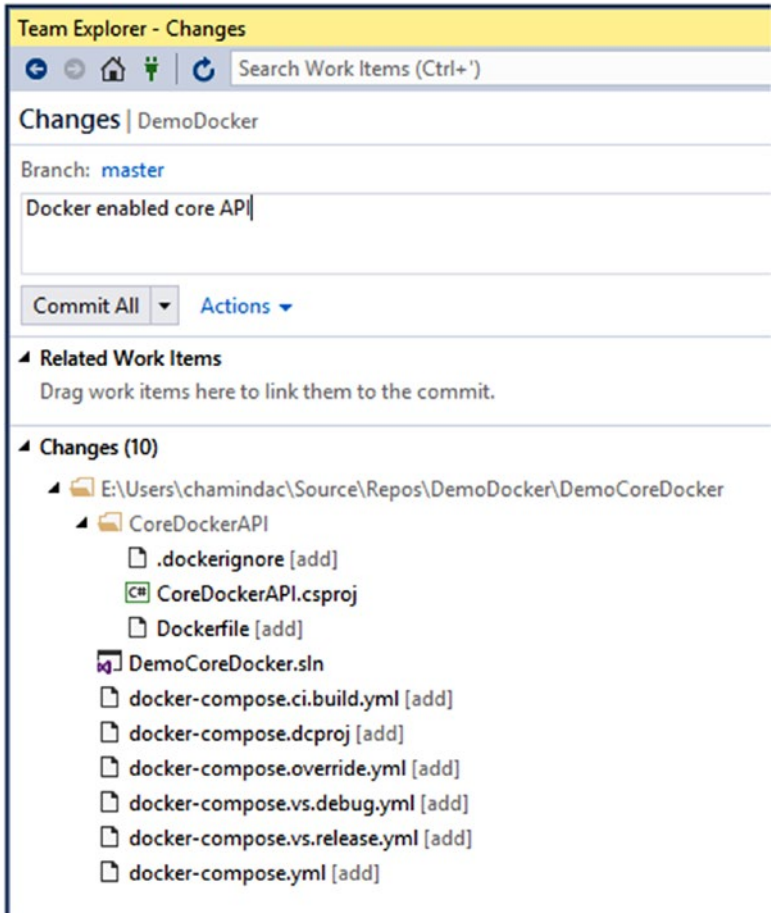


Figure 4-27. Committing Docker-enabled web API to Git repository

In this lesson, you have created a new team project with the team foundation Git as the source control repository and added a Docker-enabled web API to it. You managed to test the API locally and confirmed it can build a Docker image, which can be run on a Docker container on Linux.

Lesson 4.02 – Create Azure Container Registry

An Azure container registry lets you store images of containers, such as Docker Swarm, DC/OS, and Kubernetes, and of Azure services like Service Fabric, App Services, and so forth. It is a private Docker registry and provides you with local, network-close storage of your container images within your subscription. Learn more from <https://docs.microsoft.com/en-us/azure/container-registry/container-registry-intro>.

1. Go to the Azure portal and click the green + sign to add a new item. Type “Azure Container” in the search field and select Azure Container Registry from the list. See Figure 4-28.



Figure 4-28. Searching for Azure container registry

In the search results, select Azure Container Registry by Microsoft. See Figure 4-29.

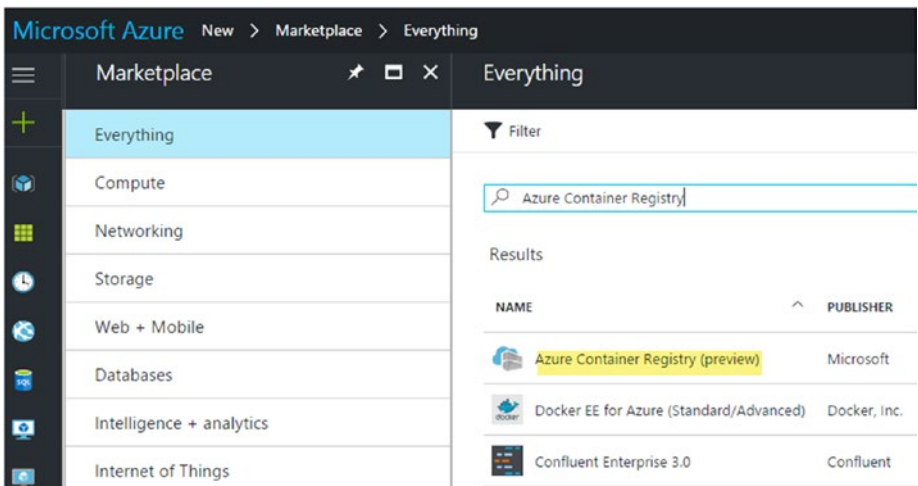


Figure 4-29. Selecting Azure Container Registry

2. Click on Create in the window that opens. See Figure 4-30.

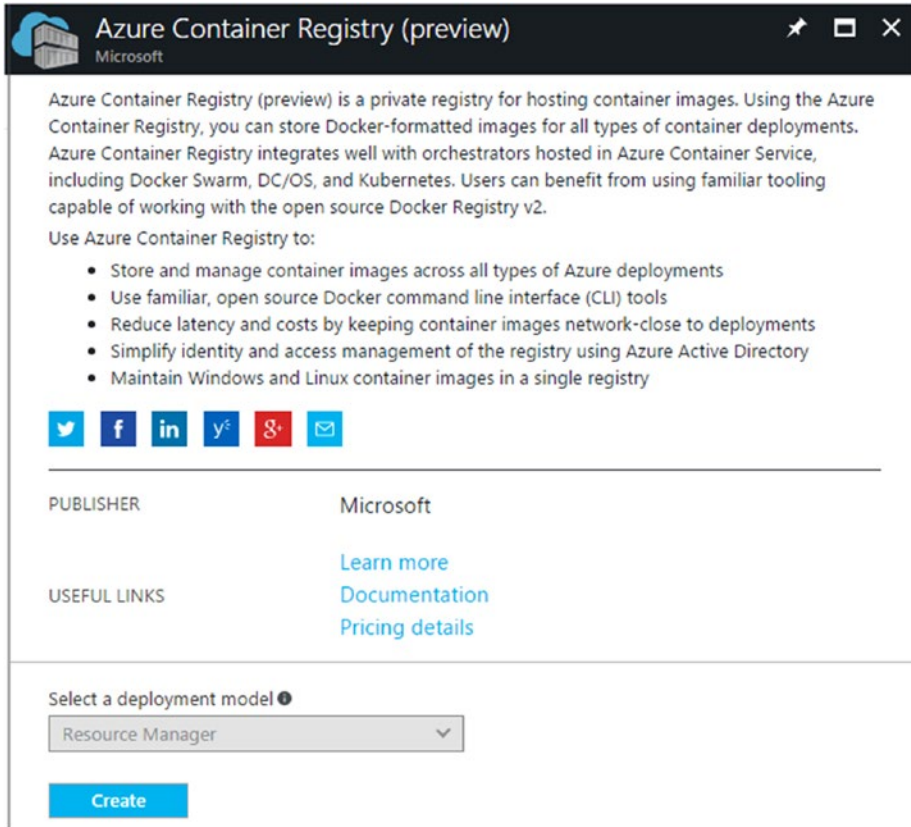


Figure 4-30. *Creating the Azure container registry*

3. Provide a name for the registry. Select West Europe or West US as the location. This is required, since as of the writing of this book, the Azure Linux app service is available for only West US, West Europe, and Southeast Asia. To make the container available to the Linux app service, we need to have both, Azure container registry and app service app on Linux, set to the same region. Select the option to create a new resource group and provide a name. Enable Admin User, as we are going to use this user's username and password (both auto-generated) to connect the group to the app service app on Linux and so forth. Click on Create after providing all the information, as shown in Figure 4-31.

Create container registry PREVIEW

* Registry name
demodockerregistry ✓

* Subscription
Azure-ChamindaC ▼

* Resource group ⓘ
☒ Create new ☐ Use existing
demodockerregistry-RG ✓

* Location
West Europe ▼

* Admin user ⓘ

* Storage account
(new) demodockerregistry160023 >

☐ Pin to dashboard

[Automation options](#)

Figure 4-31. Filling in required information to create the container registry in Azure

4. The new resource group gets created with the name provided in the previous step. You can click on Resource Groups in the Azure portal to view the new resource group. See Figure 4-32.

