

Table of Contents

Lab: Module 6 - DevOps with Containers	2
Duration: 120 minutes.....	2
Exercise 1: Setup Azure DevOps Account.....	2
Exercise 2: Setup Project and Azure Container Registry (ACR)	5
Exercise 3: Create Build Pipeline for Linux Containers	14
Exercise 4: Create Release Pipeline for AKS Cluster	29
Exercise 5: Create Build Pipeline for Windows Containers	43
Exercise 6: Create Release Pipeline for Service Fabric Windows Cluster	47

Lab: Module 6 - DevOps with Containers

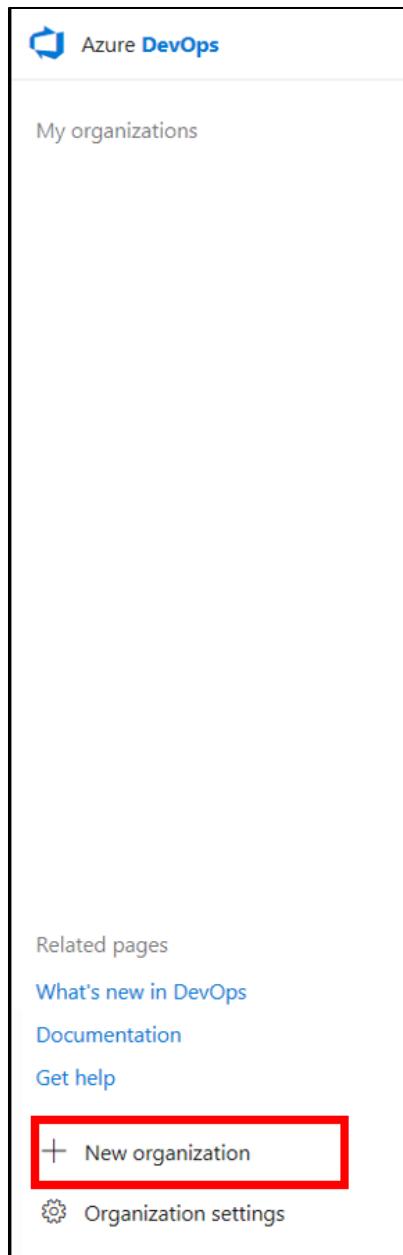
Duration: 120 minutes

Exercise 1: Setup Azure DevOps Account

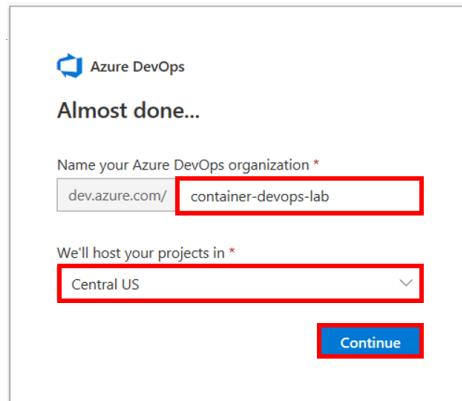
In this exercise, you are going to create an Azure DevOps account which contains toolset to create CI/CD pipeline. If you already have an Azure DevOps account, you can skip this exercise.

Tasks

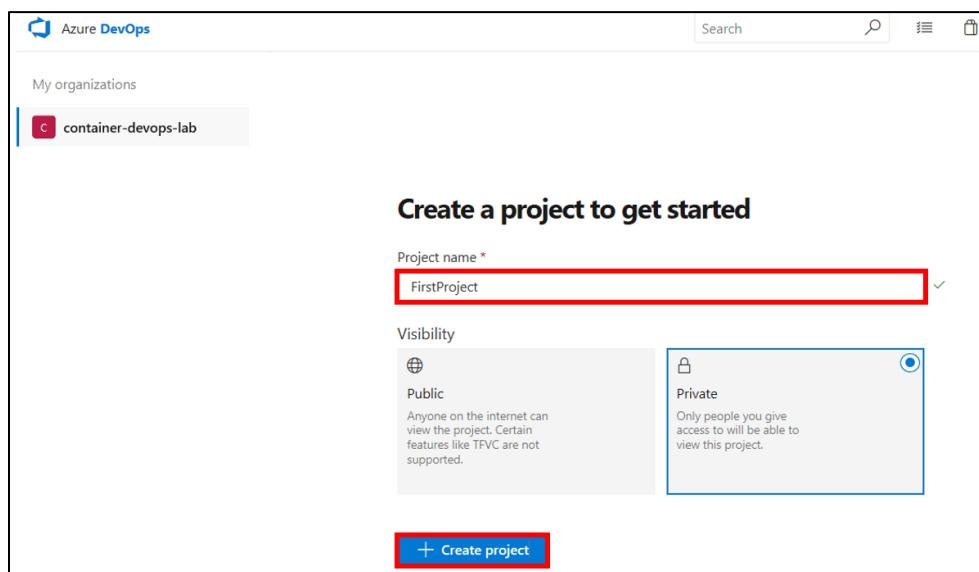
- 1. Create Azure DevOps Organization and Project**
 1. Sign onto your LOD Windows VM and open Chrome on the VM. Navigate to dev.azure.com and sign in with your Azure Pass account. Click on **Start Free** to open Azure DevOps.
 2. On Dev Portal, click **New Organization** button to add a new organization.



3. Give a unique name to your organization, select the closest datacenter and then click **Continue** to create a new Azure DevOps organization.



4. Choose a project name. Click on **Create Project**. This will take you to the URL of your project.



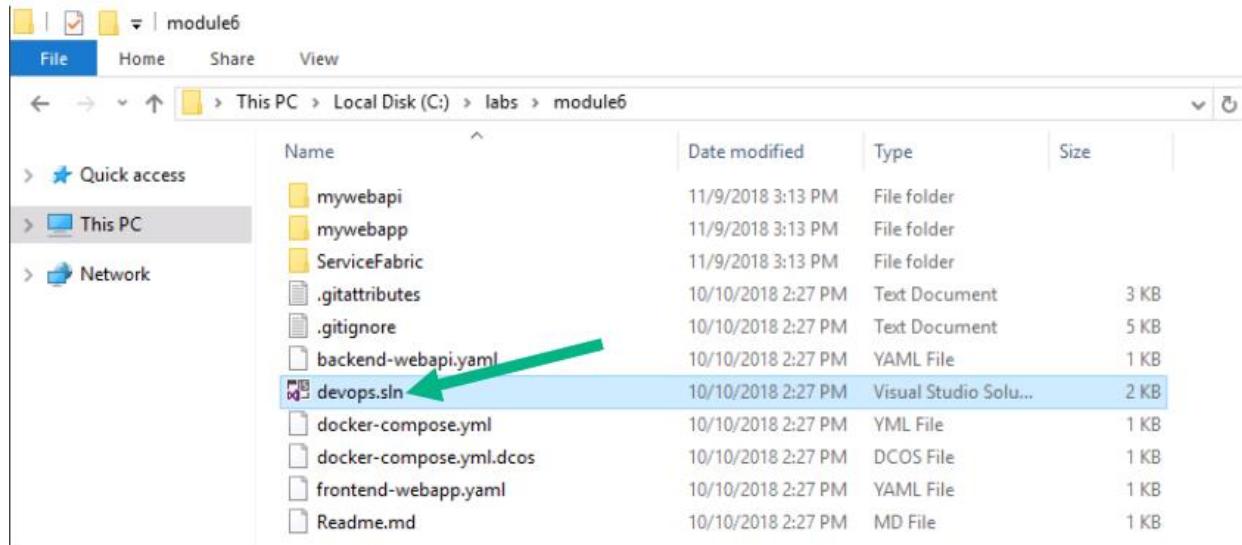
Exercise 2: Setup Project and Azure Container Registry (ACR)

In this exercise, you are going to complete first part of CI/CD pipeline by pushing the code artifacts to Azure Repository which includes sample web application and web API. You will also create an Azure Container Registry (ACR) to store Docker images in the private repository.

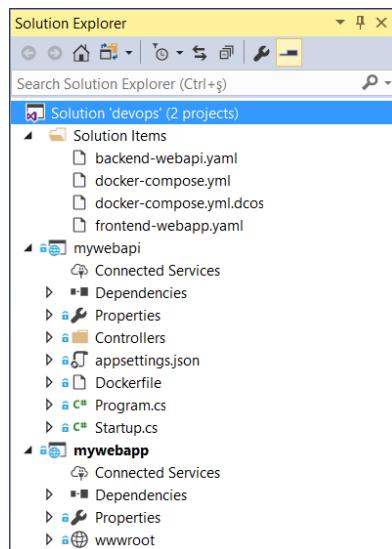
Tasks

1. Examine and Build Sample Application in Visual Studio

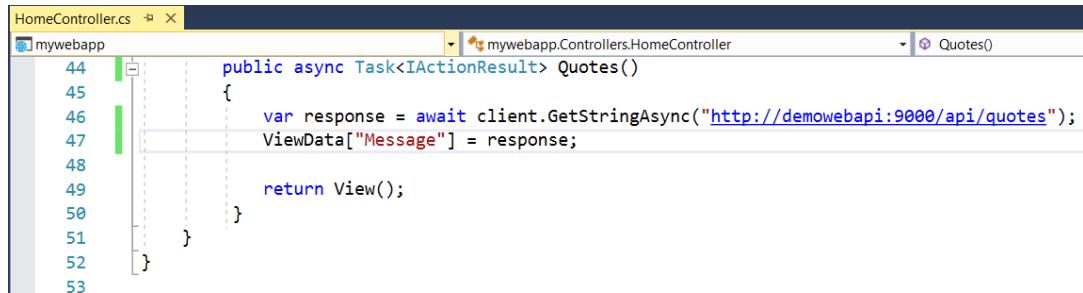
1. Go to C:\labs\module6 on your Windows VM. Double click on the “devops” solution file which will open the project in Visual Studio.



2. On the right, you will see the Solution Explorer. You will see that there are two ASP.NET Core applications, one is a web API backend project and the other is MVC Core frontend project.



- Open **HomeController.cs** in mywebapp project, under the Controllers folder. You will see that frontend application is retrieving some quotes from the web API. The URL to reach to the web API is configured by container orchestrators such as Kubernetes or Docker Swarm.

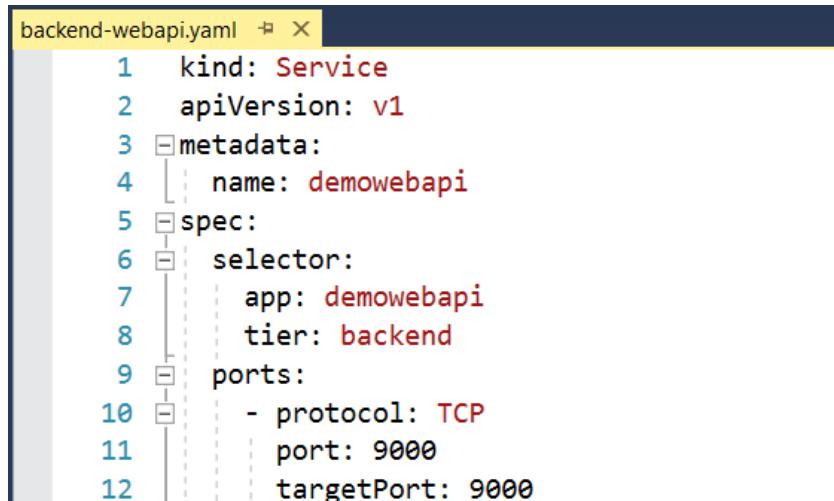


```

HomeController.cs  X
mywebapp          mywebapp.Controllers.HomeController
44  public async Task<IActionResult> Quotes()
45  {
46      var response = await client.GetStringAsync("http://demowebapi:9000/api/quotes");
47      ViewData["Message"] = response;
48
49      return View();
50  }
51
52  }
53

```

- In the Solution Items folder, open the **backend-webapi.yaml** file which is the deployment configuration file for Kubernetes cluster. Notice that here the backend application is also exposed with the name **demowebapi**.



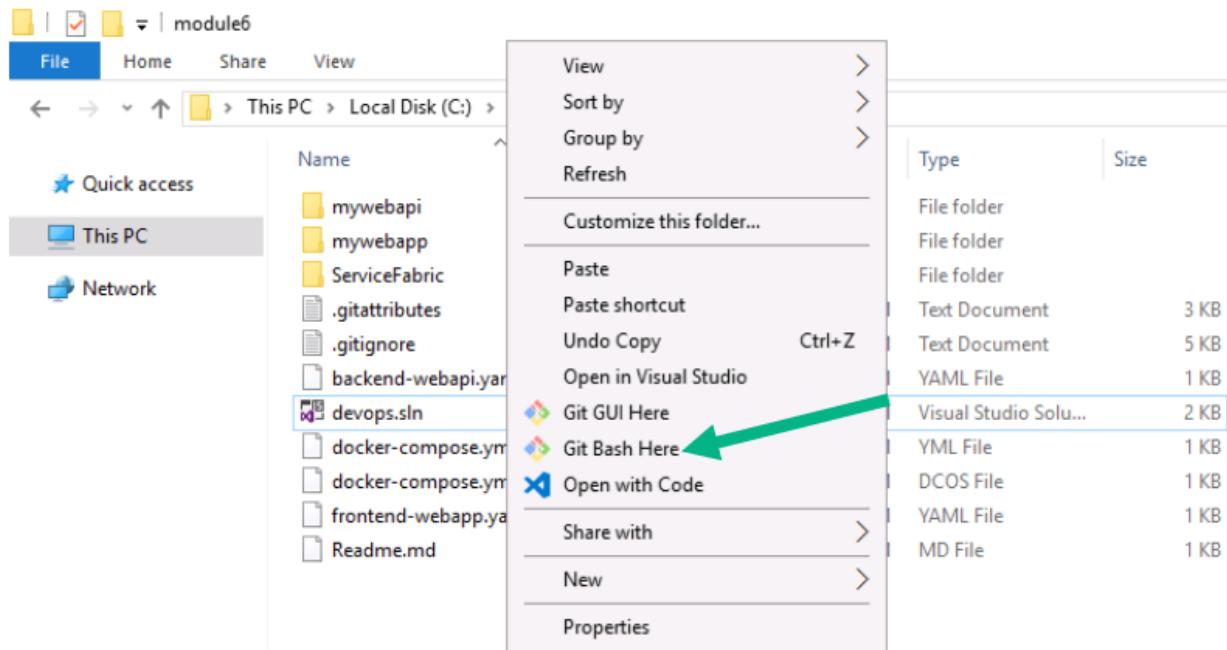
```

backend-webapi.yaml  X
1  kind: Service
2  apiVersion: v1
3  metadata:
4      name: demowebapi
5  spec:
6      selector:
7          app: demowebapi
8          tier: backend
9      ports:
10         - protocol: TCP
11             port: 9000
12             targetPort: 9000

```

2. Add Solution to Source Control (Git and Azure Repository)

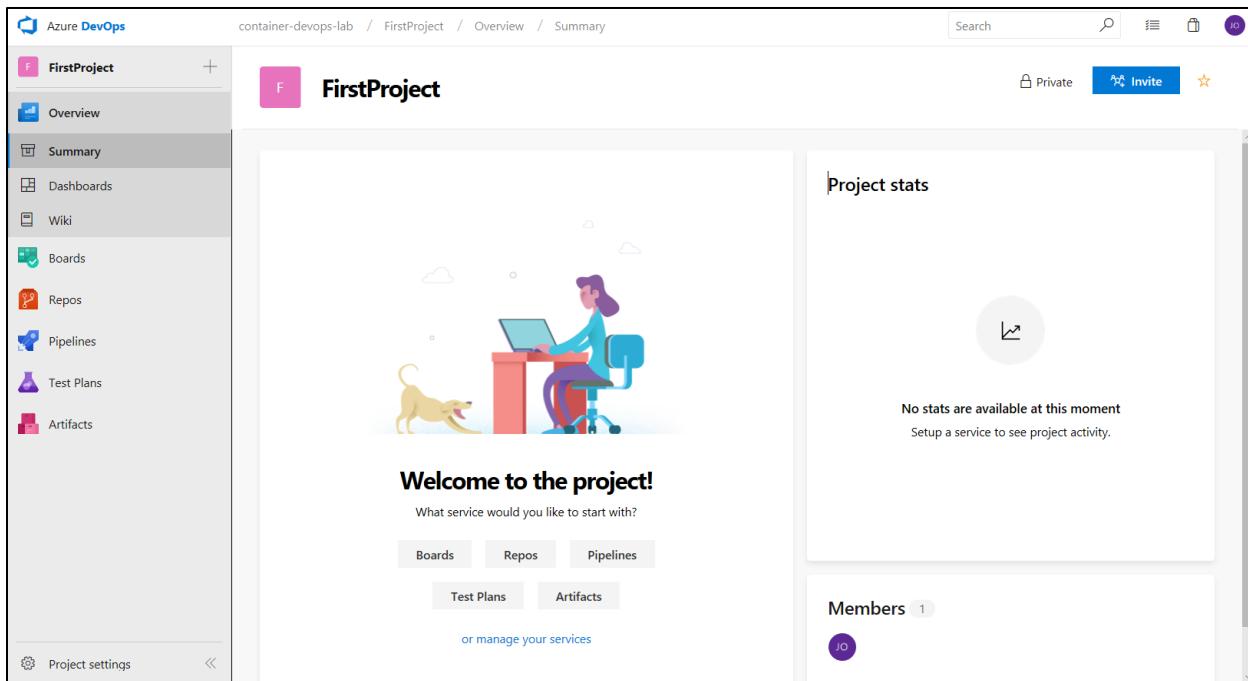
- Open File Explorer and navigate to **C:\labs\module6**.
- Right click choose Git Bash Here.



