Table of Contents

Lab: Module 6 - DevOps with Containers

Duration: 120 minutes

Exercise 1: Setup Azure DevOps Account

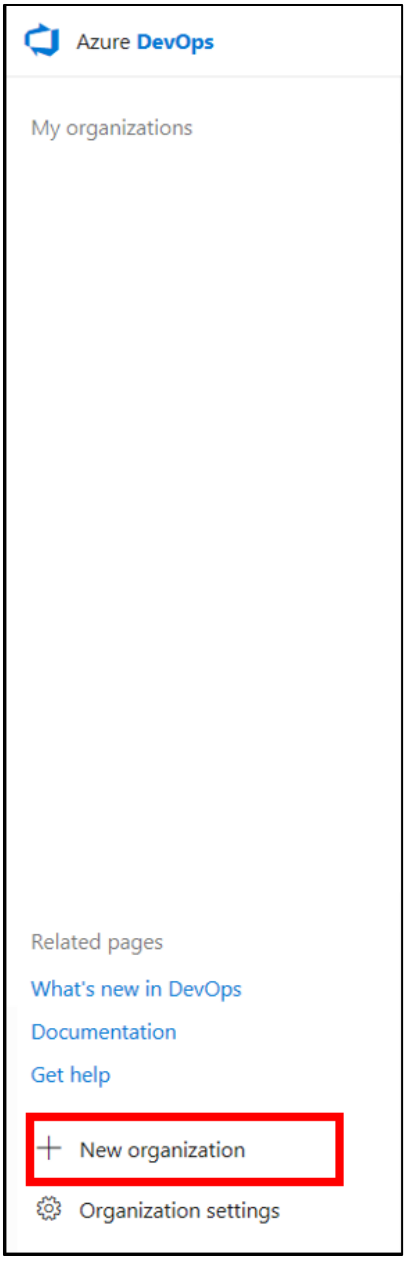
In this exercise, you are going to create an Azure DevOps account which contains toolset to create CI/CD pipeline. If you already have an Azure DevOps account, you can skip this exercise.

Tasks

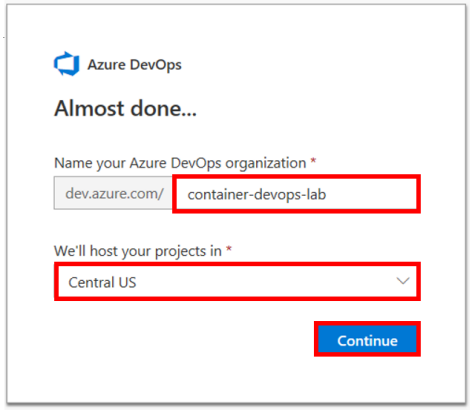
1. **Create Azure DevOps Organization and Project**

    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.

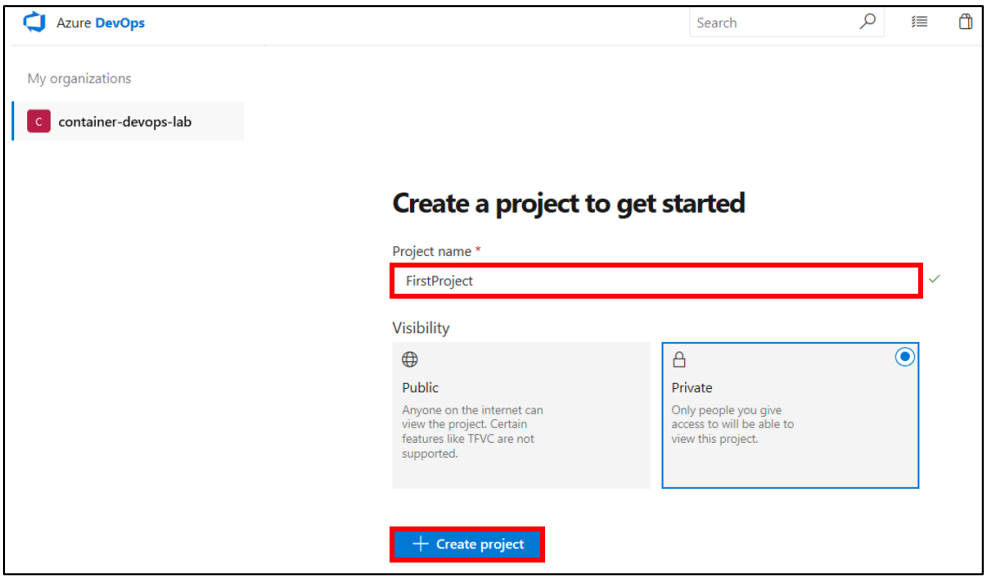
    2. On Dev Portal, click **New Organization** button to add a new organization.



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.



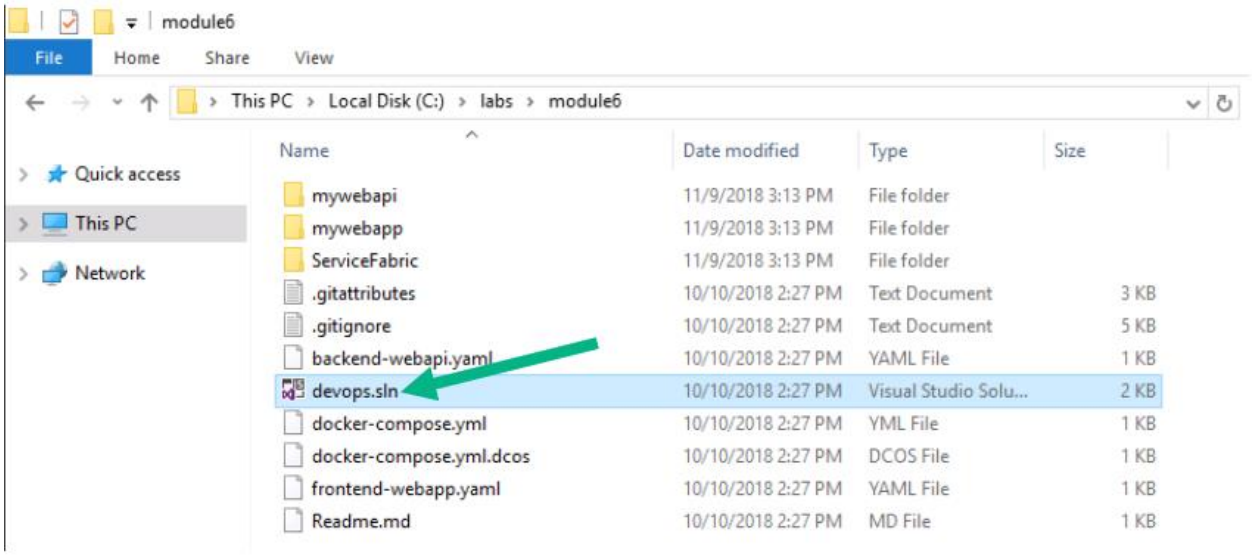
Exercise 2: Setup Project and Azure Container Registry (ACR)

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.

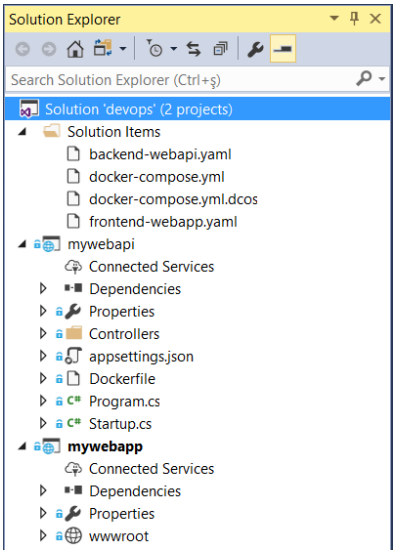
Tasks

1. **Examine and Build Sample Application in Visual Studio**

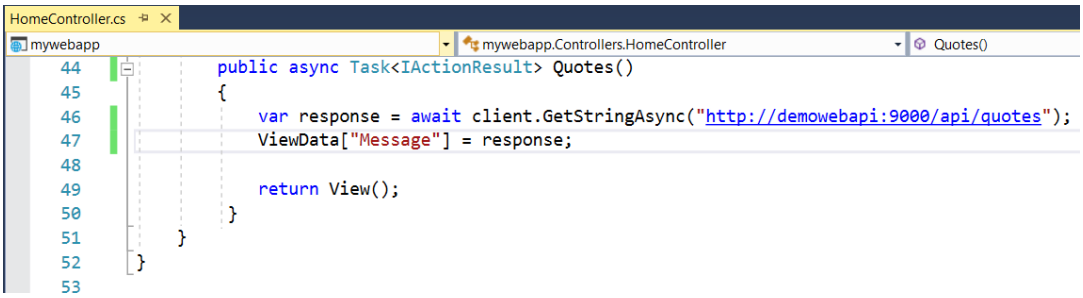
    1. Go to **C:\labs\module6** on your Windows VM. Double click on the "devops" solution file which will open the project in Visual Studio.