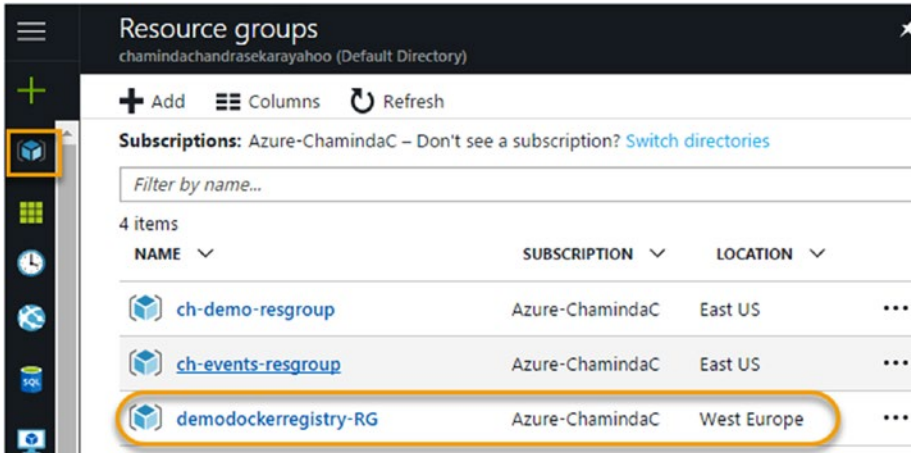


Figure 4-32. New resource group

5. Click on the new resource group name. You can see in the resource group that an Azure container registry gets created. See Figure 4-33.

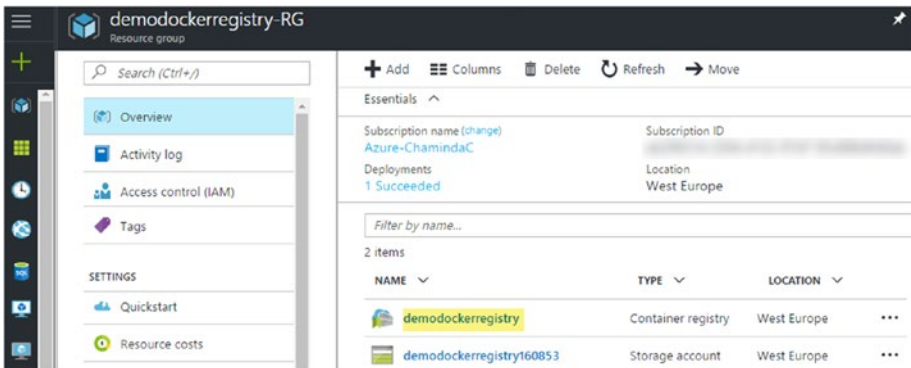


Figure 4-33. New Azure container registry created

You have created an Azure container registry using the Azure portal in this lesson. You will be using it in a future lesson in this chapter to deploy with the Docker-enabled ASP.NET Core web API, which we created in a previous lesson in this chapter.

Lesson 4.03 – Create Azure App Service on Linux App

The Azure app service on Linux can host web apps on Linux. The **app service app on Linux** you will create in this lesson will be used to host the Docker-enabled web API you created in a previous lesson in this chapter. Learn more at <https://docs.microsoft.com/en-us/azure/app-service-web/app-service-linux-intro>.

1. In the Azure portal, click on Resource Groups and click **Add**. You are going to use this new resource group to create an Azure app service app on Linux. See Figure 4-34.

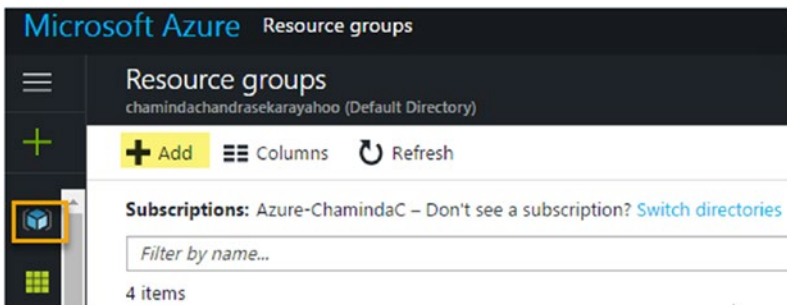


Figure 4-34. Adding a new resource group

2. Provide a name for the resource group and select the location as West Europe (region should be same region as that for the Azure container registry in the previous lesson). Click on Create. See Figure 4-35.

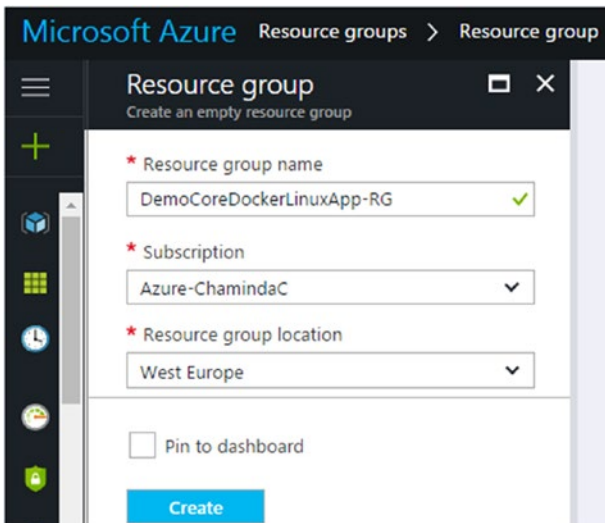


Figure 4-35. Creating new resource group

- Now, we have both an Azure container registry resource group (created in the previous lesson) and a resource group for App Service App on Linux in the same region, West Europe. See Figure 4-36.

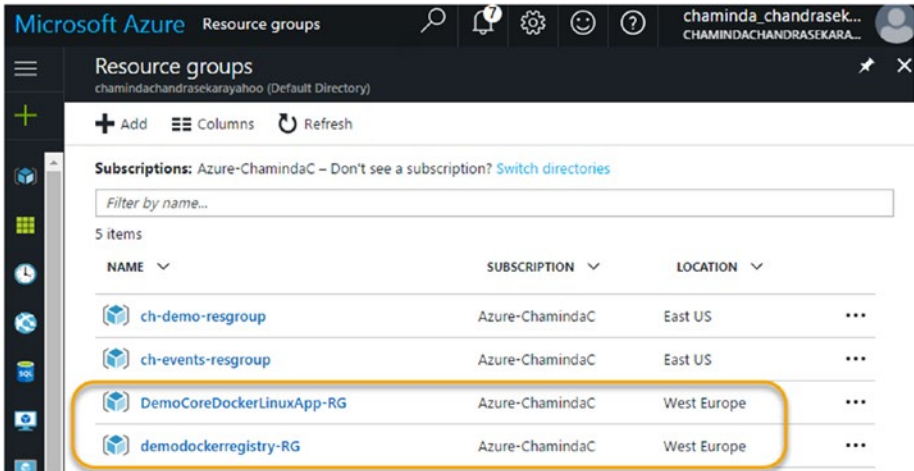


Figure 4-36. Both resource groups set to same region

- In the Azure portal, click on the green + to add a new item and search for “app on linux.” Select **Web App On Linux**. See Figure 4-37.

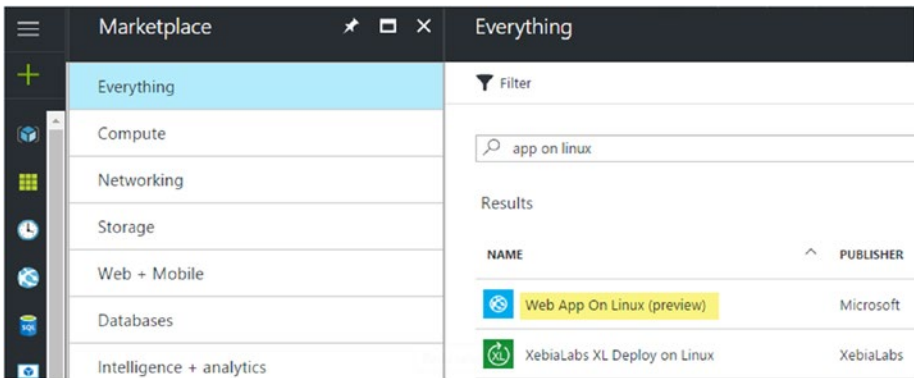


Figure 4-37. Selecting Web App On Linux

5. Click **Create** on the preview page to create a web app on Linux. See Figure 4-38.

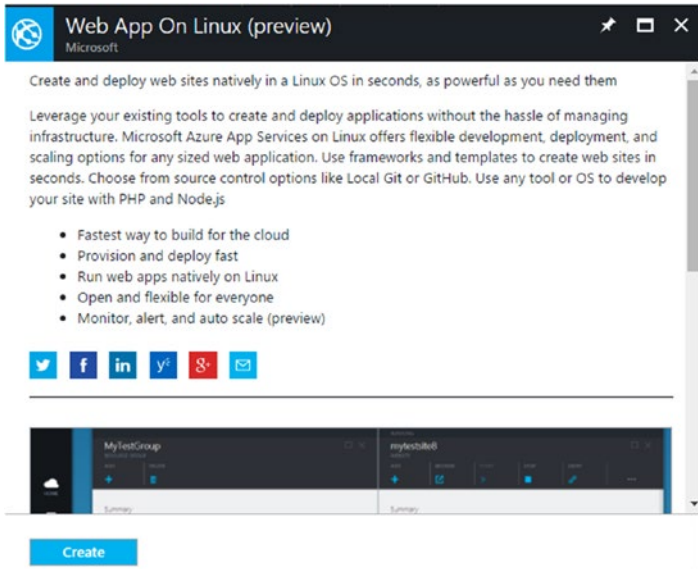


Figure 4-38. Creating web app on Linux

6. Provide a name for the app. Select the “**Use Existing**” Resource Group option and select the resource group created in this lesson. You will see that a new **service plan** is created in the West Europe region (region should be the same region that you used for the Azure container registry in the previous lesson). Click on **Configure container** to configure the container option before creating the app. See Figure 4-39.

