Hands On with Containers for .NET Applications

# Setup

### Hyper-V

1. Go to the Control Panel | Programs and Features.
2. Click **Turn Windows features on or off**.
3. Check **Hyper-V**.
4. Check **Windows Hypervisor Platform**.
5. Click **OK**.

### Docker

1. Go to <https://docs.docker.com/docker-for-windows/install/>.
2. Click **Download from Docker Hub** to install Docker for Windows.
3. Right click Docker Desktop in system tray and select **Settings**.
4. Click **Shared Drives**. Select **C drive**. Click **Apply**.
5. Open PowerShell
6. Type docker --version

### Git

1. Open the Visual Studio Installer.
2. Click **Modify**.
3. Open the Individual Components tab.
4. Enter **Git** in the search box.
5. Check both **Git for Windows** and **GitHub Extension for Visual Studio**.
6. Click **Install while downloading**.

### Visual Studio Container Tools

1. In Visual Studio, click **Extensions | Manage Extensions**.
2. Click Online.
3. Enter container in the search box.
4. Select **Visual Studio Container Tools Extension (Preview)** and click **Download**.

### Azure Command Line Interface (CLI)

1. Go to <https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest>.
2. Click **Install on Windows**.

### Azure

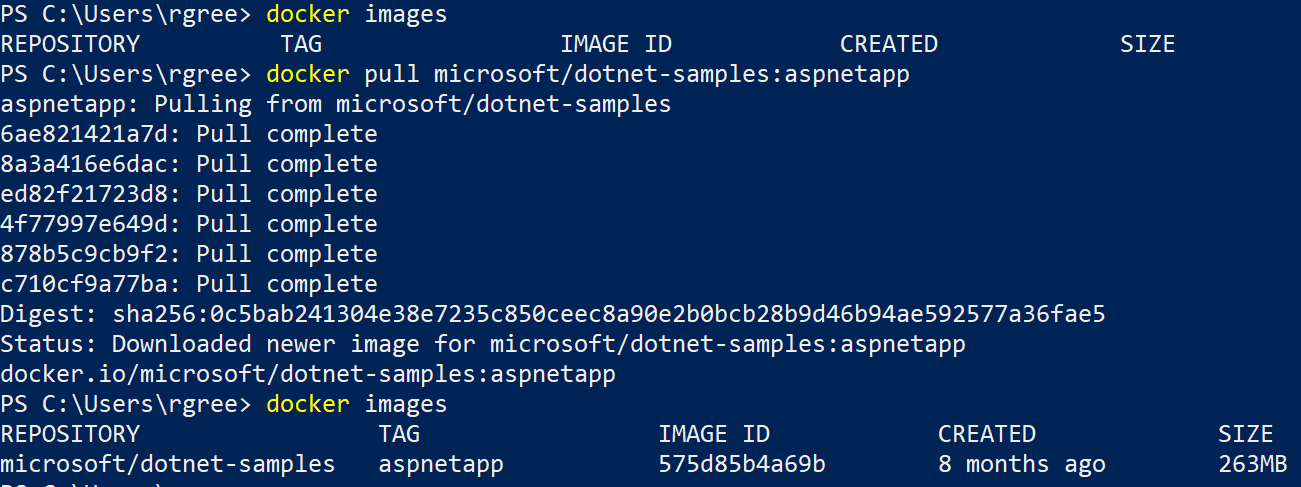
1. To create a new account, go to <https://azure.microsoft.com>.
2. Click **Start free**.

### Azure DevOps

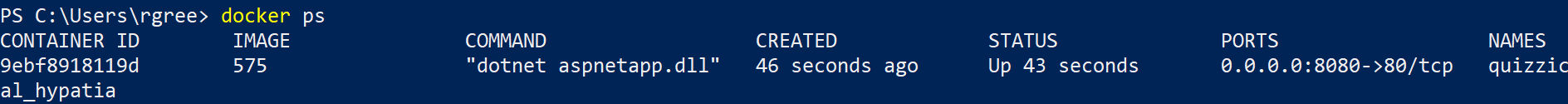
1. Go to <https://azure.microsoft.com/en-us/services/devops/>.
2. Click **Start free**.

# Prebuilt Images and Command Line

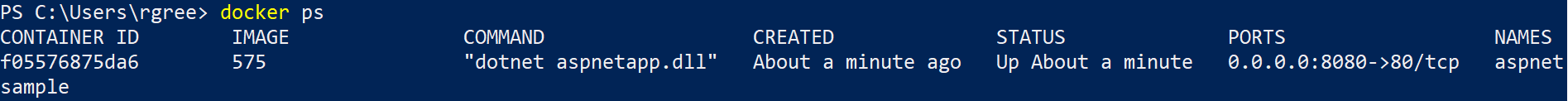
1. Open PowerShell.
2. Type docker images. There are none.
3. Navigate to DockerHub (hub.docker.com)
4. Search for Microsoft/dotnet
5. Scroll down and select **microsoft/dotnet-samples**
6. Copy docker pull microsoft/dotnet-samples:aspnetapp
7. Switch to PowerShell and paste the above.
8. Type docker images



1. Type docker run -p 8080:80 <first 3 characters of IMAGE ID>
2. Launch another instance of PowerShell and type docker ps.



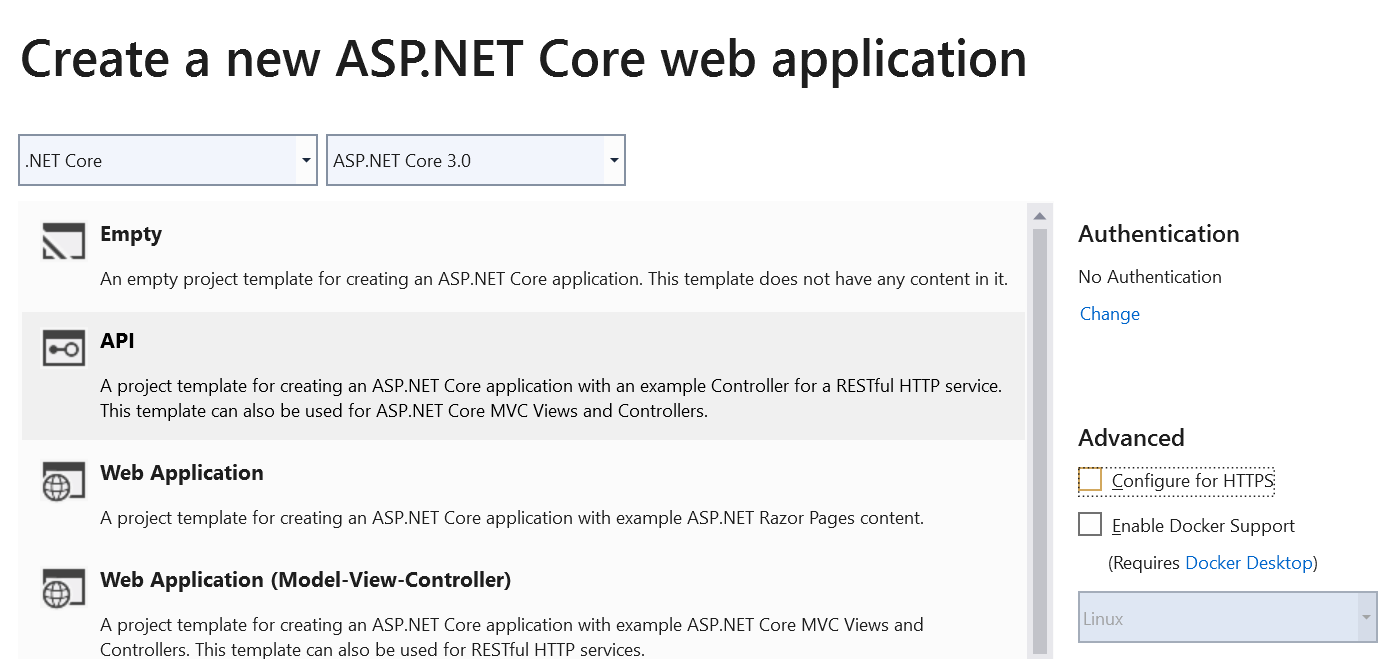
1. In the browser navigate to **localhost:8080**. You should see the sample app.
2. In PowerShell, type docker stop <first 3 characters of CONTAINER ID>
3. Refresh the browser. You should get an error.
4. In PowerShell, type docker run -p 8080:80 --name aspnetsample <first 3 characters of IMAGE ID>.
5. Refresh the browser. You should see the sample app.
6. Switch to the other instance of PowerShell. Type docker ps. Note the container has the name aspnetsample instead of a randomly assigned name.



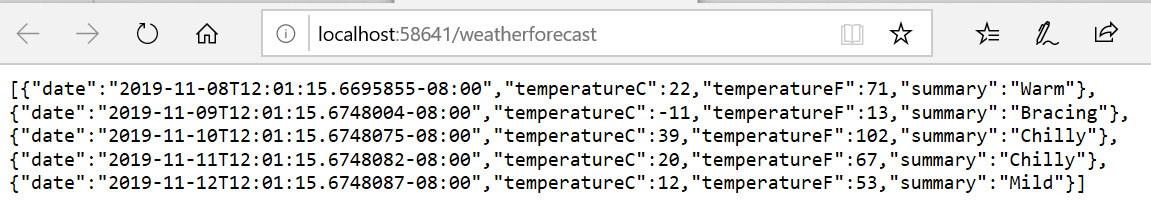
1. You can use this name to start and stop the container. Type docker stop aspnetsample. Then type docker start aspnetsample.
2. Type docker stop aspnetsample.
3. Type docker run -p 8000:80 --name aspnetsample <first 3 characters of IMAGE ID>. You will get an error that the container name is already in use.
4. Type docker rm aspnetsample
5. Type docker run -p 8000:80 --name aspnetsample <first 3 characters of IMAGE ID>.
6. Type docker stop aspnetsample
7. Type docker rm aspnetsample
8. Type docker images.
9. To delete the image, type docker rmi < first 3 characters of IMAGE ID >. Add –force if necessary.
10. Type docker images. There are no images.

# Create Images from App in Visual Studio

1. In Visual Studio, create a new ASP.NET Core Web Application named WeatherAPI. Choose the API template. Uncheck Configure for HTTPS. Do not check Enable Docker Support.



1. Run the app in IIS Express.



1. Stop the app.
2. Right-click the project and select **Add | Docker Support**. Choose Linux when prompted and click **OK**.
3. Visual Studio creates Dockerfile and builds a container.

========== Checking for Container Prerequisites ==========

Verifying that Docker Desktop is installed...

Docker Desktop is installed.

========== Verifying that Docker Desktop is running... ==========

Verifying that Docker Desktop is running...

Docker Desktop is running.

========== Verifying Docker OS ==========

Verifying that Docker Desktop's operating system mode matches the project's target operating system...

Docker Desktop's operating system mode matches the project's target operating system.

========== Pulling Required Images ==========

Checking for missing Docker images...

Pulling missing Docker images. To cancel this download, close the command prompt window.

docker pull mcr.microsoft.com/dotnet/core/aspnet:3.0-buster-slim

Docker pull completed.

Docker images are ready.

========== Warming up container(s) for WeatherAPI ==========

Starting up container(s)...

docker build -f "C:\Users\rgree\source\repos\WeatherAPI\WeatherAPI\Dockerfile" --force-rm -t weatherapi:dev --target base --label "com.microsoft.created-by=visual-studio" --label "com.microsoft.visual-studio.project-name=WeatherAPI" "C:\Users\rgree\source\repos\WeatherAPI"

Sending build context to Docker daemon 18.43kB

Step 1/5 : FROM mcr.microsoft.com/dotnet/core/aspnet:3.0-buster-slim AS base

---> 930743cb4e19

Step 2/5 : WORKDIR /app

---> Running in 676947ec2e64

Removing intermediate container 676947ec2e64

---> 955cae669f5e

Step 3/5 : EXPOSE 80

---> Running in 3b8de42d8ea5

Removing intermediate container 3b8de42d8ea5

---> 1706aaa1b4d0

Step 4/5 : LABEL com.microsoft.created-by=visual-studio

---> Running in 6e730903ee80

Removing intermediate container 6e730903ee80

---> 83b0ea1b293b

Step 5/5 : LABEL com.microsoft.visual-studio.project-name=WeatherAPI

---> Running in af0f925944ca

Removing intermediate container af0f925944ca

---> 25c6e534f4a8

Successfully built 25c6e534f4a8

Successfully tagged weatherapi:dev

SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build context will have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions for sensitive files and directories.

docker run -dt -v "C:\Users\rgree\vsdbg\vs2017u5:/remote\_debugger:rw" -v "C:\Users\rgree\source\repos\WeatherAPI\WeatherAPI:/app" -v "C:\Users\rgree\source\repos\WeatherAPI:/src" -v "C:\Users\rgree\.nuget\packages\:/root/.nuget/fallbackpackages3" -v "C:\Microsoft\Xamarin\NuGet\:/root/.nuget/fallbackpackages" -v "C:\Program Files\dotnet\sdk\NuGetFallbackFolder:/root/.nuget/fallbackpackages2" -e "DOTNET\_USE\_POLLING\_FILE\_WATCHER=1" -e "ASPNETCORE\_ENVIRONMENT=Development" -e "NUGET\_PACKAGES=/root/.nuget/fallbackpackages3" -e "NUGET\_FALLBACK\_PACKAGES=/root/.nuget/fallbackpackages;/root/.nuget/fallbackpackages2;/root/.nuget/fallbackpackages3" -p 63364:80 --entrypoint tail weatherapi:dev -f /dev/null

2392098f89ac9ef78994c6350d90eec10ac4782d4f8334d62c8907d7626c0c36

Container started successfully.