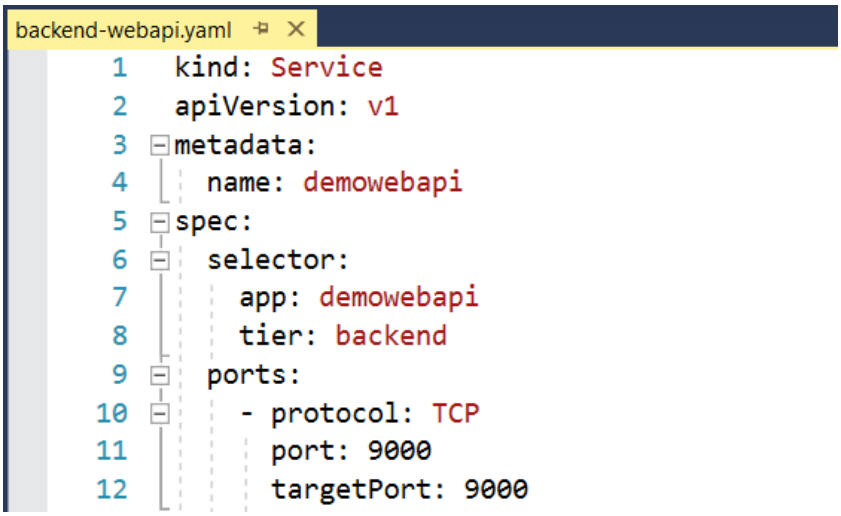
    2. On the right, you will see the Solution Explorer. You will see that there are two ASP.NET Core applications, one is a web API backend project and the other is MVC Core frontend project.



3. 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.

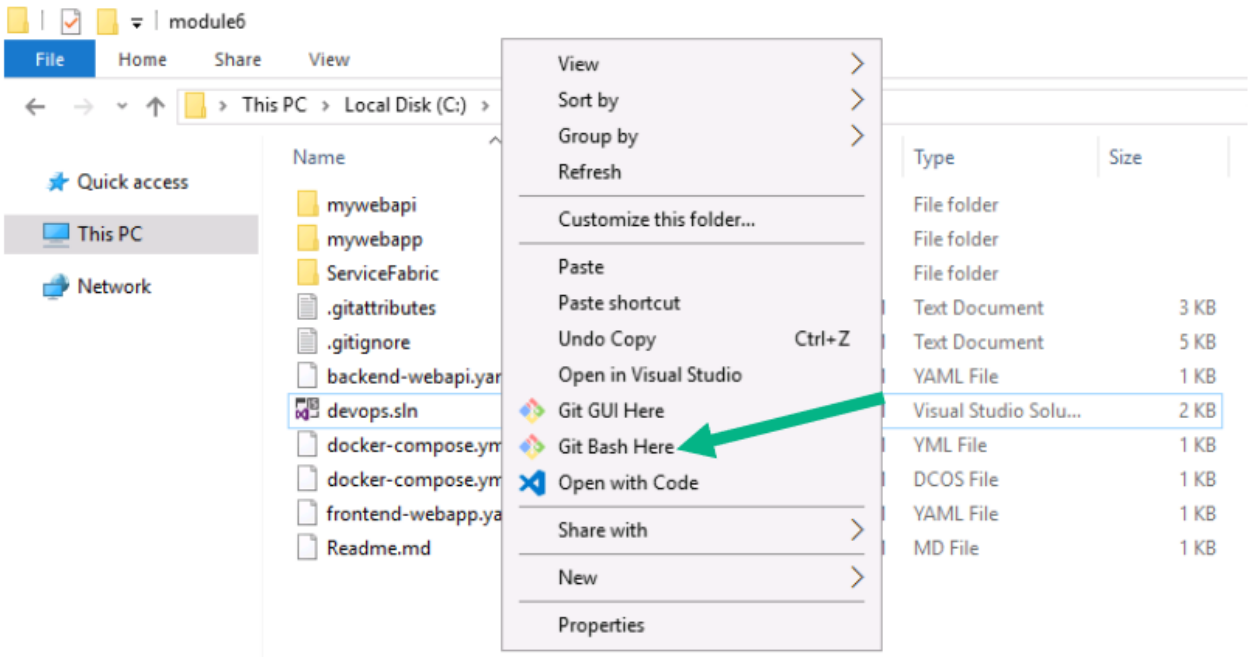


4. In the Solution Items folder, open the **backend-webapi.yaml** file which is the deployment configuration file for Kubernetes cluster. Notice that here the backend application is also exposed with the name **demowebapi**.



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

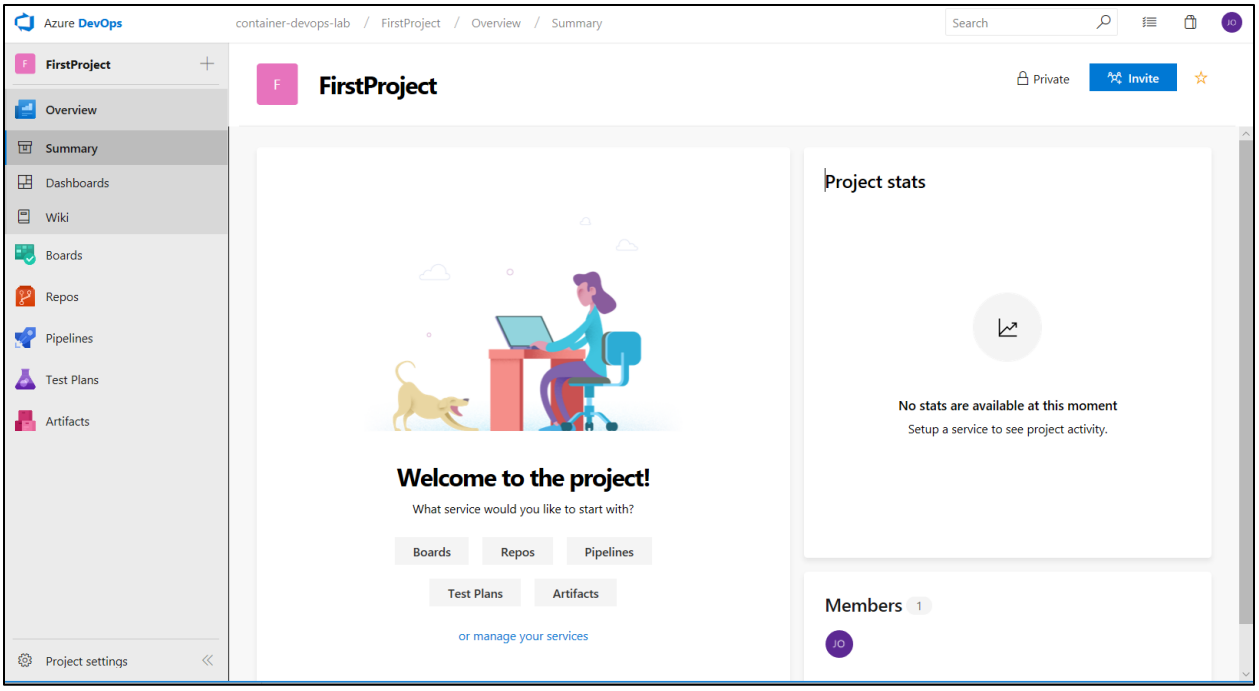
1. Open File Explorer and navigate to **C:\labs\module6**.
2. Right click choose Git Bash Here.