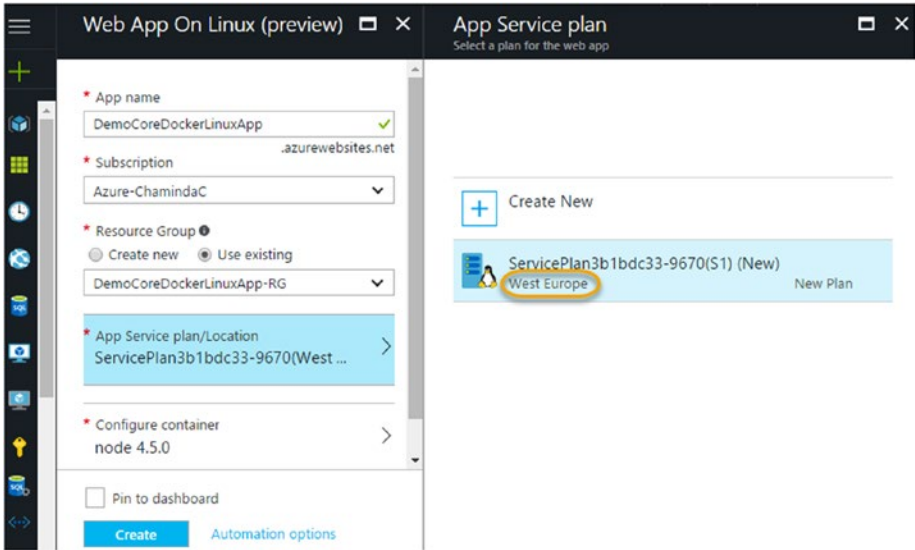


Figure 4-39. Provide information to create the web app on Linux

7. Leave the node.js container for the time being. We will set up a private registry for this later. Click Create. See Figure 4-40.

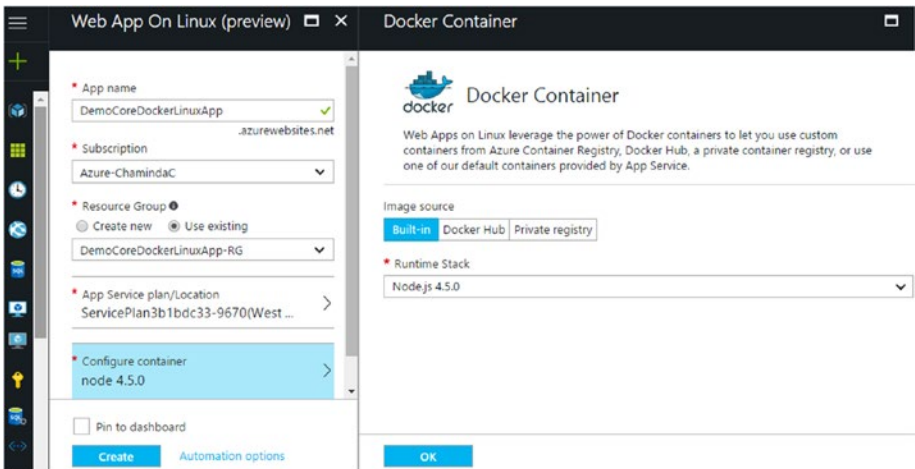


Figure 4-40. Configure container

8. A new app service app gets created. Click on its name. See Figure 4-41.

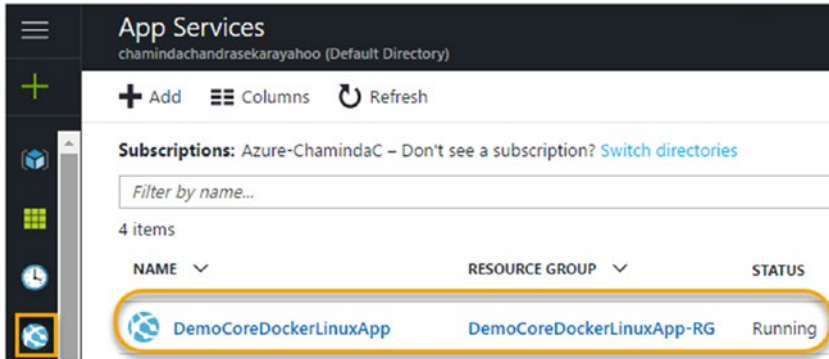


Figure 4-41. App service app on Linux is created

9. In the app overview, click on the available URL. See Figure 4-42.

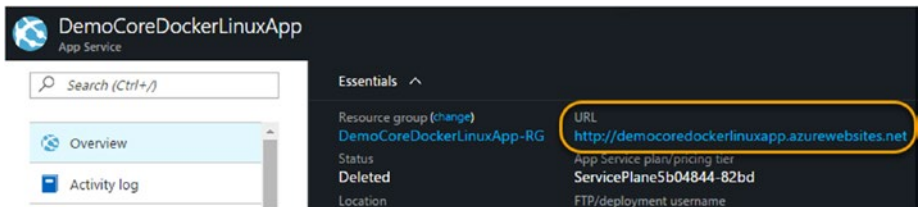


Figure 4-42. URL of the new web app

10. The new web app should load in a new tab in the browser. See Figure 4-43.

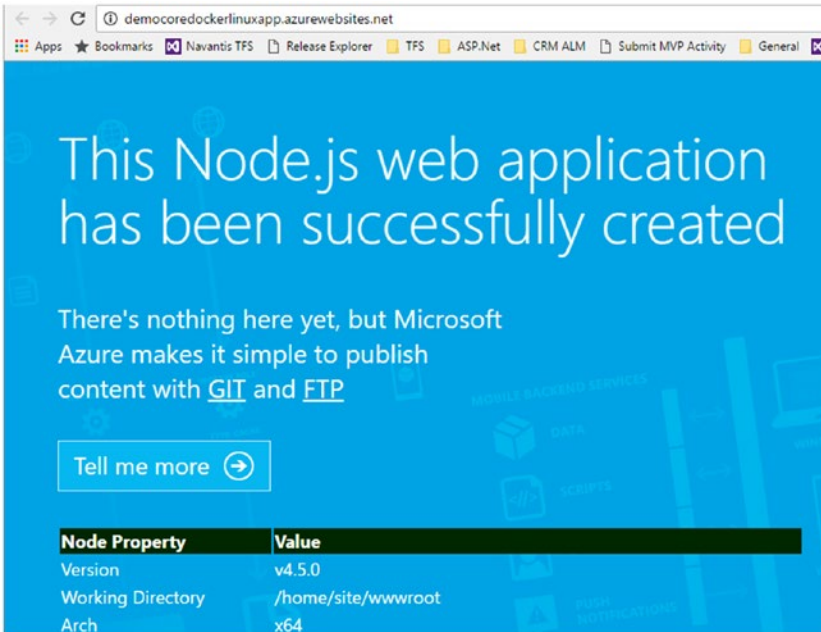


Figure 4-43. Web app on Linux is ready

In this lesson, you have created a new Azure app service app on Linux in the same region where we created the Azure container registry in the previous lesson. You will be using this app to host the Docker-enabled ASP.NET Core web API, which was created in a previous lesson.

Lesson 4.04 – Create a Build to Push Container Image to Azure Container Registry

Prerequisites: You need to install the Team Services extension called “Docker Integration” from <https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.docker> to your Team Services account. Installation of marketplace extensions is explained in Chapter 2.

1. First, link the Azure container registry to the team project. Go to **Services** tab of the settings in the team project and click **New Service Endpoint**, and then select **Docker Registry**. This will open a popup window for **Add new Docker Registry Connection**. See Figure 4-44.