========== Finished ==========

1. Explore Dockerfile, which contains instructions on how to build a container that will run the app.

FROM mcr.microsoft.com/dotnet/core/aspnet:3.0-buster-slim AS base

WORKDIR /app

EXPOSE 80

FROM mcr.microsoft.com/dotnet/core/sdk:3.0-buster AS build

WORKDIR /src

COPY ["WeatherAPI/WeatherAPI.csproj", "WeatherAPI/"]

RUN dotnet restore "WeatherAPI/WeatherAPI.csproj"

COPY . .

WORKDIR "/src/WeatherAPI"

RUN dotnet build "WeatherAPI.csproj" -c Release -o /app/build

FROM build AS publish

RUN dotnet publish "WeatherAPI.csproj" -c Release -o /app/publish

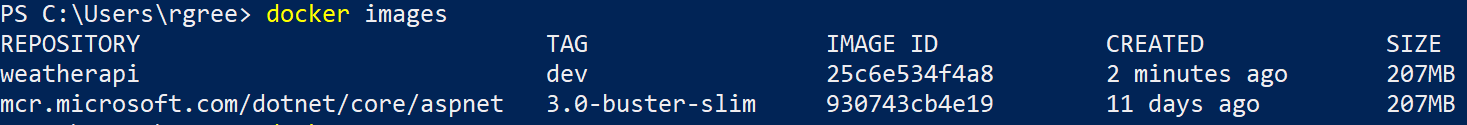
FROM base AS final

WORKDIR /app

COPY --from=publish /app/publish .

ENTRYPOINT ["dotnet", "WeatherAPI.dll"]

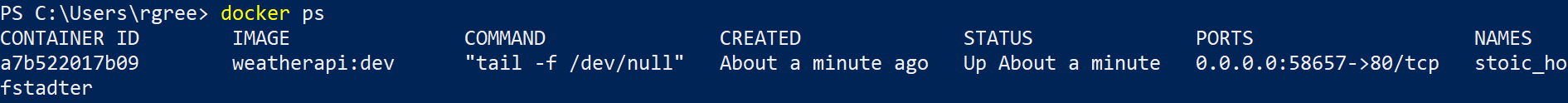
1. Open PowerShell
2. Type docker images



1. Docker downloaded aspnet:3.0-buster-slim from MCR and then tagged it weatherapi:dev.

When Docker support is added, and again when solution is opened, Visual Studio “warms up” a container with volumes mounted but the app is not running.

1. Type docker ps



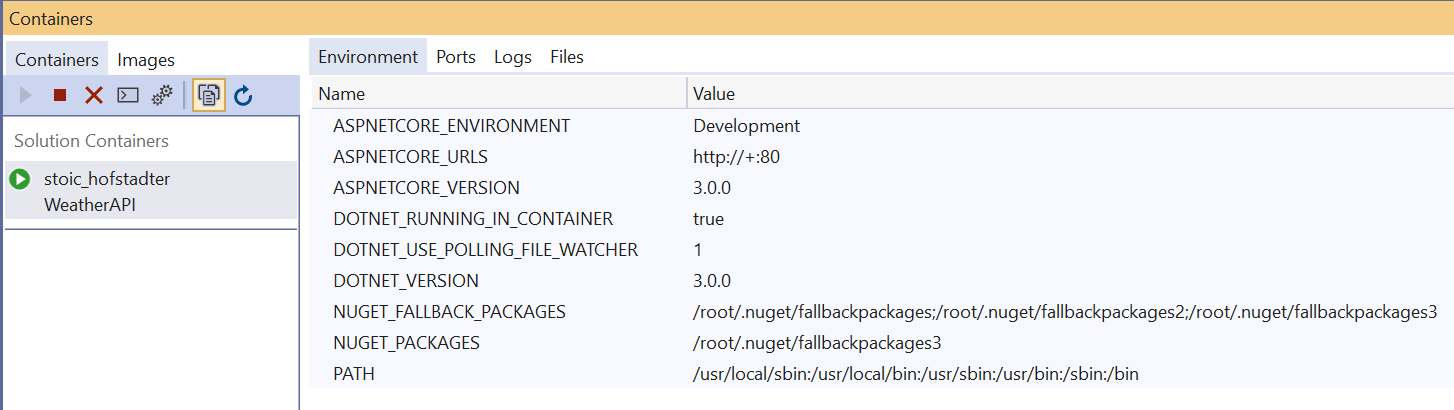
1. There is a running container on a TCP port. The container is based on the weatherapi image and is tagged dev.
2. In the browser, navigate to **localhost:<PORT NUMBER>/weatherforecast**. You should get an error that you can’t reach the page.

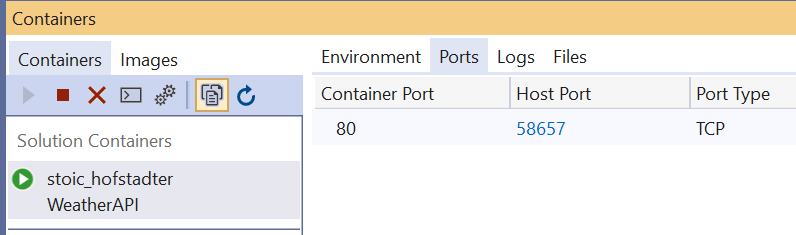
All Docker did was create container based on the base image (aspnet:3.0-buster-slim). Debug does not actually copy the app to the container. Visual Studio maps a volume when it runs. If you build with Release, it does copy the app to the container. You can see this in the Output window. Visual Studio only runs first 3 lines. Creates empty container and volume maps.

1. Click the **Docker Run** button. You should see the weather forecast.

On F5 Visual Studio builds the app on the host and then starts the app using a Docker EXEC command.

1. The Visual Studio Containers window shows the container running.





1. In Visual Studio, open WeatherForecastController.cs.
2. Put breakpoint in the Get method.
3. Refresh the browser.
4. Close the browser.
5. Reopen the browser and navigate to **localhost:< PORT NUMBER >/weatherforecast**.
6. Stop the app in Visual Studio.
7. Refresh the browser. You can’t reach the page.

When you stop debugging, Visual Studio stops the executable so the app is not available.

1. Change from Debug to Release and Ctrl+F5 to run the app.

1>------ Build started: Project: WeatherAPI, Configuration: Release Any CPU ------

1>docker exec -i 23556e3221c82a1678af8e015c199eb9c84948aeaef0e0f9e0296c4628ef0807 /bin/sh -c "if PID=$(pidof dotnet); then kill $PID; fi"

1>WeatherAPI -> C:\Users\rgree\source\repos\WeatherAPI\WeatherAPI\bin\Release\netcoreapp3.0\WeatherAPI.dll

1>docker build -f "C:\Users\rgree\source\repos\WeatherAPI\WeatherAPI\Dockerfile" --force-rm -t weatherapi --label "com.microsoft.created-by=visual-studio" --label "com.microsoft.visual-studio.project-name=WeatherAPI" "C:\Users\rgree\source\repos\WeatherAPI"

1>Sending build context to Docker daemon 18.43kB

1>

1>Step 1/18 : FROM mcr.microsoft.com/dotnet/core/aspnet:3.0-buster-slim AS base

1>Step 2/18 : WORKDIR /app

1> ---> 930743cb4e19

1> ---> Using cache

1> ---> d01f64f44bf8

1>Step 3/18 : EXPOSE 80

1> ---> Using cache

1>Step 4/18 : FROM mcr.microsoft.com/dotnet/core/sdk:3.0-buster AS build

1> ---> f97a49b2a965

1> ---> 170a7f2ec51a

1>Step 5/18 : WORKDIR /src

1> ---> Using cache

1> ---> e38322ef22f1

1>Step 6/18 : COPY ["WeatherAPI/WeatherAPI.csproj", "WeatherAPI/"]

1> ---> Using cache

1> ---> 2aae9b7fc605

1>Step 7/18 : RUN dotnet restore "WeatherAPI/WeatherAPI.csproj"

1> ---> Using cache

1>Step 8/18 : COPY . .

1> ---> 70758a4f9668

1> ---> Using cache

1> ---> 8681220422c5

1>Step 9/18 : WORKDIR "/src/WeatherAPI"

1> ---> Using cache

1>Step 10/18 : RUN dotnet build "WeatherAPI.csproj" -c Release -o /app/build

1> ---> db53b6dc7adc

1> ---> Using cache

1> ---> 60500ddc0de9

1>Step 11/18 : FROM build AS publish

1> ---> 60500ddc0de9

1>Step 12/18 : RUN dotnet publish "WeatherAPI.csproj" -c Release -o /app/publish

1> ---> Using cache

1> ---> 45807591edc5

1>Step 13/18 : FROM base AS final

1> ---> f97a49b2a965

1>Step 14/18 : WORKDIR /app

1> ---> Using cache

1> ---> d7e99a759a09

1>Step 15/18 : COPY --from=publish /app/publish .

1> ---> Using cache

1> ---> c8c91b959153

1>Step 16/18 : ENTRYPOINT ["dotnet", "WeatherAPI.dll"]

1> ---> Using cache

1> ---> cb282841d88c

1>Step 17/18 : LABEL com.microsoft.created-by=visual-studio

1> ---> Using cache

1> ---> 24b05e2c7594

1>Step 18/18 : LABEL com.microsoft.visual-studio.project-name=WeatherAPI

1> ---> Using cache

1> ---> 90b89fd22267

1>Successfully built 90b89fd22267

1>Successfully tagged weatherapi:latest

1>SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build context will have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions for sensitive files and directories.

1>docker rm -f 23556e3221c82a1678af8e015c199eb9c84948aeaef0e0f9e0296c4628ef0807

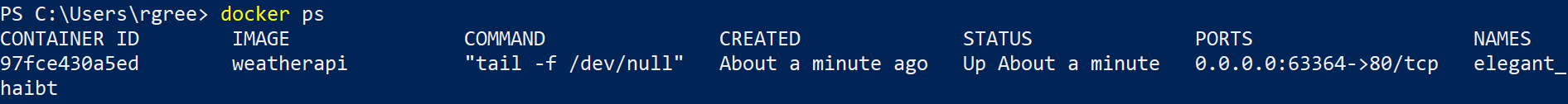
1>23556e3221c82a1678af8e015c199eb9c84948aeaef0e0f9e0296c4628ef0807

1>docker run -dt -v "C:\Users\rgree\vsdbg\vs2017u5:/remote\_debugger:rw" -e "ASPNETCORE\_ENVIRONMENT=Development" -p 51421:80 --entrypoint tail weatherapi -f /dev/null

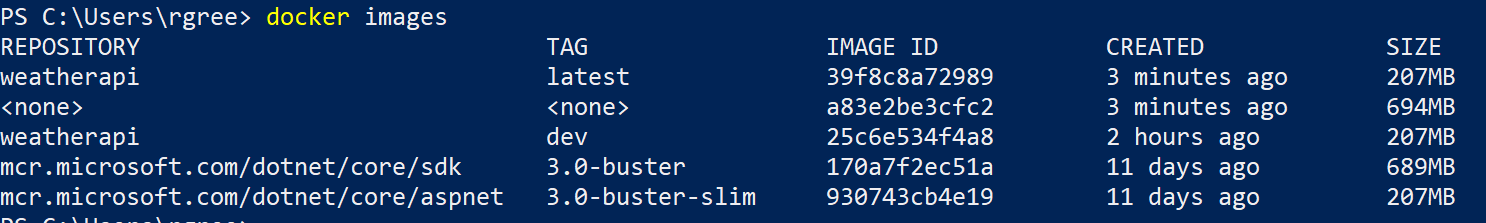
1>038420556b53979b1e2de53eff152f33621f252098ff573808b0fd25abddb2df

========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========

1. The app appears in the browser. Copy the URL.
2. Enter docker ps. Note a different container is running. It is based on the weatherapi image, however it is not tagged dev. As you will see in a moment, this is a different version of the weatherapi image.