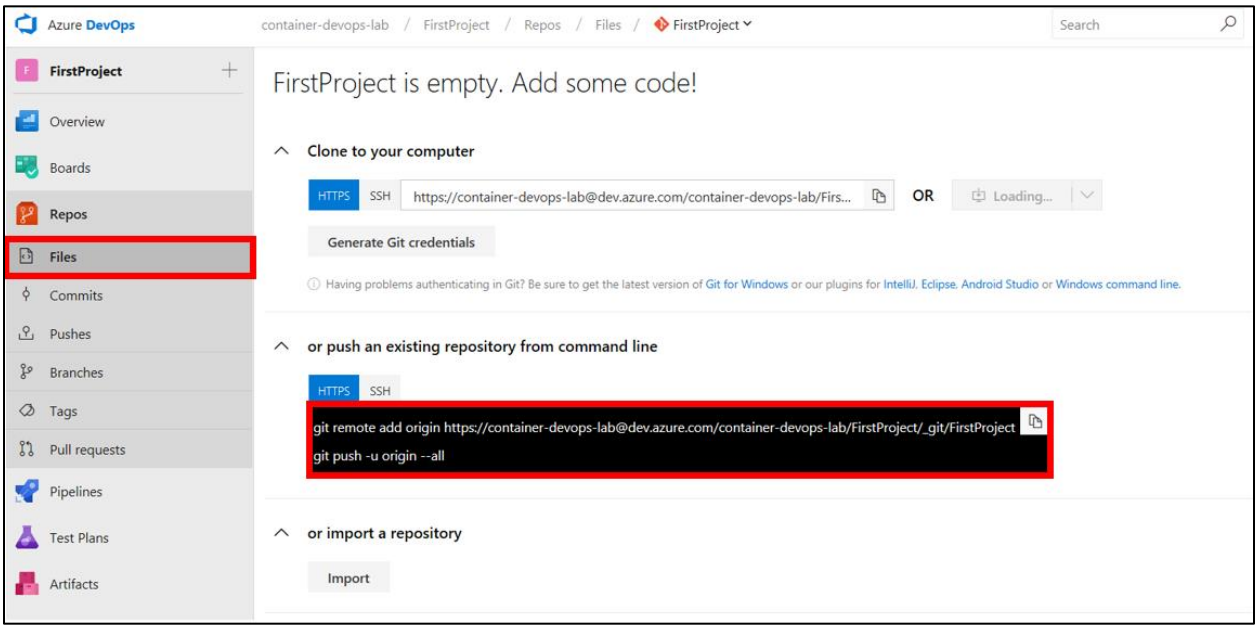
3. Run the following commands, one at a time (you can either fill in a name and email or leave it as the default). Git config (first two commands) is a one-time setup. Git init will initialize your folder as a git repository and add an empty hidden .git folder. Git add and commit are necessary steps 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'
```

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



5. 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.

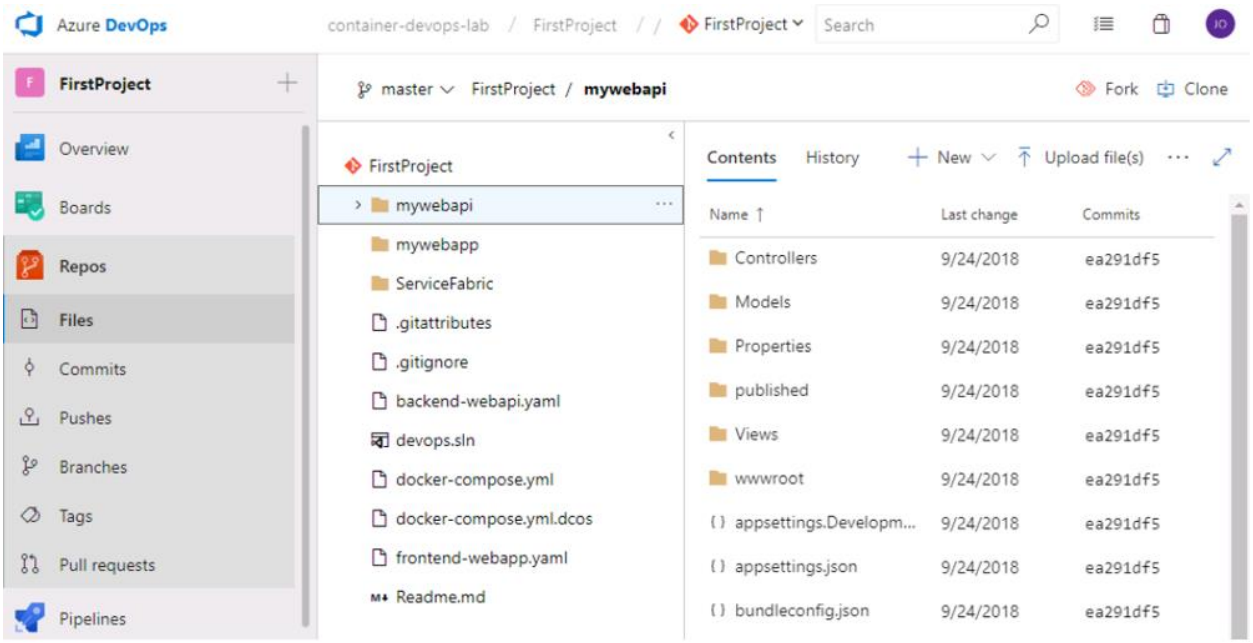


6. Paste them into Git Bash by pressing Shift + Insert (or right click and paste it into the Git Bash command line). It will automatically run the first line, hit enter to add the second line. “Git remote add origin” means to add a “remote URL such as your Azure repo URL” with the alias “origin” to your 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@5dokdkszzx3za MINGW64 /c/labs/module6 (master)
$ git remote add origin https://container-devops-lab@dev.azure.com/container-devops-lab/FirstProject/_git/FirstProject

super@5dokdkszzx3za MINGW64 /c/labs/module6 (master)
$ git push -u origin --all
```

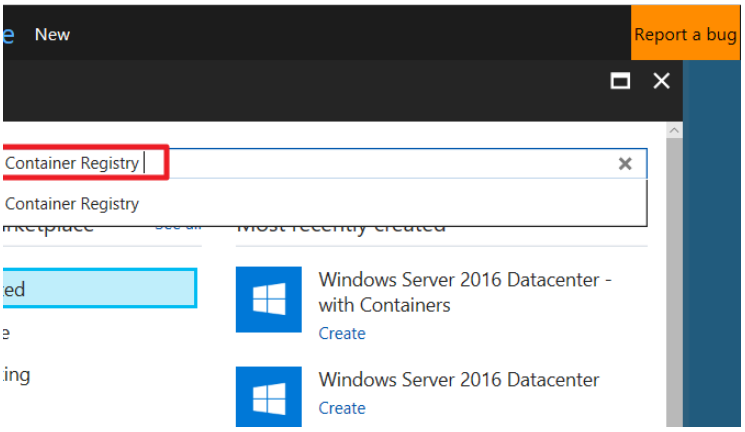
You will see a popup to authenticate to your Microsoft account. Once authenticated, your project will be pushed to remote Git repository in Azure DevOps if you refresh Azure DevOps repository in your browser.



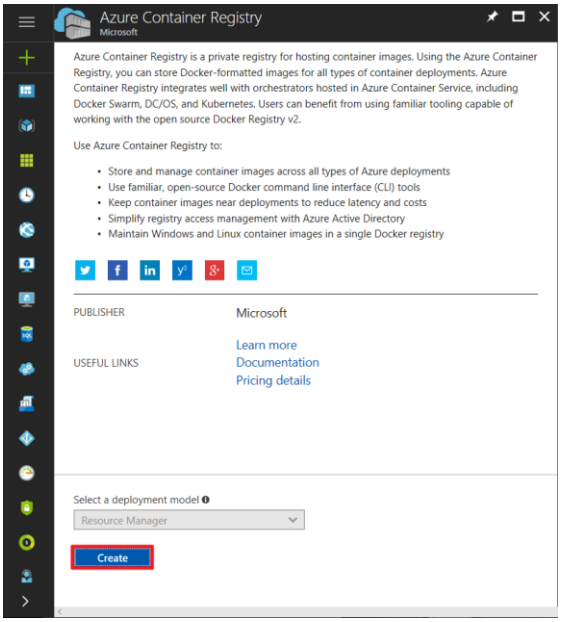
3. Create Azure Container Registry (ACR)

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.