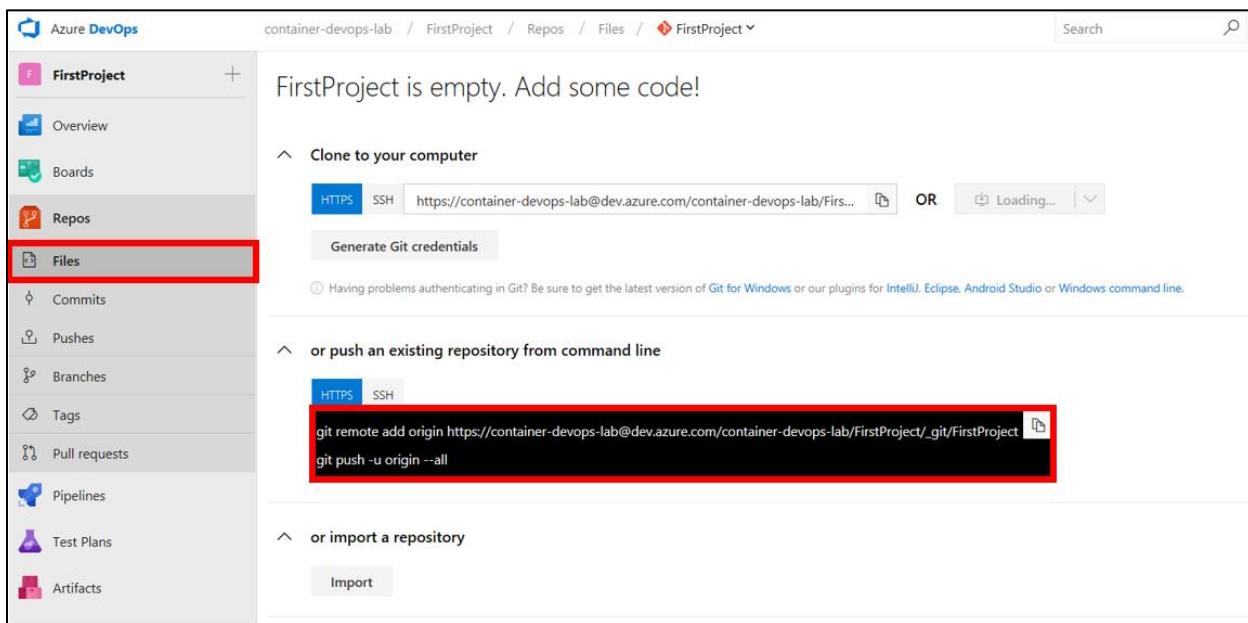
- Run the following commands, one at a time (you can either fill in a name and email or leave it as the default). Git config (first two commands) is a one-time setup. Git init will initialize your folder as a git repository and add an empty hidden .git folder. Git add and commit are necessary steps anytime you make a change to your code and want it to be staged (tracked and ready to be committed) and committed (ready to be pushed to the server)

```
git config --global user.email you@example.com
git config --global user.name "Your Name"
git init
git add .
git commit -m 'adding my web project'
```

- Navigate to <https://dev.azure.com/YourAccountName/YourProjectName> to see the environment. A sample link is <https://dev.azure.com/container-devops-lab/FirstProject>



- Now you are ready to push your local Git repository to remote repository in Azure DevOps. Go to your Code repository by clicking on **Repos - Files**. You will see that your Git repository is empty, and you are ready to add your application into it. Copy the two commands to do this.

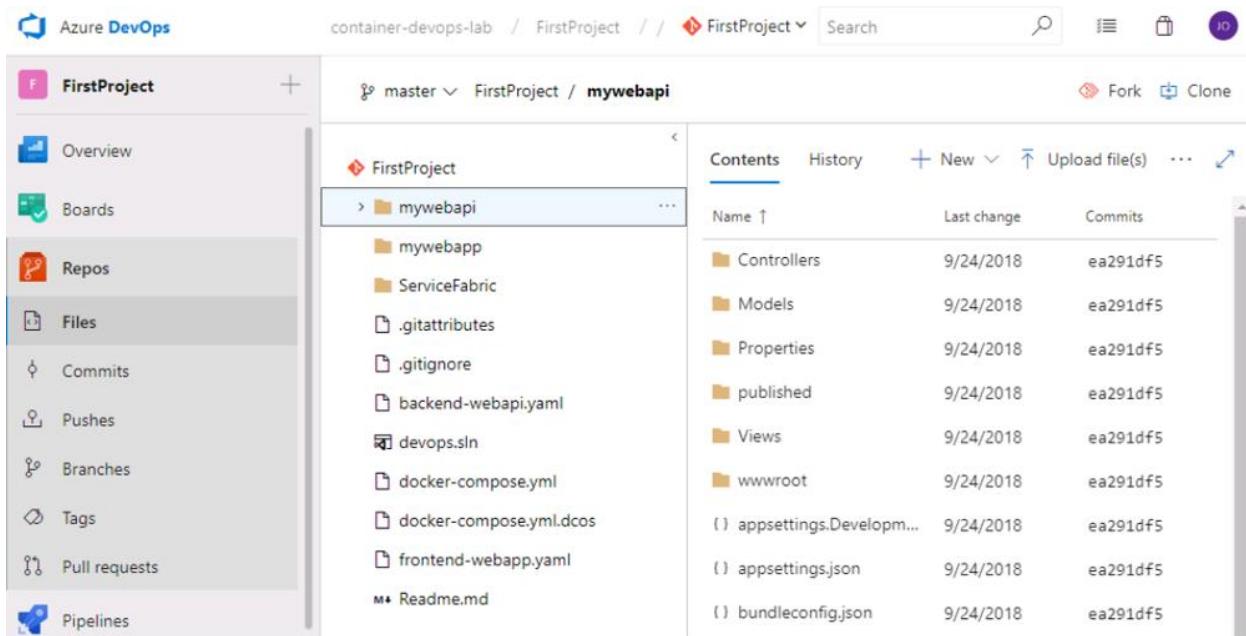


- Paste them into Git Bash by pressing Shift + Insert (or right click and paste it into the Git Bash command line). It will automatically run the first line, hit enter to add the second line. “Git remote add origin” means to add a “remote URL such as your Azure repo URL” with the alias “origin” to your local git settings for this project folder. You could name it any alias you want, “origin” is just the default. “Git push”

means to interact with the server and push code to that URL you provided as origin. The “-u” will “set-upstream” which means that it will default all pushes and pulls to the “origin URL” and you don’t have to specify “origin” in all your future commands.

```
super@5dokdksszx3za MINGW64 /c/labs/module6 (master)
$ git remote add origin https://container-devops-lab@dev.azure.com/container-devops-lab/FirstProject/_git/FirstProject
super@5dokdksszx3za MINGW64 /c/labs/module6 (master)
$ git push -u origin --all
```

You will see a popup to authenticate to your Microsoft account. Once authenticated, your project will be pushed to remote Git repository in Azure DevOps if you refresh Azure DevOps repository in your browser.



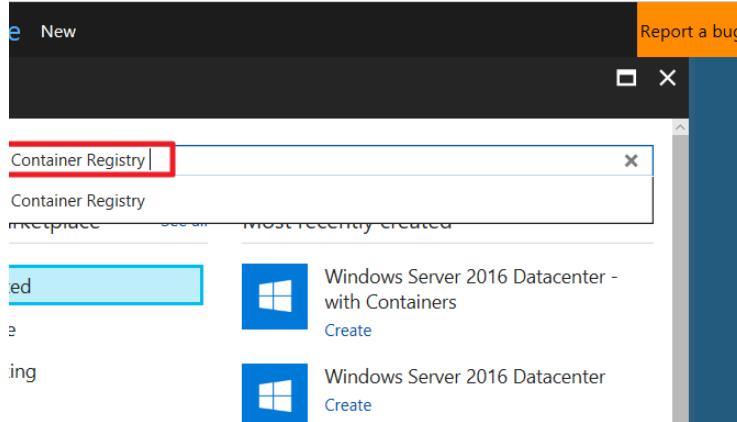
The screenshot shows the Azure DevOps interface for a project named 'FirstProject'. The left sidebar has 'Repos' selected. The main area shows a file tree under 'mywebapi' with files like mywebapp, ServiceFabric, .gitattributes, .gitignore, backend-webapi.yaml, devops.sln, docker-compose.yml, docker-compose.yml.dcos, frontend-webapp.yaml, and Readme.md. To the right is a table of contents with columns for Name, Last change, and Commits. The table lists several files with their last change date (9/24/2018) and commit hash (ea291df5).

Name ↑	Last change	Commits
Controllers	9/24/2018	ea291df5
Models	9/24/2018	ea291df5
Properties	9/24/2018	ea291df5
published	9/24/2018	ea291df5
Views	9/24/2018	ea291df5
wwwroot	9/24/2018	ea291df5
(.) appsettings.Developm...	9/24/2018	ea291df5
(.) appsettings.json	9/24/2018	ea291df5
(.) bundleconfig.json	9/24/2018	ea291df5

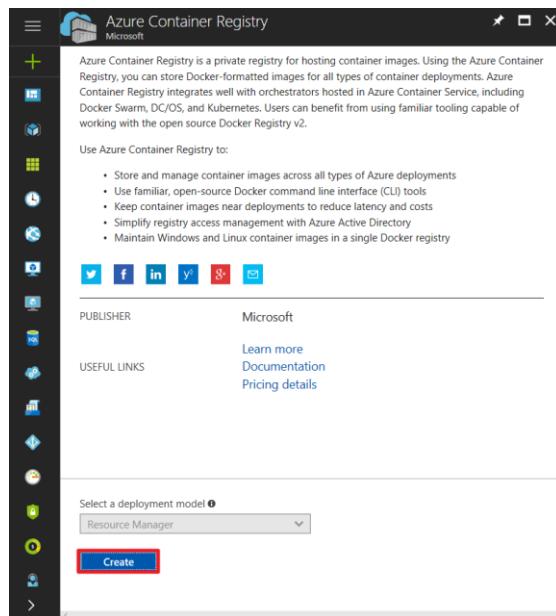
3. Create Azure Container Registry (ACR)

One of the critical components in DevOps lifecycle is container registry. Container registry allows you to store and manage your container images. You can use public registries such as docker hub, private registries installed in your on-premise environment or provided by cloud services. In this task, you are going to use Azure Container Registry to manage your Linux and Windows container images.

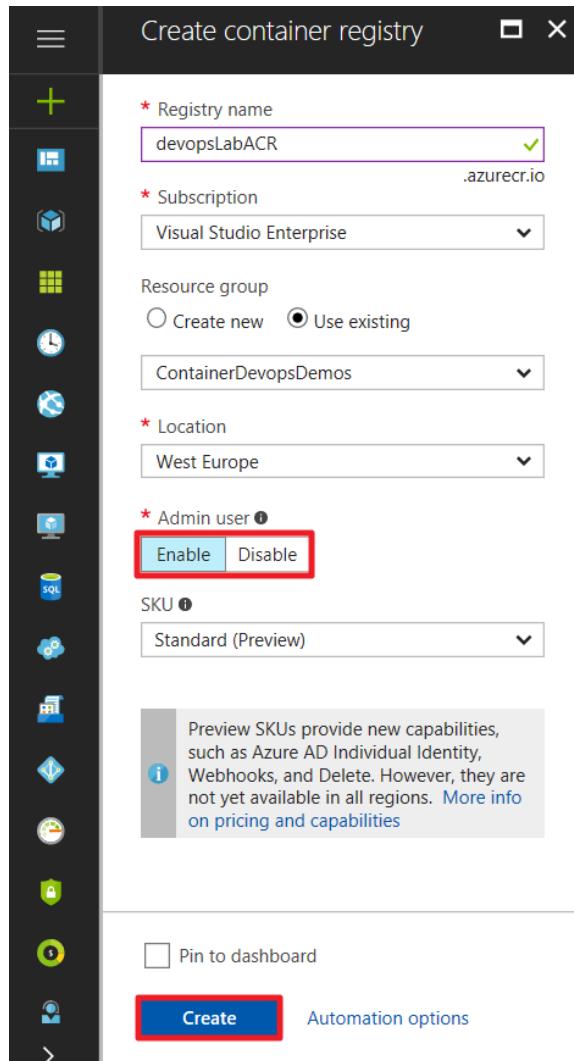
1. In Azure Portal, click plus button to add a new resource. Type “**Container Registry**” in the search textbox and click on the result.



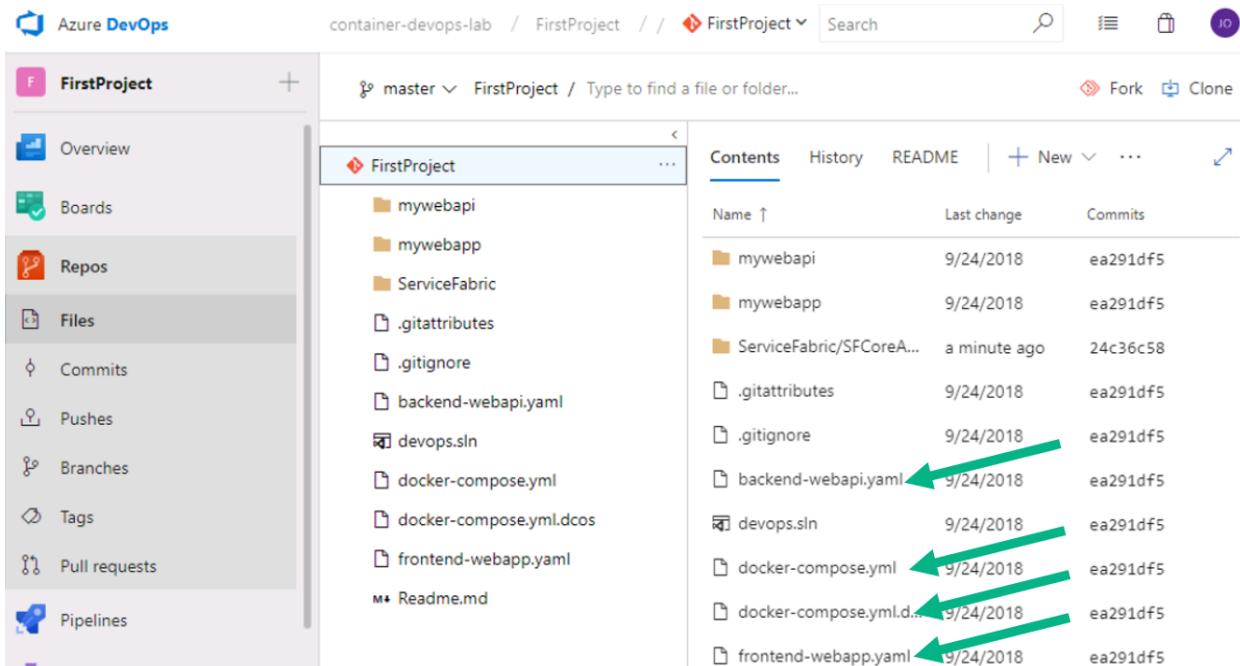
2. Click Create



3. Select a registry name which should be unique and different from the one on the screenshot. You may select the resource group you have used in previous labs to keep resources together. Make sure to enable **Admin user** so that build agents can login to ACR and push container images successfully. Note the SKU just needs to be Standard.