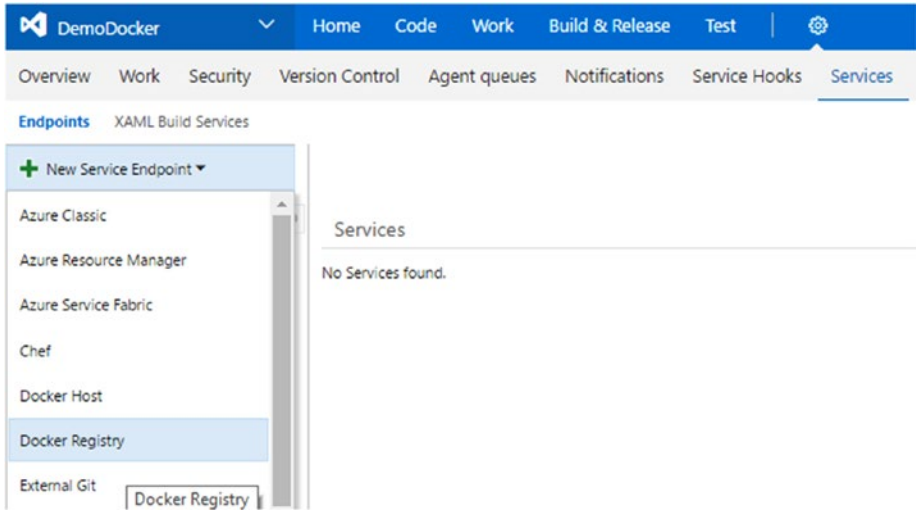


Figure 4-44. Linking Docker registry to Team Services

2. In another browser window, open the Access Key section of the container registry in the Azure portal. See Figure 4-45.

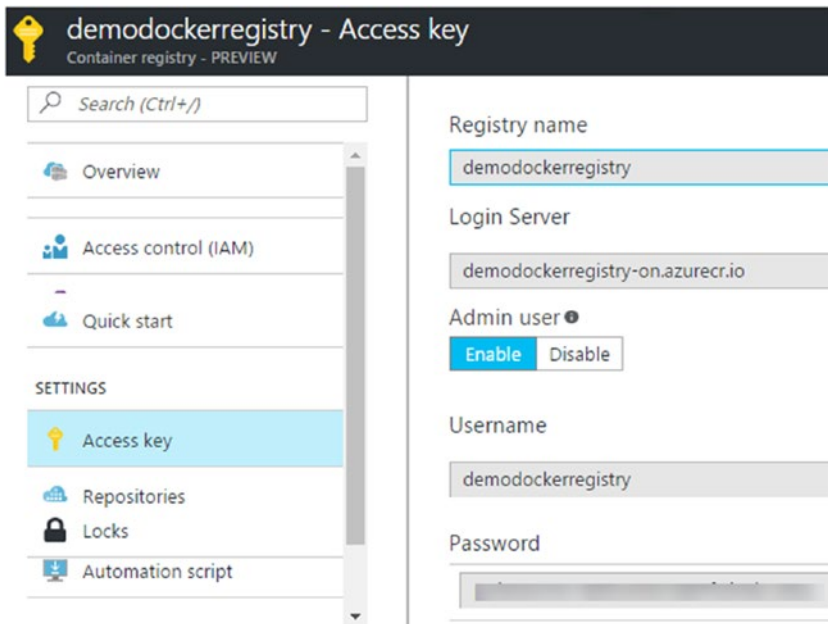


Figure 4-45. Access Key section of container registry

3. Provide the registry connection information in the **Add new Docker Registry Connection** window to connect Docker registry to the team project.

Registry - Login server = demodockerregistry-on.azurecr.io

Connection - Docker Registry = <https://demodockerregistry-on.azurecr.io>

Provide a name for the connection. Use your **username** as **Docker Id** and the **password** from the container registry in the connection and Click **OK**. Make sure **Admin user** is enabled in the registry. Save changes in Azure portal, if you have enabled **Admin user** just now (if it was left disabled when the registry was created in a previous lesson). See Figure 4-46.

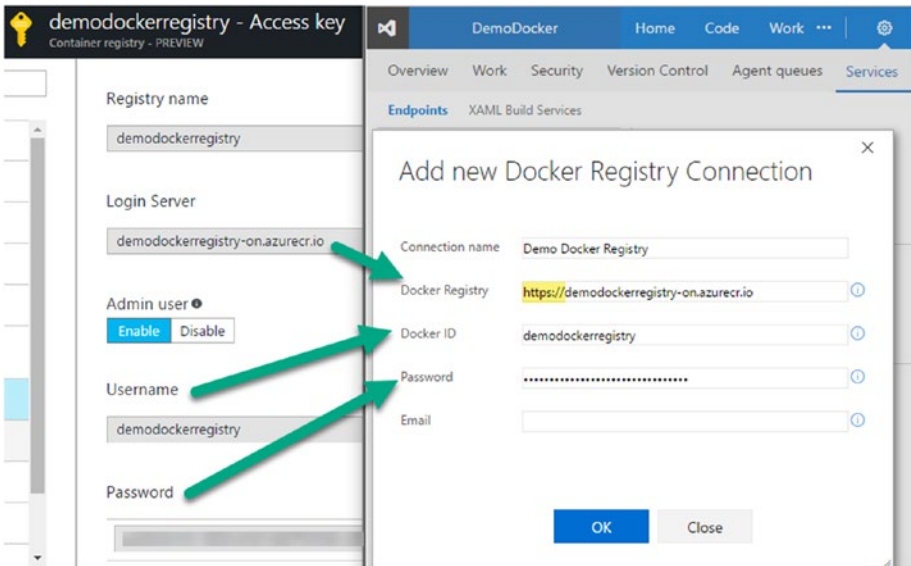


Figure 4-46. Linking container registry to Team Services project

4. A new **Demo Docker Registry** connection is now available for the team project. See Figure 4-47.

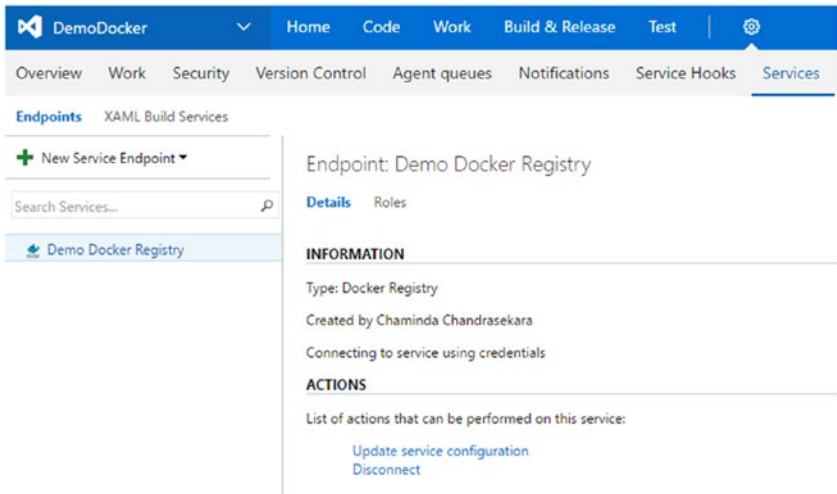


Figure 4-47. Docker registry is connected to team project

5. Create a new empty build definition named “DockerDemoBuild” in the Build & Release tab of the team project. Select agent queue Hosted Linux (Preview) in the General tab of the build definition. Provide a build number format, such as `$(date:yyyyMMdd)$(rev:.r)`. See Figure 3-48.

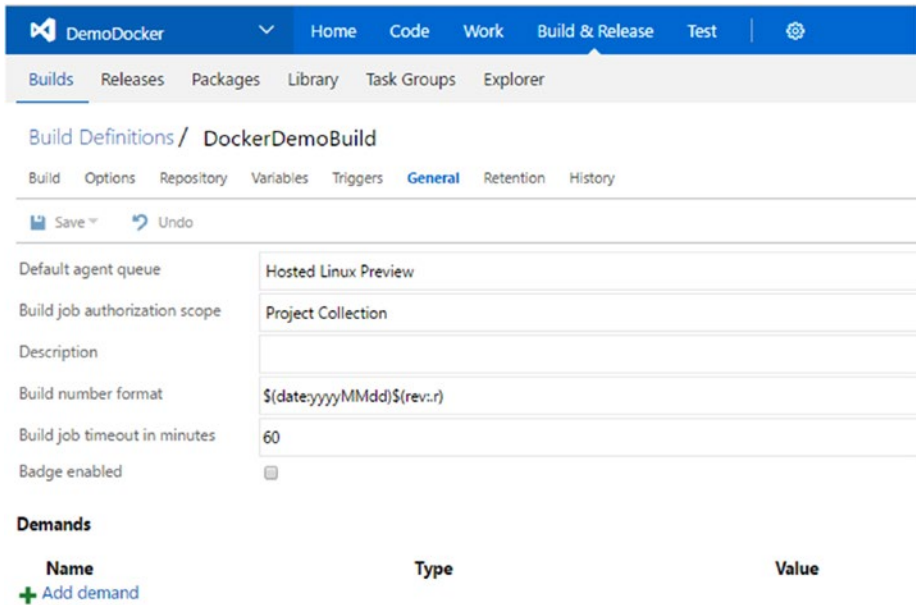


Figure 4-48. Build definition using hosted Linux agent pool

6. In the Repository tab of the build definition, select Git and select the repository of the team project. Set default branch to **master**. Uncheck all other options. The “Sources” option in the Clean options dropdown can be set to true or false. See Figure 4-49.