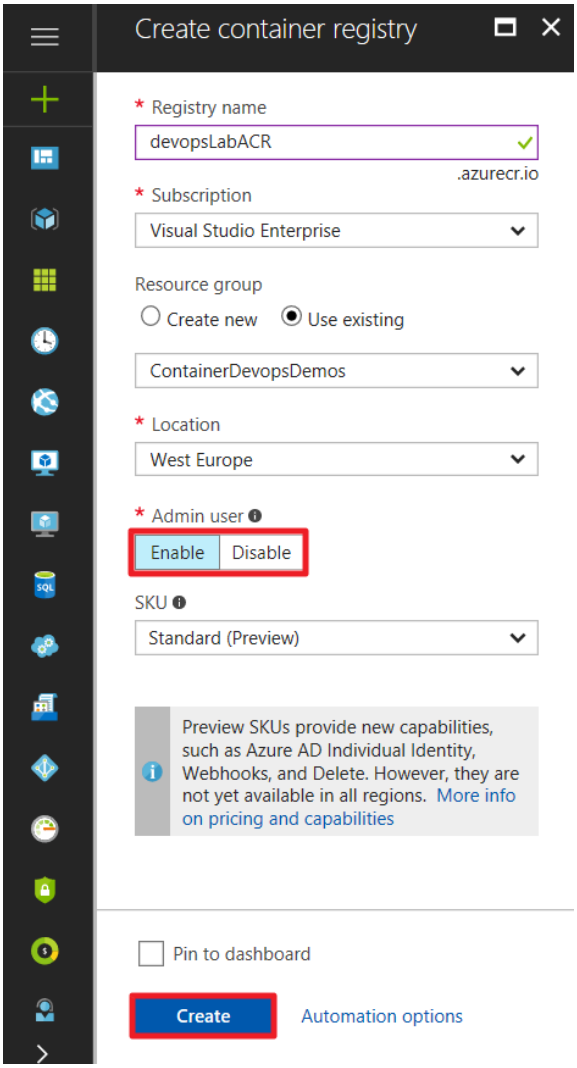
1. In Azure Portal, click plus button to add a new resource. Type **"Container Registry"** in the search textbox and click on the result.



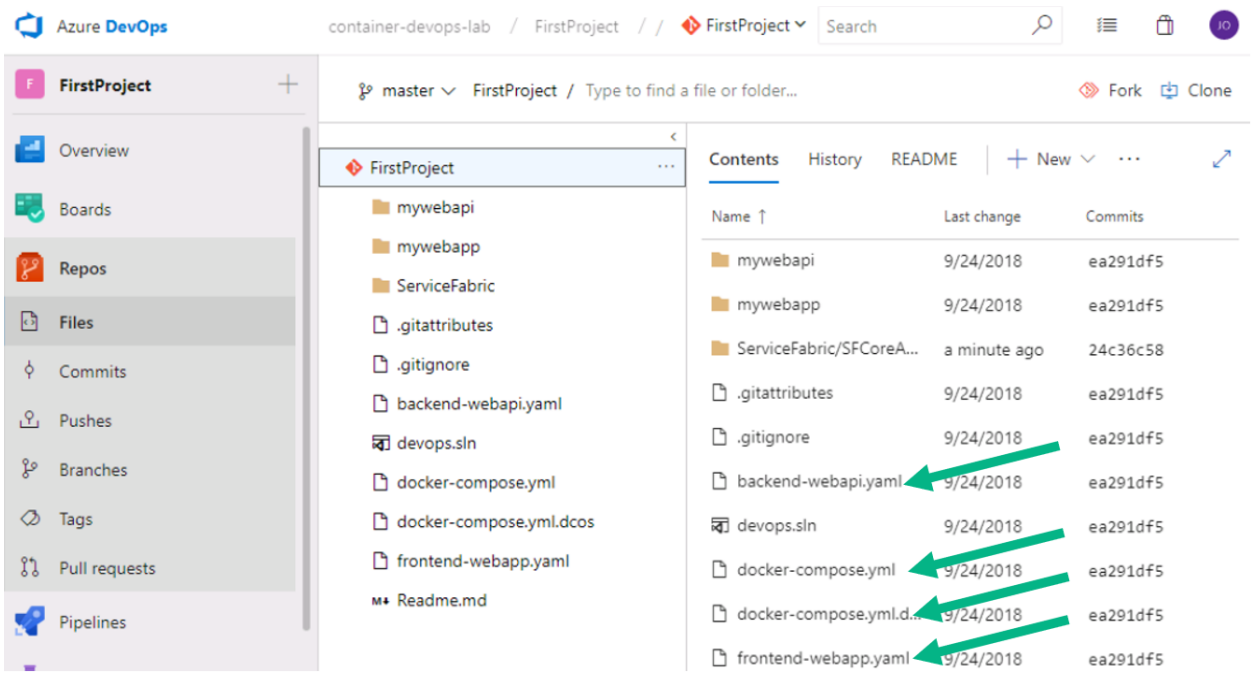
2. Click **Create**



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**.

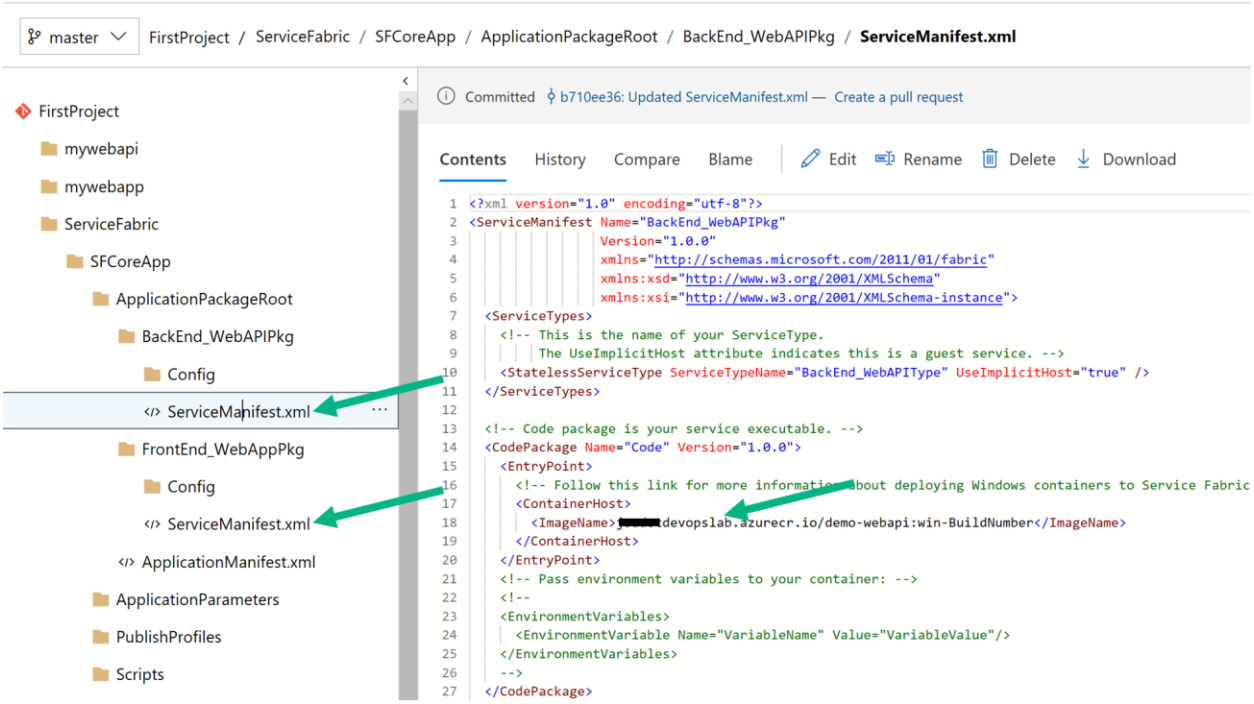


5. Go into each of the 4 files individually and replace all instances of the existing ACR name **devopslabacr** with the new ACR name that is being created **(you must put in your ACR name all in lowercase, otherwise you will get an error later!)**. There should be a total of 6 replacements.