4. While ACR is being built, you can update the container registry name in your Kubernetes configuration files. Go into Azure DevOps and hit the **Repos > Files** tab. The 4 files that you will need to be edited are: **backend-webapi.yaml**, **docker-compose.yml**, **docker-compose.yml.dcos**, and **frontend-webapp.yaml**.



5. Go into each of the 4 files individually and replace all instances of the existing ACR name **devopslabacr** with the new ACR name that is being created **(you must put in your ACR name all in lowercase, otherwise you will get an error later!)**. There should be a total of 6 replacements.

Make sure you write your ACR name in **lowercase**, e.g. `devopslabacr.azurecr.io/demo-webapp:BuildNumber`

You may face authentication issues while your Kubernetes tries pulling container images during deployment. Azure Pipeline Kubernetes task will create a secret with lowercase ACR address, and your ACR name in yaml file should match it.

6. Similarly, the two **ServiceManifest.xml** files located in **ServiceFabric/SFCoreApp/ApplicationPackageRoot/BackEnd_WebAPIpkg** and **ServiceFabric/SFCoreApp/ApplicationPackageRoot/FrontEnd_WebAppPkg** and replace the image name prefix with the new ACR that you just created instead of the default value **devopslabacr**

Page Heading Ignored

The screenshot shows a GitHub commit page for a project named 'FirstProject'. The commit message is 'Committed b710ee36: Updated ServiceManifest.xml — Create a pull request'. The commit details two changes:

- BackEnd_WebAPIPkg / ServiceManifest.xml**: This file defines a service type named 'BackEnd_WebAPIPkg' with version 1.0.0. It uses a stateless service type and specifies 'UseImplicitHost=true'. The code package is named 'Code' with version 1.0.0. It includes an entry point with an image name pointing to a private Docker registry: 'devopslab.azurecr.io/demo-webapi:win-BuildNumber'. Environment variables are also defined.
- FrontEnd_WebAppPkg / ServiceManifest.xml**: This file defines a service type named 'FrontEnd_WebAppPkg' with version 1.0.0. It uses a stateless service type and specifies 'UseImplicitHost=true'. The code package is named 'Code' with version 1.0.0. It includes an entry point with an image name pointing to a private Docker registry: 'devopslab.azurecr.io/demo-webapp:win-BuildNumber'. Environment variables are also defined.

Green arrows point from the commit message to the two ServiceManifest.xml files in the commit history.

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceManifest Name="BackEnd_WebAPIPkg"
  Version="1.0.0"
  xmlns="http://schemas.microsoft.com/2011/01/fabric"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<ServiceTypes>
  <!-- This is the name of your ServiceType.
       The UseImplicitHost attribute indicates this is a guest service. -->
  <StatelessServiceType ServiceTypeName="BackEnd_WebAPIType" UseImplicitHost="true" />
</ServiceTypes>
<!-- Code package is your service executable. -->
<CodePackage Name="Code" Version="1.0.0">
  <EntryPoint>
    <!-- Follow this link for more information about deploying Windows containers to Service Fabric
        <ContainerHost>
          <ImageName>[REDACTED]devopslab.azurecr.io/demo-webapi:win-BuildNumber</ImageName>
        </ContainerHost>
      </EntryPoint>
    <!-- Pass environment variables to your container: -->
    <!--
      <EnvironmentVariables>
        <EnvironmentVariable Name="VariableName" Value="VariableValue"/>
      </EnvironmentVariables>
    -->
  </CodePackage>
```

Exercise 3: Create Build Pipeline for Linux Containers

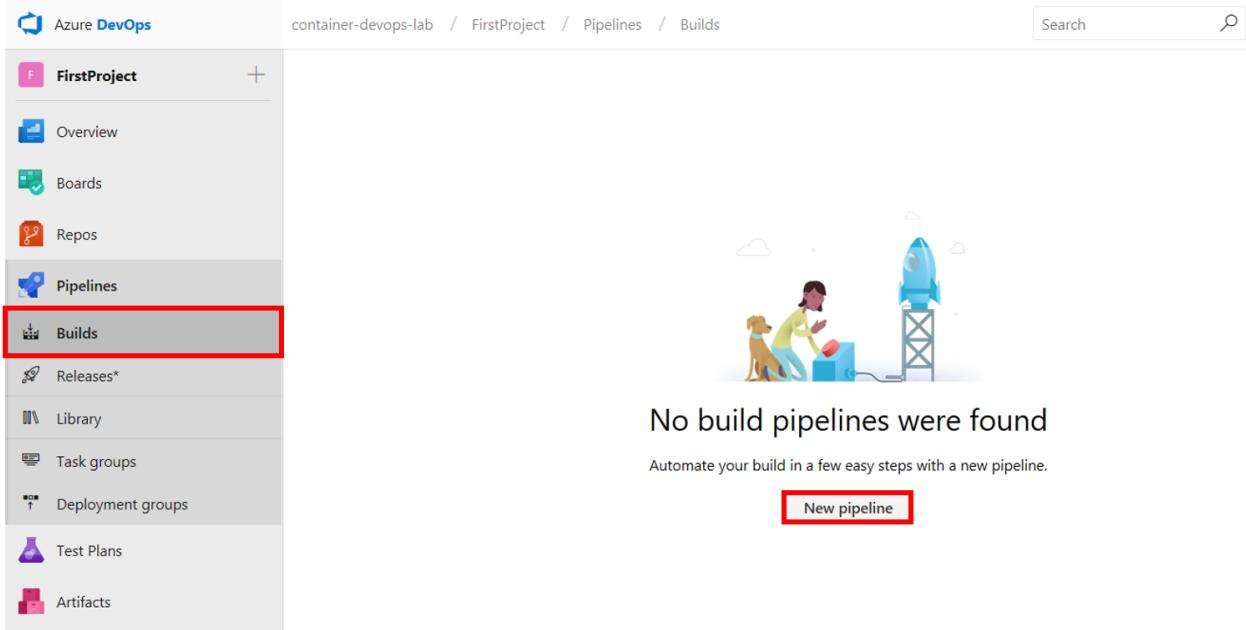
Tasks

If you want to avoid creating the pipeline step by step, you can skip this task and jump to [Optional: Import a build pipeline](#).

1. Create new build pipeline

In this task, you are going to create a build pipeline which will be executed on Linux agent and will produce two Linux container images of your applications. These images will be tagged automatically with appropriate build number and then pushed into the Azure Container Registry (ACR).

1. Navigate to <https://dev.azure.com/YourAccountName/YourProjectName> and go to **Pipelines - Builds** page and click **New pipeline**.



2. Choose **Use the visual designer**.

New pipeline

Build your code in a few easy steps

- 1 Location
- 2 Repository
- 3 Template
- 4 Save and run

Where is your code?



GitHub

Home to the world's largest community of developers



Azure Repos

Free private Git repositories, pull requests, and code search

[Use the visual designer](#) to create a pipeline without YAML.

3. Create the build pipeline by selecting the Azure repository and click **Continue**.

The screenshot shows the 'Select your repository' step in the Azure DevOps Pipelines setup. On the left, the 'Pipelines' menu is open, showing options like Builds, Releases, Library, Task groups, Deployment groups, Test Plans, and Artifacts. The 'Pipelines' item is selected. In the center, a large arrow points to the right. Below it, the text 'Select your repository' is displayed, along with instructions: 'Tell us where your sources are. You can customize how to get these sources from the repository later.' To the right, there is a grid of source selection options: GitHub, GitHub Enterprise, Subversion, Bitbucket Cloud, and External Git. The 'Azure Repos Git' option is highlighted with a red box. Below this, a form is shown for selecting a team project, repository, and default branch. The 'Team project' dropdown is set to 'FirstProject', the 'Repository' dropdown is set to 'FirstProject', and the 'Default branch for manual and scheduled builds' dropdown is set to 'master'. A large red box highlights the 'Continue' button at the bottom right of the form.

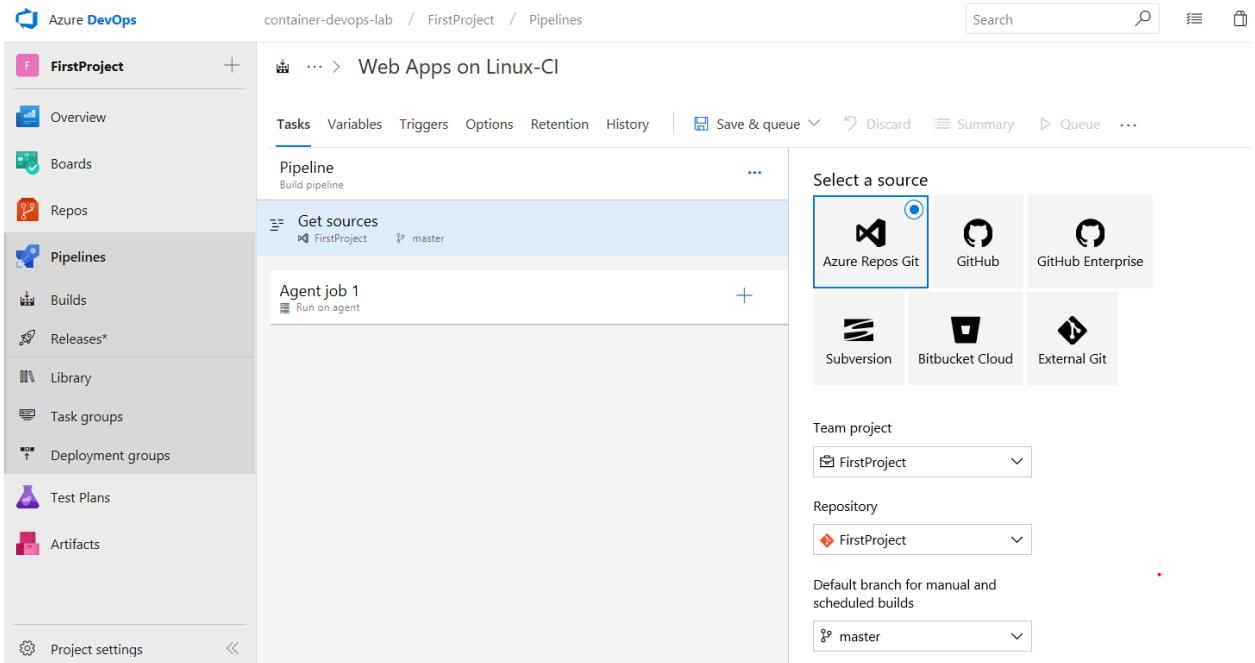
4. Then, select an **Empty job** to start from a clean state.

The screenshot shows the Azure DevOps interface for creating a new pipeline. The left sidebar is titled 'FirstProject' and includes options like Overview, Boards, Repos, Pipelines, Builds, Releases*, Library, Task groups, Deployment groups, Test Plans, and Artifacts. The 'Pipelines' option is selected. The main area is titled 'Choose a template' with a large circular arrow icon. Below it, text says 'Choose a template that builds your kind of app. Don't worry if it's not an exact match; you can add and customize the tasks later.' To the right, there's a section titled 'Select a template' with a search bar and a link to 'Empty job'. Below this, a note says 'Configuration as code' and 'Looking for a better experience to configure your pipelines using YAML files? Try the new YAML pipeline creation experience. Learn more'. A 'Featured' section lists several pipeline types: '.NET Desktop', 'Android', 'ASP.NET', and 'Azure Web App for ASP.NET', each with a brief description.

5. Before adding build tasks to the pipeline, you are going to select an agent queue containing agents which will be assigned to handle build requests. **Select Hosted Ubuntu 1604** agent queue which contains agents installed on Linux OS.

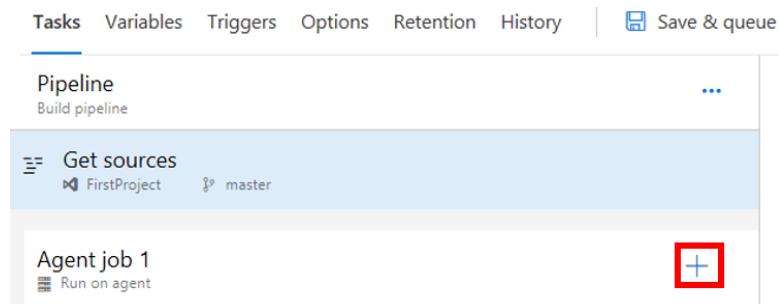
This screenshot shows the 'Agent pool' configuration for a pipeline. It includes fields for 'Name' (set to 'Web Apps on Linux-Cl'), 'Agent pool' (set to 'Hosted Ubuntu 1604'), and 'Parameters'. A note states: 'This pipeline doesn't have any pipeline parameters. Create them to share the most important settings between tasks and change them in one place.' A 'Learn more' link is also present.

6. Click on **Get sources** task and leave the default options.



The screenshot shows the Azure DevOps Pipelines interface for a project named 'FirstProject'. On the left, there's a sidebar with options like Overview, Boards, Repos, Pipelines, Builds, Releases, Library, Task groups, Deployment groups, Test Plans, Artifacts, and Project settings. The 'Pipelines' option is selected. The main area shows a 'Pipeline' build pipeline with two tasks: 'Get sources' (using FirstProject master branch) and 'Agent job 1' (Run on agent). To the right, a 'Select a source' modal is open, displaying various source control options with 'Azure Repos Git' highlighted.

7. Now we can start adding necessary tasks to build the web application and web API. You will click the + sign to add new tasks:



The screenshot shows the same Azure DevOps Pipelines interface as before, but now the 'Agent job 1' task card has a red box drawn around its '+' icon in the bottom right corner, indicating where to click to add new tasks.

8. Normally, you might need to add dotnet build and publish steps, however these are included inside of the Dockerfile and are completed as part of the Docker build process. Just remember when you are creating your own CI/CD pipelines to either include the dotnet build and publish steps in the Dockerfile or the CI Build pipeline. For your reference, here is the Dockerfile:

```
FROM microsoft/aspnetcore-build:2.0 AS build-env  
WORKDIR /app
```

```
# Copy csproj and restore as distinct layers  
COPY *.csproj ./
```

```

RUN dotnet restore

# Copy everything else and build
COPY . .

RUN dotnet publish -c Release -o out

# Build runtime image
FROM microsoft/aspnetcore:2.0
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "mywebapp.dll"]

```

9. Add a **Docker** task to build container image for web API application.



10. Select an Azure subscription and click **Authorize** to allow Azure DevOps account access to the resources in the subscription.

Docker ⓘ

Version 1.*

Display name *

Build an image

Container Registry ^

Container registry type * ⓘ

Azure Container Registry

Azure subscription ⓘ | Manage ⓘ

https://devops-test-12345678900000000000000000000000 | Authorize | ⚡

ⓘ Click Authorize to configure an Azure service connection

Now configure this task's other properties as follows:

- **Display Name:** Build WebApi Container Image

- **Container Registry Type:** Azure Container Registry
- **Azure Container Registry:** Select the container registry you have created..
- **Command:** build
- **Docker File:** mywebapi/Dockerfile
- Unselect **Use default build context**
- **Build Context:** mywebapi
- **Image Name:** demo-webapi:\${Build.BuildId}

Display name *

Container Registry ^

Container registry type * ⓘ

Azure subscription ⓘ | Manage ⓘ

 (1) Scoped to subscription 'Visual Studio Enterprise'

Azure container registry ⓘ

 (1)

Commands ^

Command * ⓘ

Dockerfile * ⓘ

 ...

Arguments ⓘ

Use default build context ⓘ

Build context ⓘ

 ...

Image name * ⓘ

Qualify image name ⓘ

11. Now add another **Docker** task to push the image to ACR.

- **Display Name:** Push WebApi Container Image
- **Container Registry Type:** Azure Container Registry

- **Azure Container Registry:** Select the container registry you have created.
- **Command:** push

Image Name: demo-webapi:\${Build.BuildId}

The screenshot shows the 'Push WebAPI Container Image' task configuration in an Azure DevOps pipeline. It includes fields for Display name, Container Registry type (Azure Container Registry), Azure subscription, Azure container registry, Command (push), Arguments, and Image name (demo-webapi:\${Build.BuildId}). Checkboxes for Push multiple images, Qualify image name (which is checked), and Include source tags are also present.

Display name *

Push WebAPI Container Image

Container Registry ^

Container registry type * ⓘ

Azure Container Registry

Azure subscription ⓘ | Manage ↗

① Scoped to subscription 'Visual Studio Enterprise'

Azure container registry ⓘ

Commands ^

Command * ⓘ

push

Arguments ⓘ

Push multiple images ⓘ

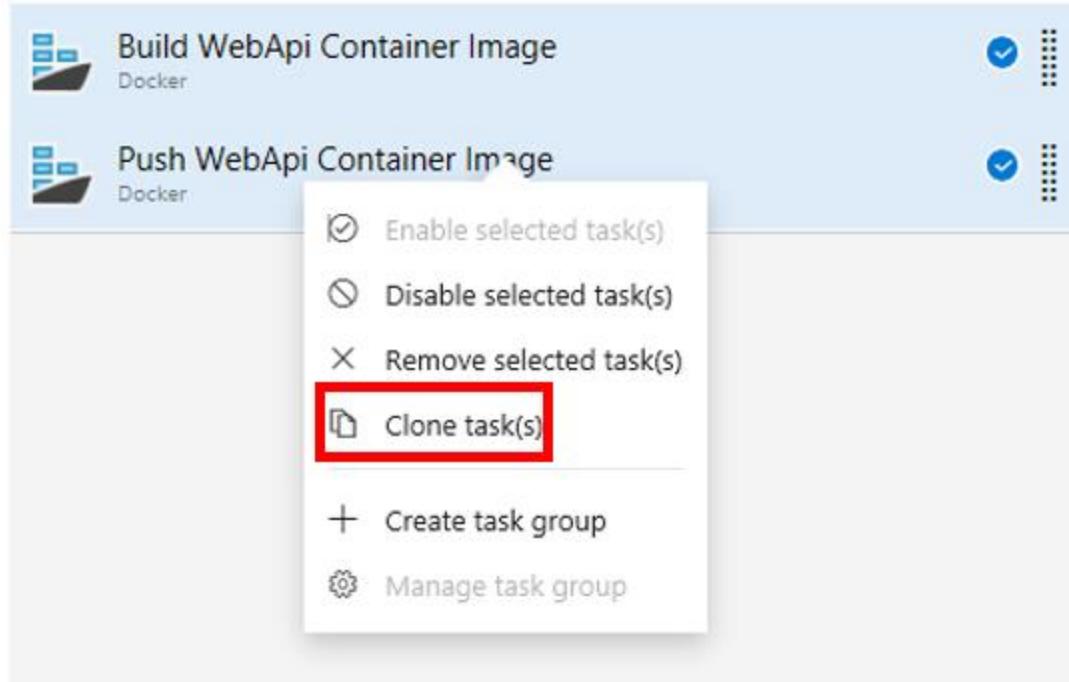
Image name * ⓘ

demo-webapi:\${Build.BuildId}

Qualify image name ⓘ

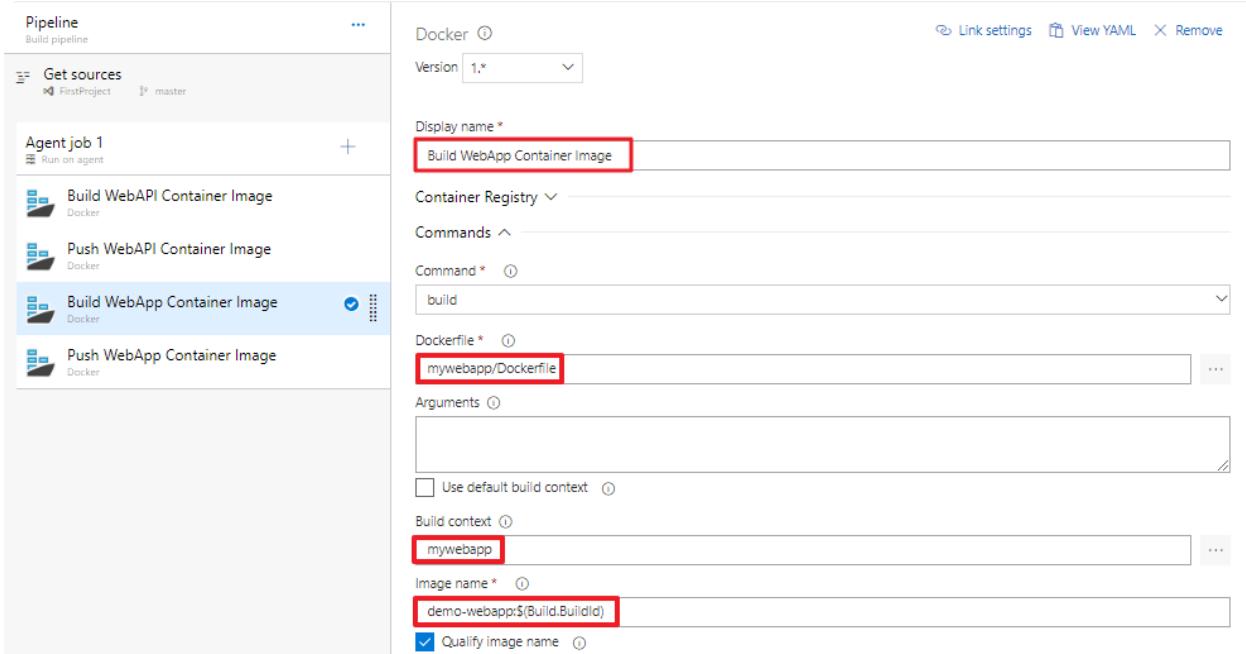
Include source tags ⓘ

12. You added two Docker tasks for webapi project. Now you will add another two Docker tasks for webapp project. Highlight your Build and Push Docker tasks with **select** and clicking, then **right** click and choose **Clone task(s)**.



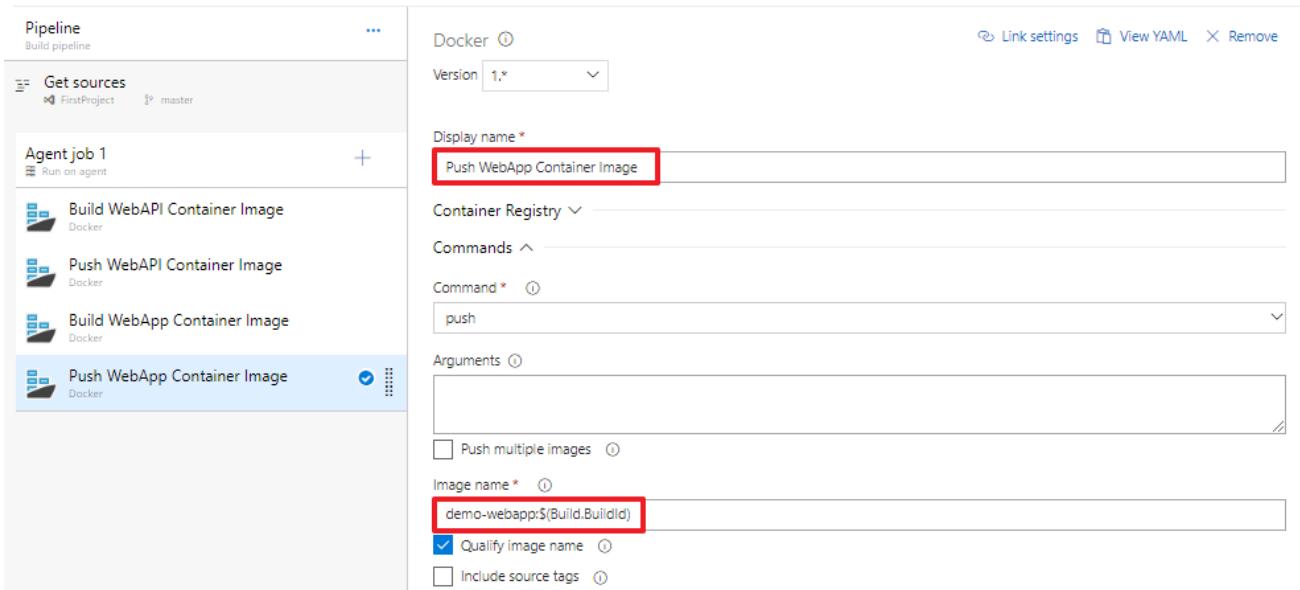
13. Change the **Build WebApi Container Image** copy task to the following attributes:

- **Display Name:** Build WebApp Container Image
- **Docker File:** mywebapp/Dockerfile
- **Build Context:** mywebapp
- **Image Name:** demo-webapp:\${Build.BuildId}

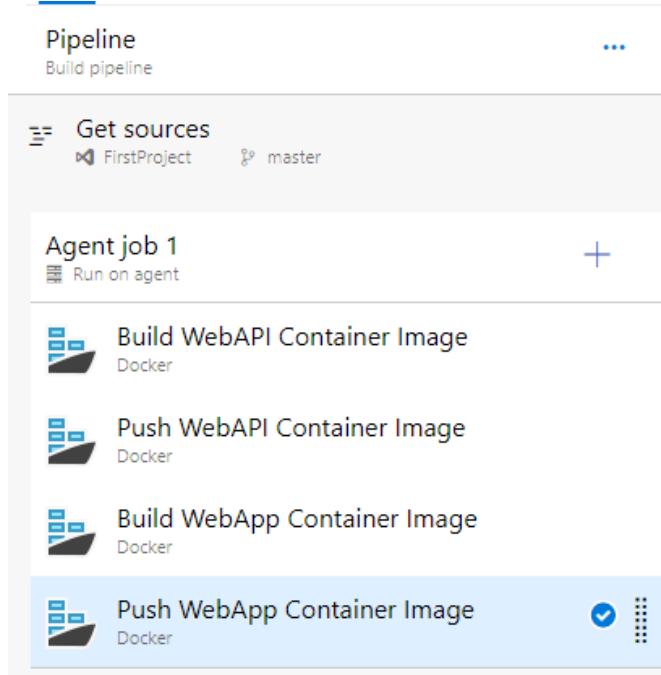


14. Change the **Push WebApi Container Image** copy task to the following attributes:

- **Display Name**: Push WebApp Container Image
- **Image Name**: demo-webapp:\${Build.BuildId}



15. Verify you have the following tasks so far:



Your images are now configured to be pushed to ACR and they will be tagged by build number, since you setup image name as follows: demo-webapp:\${Build.BuildId}. So, every time a build is triggered, its generated build number will be used to tag images in ACR.

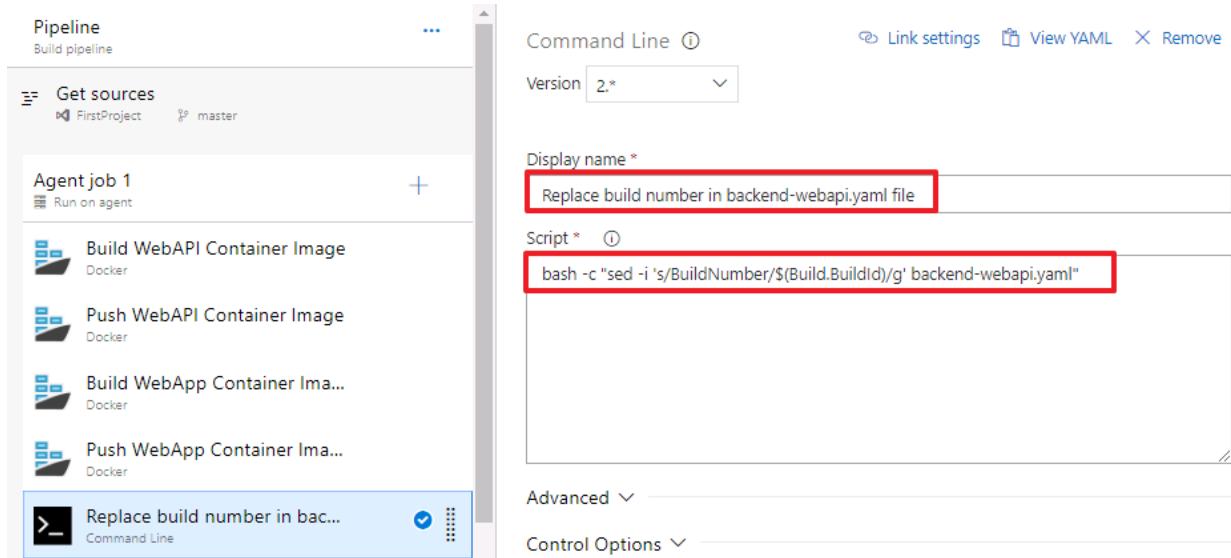
*To pick these images up from ACR and deploy to Kubernetes container orchestrator, you will need to use the same build number in release pipeline. Therefore you need to place it in Kubernetes deployment configuration files: **backend-webapi.yaml** and **frontend-webapp.yaml**.*

16. Add a **Command Line** task after all your other tasks.

A screenshot of the Azure DevOps Pipeline interface. The pipeline structure is identical to the one above, with "Get sources" and "Agent job 1" steps. In the bottom right corner, there is a search bar with the placeholder text "Command Line" and a "Refresh" button. The "Add tasks" button is visible, indicating where a new task can be added.

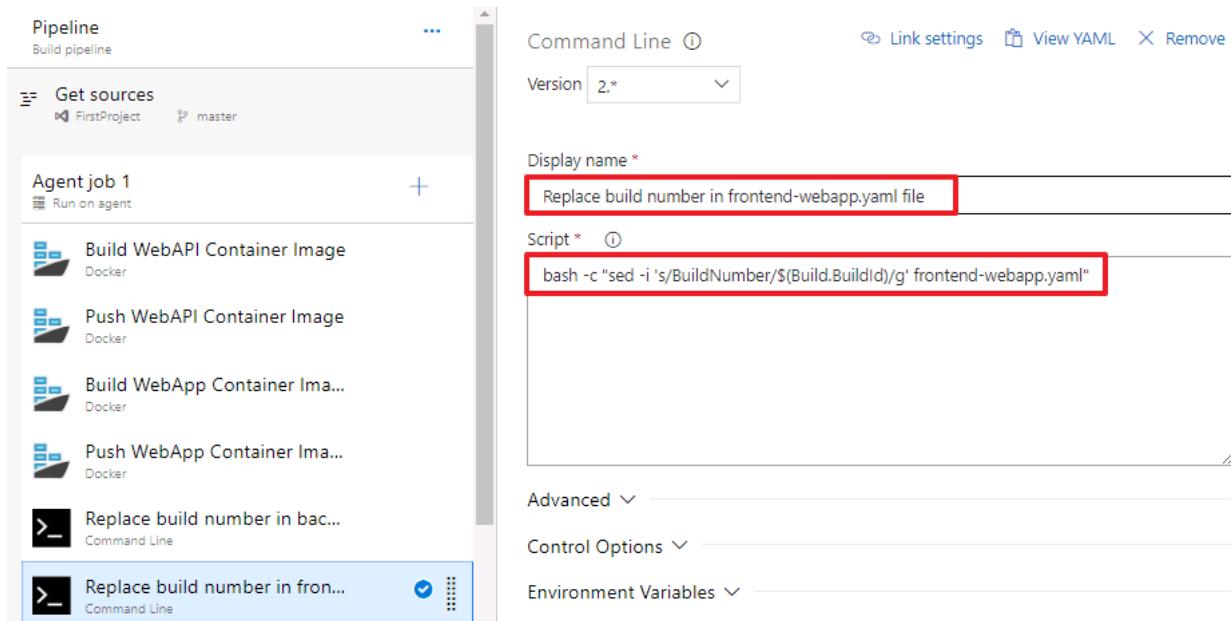
Configure the first **Command Line** task as follows:

- **Display Name:** Replace build number in backend-webapi.yaml file
- **Script:** bash -c "sed -i 's/BuildNumber/\${Build.BuildId}/g' backend-webapi.yaml" (it is recommended to write it manually to avoid copy past issues)



17. Clone your first command line task, and configure the second **Command Line task** as follows:

- **Display Name:** Replace build number in frontend-webapp.yaml file
- **Script:** bash -c "sed -i 's/BuildNumber/\${Build.BuildId}/g' frontend-webapp.yaml"



The previous task ensures that the Yaml file's content is updated with build number, now it is time to upload it as build artifact to be used in release pipeline later.

18. Add a new **Publish Build Artifacts** task and configure as follows:

- **Display Name:** Publish Artifact: frontend-webapp.yaml
- **Path to Publish:** frontend-webapp.yaml
- **Artifact Name:** frontend-webapp

The screenshot shows the Azure Pipelines interface for a pipeline named 'MyFirstProject'. On the left, there is a list of tasks under 'Agent job 1': 'Get sources', 'Build WebApi Container Image', 'Push WebApi Container Image', 'Build WebApp Container Image', 'Push WebApp Container Image', 'Replace build number in back...', 'Replace build number in front...', and 'Publish Artifact: frontend-web...'. The 'Publish Artifact' task is highlighted with a blue selection bar at the bottom. On the right, the configuration for this task is shown. It has a 'Display name' field set to 'Publish Artifact: frontend-webapp.yaml', a 'Path to publish' field set to 'frontend-webapp.yaml', and an 'Artifact name' field set to 'frontend-webapp'. The 'Path to publish' and 'Artifact name' fields are both highlighted with red boxes. The pipeline has a version '1.*' and is set to 'Save & queue'.

19. Clone your Publish Build Artifact task and modify the second one to be configured as follows:

- **Display Name:** Publish Artifact: backend-webapi.yaml
- **Path to Publish:** backend-webapi.yaml
- **Artifact Name:** backend-webapi

Finally change the build pipeline's name to **Web Apps on Linux - CI**. Click **Save & queue** to save and trigger a build (Note: you will see a pop-up asking for a comment, you can leave it blank and press Save & Queue on the popup to continue).

20. Your finished build should look like this (if you get an error please click on the step with the error and try debugging or the instructor will come by and help you):

Logs	Summary	Tests	Artifacts	Release	Edit	Queue	...
							Started: 10/4/2018, 5:42:09 AM
							... 3:51.020
Agent job 1 Job							
Pool: Hosted Ubuntu 1604 · Agent: Hosted Agent							
✓ Prepare job · succeeded							0.000
✓ Initialize Agent · succeeded							1.453
✓ Initialize job · succeeded							1.143
✓ Checkout · succeeded							7.134
✓ Build WebAPI Container Image · succeeded							2:41.960
✓ Push WebAPI Container Image · succeeded							10.856
✓ Build WebApp Container Image · succeeded							37.210