Build Definitions / DockerDemoBuild

Build Options **Repository** Variables Triggers General Retention History

Save Undo

Repository type: Git

Repository: DemoDocker

Default branch: master

Label sources: Don't label sources

Report build status: ☐

Checkout submodules: ☐

Checkout files from LFS: ☐

Don't sync sources: ☐

Shallow fetch: ☐ Depth: 0

Clean: false Clean options: Sources

Figure 4-49. Repository selected for the build definition

7. In the Triggers tab, check the “Continuous integration” option to enable build triggering for each commit made to the repository master branch. Select “master branch” for the Branch filters dropdown. See Figure 4-50.

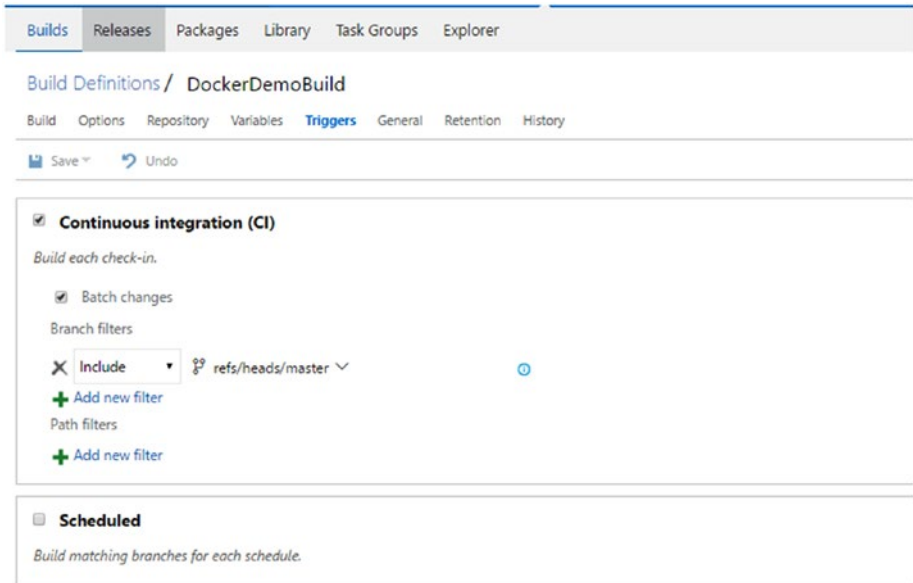


Figure 4-50. Build all commits to master branch

8. Leave Variables tab as it is.
9. For Options and Retention tabs, do not make any changes.
10. In the Build tab, add three build steps using the Docker Compose task that comes with the Docker Integration extension (<https://marketplace.visualstudio.com/items?itemName=ms-vscode.docker>). See Figure 4-51.

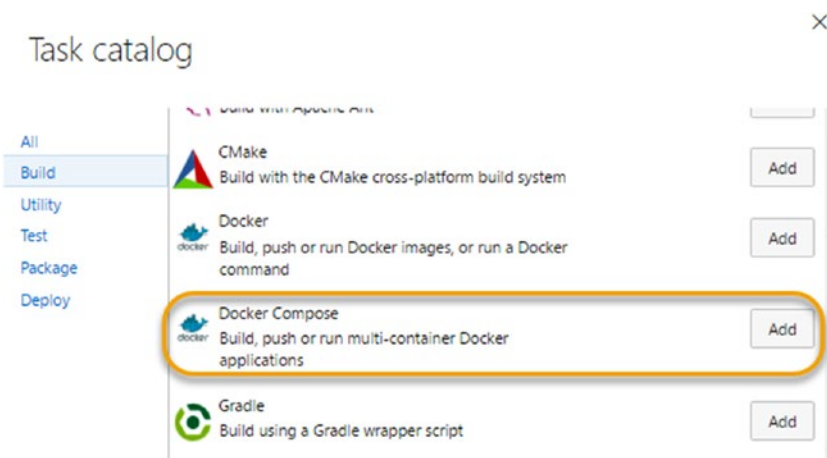


Figure 4-51. Add Docker Compose tasks to build definition

11. i. Name the first Docker Compose step “Build Repository and Create Container” since that is what happens in this step.
 - Select the Demo Docker Registry connection created earlier.
 - Provide `**/docker-compose.ci.build.yml` as the Docker Compose file. This was added when we enabled Docker for the project.
 - Make sure to uncheck both the “**Qualify Image Names**” and the “**Run In Background**” options.
 - Provide Project name as `$(Build.RepositoryName)`.
 - Select “Run a specific service image” for the Action dropdown and provide service name as `ci-build`.
 - In Advanced Options, check the “**No-op if no Docker Compose File**” box and provide `$(System.DefaultWorkingDirectory)` as the Working Directory. See Figure 4-52.

Build Definitions / DockerDemoBuild

in progress Queue new build... Summary Security

Build Options Repository Variables Triggers General Retention History

Save Undo

+ Add build step...

- Build Repository and Create Container Docker Compose
- Build services image Docker Compose
- Push services Docker Compose

Build Repository and Create Container

Version 0.1

Docker Registry Connection: Demo Docker Registry

Docker Compose File: `**/docker-compose.ci.build.yml`

Additional Docker Compose Files:

Environment Variables:

Project Name: `$(Build.RepositoryName)`

Qualify Image Names: ☐

Action: Run a specific service image

Service Name: `ci-build`

Container Name:

Ports:

Working Directory:

Endpoint Override:

Command:

Run In Background: ☐

Advanced Options

Docker Host Connection:

No-op if no Docker Compose File: ☒

Require Additional Docker Compose Files: ☐

Working Directory: `$(System.DefaultWorkingDirectory)`

Control Options

Enabled: ☒

Continue on error: ☐

Always run: ☐

Timeout: 0

Figure 4-52. Build repository and create container

ii. Name the second Docker Compose step “Build Services Image.” The container generated in the previous step will be packaged as an image in this step.

- Select the Demo Docker Registry and provide `**/docker-compose.yml` as the Docker Compose file.
- For Additional Docker compose file, provide `docker-compose.ci.yml`.
- Set Environment Variable to `DOCKER_BUILD_SOURCE=.`. This will set its value to empty while running the task.
- For Project Name, type in `$(Build.Repository.Name)` and check the “Qualify Image Names” box.
- Select “Build service images” from the Action dropdown.
- Type “RTM” in Additional Image Tags field.
- Check options for “Include Source Tags” and “Include Latest Tag.” These tags are useful when referring to a container image. You can see the RTM tag being used when the wiring up of the Azure Container Registry with the Azure app service app on Linux is done. See Figure 4-53.

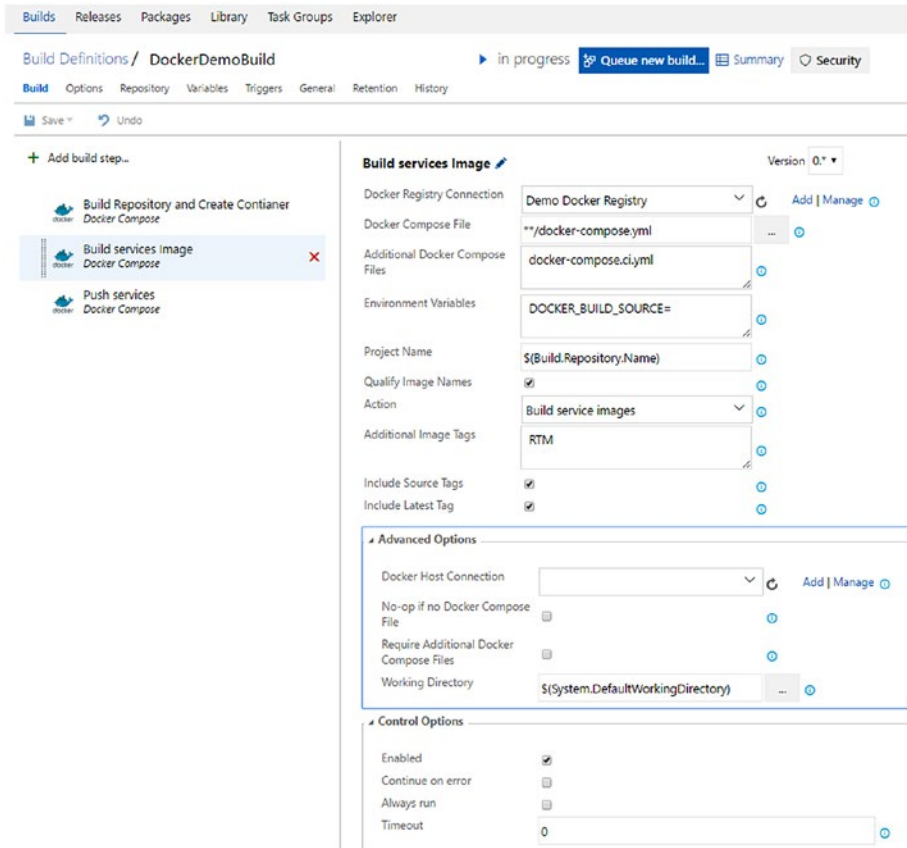


Figure 4-53. Build container service image

iii. In the third Docker Compose task, we are pushing the container image to the Azure Container Registry.