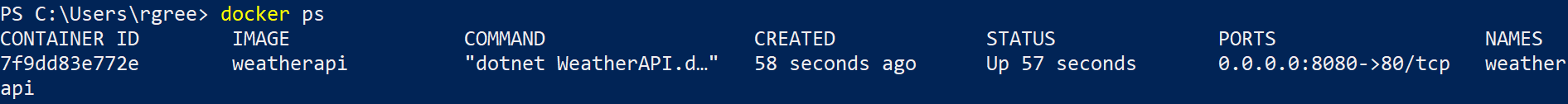


1. Type docker images

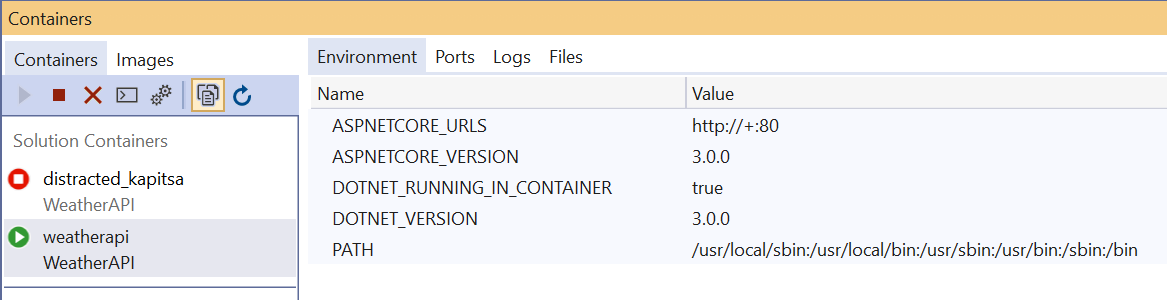


See <https://docs.microsoft.com/en-us/visualstudio/containers/container-build?view=vs-2019> for a good explanation of the multi-stage build.

1. Open a new browser window and paste the URL. It works.
2. Enter docker stop <first 3 characters of CONTAINER ID> to stop the container.
3. In the browser, the app stops running.
4. In Visual Studio, the Containers window shows the container is not running.
5. Enter docker run -p 8080:80 --name weatherapi weatherapi
6. Open a new instance of PowerShell and enter docker ps.



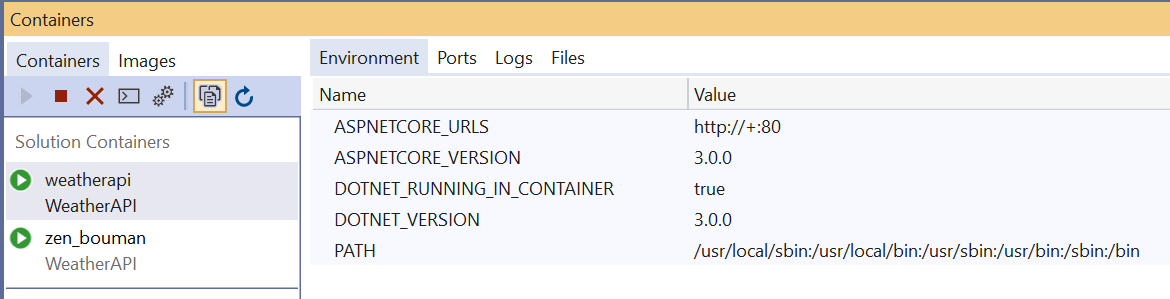
1. In the browser, navigate to **http://localhost:8080/weatherforecast**. The weather forecast appears.
2. In Visual Studio, the Containers window shows the container running.



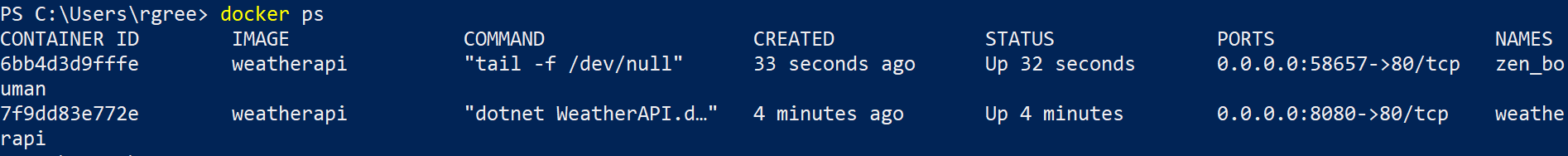
1. In Visual Studio, right click on DockerFile and select **Build Docker Image**.
2. In the Output window, see the docker build command used.

docker build -f "c:\users\rgree\source\repos\weatherapi\weatherapi\dockerfile" --force-rm -t weatherapi --label "com.microsoft.created-by=visual-studio" --label "com.microsoft.visual-studio.project-name=WeatherAPI" "c:\users\rgree\source\repos\weatherapi"

1. Go to C:\Users\<<YourName>>\source\repos\WeatherAPI and enter docker build -f "c:\users\< <YourName>>\source\repos\weatherapi\weatherapi\dockerfile " -t weatherapi.. <Notice the Docker build command ends in a dot>. This builds the image by hand.
2. Enter Docker images.
3. To see Visual Studio warming up the container in action, quit Visual Studio. Then start it again and load the WeatherAPI solution.
4. Look in the Container Tools tab of the Output window.
5. Look in the Containers tool window.



1. Type docker ps.

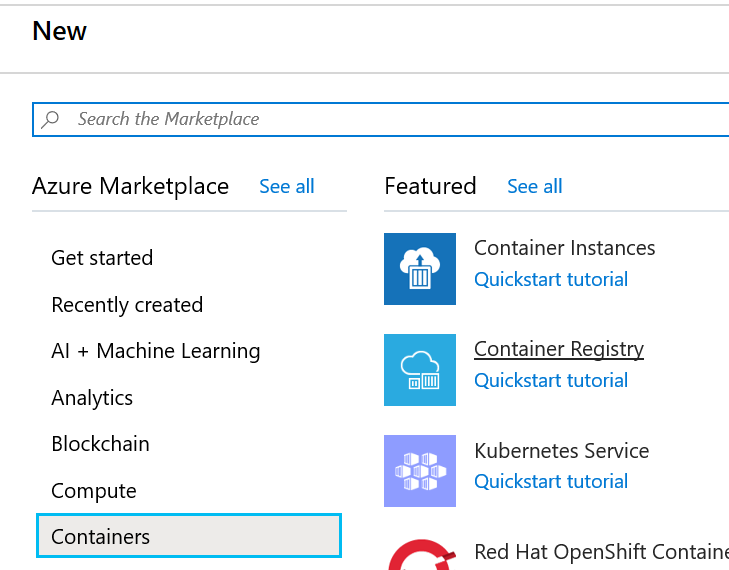


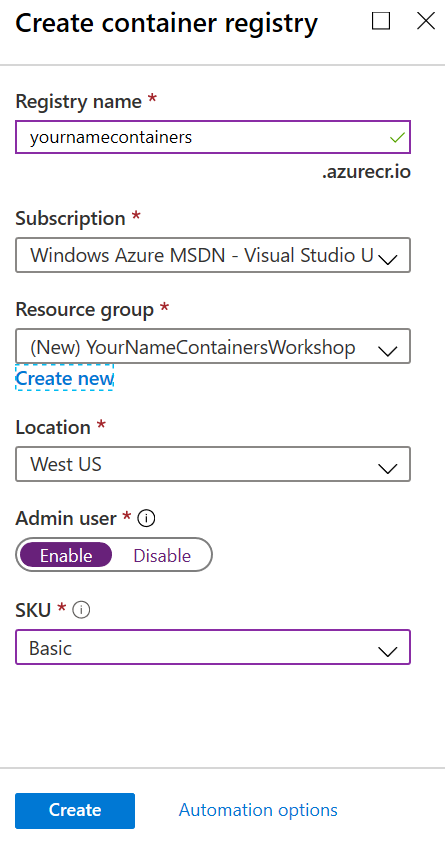
1. Stop all running containers.

# Push the image to Azure Container Registry

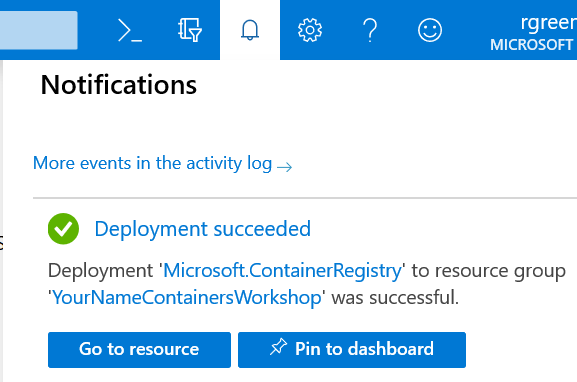
## Manual

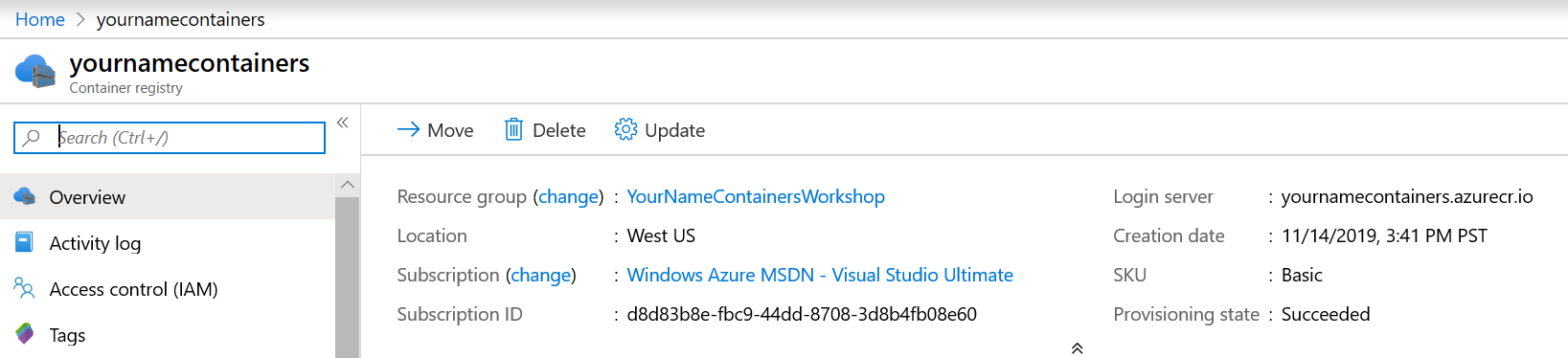
1. In the Azure Portal, click **Create a Resource**.
2. Select **Containers** in the list on the left and click **Container Registry**.





1. Enter **<YourName>containers** as the Registry name. The Registry name needs to be unique, so replace <YourName> with something you will remember.
2. Click **Create** **new** to create a new resource group.
3. Name the new resource group **<YourName>ContainersWorkshop**.
4. **Enable** Admin user.
5. Select **Basic** in the SKU drop-down list.
6. Click **Create**.
7. When the deployment succeeds, click **Go to Resource**. If necessary, click the notifications icon to see this option.





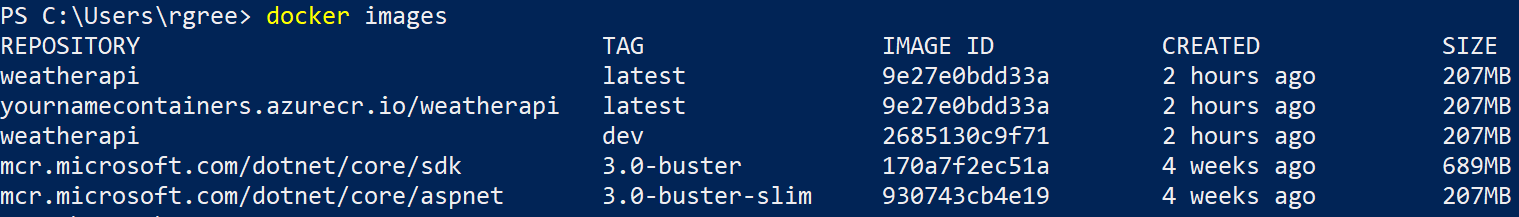
1. In PowerShell type az login to login to Azure. You will be prompted to login to Azure.

If you have more than one Azure subscription, they will be displayed. Enter az account set --subscription <subscriptionid> to use the same subscription you used to create the Azure Container Registry.

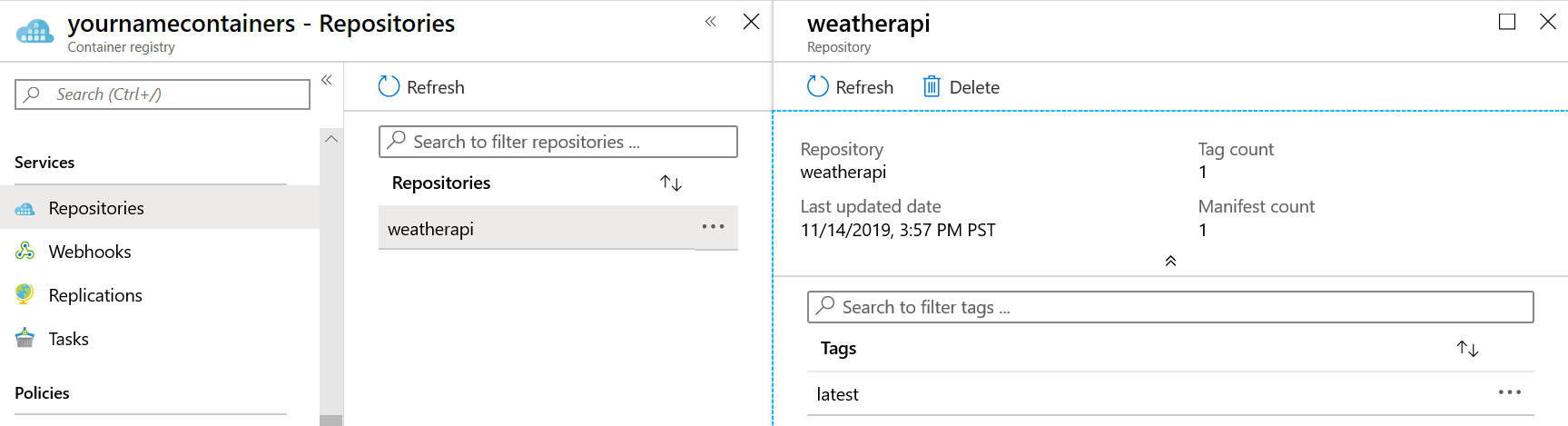
1. Type az acr login --name <YourName>containers to login to the container registry.
2. Type docker tag weatherapi <YourName>containers.azurecr.io/weatherapi.