✓ Push WebApp Container Image · succeeded							2.477
✓ Replace build number in backend-webapi.yaml file · succeeded							0.243
✓ Replace build number in frontend-webapp.yaml file · succeeded							0.237
✓ Publish Artifact: frontend-webapp.yaml · succeeded							3.837
✓ Publish Artifact: backend-webapi.yaml · succeeded							3.733
✓ Post-job: Checkout · succeeded							0.547
✓ Report build status · succeeded							0.207

Optional: Import a build pipeline

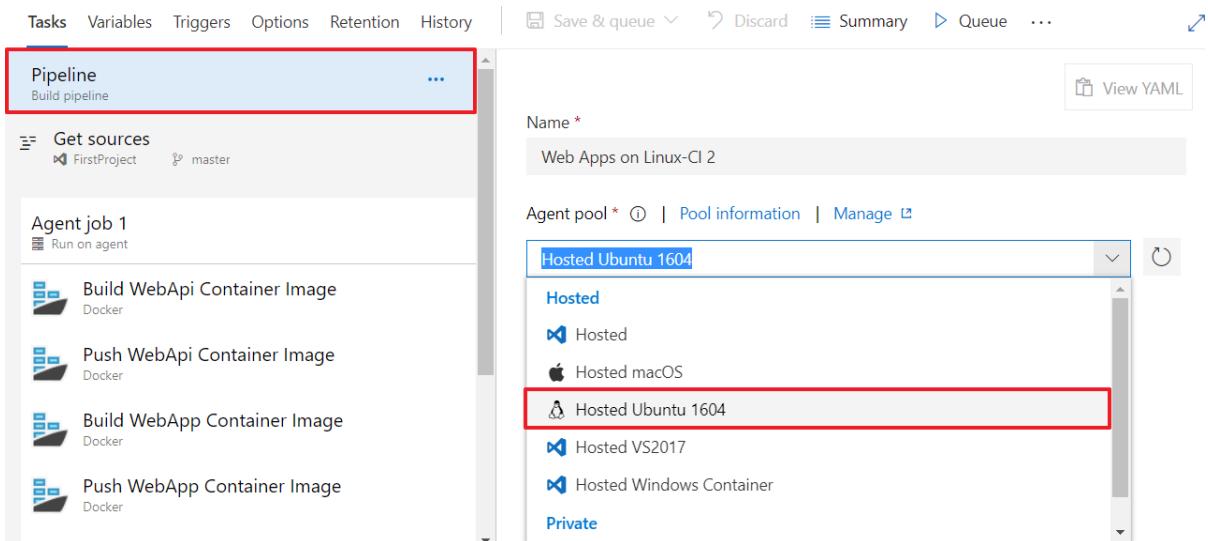
Skip this optional task if you already created the build pipeline.

1. Navigate to **Pipelines - Builds** and click **New - Import a pipeline**.

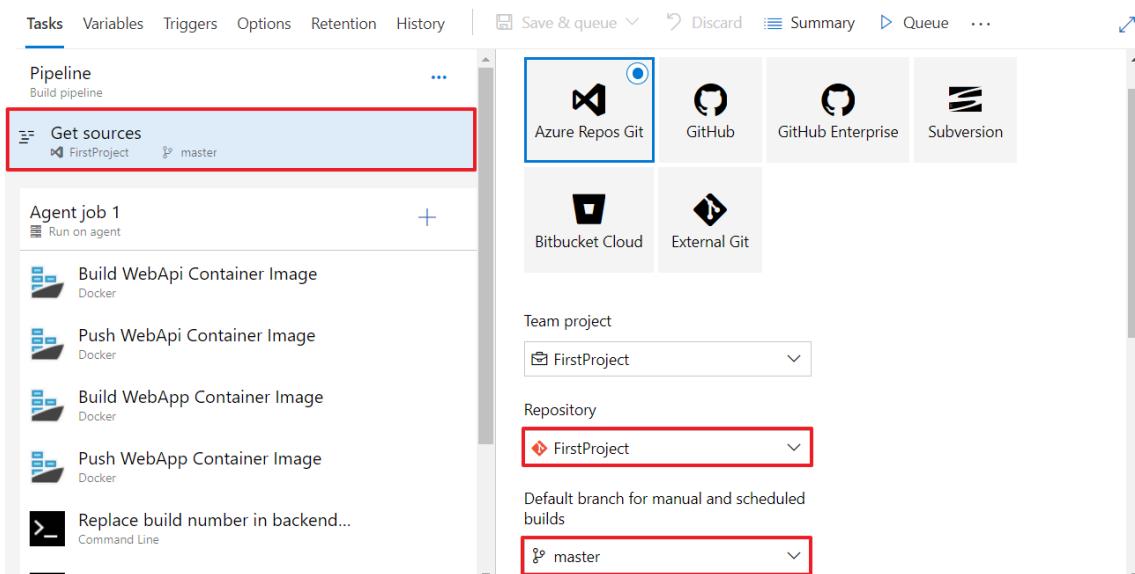
Note: Currently, we can only do this when at least one pipeline exists. So, you will need to create an empty pipeline first.

The screenshot shows the Azure DevOps interface for a project named 'JoudotFirstProject'. The left sidebar has a red box around the 'Builds' option under the 'Pipelines' section. The main area shows a search bar and a button for 'New'. A red box highlights the 'Import a pipeline' button, which is located next to the 'New build pipeline' button. The top navigation bar shows 'Azure DevOps' and the project name 'JoudotFirstProject'.

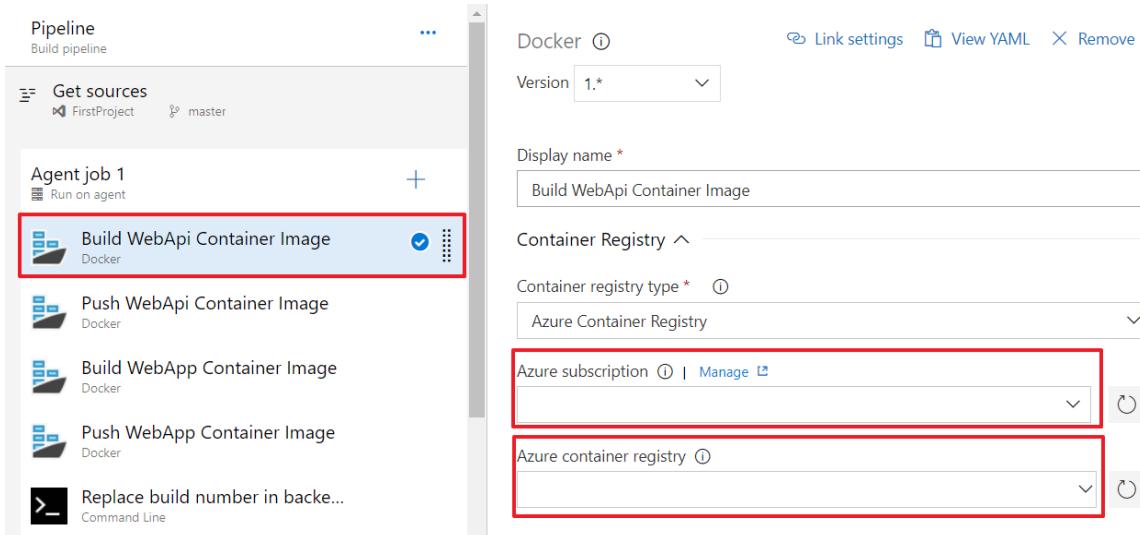
2. Select C:\labs\module6-ext\Web Apps on Linux-Cl.json and click Import
3. Select Hosted Ubuntu 1604 as an agent pool



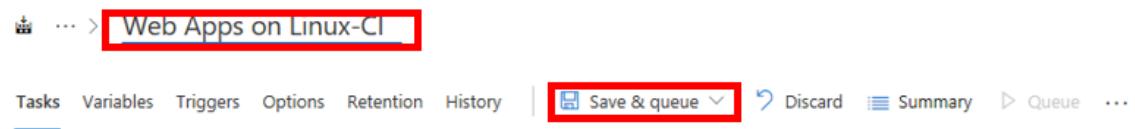
4. Click on Get sources and make sure that correct Git repository and branch is selected.



5. Select the docker tasks one by one and enter the subscription and ACR information.



6. Make sure the Build pipeline is named Web Apps on Linux-CI to stay aligned with the following screenshots. Click **Save & queue**.



Exercise 4: Create Release Pipeline for AKS Cluster

In this exercise, you will create a Release Pipeline which will pull container images from ACR (result of a previous build task), and then deploy these containers to Kubernetes cluster on AKS, managed Kubernetes service.

Tasks

If you want to avoid creating the pipeline step by step, you can skip this task and jump to [Optional: import a release pipeline](#).

1. Create Release Pipeline for Kubernetes Deployment

Build pipeline is used for compiling the application, building a container image, and then pushing container images to container registry. Release pipeline, on the other hand, is used to pull the container image from registry and then deploy to container cluster.

1. Navigate to <https://dev.azure.com/YourAccountName/YourProjectName> and go to Pipelines - Releases page and click New pipeline.

Azure DevOps

container-devops-lab / FirstProject / Pipelines / Releases*

Search

F FirstProject

Overview Boards Repos Pipelines Builds Releases*

Library Task groups Deployment groups Test Plans Artifacts

No release pipelines found

Automate your release process in a few easy steps with a new pipeline

New pipeline

2. Although there are templates about Kubernetes deployment, you are going to start with empty template. Click **Empty job** to create an empty release pipeline.

Select a template
Or start with an **Empty job**

Search

3. Click on release pipeline name and change it to **AKS-Release**. Then click on **Add an artifact** to link this release pipeline with build artifacts.

The screenshot shows the Azure DevOps Pipeline Editor interface. At the top, it says "All pipelines > AKS - Release". Below that is a navigation bar with tabs: Pipeline (which is selected), Tasks, Variables, Retention, Options, and History. The main area is divided into two sections: "Artifacts" on the left and "Stages" on the right.

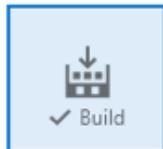
In the "Artifacts" section, there is a button labeled "+ Add" and a sub-section titled "Add an artifact". This sub-section is highlighted with a red box. Below it, there is a small icon and the text "Schedule not set".

In the "Stages" section, there is a single stage named "Stage 1" which contains "1 job, 0 task". There are edit and search icons next to the stage name.

4. On the **Add artifact** popup, select **Web Apps on Linux -CI** build pipeline. Finally select **Add**.

Add an artifact

Source type



Azure Repos...

Github

Team Found...

[4 more artifact types ▾](#)

Project * ⓘ

FirstProject

Source (build pipeline) * ⓘ

Web Apps on Linux-CI

Default version * ⓘ

Latest

Source alias ⓘ

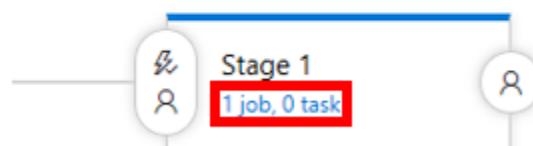
_Web Apps on Linux-CI

ⓘ The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of **Web Apps on Linux-CI** published the following artifacts: **frontend-webapp, backend-webapi.yaml**.

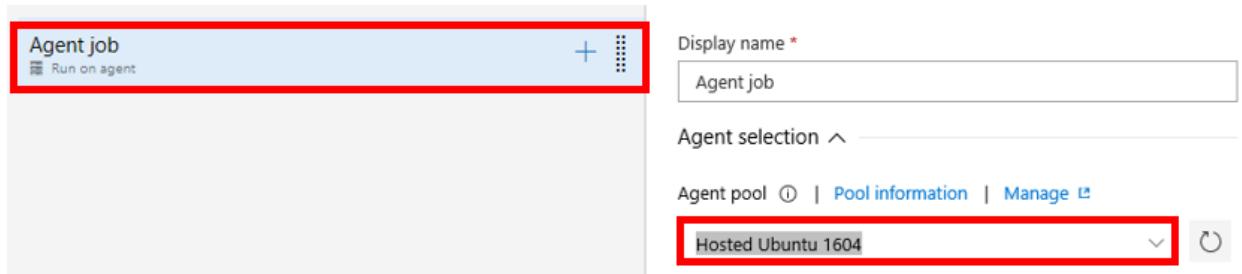
Add

5. Click "**1 job, 0 task**" link available inside the Stage 1.

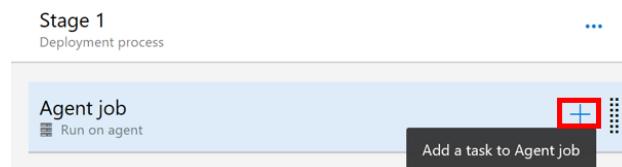
[Stages | + Add ▾](#)



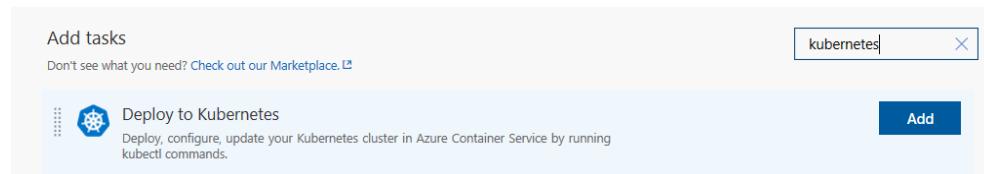
6. Click on the **Agent job** and then select **Hosted Ubuntu 1604** from the Agent pool dropdown.



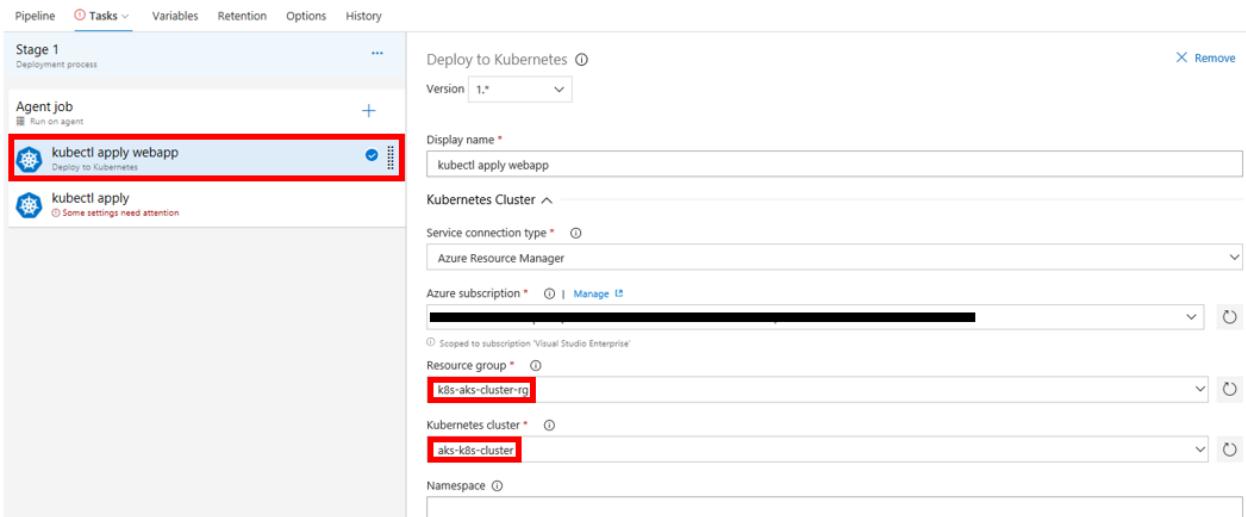
- Now you are ready to add tasks to deploy your containers to Kubernetes cluster. Click plus button on **Agent phase**, and then find **Deploy to Kubernetes** task.



Click **Add** to add the task. Make sure to add two copies of this task, one for deploying webapi and another for webapp.

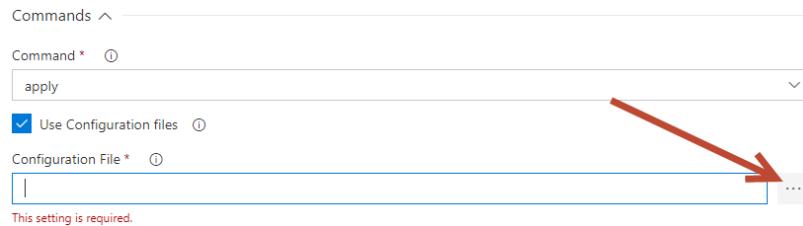


- Change the first task name to **kubectl apply webapp**. Then to add a connection to your AKS Kubernetes cluster by selecting the Azure subscription, resource group and cluster.



- In the **Commands** section:

- Check the box for **Use configuration files** and click on the 3 dots to the right of the input box to explore files



- **Select the main folder, select frontend-webapp folder, select frontend-webapp.yaml, then hit OK.**

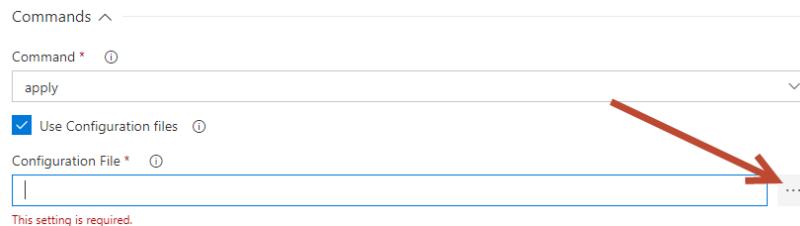
10. Now fill other parameters in the **Secrets** section of the task:

- **Type of secret:** DockerRegistry
- **Container Registry type:** Azure Container Registry
- **Azure Container Registry:** devopsLabACR (this registry should be the same as the one you have used in your build pipeline, since container images will be pulled from this registry)
- **Secret name:** acr (this is the secret which will be created using your ACR credentials, and which will be stored in Kubernetes cluster. You will notice that this is same as *imagePullSecrets* parameter in yaml deployment files, frontend-webapp.yaml and backend-webapi.yaml)
- **Force update secret:** Selected (This parameter will delete the secret and recreate it, and prevent getting errors caused by existing secret with same name)

A screenshot of the 'Secrets' configuration section. It includes fields for 'Type of secret' (set to 'dockerRegistry'), 'Container registry type' (set to 'Azure Container Registry'), 'Azure subscription' (with a note about being scoped to 'Visual Studio Enterprise'), 'Azure container registry' (redacted), 'Secret name' (set to 'acr'), and a checked 'Force update secret' checkbox. Red boxes highlight the 'dockerRegistry' selection, the 'Azure Container Registry' selection, and the 'acr' secret name.

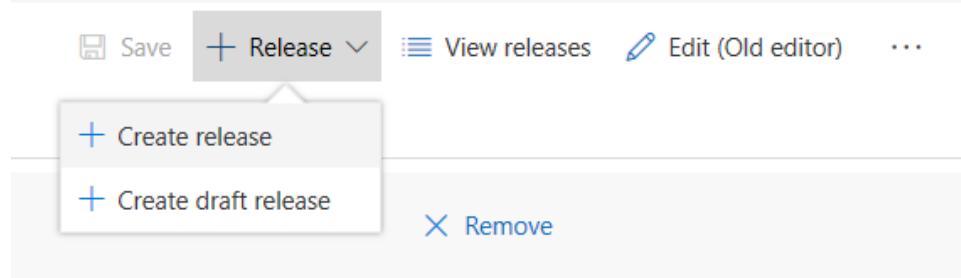
11. Similarly, configure second Kubernetes task:

- **Display Name:** kubectl apply webapi
- **Select your AKS cluster**
- Check the box for **Use Configuration files**, and then the below input box will open:



- Click on the 3 dots to the right of the input box to explore files, select the main folder, select backend-webapi folder, select **backend-webapi.yaml**, then hit OK.
- **Container Registry type:** Azure Container Registry