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**


master
FirstProject / ServiceFabric / SFCoreApp / ApplicationPackageRoot / BackEnd_WebAPIPkg / ServiceManifest.xml
FirstProject
mywebapi
mywebapp
ServiceFabric
SFCoreApp
ApplicationPackageRoot
BackEnd_WebAPIPkg
Config
ServiceManifest.xml
FrontEnd_WebAppPkg
Config
ServiceManifest.xml
ApplicationManifest.xml
ApplicationParameters
PublishProfiles
Scripts
Committed b710ee36: Updated ServiceManifest.xml — Create a pull request
Contents History Compare Blame Edit Rename Delete Download
<?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

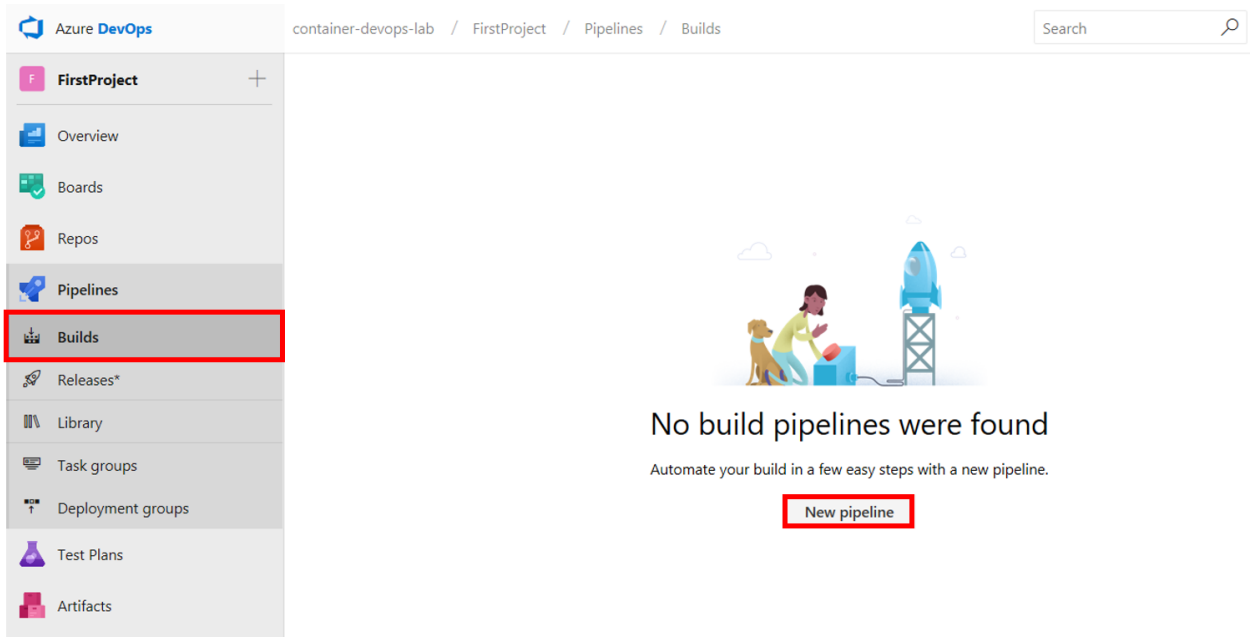
Tasks

> If you want to avoid creating the pipeline step by step, you can skip this task and jump to Optional: Import a build pipeline.

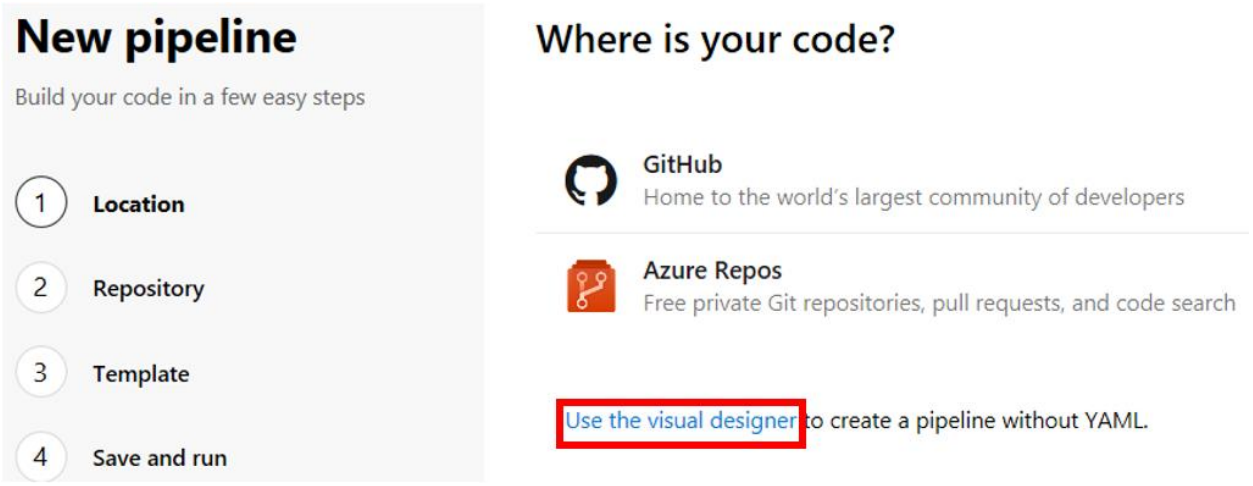
1. **Create new build pipeline**

   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).

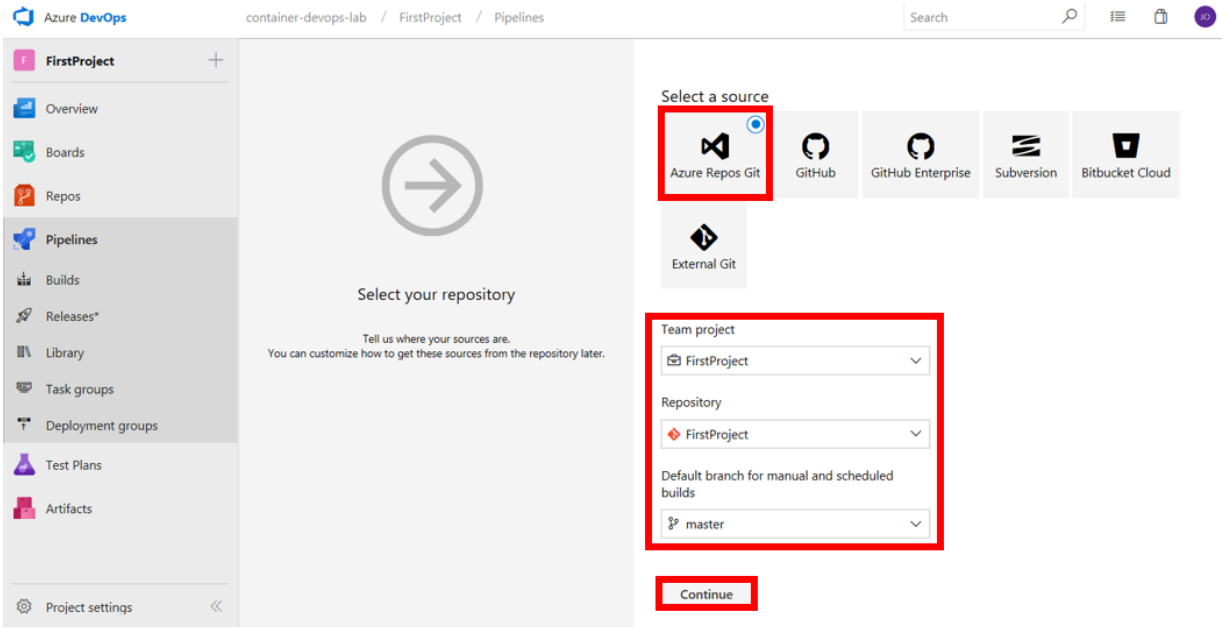
   1. Navigate to https://dev.azure.com/YourAccountName/YourProjectName and go to **Pipelines - Builds** page and click **New pipeline**.