NOTE: Use all lower case for the tag name.

1. Type docker images

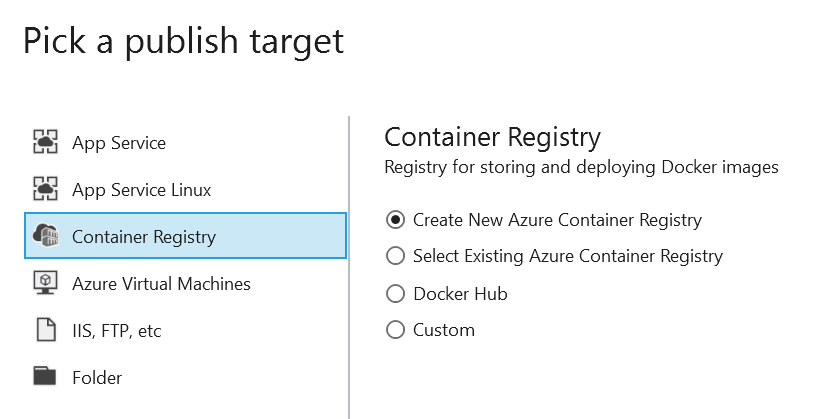


1. Type docker push <YourName>containers.azurecr.io/weatherapi. This will copy the container image to the container registry.
2. In the Azure portal click on Repositories on the left-hand navigation.
3. Click on weatherapi.

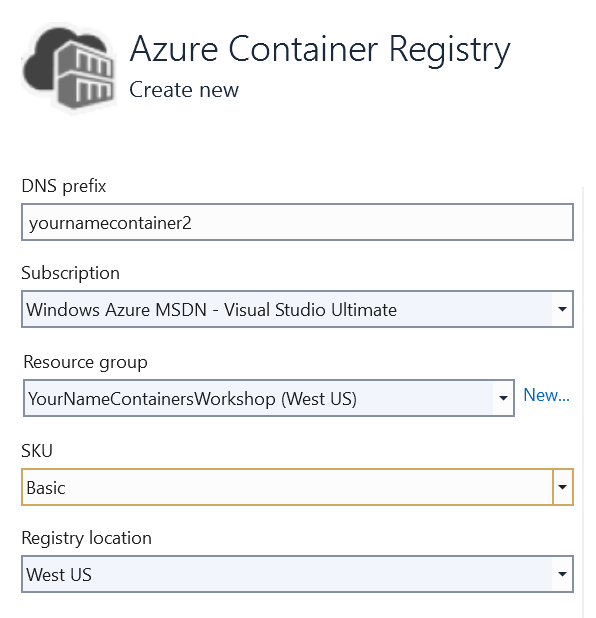


## Visual Studio

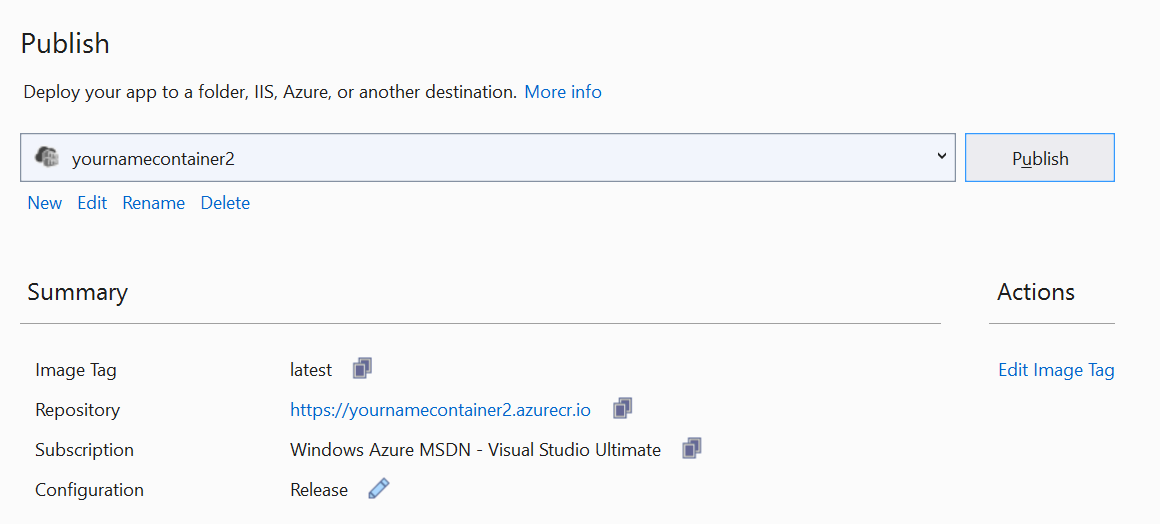
1. Right click the WeatherAPI project and select **Publish**.
2. Click **Container Registry** and then **Create New Azure Container Registry**.



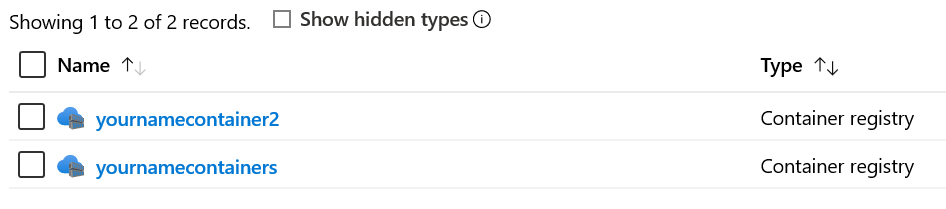
1. Click **Create Profile**.
2. Enter **<YourName>container2** as the DNS Prefix.
3. Select **<YourName>ContainersWorkshop** from the Resource group drop-down.
4. Select **Basic** from SKU drop-down.



1. Click **Create**.
2. When Visual Studio displays the Publish page, click **Publish**.



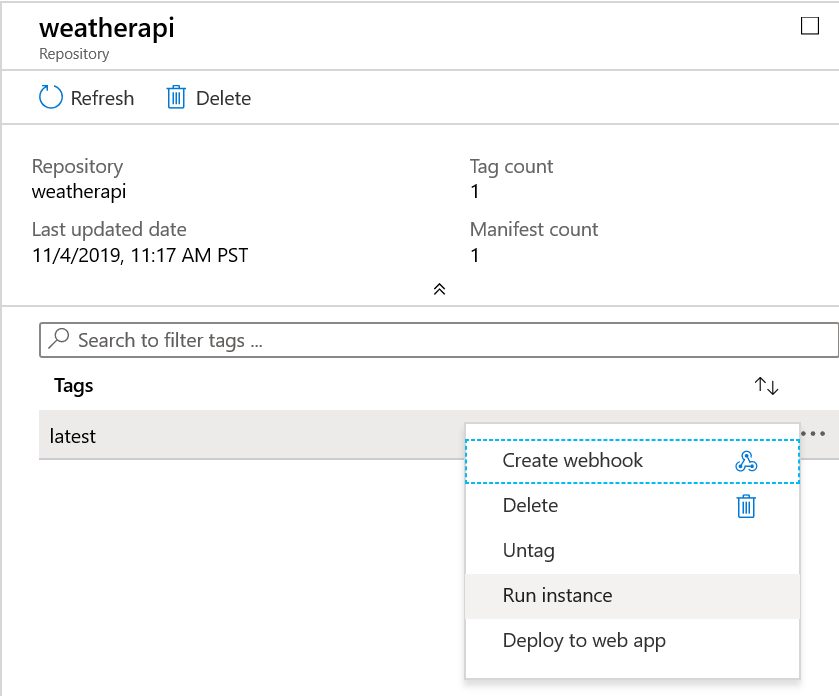
1. Return to the Azure portal.
2. Click **Home** to return to the home view.
3. Click the Resource groups icon to display your resource groups.
4. Click **<YourName>ContainersWorkshop**.
5. Confirm you now have 2 container registries.



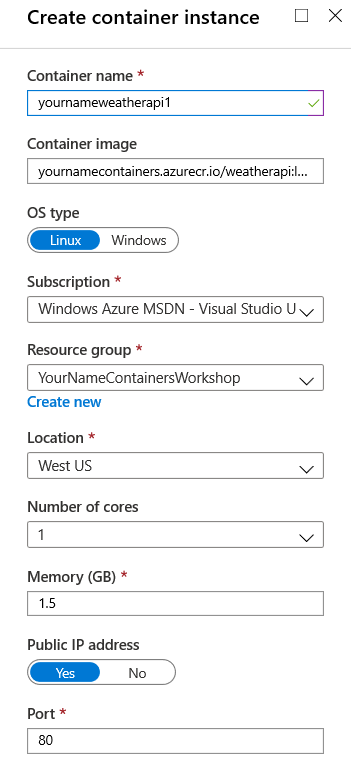
1. Click **<YourName>containers2**.
2. Click **Repositories** and confirm the container has been pushed to the registry.

# Deploy to Azure Container Instance

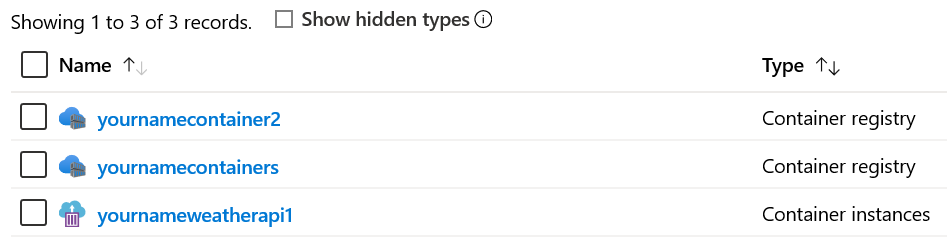
1. In the Azure Portal, navigate back up to the resource group.
2. Click the first container registry you created.
3. Click **Repositories** and then click **weatherapi**.
4. Click the ellipsis to the right of **latest** and select **Run instance**.

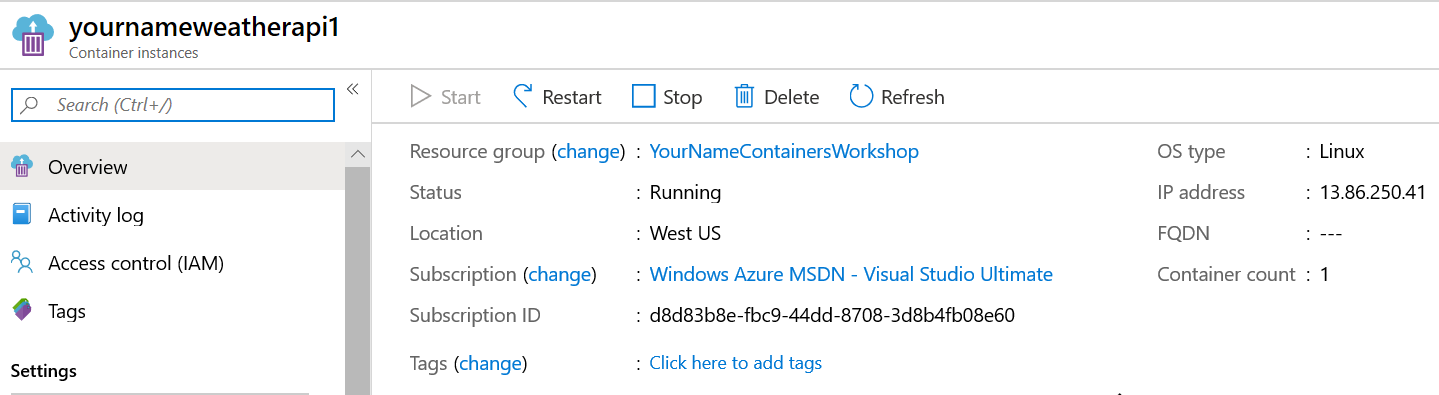


1. Enter **<YourName>weatherapi1** as the Container name.



1. Click **OK** to deploy the container to an Azure Container Instance.
2. When deployment is complete, navigate to the container instance, either from the deployment succeeded message or via the resource group page.