- Provide all options similar to previous task, except the Action. Action should be set to “Push service images.”
- Follow the screenshot in Figure 4-54 to set the parameters properly.

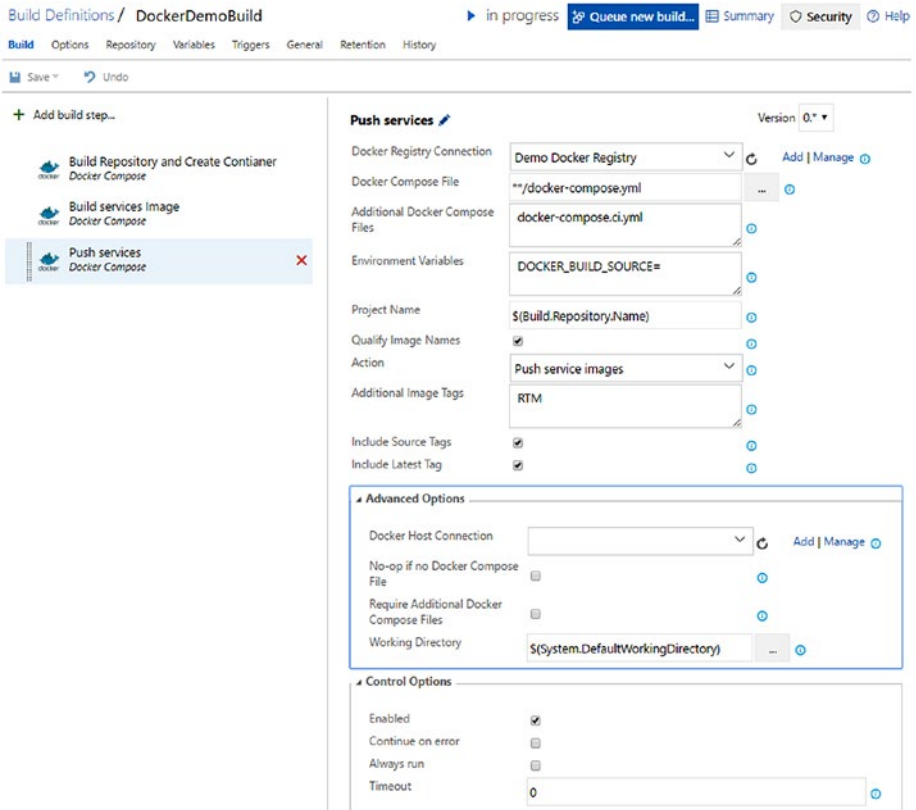


Figure 4-54. Pushing image to Azure container registry

12. Save the build definition and queue a new build. You can see the build is pushing a container image with the application's API name and using the RTM tag specified. See Figure 4-55.

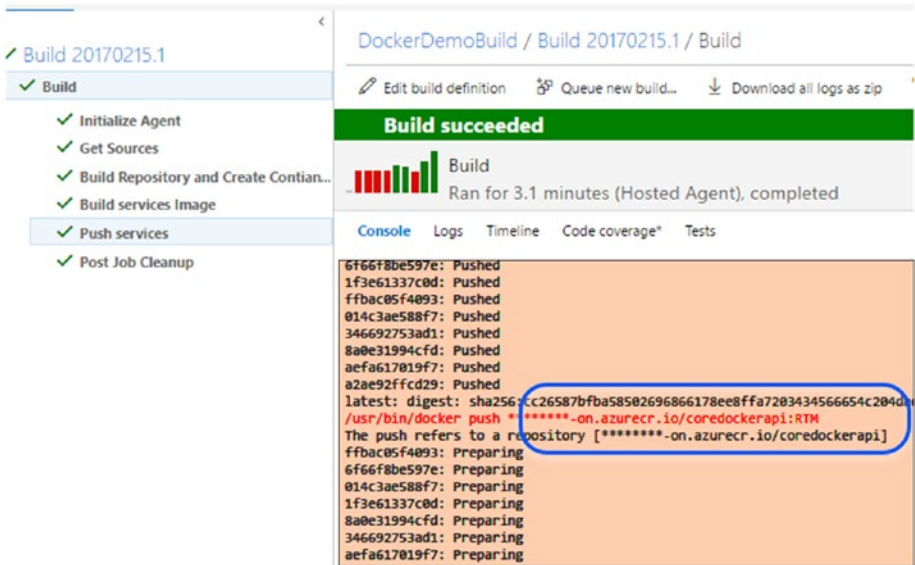


Figure 4-55. Building the Docker image and pushing it to registry

You have created a build in this lesson that can build a Docker-enabled ASP.NET Core web API. This build can create the Docker image and push it to the Azure container registry.

Lesson 4.05 – Wire Up Container Registry and the App Service App on Linux

The next step is to wire this Azure app service app on Linux with the Azure container registry that was pushed with a Docker container image in the previous lesson.

1. Open the app service app on Linux and the Azure container registry in two browser windows. Navigate to the Azure container registry's Access key and make sure to enable Admin user. See Figure 4-56.

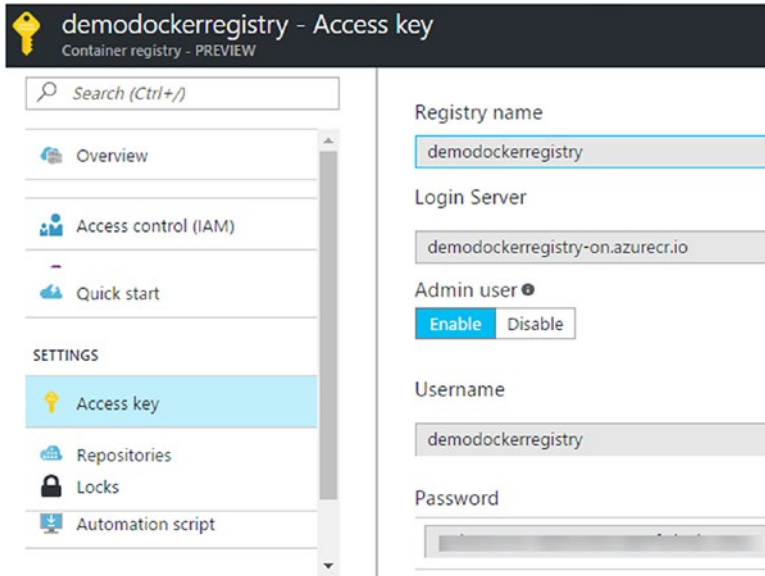


Figure 4-56. Make sure admin user enabled in access key of registry

2. In a third browser window, navigate to Web App on Linux and go to Docker Containers section. Select Private registry. See Figure 4-57.