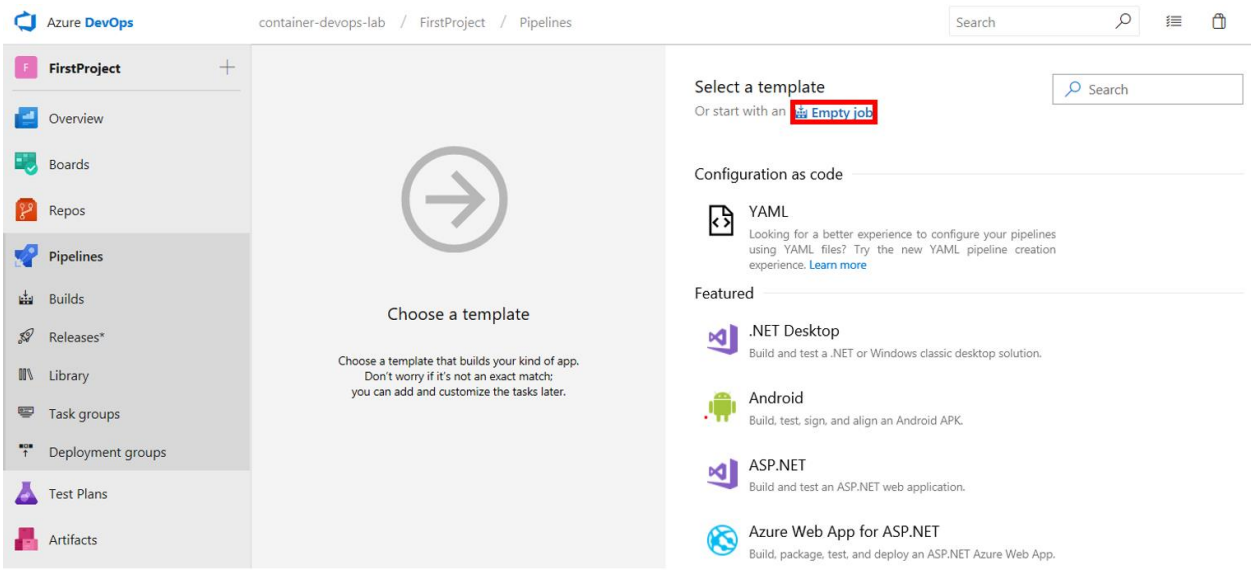
   2. Choose **Use the visual designer.**



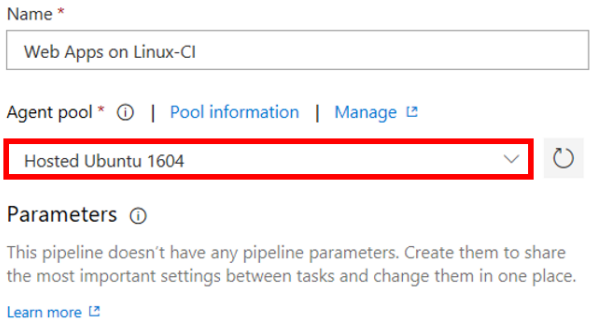
3. Create the build pipeline by selecting the Azure repository and click **Continue**.



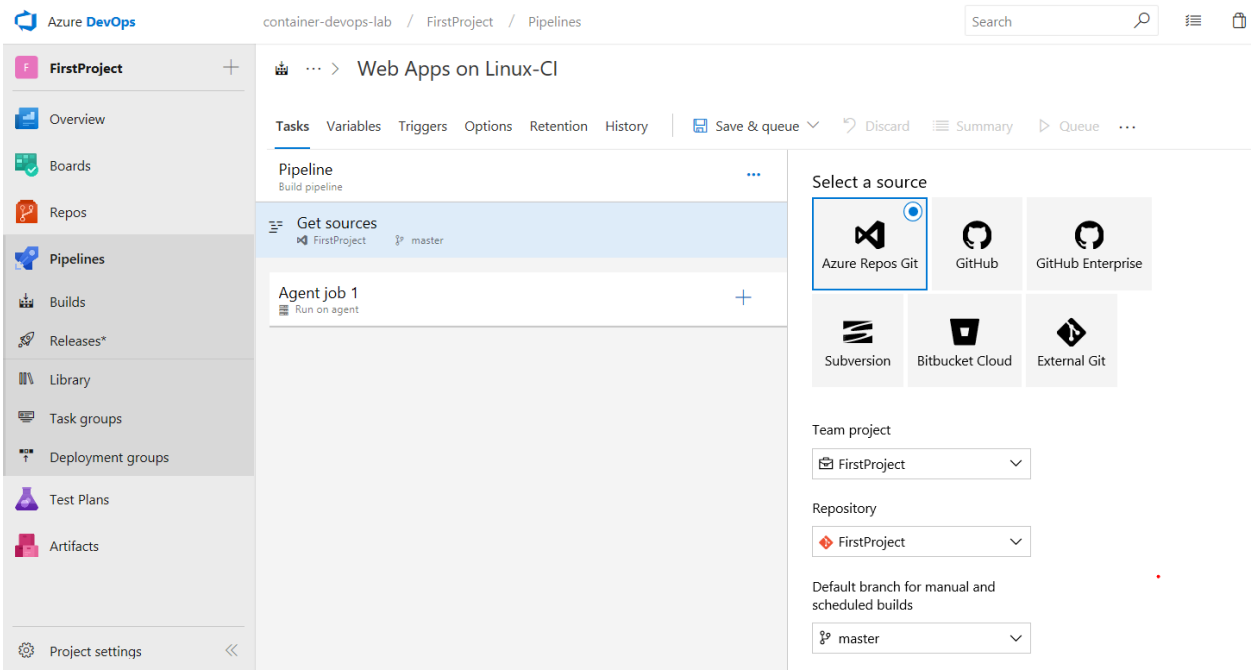
4. Then, select an **Empty job** to start from a clean state.



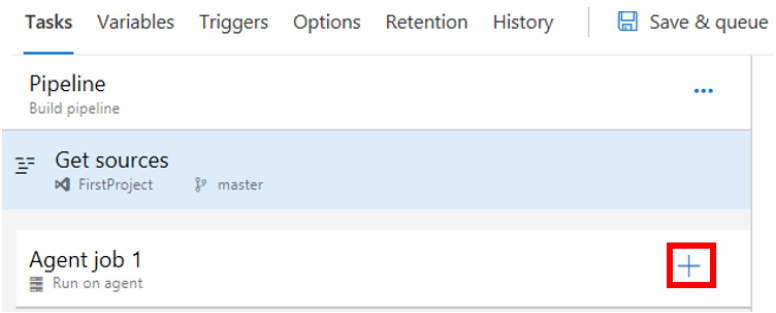
5. Before adding build tasks to the pipeline, you are going to select an agent queue containing agents which will be assigned to handle build requests. Select **Hosted Ubuntu 1604** agent queue which contains agents installed on Linux OS.



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



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:

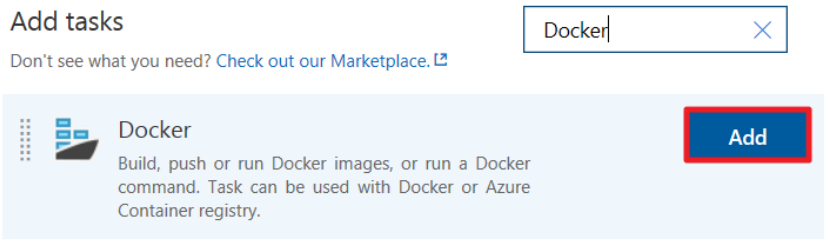


8. Normally, you might need to add dotnet build and publish steps, however these are included inside of the Dockerfile and are completed as part of the Docker build process. Just remember when you are creating your own CI/CD pipelines to either include the dotnet build and publish steps in the Dockerfile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