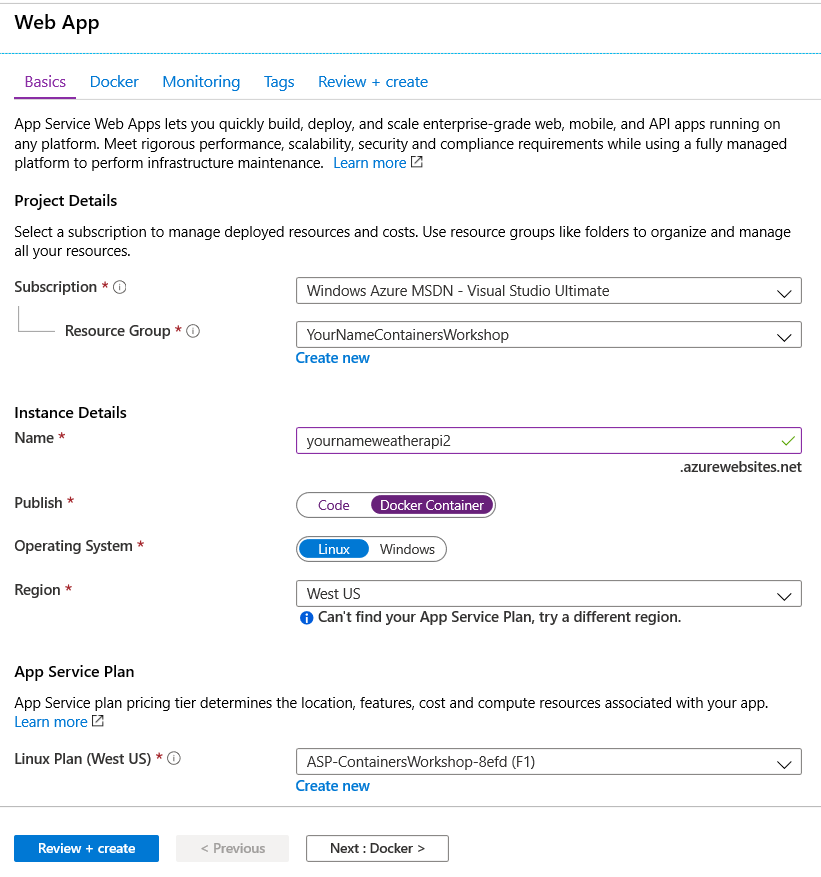


1. Open the browser and navigate to **<IP address>/weatherforecast**.
2. In the Azure portal, click **Stop** to stop the instance. Confirm you can’t reach it in the browser. Note this may take a few seconds.
3. Click **Start** to start the instance. Note you likely have a different IP address. Confirm you can reach the instance in the browser.
4. Stop the instance.

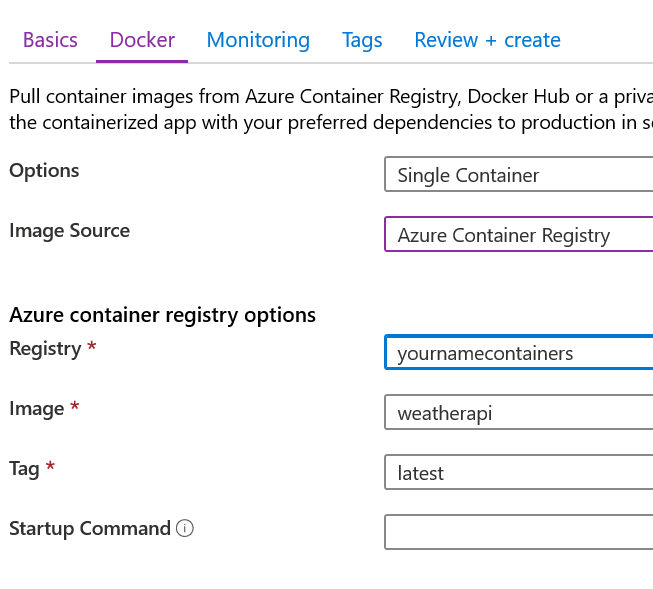
# Deploy to Web App for Containers

## Manual 1

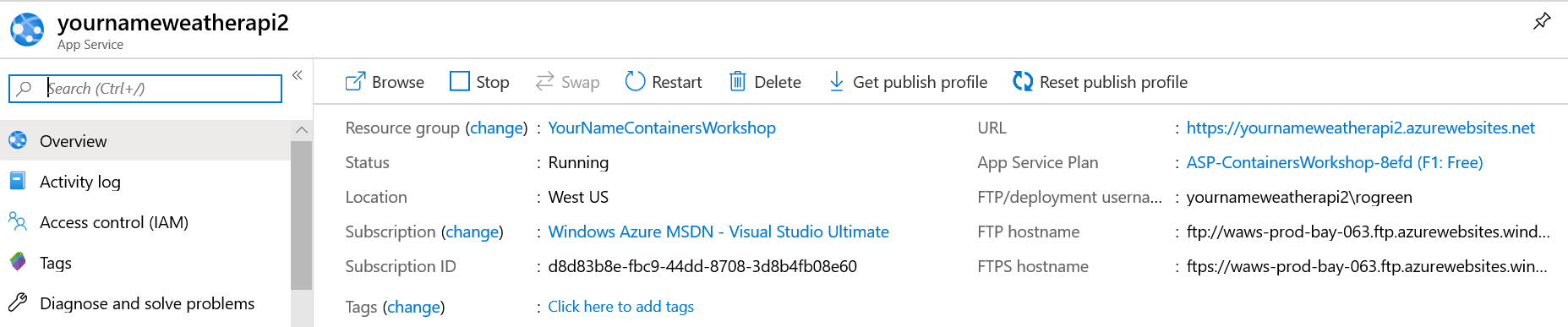
1. In the Azure portal, click **Home** to return to the home page.
2. Click **Create a resource**.
3. Select **Web App**.
4. In the Basics tab, select **<YourName>ContainersWorkshop** from the Resource Group drop-down.
5. Enter **<YourName>weatherapi2** as the instance name.
6. Select Docker Container.
7. Set the appropriate region.
8. The App Service Plan drop-down should show an app plan you created earlier.



1. Click **Next: Docker**.
2. In the Docker tab, select **Azure Container Registry** from the Image Source drop-down.
3. Select **<YourName>containers** from the Registry drop-down.
4. The Image is automatically set to **weatherapi** and the Tag is automatically set to **latest**.



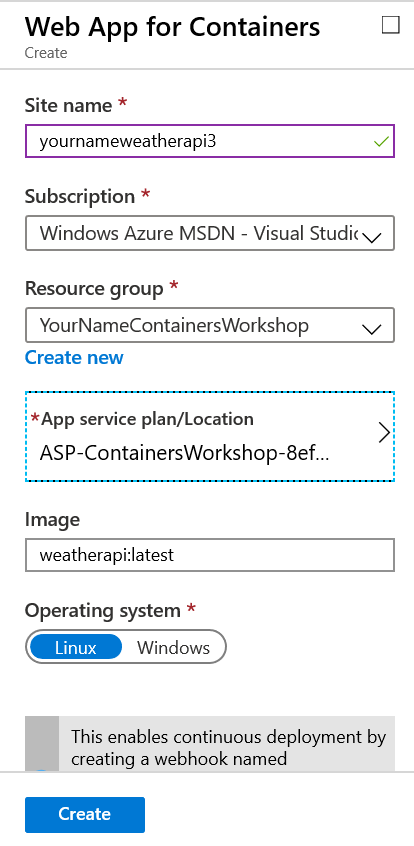
1. Scroll through the remaining tabs then click **Review + create**.
2. When you get to the Review + create tab, click **Create**.
3. When the deployment has completed, go to the resource.



1. Click the URL to launch the Web app. You should see an error.
2. Add /weatherforecast to the URL to see the API.

## Manual 2

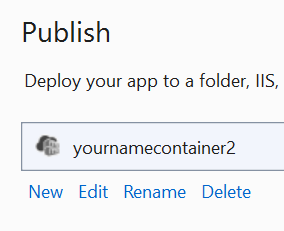
1. In the Azure portal, navigate to the **<YourName>containers2** container registry.
2. Drill down to the weatherapi repository.
3. Click the ellipsis to the right of latest and select **Deploy to web app**.
4. Enter **<YourName>weatherapi3** as the Site name.
5. Click **App Service Plan/Location** and select the Containers Workshop service plan you used previously.



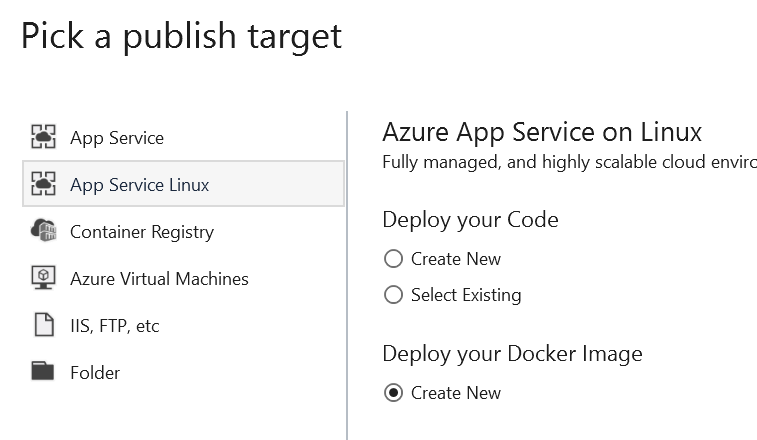
1. Click **Create**.
2. When the deployment has completed, go to the resource.
3. Click the URL to launch the Web app.
4. Add /weatherforecast to the URL to see the API.

## Visual Studio

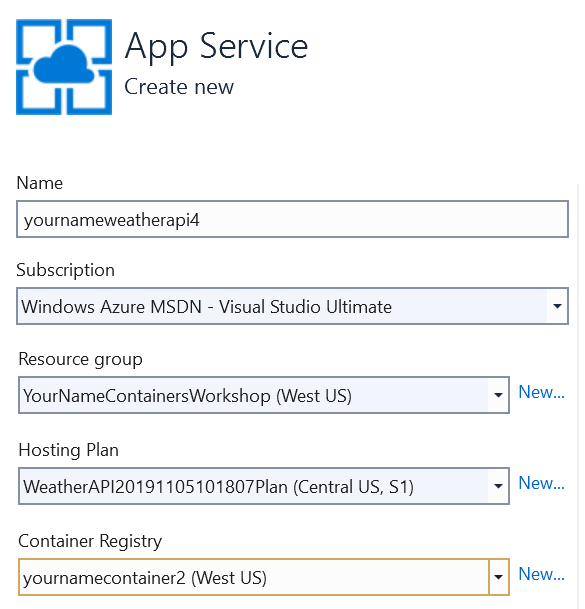
1. In Visual Studio, right-click the WeatherAPI project and select **Publish**.
2. Click **New** under the container registry name.



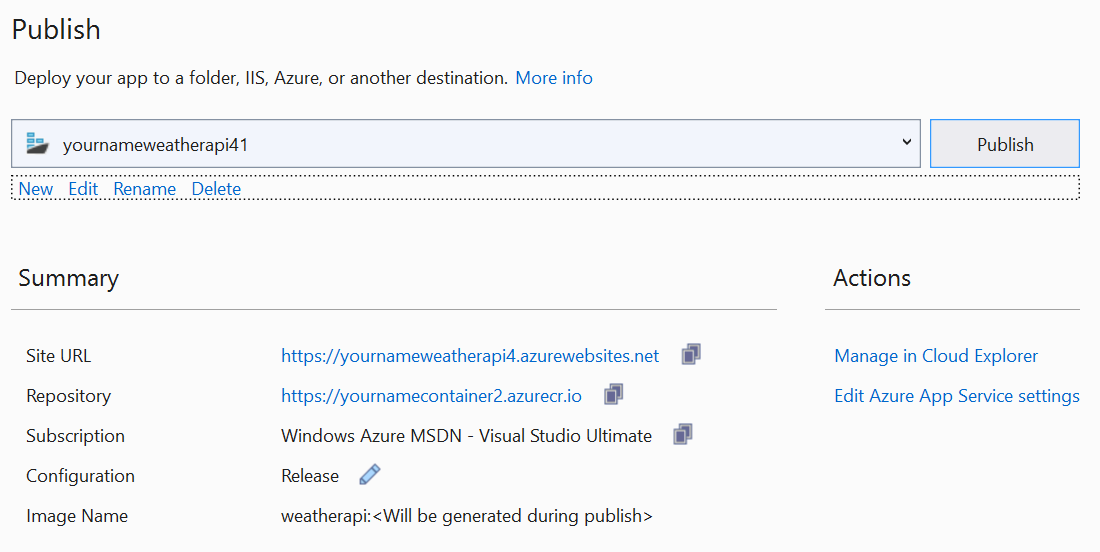
1. Select **App Service Linux**.
2. Select **Create New** under Deploy your Docker Image.



1. Click **Create Profile**.
2. Set the Name to **<YourName>weatherapi4**.
3. Select **<YourName>ContainersWorkshop** from the Resource group drop-down.
4. Select **<YourName>containers2** from the Container Registry drop-down.