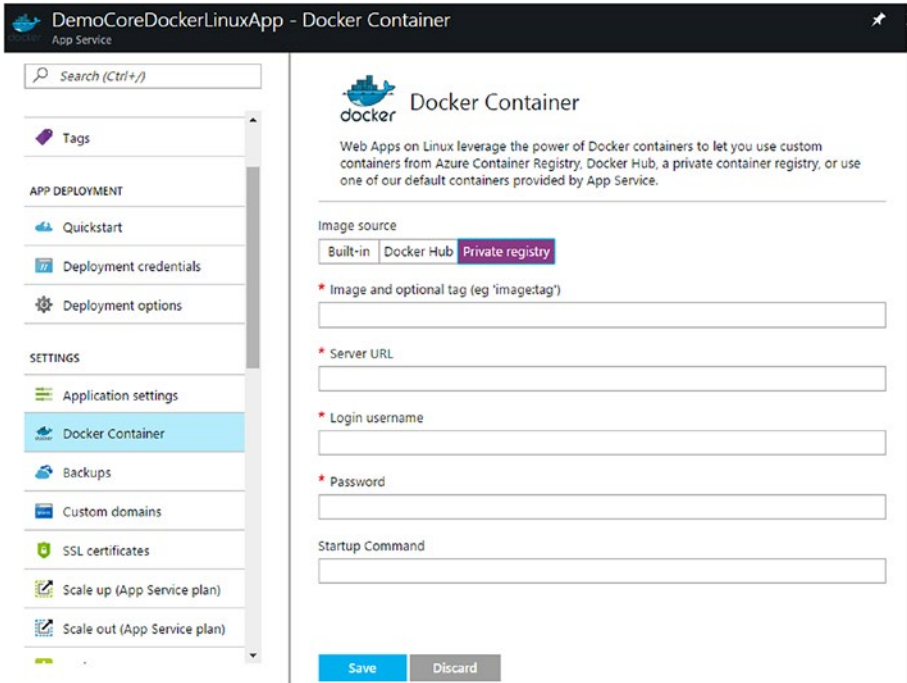


Figure 4-57. Private registry for Docker container in Linux app

3. Set up private registry information as follows.

The Login Server name in the registry should be used to create the Image Source and Server URL, as shown here:

Registry - Login server = demodockerregistry-on.azurecr.io

App - Image and Tag = demodockerregistry-on.azurecr.io/ap
pnameinsourcecode:imagetag

demodockerregistry-on.azurecr.io/coredockerapi:RTM

(RTM tag was used in the build definition and the image was pushed with the tag. We are setting the container to use that image by providing the tag.)

App - Server URL = [http://demodockerregistry-on.
azurecr.io](http://demodockerregistry-on.azurecr.io)

For username and password, copy from the registry form, and save. See Figure 4-58.

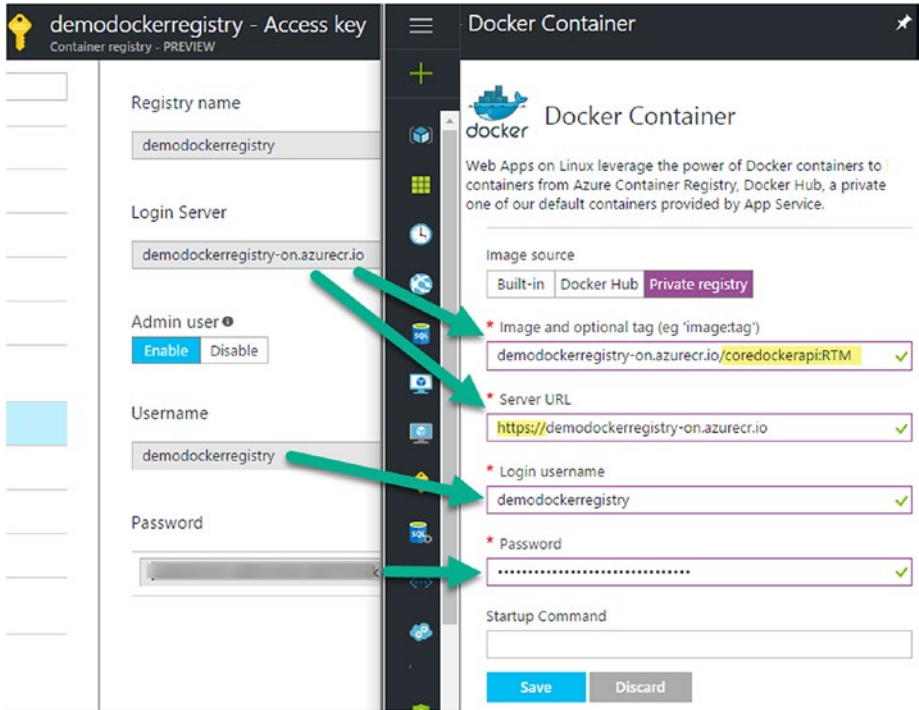


Figure 4-58. Configure private registry as Docker container on Linux app

4. You can now browse the app using the URL of the app service app on Linux plus `api/values`. See the app running in the Docker container and showing as running on Ubuntu in the browser. See Figure 4-59.

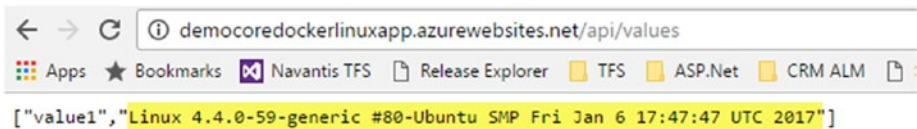


Figure 4-59. ASP.NET Core app running on Ubuntu as Linux app service

5. Let's carry out some changes to the code and get them deployed to the app service app on Linux. To do this, add an additional step to the build definition created previously. Add the Azure App Service Manage task to the build definition. See Figure 4-60.

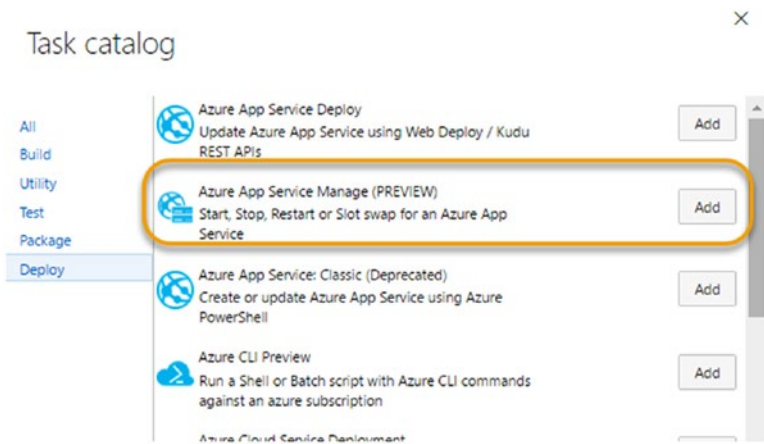


Figure 4-60. Adding App Service Manage task to build definition

6. Select the Azure subscription. Linking the Azure subscription to a Team Services team project is explained in Chapter 3. From the Action dropdown choose “**Restart App Service.**” Select the app service on Linux from the App Service Name dropdown. See Figure 4-61.

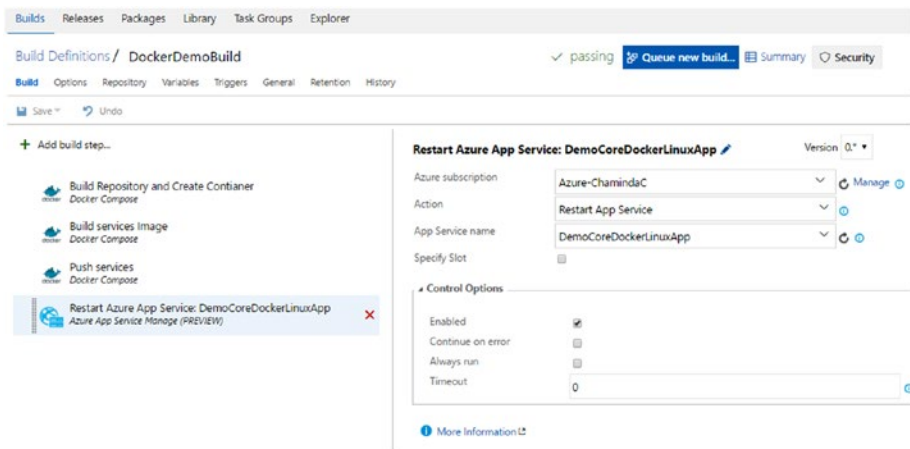


Figure 4-61. Restart Linux app on Azure task

7. In the ValuesController.cs file's Get method, change following line"

```
return new string[] { "value1", System.Runtime.InteropServices.
RuntimeInformation.OSDescription };
```

to

```
return new string[] { "Hello World", System.Runtime.
InteropServices.RuntimeInformation.OSDescription };
```

and commit and sync the code changes to the team project Git repository using the team explorer **Changes** tab. See Figure 4-62.

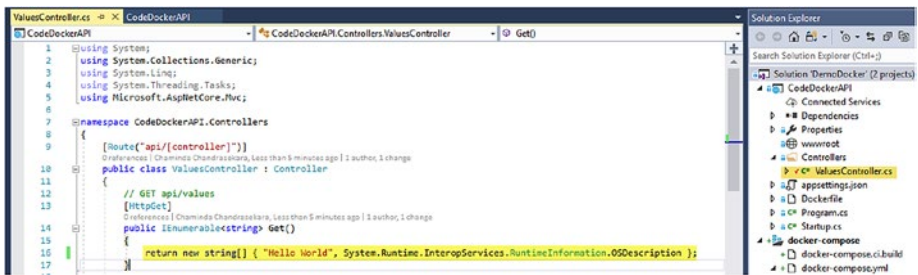


Figure 4-62. Making a code change

8. The build gets kicked in once the code syncs to the Git repository, since we have enabled continuous integration. Refresh the URL of the Azure app service on Linux, and the change will be available. See Figure 4-63.