- **Azure Container Registry:** devopsLabACR (this registry should be the same as the one you have used in your build pipeline, since container images will be pulled from this registry)
- **Secret name:** acr (this is the secret which will be created using your ACR credentials, and which will be stored in Kubernetes cluster. You will notice that this is same as *imagePullSecrets* parameter in yaml deployment files, frontend-webapp.yaml and backend-webapi.yaml)
- **Force update secret:** Selected (This parameter will delete the secret and recreate it, and prevent getting errors caused by existing secret with same name)

12. Look at the top right of your page and you will see the buttons as in the screenshot below. First, Save the release pipeline. Then click on the Release button and choose "+ Create Release" to trigger a release.



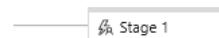
On **Create a new release** window select the Stage 1 from the dropdown. Keep all defaults and press **Create**.

Create a new release

AKS - Release

Pipeline ▾

Click on a stage to change its trigger from automated to manual.



Stages for a trigger change from automated to manual. ⓘ

✓ Stage 1



Artifacts ▾

Select the version for the artifact sources for this release

Release description

Go back to the Release page by clicking on the name of the created release. Click **Deploy**.

A screenshot of the Azure DevOps 'AKS - Release' page. The top navigation bar shows 'AKS - Release > Release-1'. The main interface is divided into two sections: 'Release' on the left and 'Stages' on the right. In the 'Release' section, it says 'Manually triggered by Julien Oudot 9/25/2018 2:08 PM' and lists artifacts: '_Web Apps on Linux-Cl 5 master'. In the 'Stages' section, there is one stage named 'Stage 1' with the status 'Not deployed'. Below the stage name is a 'Deploy' button, which is highlighted with a red rectangular box. Other buttons in the same row are 'Logs' and 'Edit release'.

Finally, press **Deploy** to kick off the deployment to AKS cluster.

Page Heading Ignored

The screenshot shows the 'Stage 1 Deploy release' screen in Azure Pipelines. At the top, it says 'Stage 1 Deploy release'. Below that is a navigation bar with 'Overview' (which is underlined), 'Commits', and 'Work Items'. A message 'To be deployed (Deploying for the first time)' and 'Release-1' is displayed. Under 'Artifacts', there is a list for '_Web Apps on Linux-CI' showing 5 files. There is a large empty 'Comment' box. At the bottom right are 'Deploy' and 'Cancel' buttons, with 'Deploy' being highlighted.

When it succeeds you will see a similar log as below:

The screenshot shows the 'Agent job' logs for a pipeline stage. The left sidebar shows a 'Deployment process' with one step 'Agent job' marked as 'Succeeded'. The main area shows the 'Agent job' logs for a 'Hosted Ubuntu 1604' agent. The log details the execution of several tasks: 'Initialize Agent' (succeeded, 0.557), 'Initialize job' (succeeded, 4.850), 'Download artifact - _Web Apps on Linux-CI - frontend-webapp' (succeeded, 5.733), 'Download artifact - _Web Apps on Linux-CI - backend-webapi' (succeeded, 3.926), 'kubectl apply webapp' (succeeded, 11.790), and 'kubectl apply webapi' (succeeded, 3.394). The log was started at 9/25/2018 10:12:03 AM and took 30.114 seconds.

13. Now your deployment in Azure Pipelines is completed, but the deployment in Kubernetes cluster is just beginning. Go to Kubernetes cluster dashboard and check deployment status. After waiting for a while, you will see that the deployment is completed.

NOTE: If you haven't already open the Kubernetes dashboard for AKS cluster use the following command to open it.

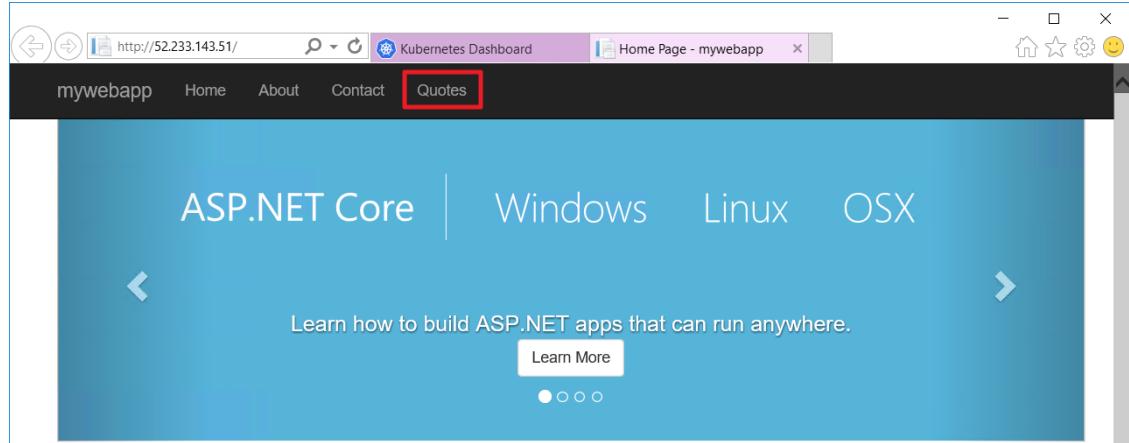
```
az aks browse --resource-group k8s-aks-cluster-rg-INITIALS --name aks-k8s-cluster
```

Name	Labels	Pods
demowebapp	app: demowebapp tier: frontend track: stable	1 / 1
demowebapi	app: demowebapi tier: backend track: stable	2 / 2

14. Go to **Service** page and check the status of your applications. Notice that frontend web application has an external endpoint, so you can reach it by clicking on IP address.

Name	Labels	Cluster IP	Internal endpoints	External endpoints
demowebapp	-	10.0.101.239	demowebapp:80... demowebapp:30...	52.233.143.51:80
demowebapi	-	10.0.199.93	demowebapi:900... demowebapi:0 T...	-
kubernetes	component: a... provider: kuber...	10.0.0.1	kubernetes:443 T... kubernetes:0 TCP	-

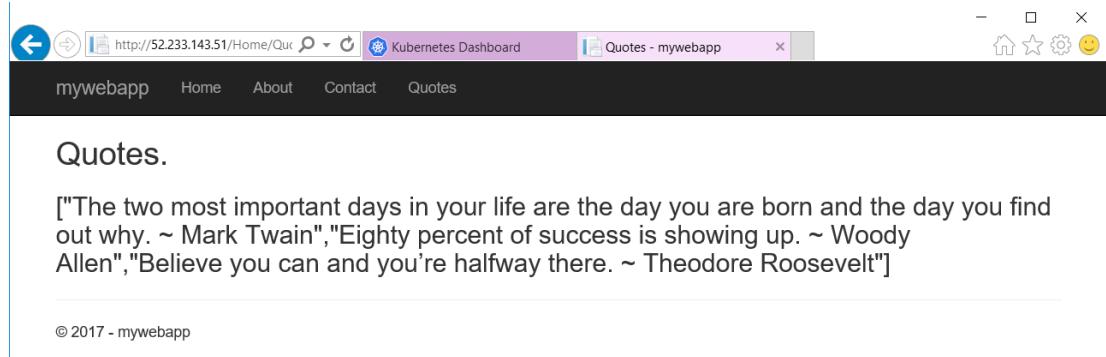
15. When the web site is loaded, click on **Quotes** link.



Notice that frontend web application is successfully fetching quotes by connecting to backend web API.

16. If you are using an Azure Pass, please delete the deployment for the web app and web api project, along with the Services for each in order to release enough cores for the next exercise.

You can check HomeController.cs in webapp project, and see that the frontend application is connecting to the backend Web API with the following URL <http://demowebapi:9000/api/quotes>. The URL's hostname is "demowebapi" which is the service name of the backend application configured in backend-webapi.yaml file.



Optional: Import a release pipeline

Skip this optional task if you already created the release pipeline.

1. Navigate to **Pipelines – Releases** and click **New – Import a pipeline**.

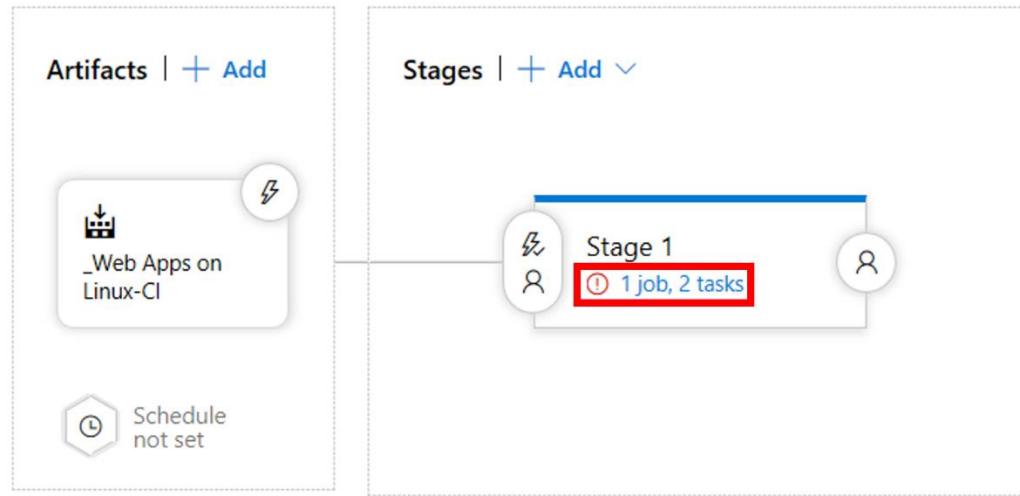
Note: it seems that we can only do that when at least one pipeline exists. So you will need to create an empty pipeline first.

The screenshot shows the Azure DevOps interface for a project named 'JoudotFirstProject'. The left sidebar has 'Pipelines' selected and 'Releases' highlighted with a red box. The main area shows 'Release pipelines' with tabs for 'Active' and 'All pipelines'. A dropdown menu is open over the 'New' button, with 'Import a pipeline' highlighted by a red box. Below the menu, there's a search bar and a message 'No favorites yet'. A 'Recent' section is shown below, and a 'New release pipeline' card is visible.

1. Select **C:\labs\module6-ext\AKS - Release-CD.json** and click **Import**
2. Make sure that the input Artifact is valid by deleting the default one: click on the Artifact Name and then **Delete**. Finally, click on **Add an Artifact** to add the artifact coming from your new build.

The screenshot shows the 'Artifacts' section of a release pipeline. It includes a header 'Artifacts' with a 'Add' button highlighted by a red box. Below is a list of artifacts: '_Web Apps on Linux-Cl' (with a delete icon) and 'Schedule not set' (with a circular icon).

3. Click on the list of tasks under **Stage 1**



4. Under agent job, select **Hosted Ubuntu 1604** as an agent pool

This screenshot shows the 'Tasks' tab for the 'Stage 1' deployment process. It lists two tasks: 'Agent job' (with a red box around it) and 'kubectl apply webapp'. To the right, a modal window titled 'Agent job' is open, showing 'Display name *' set to 'Agent job'. In the 'Agent pool' dropdown, 'Hosted Ubuntu 1604' is selected and highlighted with a red box. Other options like 'Pool information' and 'Manage' are visible.

5. Select the Kubernetes tasks one by one and enter the subscription, resource group and cluster information. Also enter the ACR information under **Secrets**

This screenshot shows the 'Tasks' tab for the 'Stage 1' deployment process. It lists two tasks: 'Agent job' (with a red box around it) and 'kubectl apply webapp'. To the right, a modal window titled 'Deploy to Kubernetes' is open. It shows 'Version' set to '1.*', 'Display name *' set to 'kubectl apply webapp', and 'Kubernetes Cluster' expanded. Under 'Service connection type', 'Azure Resource Manager' is selected. The 'Azure subscription' dropdown is open and highlighted with a red box. Other fields include 'Resource group' set to 'k8s-aks-cluster-rg' and 'Kubernetes cluster' set to 'aks-k8s-cluster'.