1. Click **Create**.

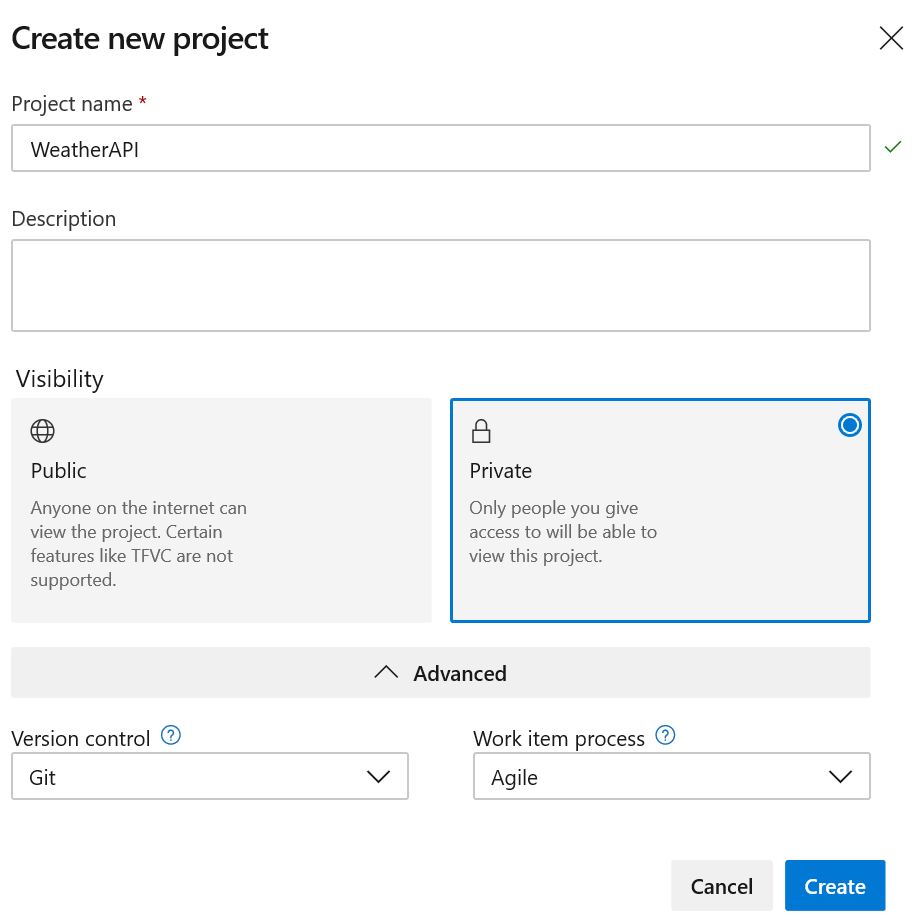


1. Click **Publish**.
2. When the Web app is published, click on the Site URL to launch the browser.
3. Add /weatherforecast to the URL to see the API.

# DevOps

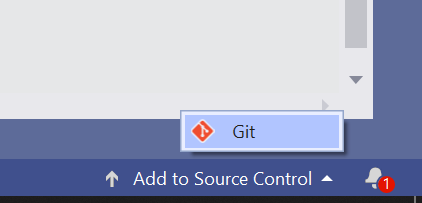
## Setup

1. Sign into the Azure DevOps portal.
2. Create a new project named **WeatherAPI**.
3. Choose **Git** and **Agile** under Advanced.
4. Click **Create**.

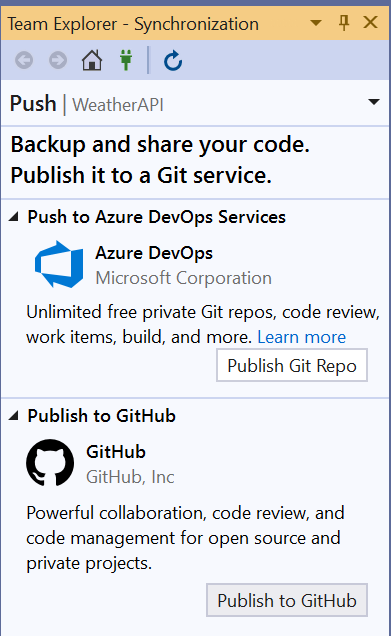


## Add Project to Source Control

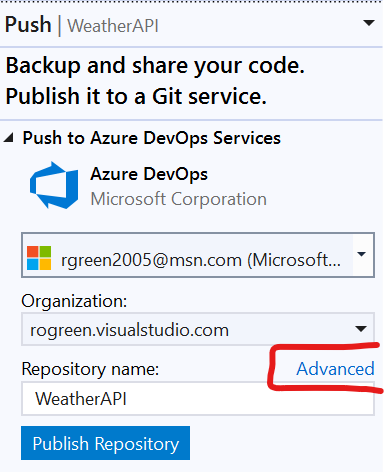
1. In Visual Studio, make sure the WeatherAPI project is open.
2. Click **Add to Source Control** in the status bar, then select **Git**.



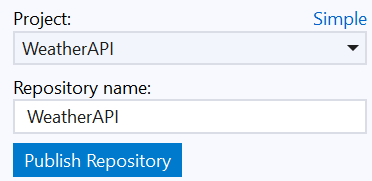
1. Click **Publish Git Repo** under Push to Azure DevOps Services. Wait while Visual Studio communicates with Azure DevOps Services.



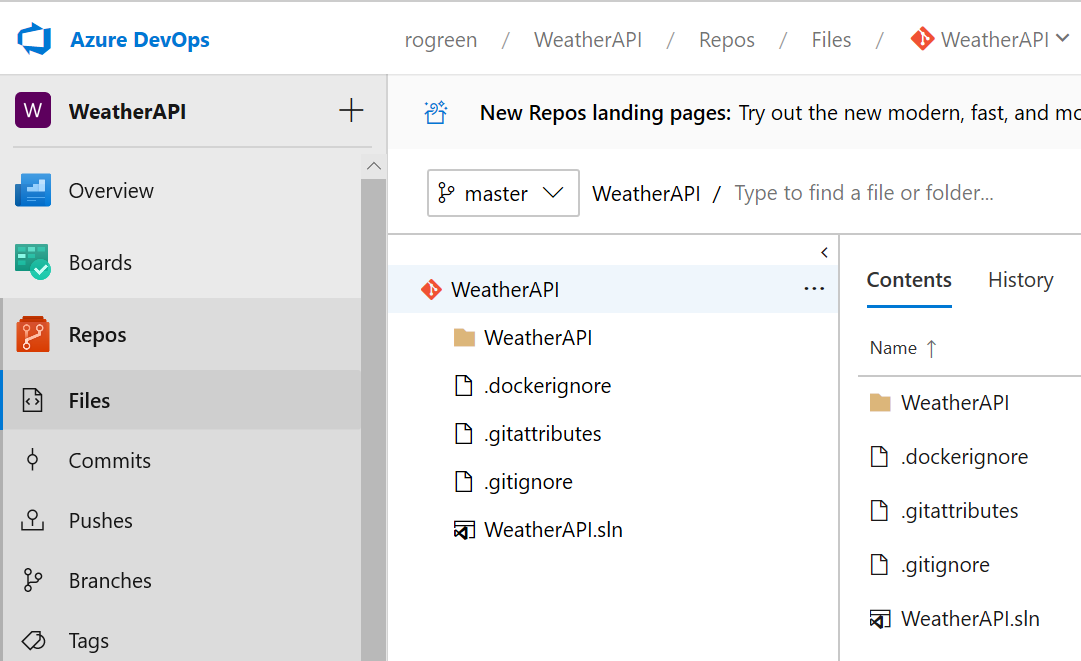
1. When it becomes enabled, click **Advanced**.



1. Select **WeatherAPI** from the Project drop-down list. Visual Studio queries the project and discovers the repository with the same name.



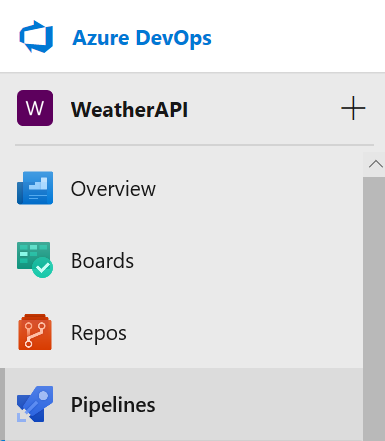
1. Click **Publish Repository**.
2. In Azure DevOps, click **Repos** on the left navigation bar and then select **Files**. The project is now under source control.



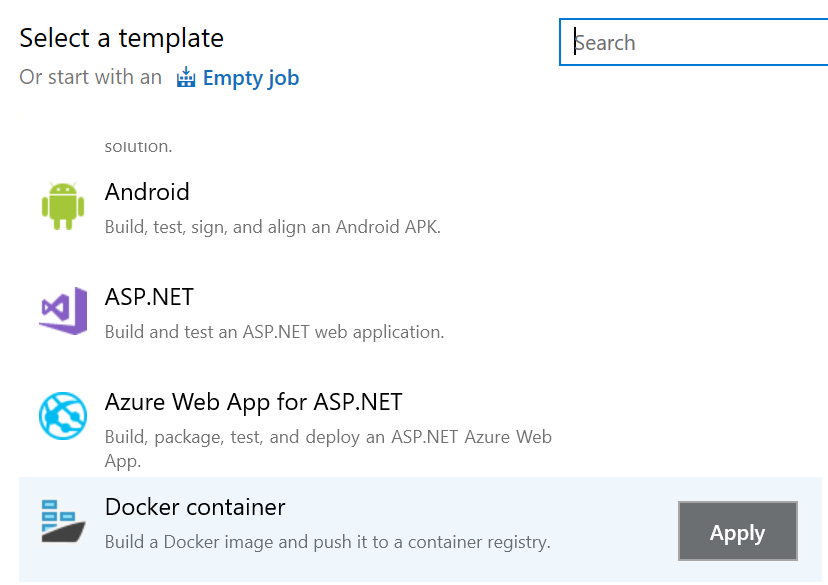
## Create Build and Release Pipelines

### Create a Build Pipeline

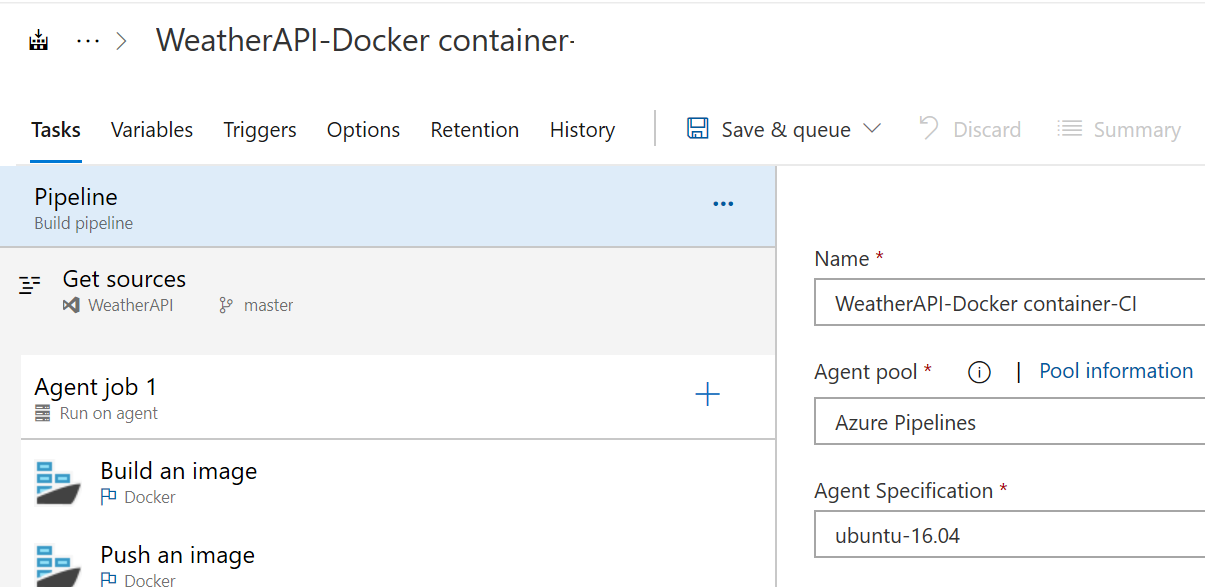
1. Click **Pipelines** on the left navigation bar.



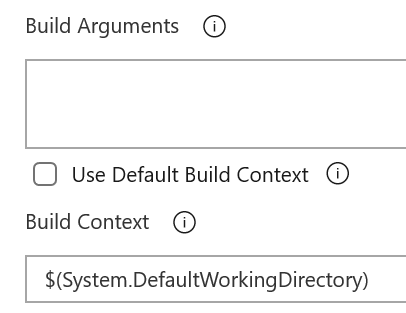
1. Click **Create Pipeline**.
2. At the bottom of the page, click **Use the classic editor**.
3. You are asked where your source code is. The repository you just created is the default. Leave it selected and click **Continue**.
4. You can start with an empty build template or write YAML or select pre-defined template. Scroll up and down to see them all.
5. In the list of templates, select **Docker Container** and click **Apply**. If you don’t see it, type Docker in the search box.



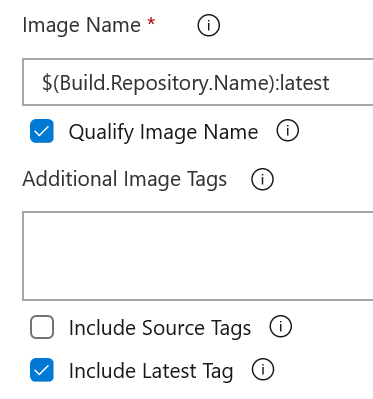
1. The pipeline contains two tasks: one to build the container image (Build an image) and another to push it to an Azure Container Registry (Push an image).
2. Selecting **Pipeline** shows you that the build occurs on a Linux machine.