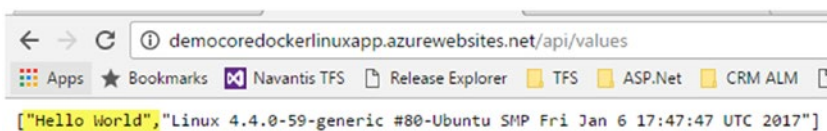


Figure 4-63. Code change available in the Linux app on Azure

You have learned how to deploy a Docker-enabled ASP.NET Core web API to the Azure app service app on Linux using a private container registry in this lesson.

Summary

In this chapter, you have learned how to build a Docker container with a Team Services build and push it to the Azure container registry. Using the Azure app service on Linux, you were able to host the container as a private registry. This is not an ideal production-ready scenario, since we have omitted usage of release management steps here. Deploying the containers to Azure container service (<https://docs.microsoft.com/enus/azure/container-service/container-service-intro>) would be the robust way of hosting Docker containers in Azure. But for learning purposes, this was sufficient. Instead of the Azure container registry, you could use a Docker hub registry with the Team Services builds that use Docker integration extensions.

Azure Container Services and Team Services

You can explore the capabilities of Azure container services to host production systems. Azure container services allow you to deploy and manage containers. For more information, visit the following:

<https://azure.microsoft.com/en-us/services/container-service/>
<https://docs.microsoft.com/en-us/azure/container-service/container-service-intro>
<https://docs.microsoft.com/en-us/azure/container-service/>

Visual Studio 2017 + Visual Studio Team Services offer a great capability to generate build and release pipelines automatically when using the extension <https://marketplace.visualstudio.com/items?itemName=VSIdeDevOpsMSFT.ContinuousDeliveryToolsforVisualStudio> with Visual Studio 2017. You can add extensions directly via Visual Studio 2017 Extensions and Updates. See Figure 4-64.

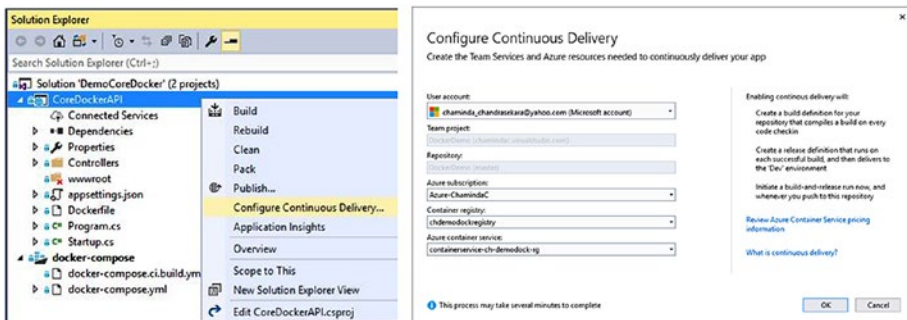


Figure 4-64. Configuring continuous delivery for Azure container services

The build and release definitions will be generated by linking to required Azure resources as well. See Figure 4-65.

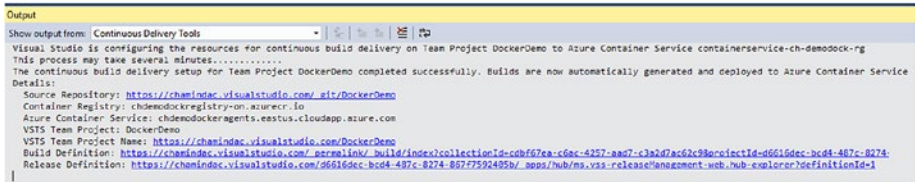


Figure 4-65. Visual Studio 2017 generates the build and release definitions

Build definitions are capable of building Docker containers and pushing them to the Azure container registry. See Figure 4-66.

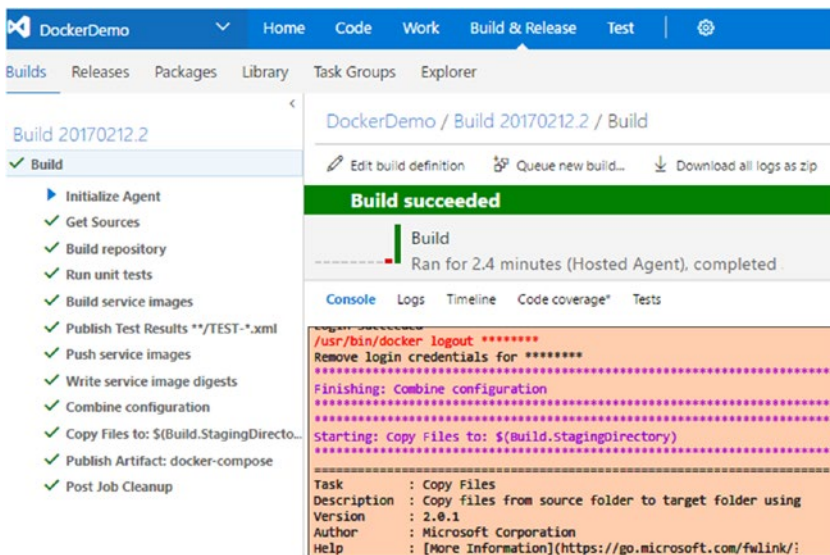


Figure 4-66. Building and pushing to the Azure container registry

A release definition capable of deploying to Azure container services is created as DC/OS (Data Centre OS - allows deploying and scaling clustered workloads while abstracting underlying hardware - for more information visit <https://docs.microsoft.com/en-us/azure/container-service/container-service-mesos-marathon-ui> and <https://docs.microsoft.com/en-us/azure/container-service/container-service-intro#deploying-anapplication>). The capabilities found in the Docker Deploy component (the Docker Integration Marketplace extension adds this release task to Team Services) will be enhanced to support many other types of deployments soon. See Figure 4-67.

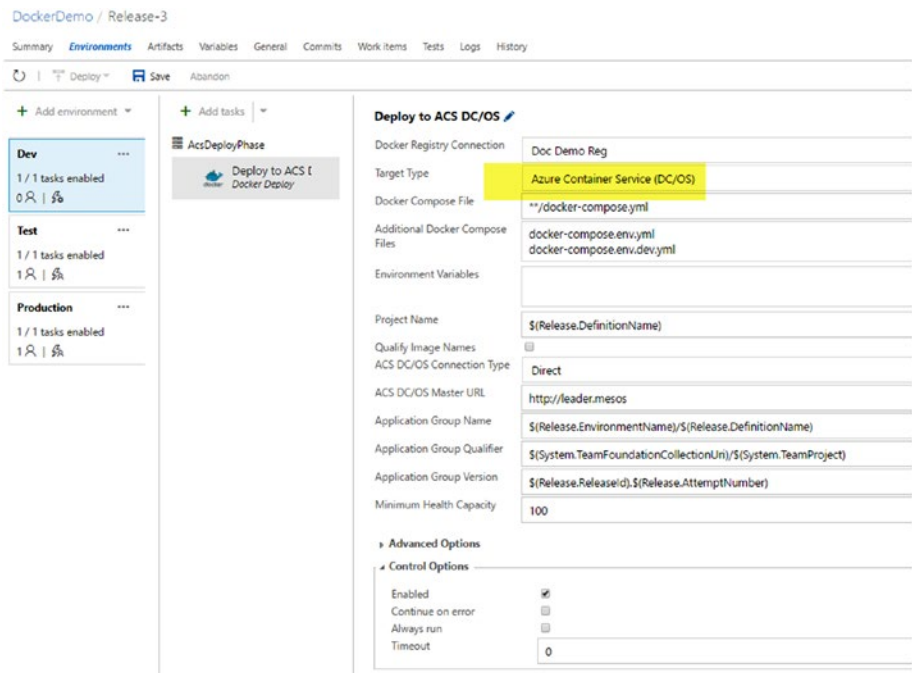


Figure 4-67. A release definition generated by VS 2017 targeting an Azure container service

These features in Azure container services, Azure container registry, and Azure app services apps on Linux with Team Services and Visual Studio show signs of evolving into a rich set of tools and platforms providing an enormous set of capabilities.

In the next chapter, you will be learning about SQL database deployment, targeting Azure SQL databases using Team Services build and release management.