Secrets ^

Type of secret * ⓘ
dockerRegistry

Container registry type * ⓘ
Azure Container Registry

Azure subscription ⓘ | Manage ↗
[REDACTED] ⚙️ ⏪ ⏴
 ⓘ Scoped to subscription 'Visual Studio Enterprise'

Azure container registry ⓘ
[REDACTED] ⚙️ ⏪ ⏴

Secret name ⓘ
acr

Force update secret ⓘ

6. Once all the Kubernetes tasks are updated, “Some settings need attention” should disappear and you can save the pipeline.

Go back to [Step 12](#) of the previous task to start a release and deploy the application to your kubernetes cluster. Then, you will check status of the deployment on kubernetes dashboard and test the application.

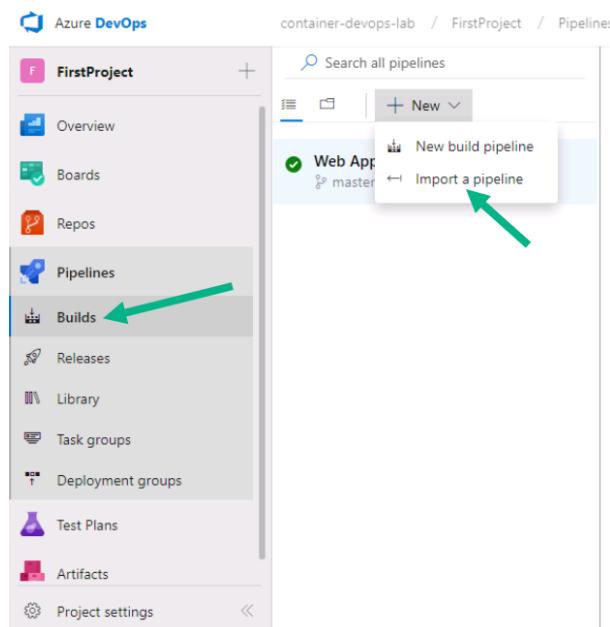
Exercise 5: Create Build Pipeline for Windows Containers

In this task, you are going to create a build pipeline which will be executed on Windows agent and will produce two Windows container images of your applications. These images will be tagged automatically with appropriate build number and then pushed into the Azure Container Registry (ACR). Instead of creating the pipeline from scratch, we will import JSON definition of the pipeline.

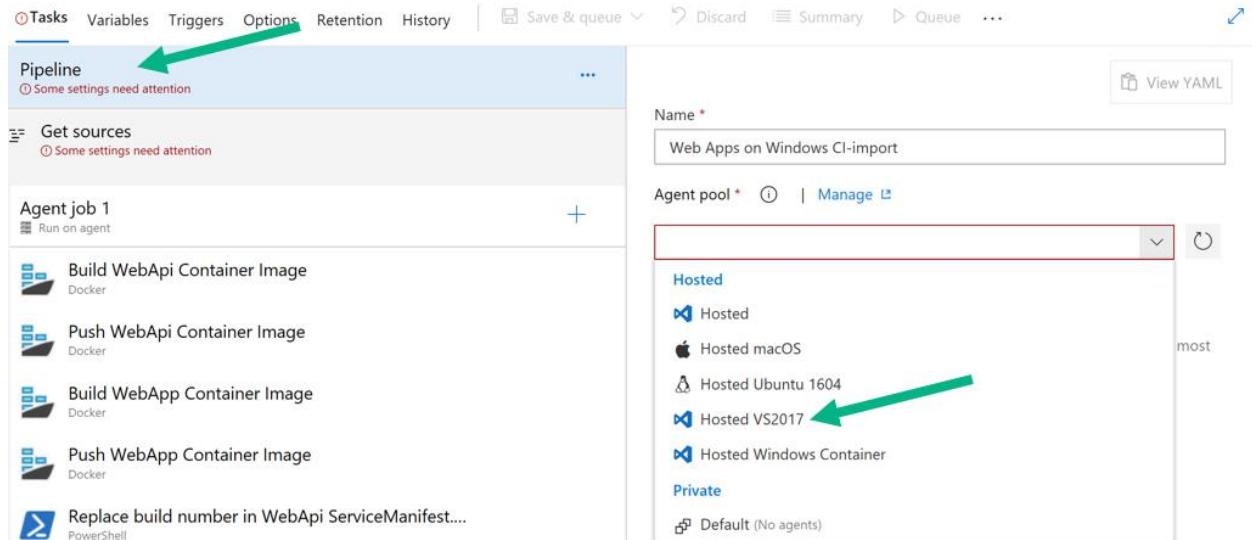
Tasks

Import a build pipeline

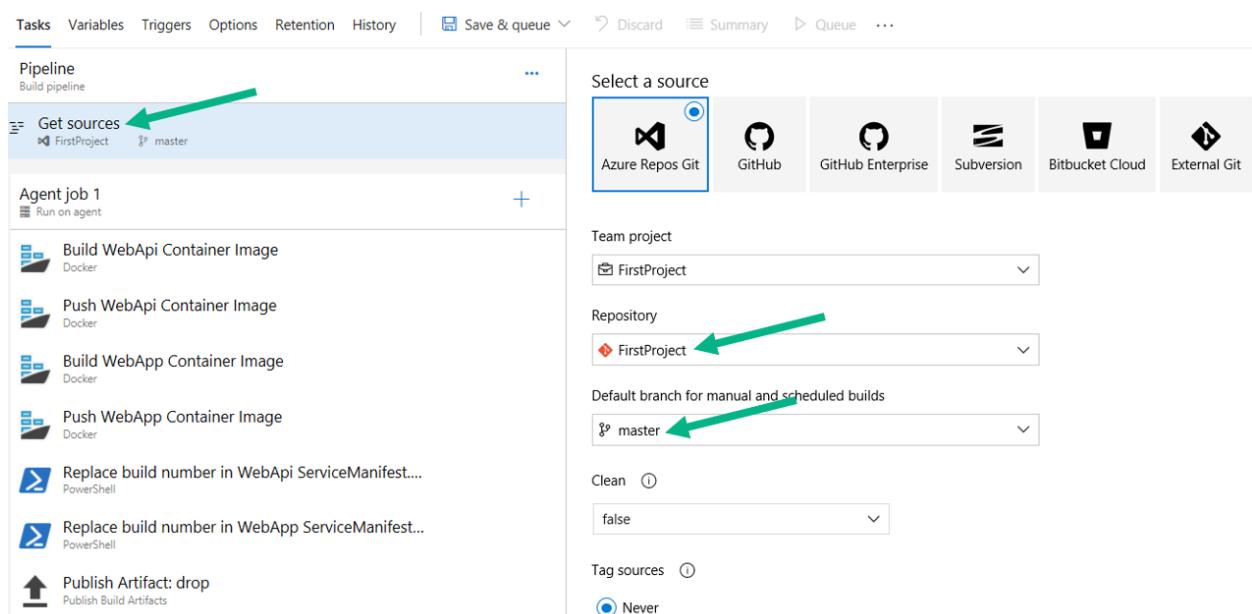
1. Navigate to **Pipelines – Builds** and click **New – Import a pipeline**.



2. Select **C:\labs\module6-ext\Web Apps on Windows-Cl.json** and click **Import**
3. Select **Hosted VS2017** as an agent pool. This is a Windows based agent.



- Click on **Get sources** and make sure that correct Git repository and branch is selected.



- Select **all docker tasks** one by one and enter the **subscription and ACR** information.

Page Heading Ignored

Pipeline
Build pipeline

Get sources
Agent Job 1
Run on agent
Build WebApi Container Image
Push WebApi Container Image
Build WebApp Container Image
Push WebApp Container Image
Replace build number in WebApi ServiceManifest...
Replace build number in WebApp ServiceManifest...
Publish Artifact: drop

Docker
Version 1.*
Display name * Build WebApi Container Image
Container Registry ^
Container registry type * Docker
Azure Container Registry
Azure subscription ⓘ | Manage ↗
Azure container registry ⓘ
Commands ^
Command * build

Link settings View YAML Remove

6. Make sure the Build pipeline is named **Web Apps on Windows-CI** to stay aligned with the following screenshots. Click **Save & queue**.

... > Web Apps on Windows-CI

Tasks Variables Triggers Options Retention History | Save & queue

7. Your finished build should look like this (if you get an error please click on the step with the error and try debugging or the instructor will come by and help you):

Logs	Summary	Tests	Release	Artifacts	Edit	Queue	...
Agent job 1 Job							
Pool: Hosted VS2017 · Agent: Hosted Agent							
	Started: 11/9/2018 7:44:07 PM						
						...	14m 48s
✓ Prepare job · succeeded	<1s						
✓ Initialize Agent · succeeded	1s						
✓ Initialize job · succeeded	6s						
✓ Checkout · succeeded	11m 9s						
✓ Build WebApi Container Image · succeeded	8s						
✓ Push WebApi Container Image · succeeded	3m 9s						
✓ Build WebApp Container Image · succeeded	6s						
✓ Push WebApp Container Image · succeeded	2s						
✓ Replace build number in WebApi ServiceManifest.xml · succeeded	1s						
✓ Replace build number in WebApp ServiceManifest.xml · succeeded	1s						
✓ Publish Artifact: drop · succeeded	1s						
✓ Post-job: Checkout · succeeded	<1s						
✓ Report build status · succeeded	<1s						

Exercise 6: Create Release Pipeline for Service Fabric Windows Cluster

In this exercise, you will create a Release Pipeline which will pull container images from ACR (result of a previous build task), and then deploy these containers to Windows cluster on Service Fabric.

Build pipeline is used for compiling the application, building a container image, and then pushing container images to container registry. Release pipeline, on the other hand, is used to pull the container image from registry and then deploy to container cluster. Instead of creating the pipeline from scratch, we will import JSON definition of the pipeline.

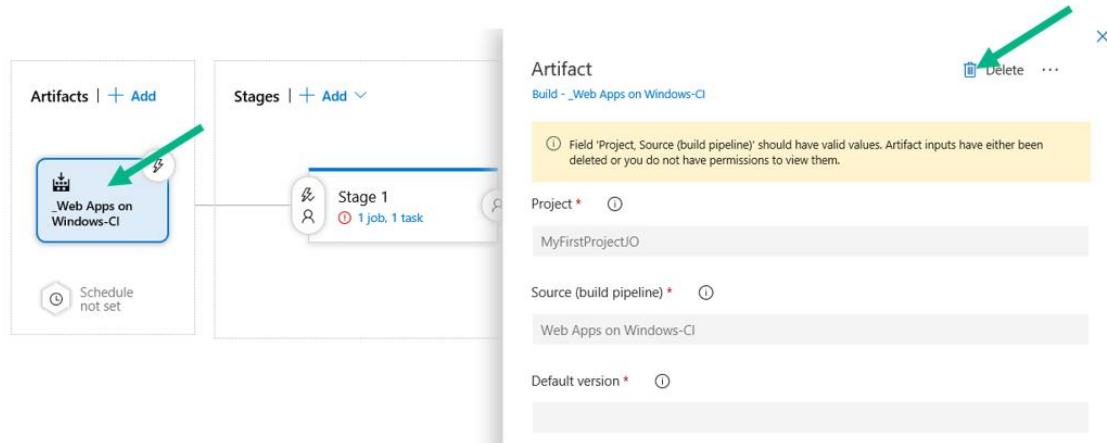
Tasks

Import a release pipeline

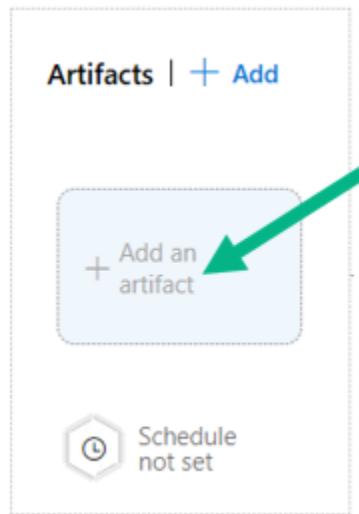
1. Navigate to **Pipelines – Releases** and click **New – Import a pipeline**.

The screenshot shows the Azure DevOps interface for a project named 'FirstProject'. The left sidebar lists various options: Overview, Boards, Repos, Pipelines (selected), Builds, Releases (highlighted with a green arrow), Library, Task groups, Deployment groups, Test Plans, and Artifacts. The main area displays a pipeline named 'AKS - Release' with 'Stage 1' checked. A search bar at the top right says 'Search all pipelines'. Below it, a 'New' button dropdown menu is open, showing 'New release pipeline' and 'Import release pipeline' (also highlighted with a green arrow).

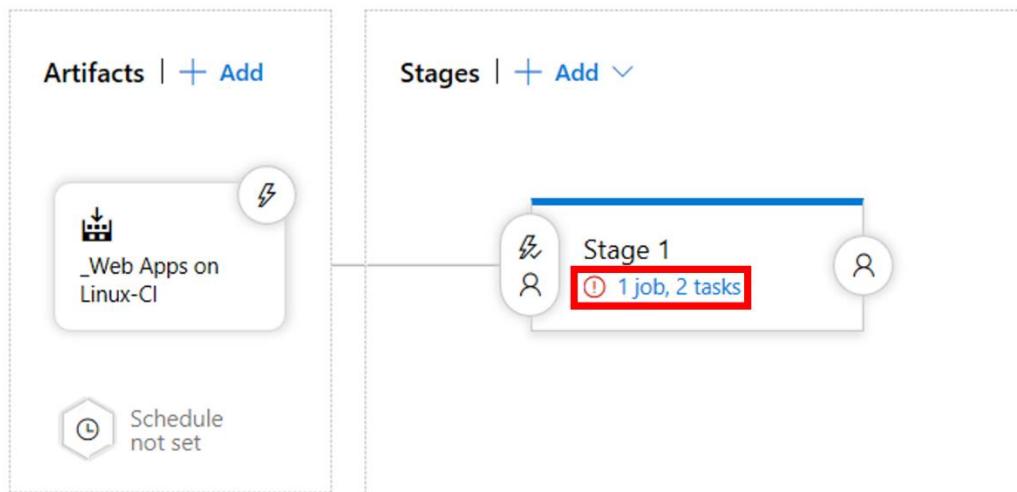
2. Select **C:\labs\module6-ext\SF - Release-CD.json** and click **Import**
3. Make sure that the input Artifact is valid by deleting the default one: click on the Artifact Name and then **Delete**.



- Finally, click on **Add an Artifact** to add the artifact coming from your **Web Apps on Windows-CI** build.



- Click on the list of tasks under **Stage 1**



6. In **Stage 1**, click on **New** to add a new cluster connection.

The screenshot shows the Azure Pipelines interface. At the top, there are tabs: Pipeline, Tasks (which is selected), Variables, Retention, Options, and History. Below the tabs, there's a list of stages: Stage 1, Agent job, and Deploy Service Fabric Application. Stage 1 has a red warning icon and the text "Some settings need attention". To the right of Stage 1, there's a "..." button. The Deploy Service Fabric Application stage also has a red warning icon and the same text. On the far right, there are buttons for "Stage name", "Parameters", "Application package", "Cluster connection" (which is highlighted with a red box and has a red note "This setting is required."), and "Manage". A green arrow points from the left towards the Stage 1 header. Another green arrow points from the bottom right towards the "New" button in the Cluster connection dropdown.

7. In the **Add Service Fabric service connection** window fill in the following parameters:

- **Connection name:** SF-Connection (it can be any name you choose)
- **Cluster Endpoint:** tcp://<cluster-name>.eastus.cloudapp.azure.com:19000 (the cluster name will be what you chose in the module 5. If you choose as instructed, it should be containers-lab-initials)
- **Service Certificate Thumprint:** copy the value of the **Security** section of the **Service Fabric** cluster. You can find it from portal.azure.com.

The screenshot shows the 'containers-lab - Security' page in the Azure portal. The left sidebar lists various settings like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Node types, Nodes, Applications, and Security. The Security link is highlighted with a green arrow. The main content area shows the Security mode set to Secure. Under Cluster certificates, there is one primary certificate with the following details:

CERTIFICATE TYPE	COMMON NAME	THUMBPRINT
Primary	N/A	95B543483F7F427CDB93345D20C0C36B8827BC874

Under Client certificates, it says 'No client certificates found.' In the Azure Active Directory section, it says 'No Azure Active Directory tenants found.'

- **Client Certificate:** You can use the following command to obtain a Base64-encoded representation of the client certificate:

```
[System.Convert]::ToBase64String([System.IO.File]::ReadAllBytes("C:\labs\module5\certificateFile.pfx")) > output.txt
```

```
PS C:\labs\module5> [System.Convert]::ToBase64String([System.IO.File]::ReadAllBytes('C:\labs\module5\containers-lab201811091411.pfx')) > output.txt
PS C:\labs\module5>
```

Copy the content of **output.txt** into **Client Certificate** field (do it from notepad to make sure this is a clean copy).

- **Password:** empty (because our certificate does not have any password)

Add Service Fabric service connection

● Certificate Based ○ Azure Active Directory credential ○ Others

Connection name: SF-Connection

Cluster Endpoint: tcp://containers-lab.eastus.cloudapp.azure.com:19000

Server Certificate Thumbprint(s): 95B543483F7F427CDB93345D20C0C36B827BC874

Client Certificate: DHLZGCdrSUsrSKp74ZZiBBTfQ8guhq02StA5ArIw031un
dxIQ==

Password:

[Learn More](#)

OK **Close**

- Under **Agent job**, select **Hosted VS2017** as an agent pool. This is a Windows agent.

Pipeline Tasks **Tasks** Variables Retention Options History

Stage 1 Deployment process

Agent job **Run on agent**

Deploy Service Fabric Application Service Fabric Application Deployment

Agent job **Hosted VS2017**

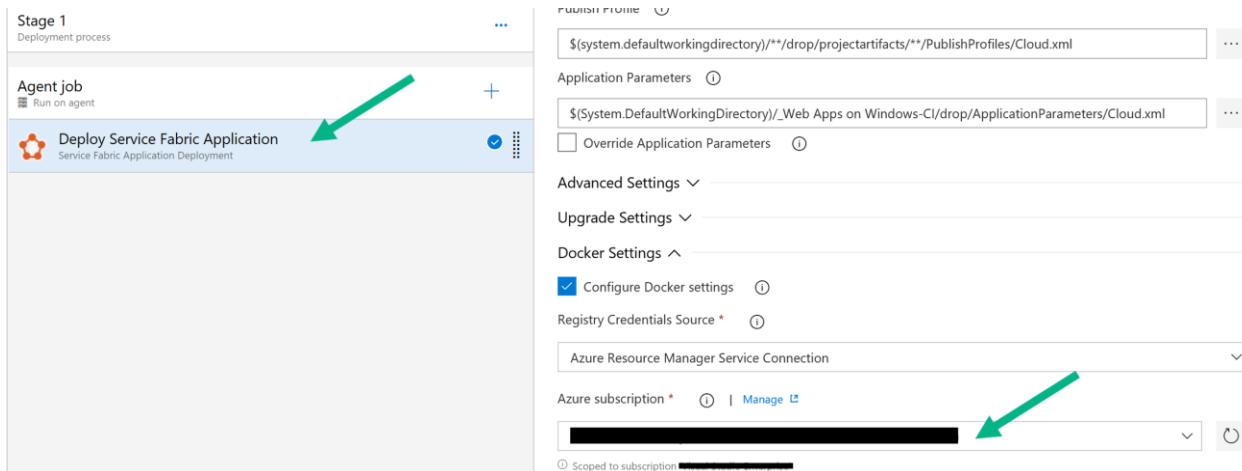
Display name * Agent job

Agent selection Hosted VS2017

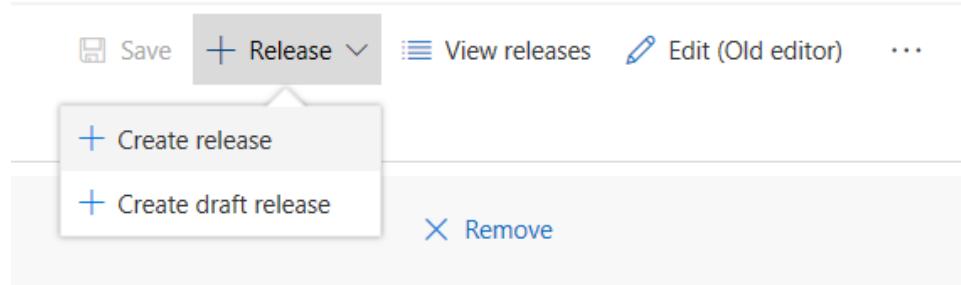
Agent pool **Hosted VS2017**

- Select the **Deploy Service Fabric Application** task and under **Docker Settings** select **Azure Resource Manager Service Connection** and then select your Azure subscription. This will allow the cluster to pull the container images located in the Azure registry.

Page Heading Ignored



10. Look at the top right of your page and you will see the buttons as in the screenshot below. First, **Save** the release pipeline. Then click on the **Release** button and choose “**+ Create Release**” to trigger a release.



On the **Create a new release** dialog window select the **Stage 1** from the dropdown, select the last artifact version and click **Create**.

Create a new release
SF - Release - Copy

Pipeline ^
Click on a stage to change its trigger from automated to manual.

Stage 1

Artifacts ^
Select the version for the artifact sources for this release

Source alias	Version
_Web Apps on Windows-CI	11

Release description

Create Cancel

Go back to the Release page by clicking on the name of the created release at the top of the screen. Click **Deploy**.

SF - Release - Copy > Release-1 ^

Pipeline Variables History | + Deploy ▾ Cancel Refresh Edit release ...

Release

Manually triggered
by Julien Oudot
11/12/2018 4:23 PM

Artifacts

_Web Apps on Window...
11
 master

Stages

Stage 1
 Not deployed

Deploy Logs

Finally, press **Deploy** to kick off the deployment to Service Fabric cluster.

When it succeeds you will see a similar log as below:

Azure Pipelines Agent job logs for a deployment process. The job has succeeded and contains four steps:

- Initialize Agent · succeeded <1s
- Initialize job · succeeded 2s
- Download artifact - _Web Apps on Windows-Cl - drop · succeeded 3s
- Deploy Service Fabric Application · succeeded 2 warnings 24s

11. Now your deployment in Azure Pipelines is completed, but the deployment in Service Fabric cluster is just beginning. Go to Service Fabric explorer and check the deployment status. After waiting for a while, you will see that the deployment goes from **InBuild** to **Ready** state.

Microsoft Azure Service Fabric Explorer interface. The URL bar shows a warning: **Not secure | https://containers-lab.eastus.cloudapp.azure.com:19080/Explorer/index.html#/apptype/SFCoreAppType/app...**. The main view shows the deployment status for an instance:

Instance 131865320687722702	
ESSENTIALS DETAILS	
Id 131865320687722702	Node Name _nt1vm_0
Health State OK	Status InBuild

The left sidebar shows the cluster structure:

- Cluster
 - Applications
 - SFCoreAppType
 - fabric/SFCoreApp
 - fabric/SFCoreApp/BackEnd_WebAPI
 - dd0e5b77-648e-4349-8f3d-63de2a9810...
 - _nt1vm_0
 - _nt1vm_1
 - _nt1vm_2
 - fabric/SFCoreApp/FrontEnd_WebApp
 - 3887a322-4576-4759-e6a-cd49ae9b1b...
 - _nt1vm_0
 - _nt1vm_1
 - _nt1vm_2

Microsoft Azure Service Fabric Explorer

Instance 131866203208825636

ESSENTIALS

Id 131866203208825636	Node Name _nt1vm_1
Health State OK	Status Ready

ADDRESS

Endpoints

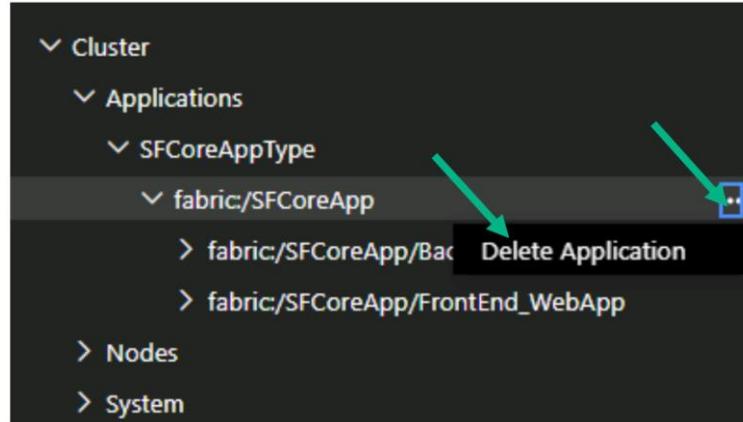
Front End Web App Type Endpoint	10.0.0.5:80
---------------------------------	-------------

If you still have the IIS application deployed, you will see an error saying that the port 80 is not available. To fix that, click on the thee dots next to the **fabric:/SFIsApplication** and **Delete Application**.

- Cluster
- Applications
 - SFCoreAppType
 - SFIsApplicationType
 - fabric:/SFIsApplication

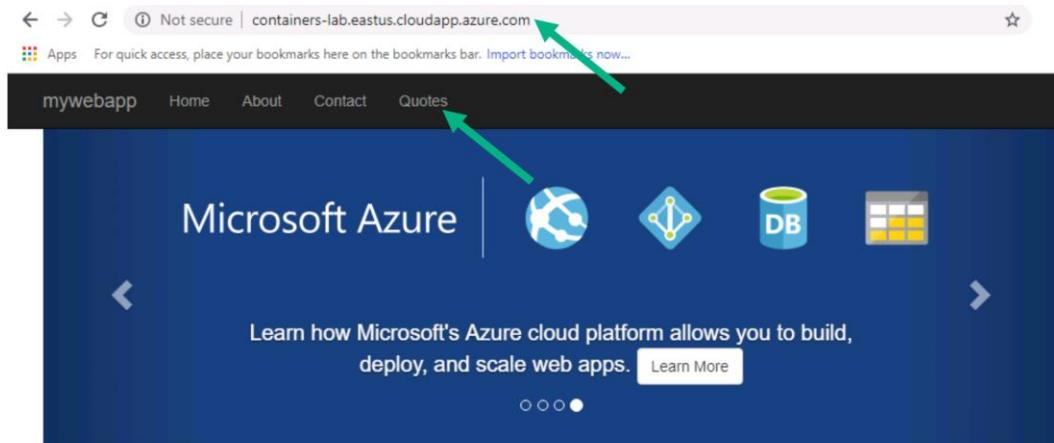
Delete Application

If there is an error while deploying the content, you will need to remove the application from the cluster to be able to deploy again. Do do it, click on the application name **fabric:/SFCoreApp** and **Delete Application**.



12. Once the application is ready, open any browser and navigate to `http://<cluster-name>.eastus.cloudapp.azure.com`

13. When the web site is loaded, click on **Quotes** to display the quotes.



Application uses

- Sample pages using ASP.NET Core MVC
- Bower for managing client-side libraries
- Theming using Bootstrap

How to

- Add a Controller and View
- Manage User Secrets using Secret Manager.
- Use logging to log a message.
- Add packages using NuGet.

Overview

- Conceptual overview of what is ASP.NET Core
- Fundamentals of ASP.NET Core such as Startup and middleware.
- Working with Data
- Security

Run & Deploy

- Run your app
- Run tools such as EF migrations and more
- Publish to Microsoft Azure Web Apps

Notice that frontend web application is successfully fetching quotes by connecting to backend web API.