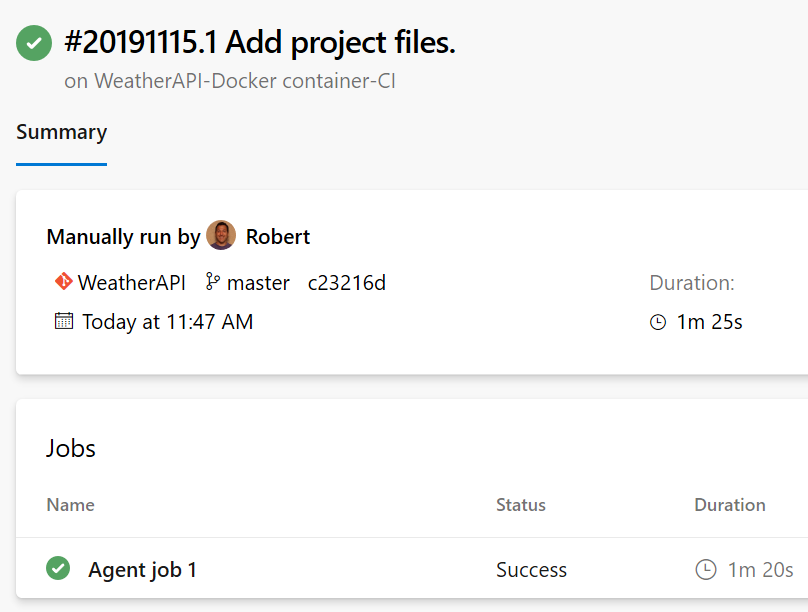
1. Selecting **Get sources** shows you the repository you selected when creating the pipeline.
2. Select the **Build an image** task.
3. Select your Azure subscription from the Azure subscription drop-down.
4. Click **Authorize**. You will be prompted to log in to Azure.
5. Select **<YourName>containers** from the Azure Container Registry drop-down.
6. Uncheck **Use Default Build Context**.
7. Enter **$(System.DefaultWorkingDirectory)** as the Build Context. This is necessary due to the pathing Visual Studio uses when it creates the Dockerfile.



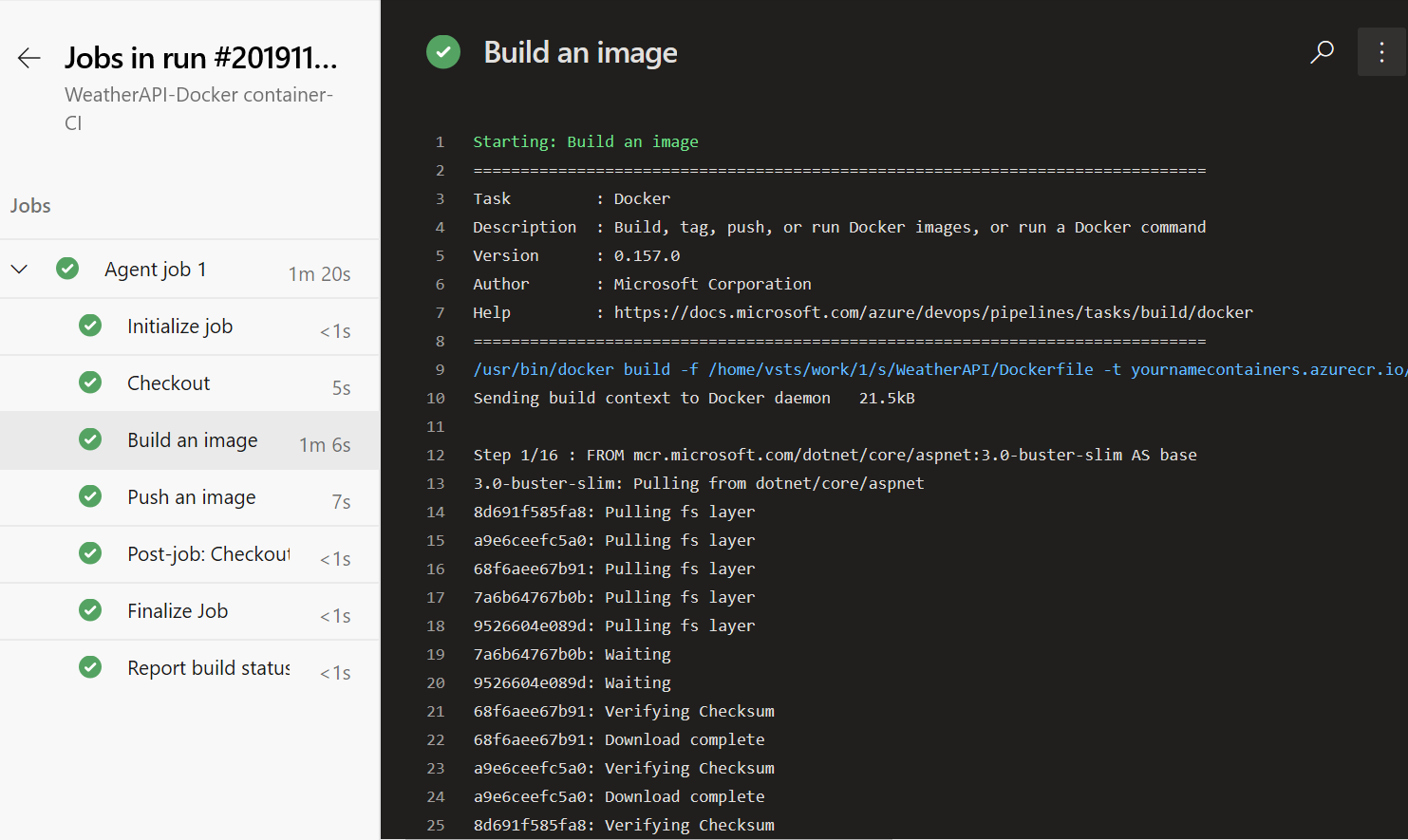
1. Change the Image Name to **$(Build.Repository.Name):latest**. The default would be to tag the image with the build number each time you built it. Then you update a different image every time. This way you will create one image and tag it latest.
2. Check **Include Latest Tag**.



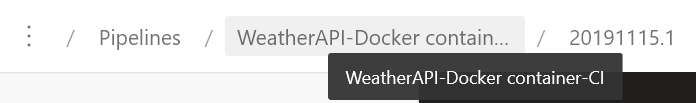
1. Select the **Push an image** task.
2. In the Azure subscription drop-down, select the available Azure service connection you set up in the previous task.
3. Select **<YourName>containers** from the Azure Container Registry drop-down.
4. Change the Image Name to **$(Build.Repository.Name):latest**.
5. Check **Include Latest Tag**.
6. In the Triggers tab, check **Enable continuous integration**. That way, each time you check code into master the container will automatically be built and pushed to the container registry.
7. Select **Save & queue** from the Save & queue drop-down.
8. In the Run pipeline dialog, click **Save and run**. This will manually trigger a build. You will be alerted as to whether the build succeeded or failed.



1. Click **Agent job 1**.
2. Click **Build an image** to see that in this task the container was built.

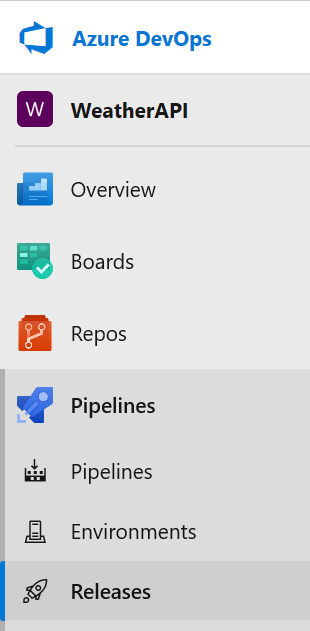


1. Click **Push an image** to see that in this task the container was pushed to the ACR.
2. Select the build in the task bar. This returns you to the build’s main page.

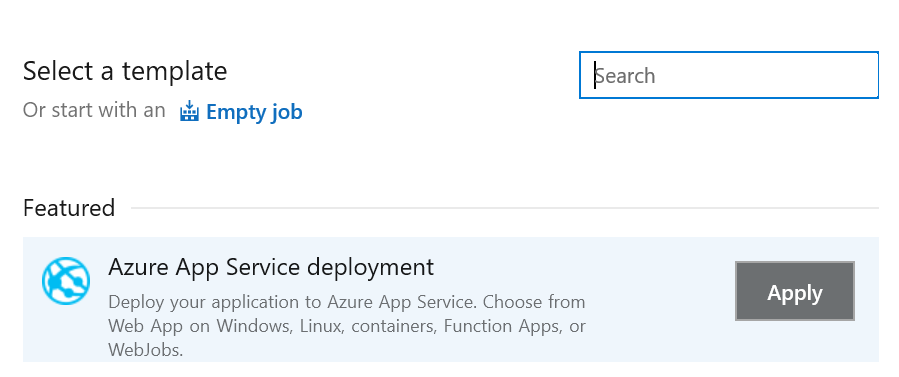


### Create a Release Pipeline

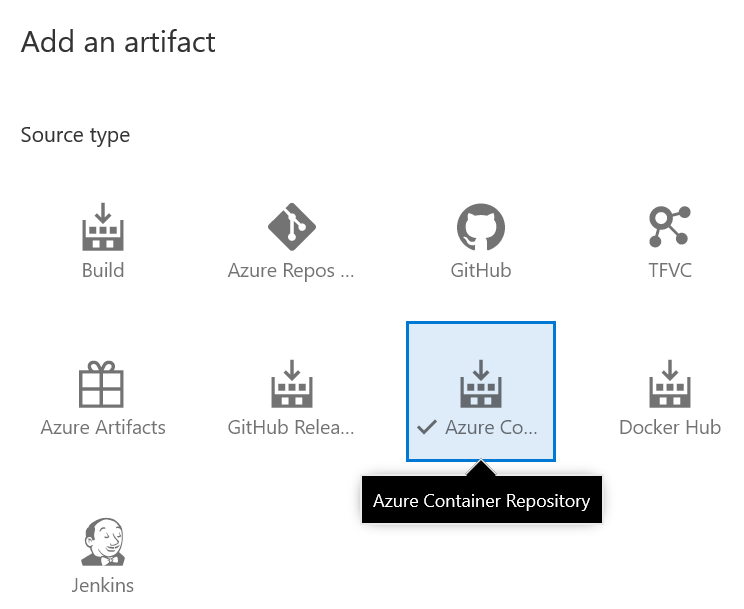
1. Select **Releases** under Pipelines on the left navigation bar.



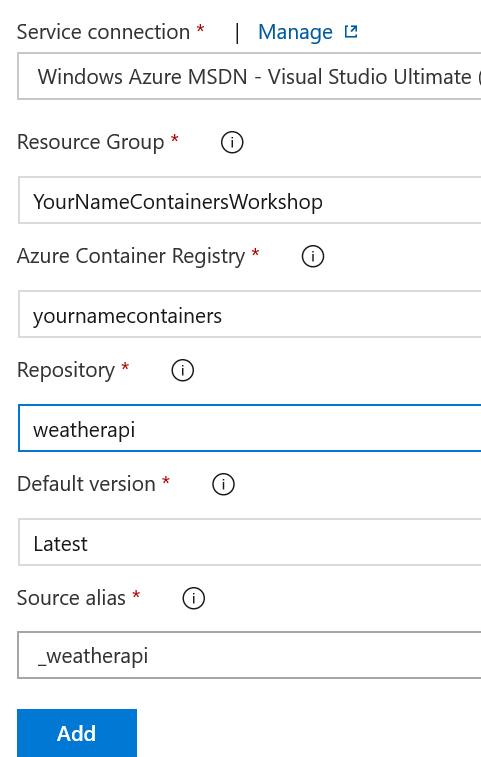
1. Click **New pipeline**.
2. Select the **Azure App Service Deployment** template and click **Apply**.



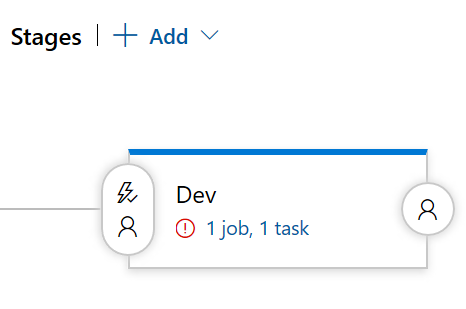
1. Change the name of the Stage to **Dev**.
2. Close the Stage window.
3. Click on **Add an artifact**. Here you need to identify what you are publishing, so you need to select the container in the container registry.
4. Click on the more artifact types link and select **Azure Container Registry**.