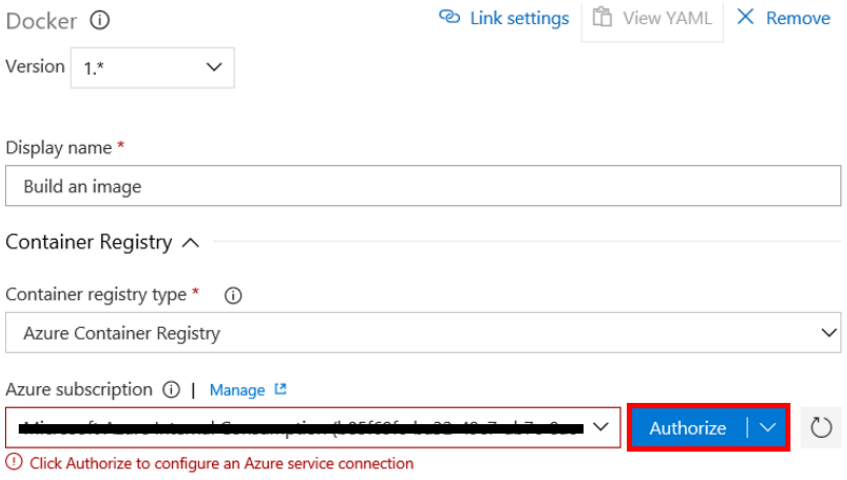
```
FROM microsoft/aspnetcore-build:2.0 AS build-env

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.



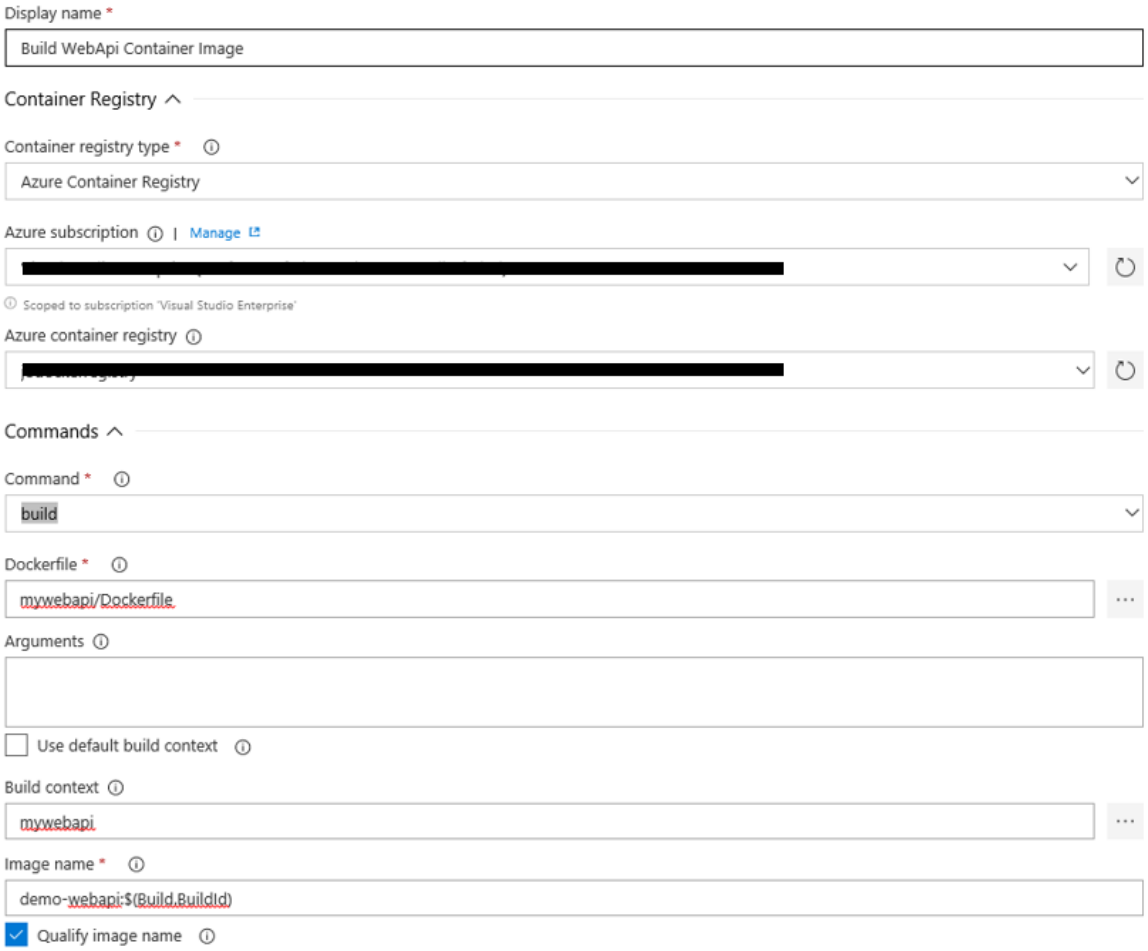
10. Select an Azure subscription and click **Authorize** to allow Azure DevOps account access to the resources in the subscription.



Now configure this task's other properties as follows:

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

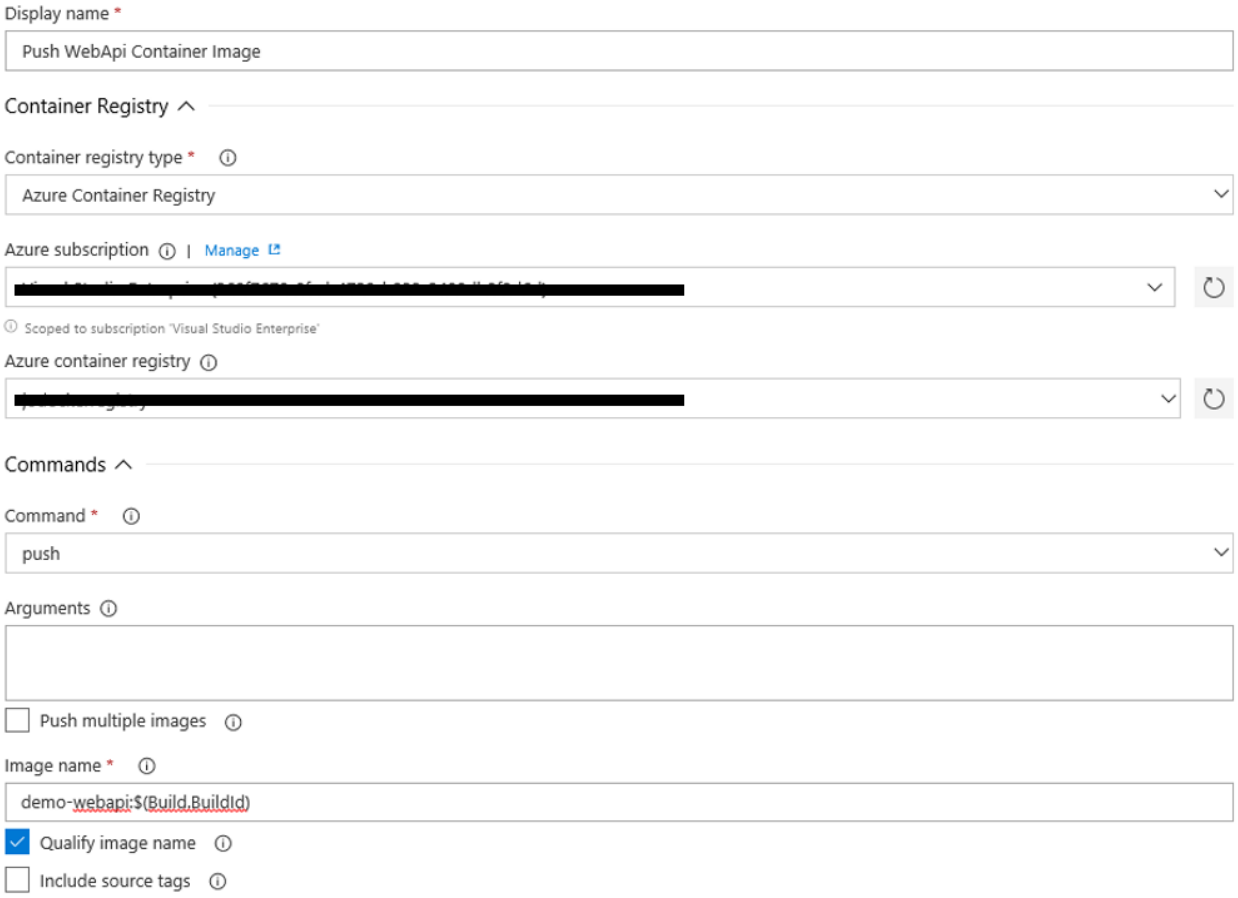
- **Container Registry Type**: Azure Container Registry
- **Azure Container Registry**: Select the container registry you have created..
- **Command**: build
- **Docker File**: mywebapi/Dockerfile
- Unselect **Use default build context**
- **Build Context**: mywebapi