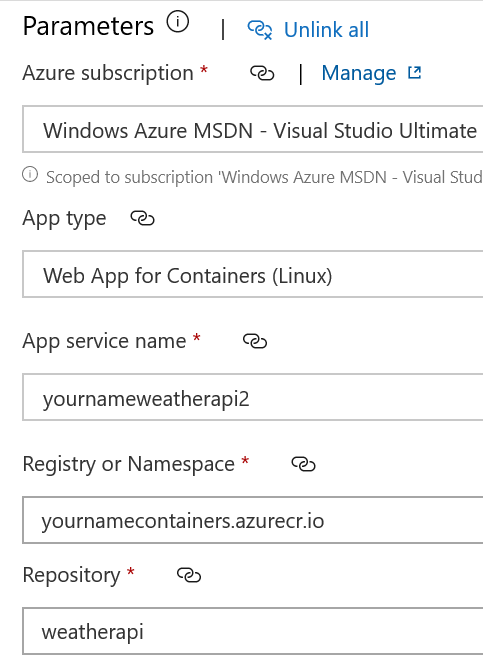
1. In the Service connection drop-down, select the Azure service connection you set up earlier.
2. Select **<YourName>ContainersWorkshop** from the Resource Group list.
3. Select **<YourName>containers** from the Azure Container Registry drop-down.
4. Select **weatherapi** from the Repository drop-down list.



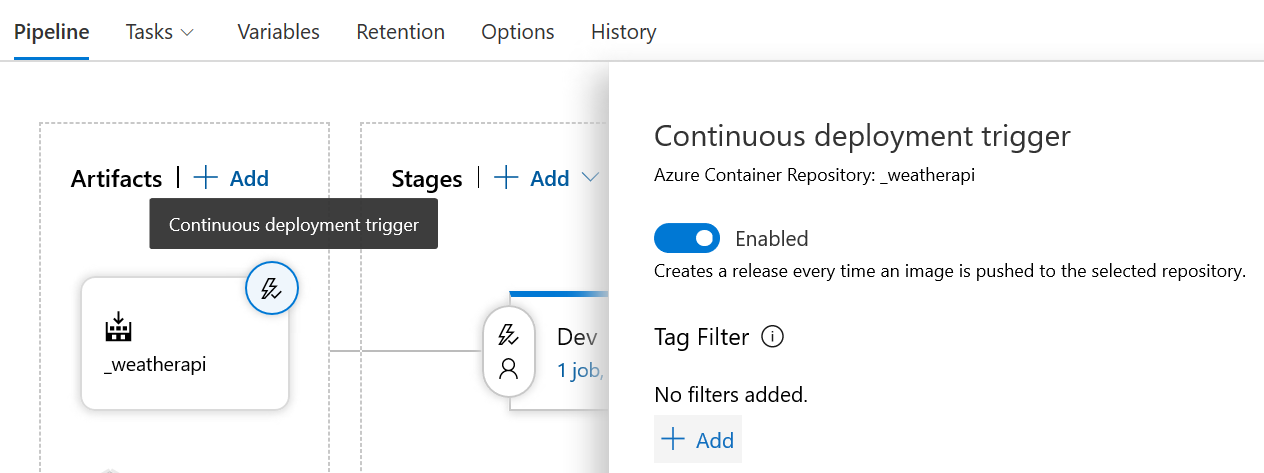
1. Click **Add**. You have now specified that you will publish that container. You now need to specify where you will publish it.
2. Click **1 job, 1 task** in the Dev stage.



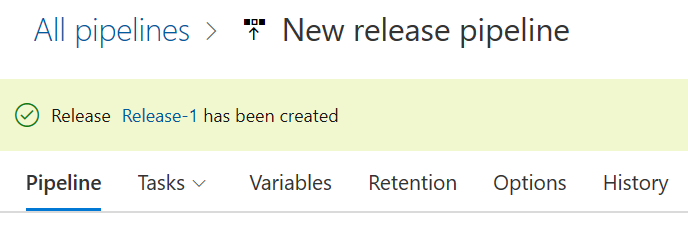
1. In the Azure subscription drop-down, select the Azure service connection you set up earlier.
2. Select **Web App for Containers (Linux)** from the App type drop-down list.
3. Select **<YourName>weatherapi2** from the App service name drop-down list. You will use an existing App service here. You could of course create a new one.
4. Enter **<YourName>containers.azurecr.io** as the Registry or Namespace.
5. Enter **weatherapi** as the Repository.

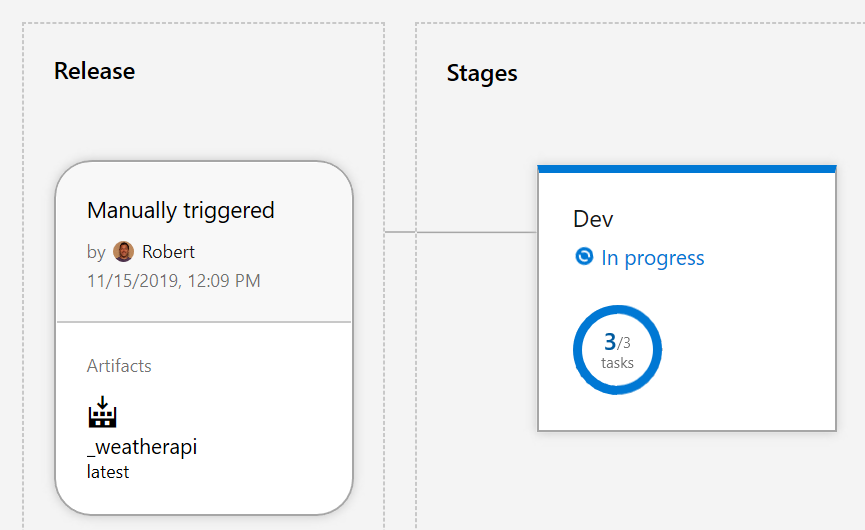


1. Select **Pipeline** to go back to the main pipeline tab.
2. Click the Continuous Deployment trigger button.
3. Set this to **Enabled**. That way, a successful build will automatically trigger a release.

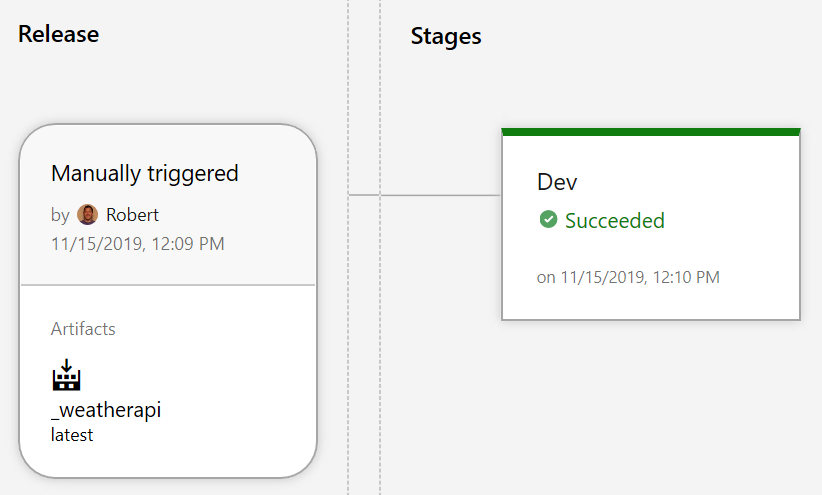


1. Close the Continuous deployment trigger window.
2. Click **Save**. In the Save dialog, click **OK**.
3. Click **Create Release**.
4. In the Create a new release window, click **Create** to manually trigger a release.
5. Click Release-1 to view the progress.





1. You will be alerted as to whether the release succeeded or failed.



1. In the browser, navigate to <YourName>weatherapi2.azurewebsites.net/weatherforecast. You should see the weather data.

### Automatically Trigger a Build and Release

1. In Visual Studio, open the WeatherForecastController.cs file.
2. Modify the Summaries array as follows:

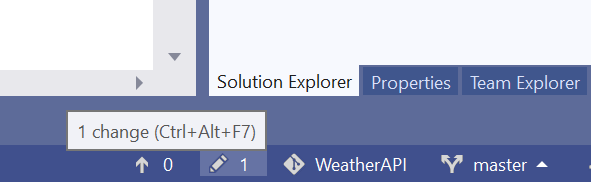
private static readonly string[] Summaries = new[]

{

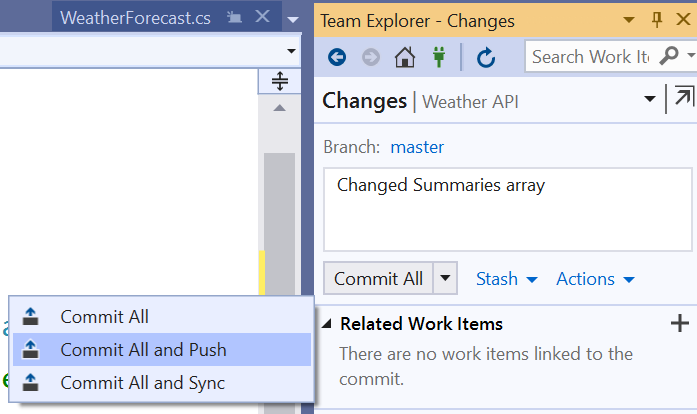
    "Way too cold", "Too cold", "Cold", "Coolish", "Warmish", "Nice", "Uncomfortable", "Hot", "Too hot", "Way too hot"

};

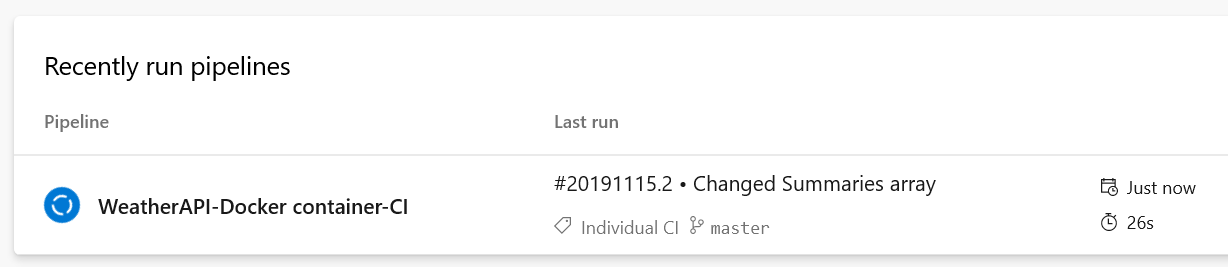
1. Save the file.
2. Click the changes icon in the status bar.



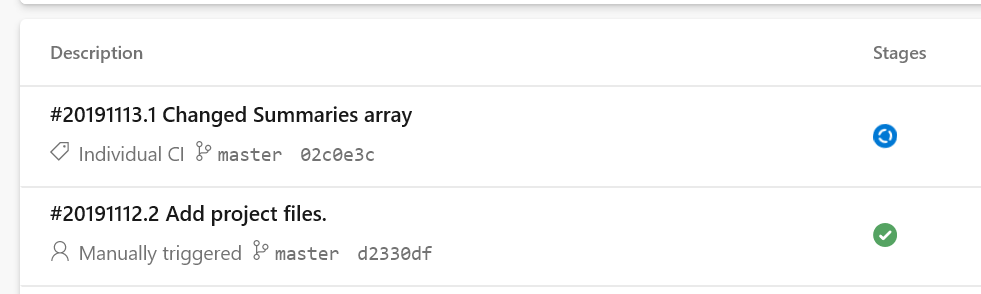
1. Enter a commit message and then select **Commit All and Push** to send this change to the source code repository.



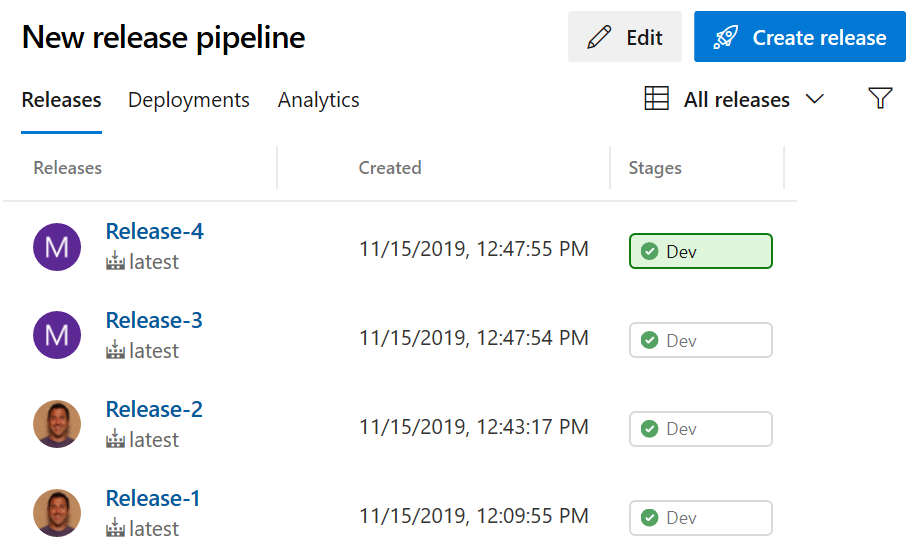
1. When the changes have successfully pushed, go to Azure DevOps in the browser.
2. Click on **Pipelines** on the left navigation and then **Pipelines**. See that a build has started automatically.



1. Click on the pipeline to see the previous and current build.



1. When the build succeeds, click **Releases** under Pipelines. See that a release has started automatically.



1. When the release succeeds, open a new browser instance (to avoid seeing the old values due to caching) and navigate to <YourName>weatherapi2.azurewebsites.net/weatherforecast. You should see the new summaries displayed.