- **Image Name**: demo-webapi:$(Build.BuildId)



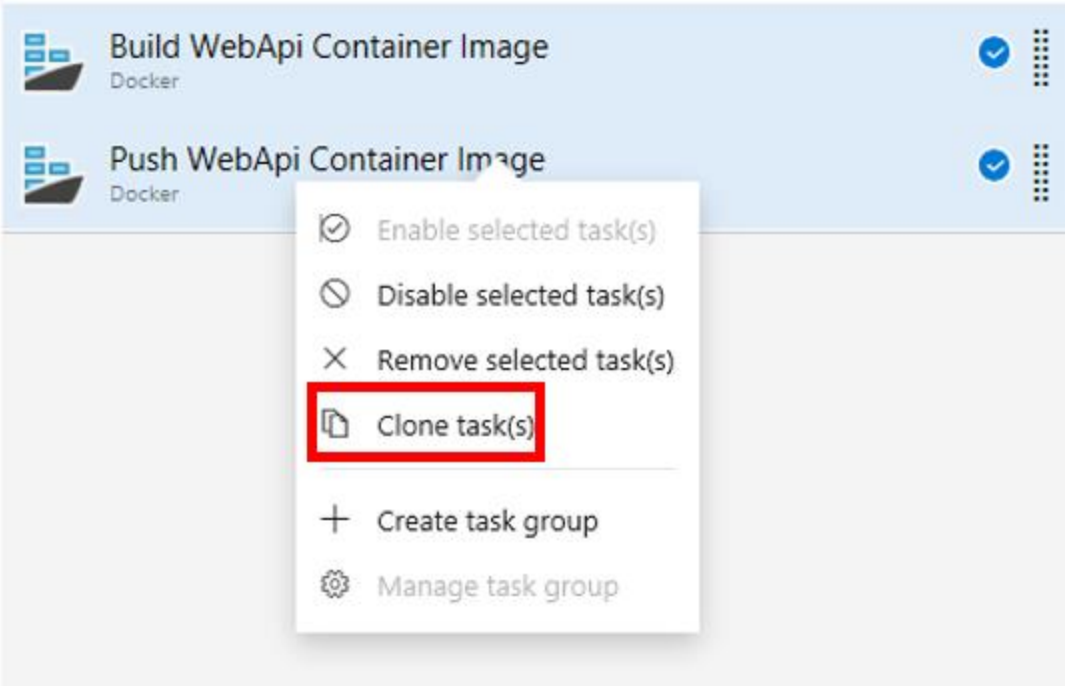
11. Now add another **Docker** task to push the image to ACR.
    - **Display Name**: Push WebApi Container Image
    - **Container Registry Type**: Azure Container Registry

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

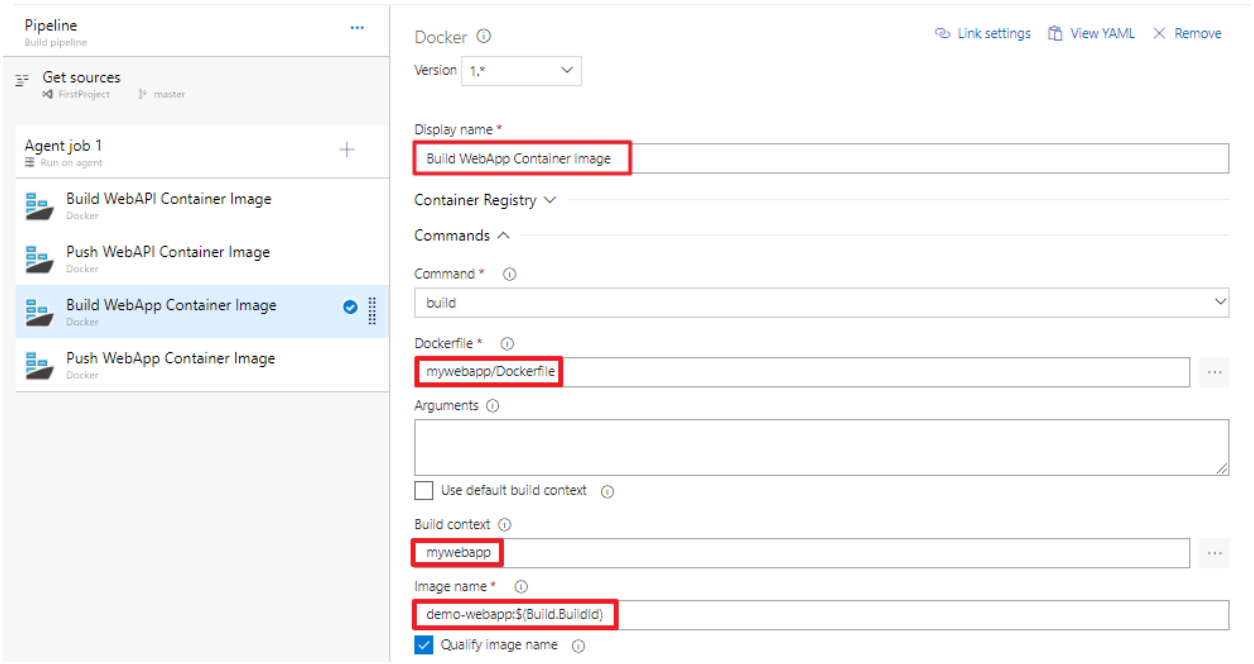
  Image Name: demo-webapi:$(Build.BuildId)



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).**

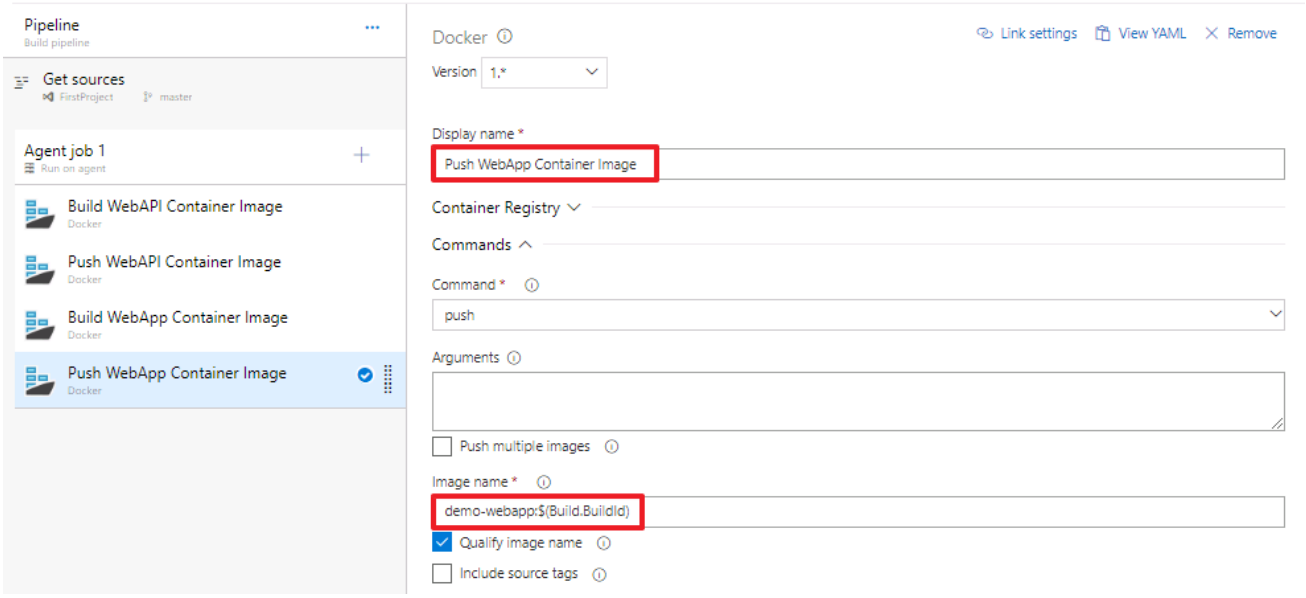


13. Change the **Build WebApi Container Image copy** task to the following attributes:
    - **Display Name**: Build WebApp Container Image
    - **Docker File**: mywebapp/Dockerfile
    - **Build Context**: mywebapp
    - **Image Name**: demo-webapp:$(Build.BuildId)

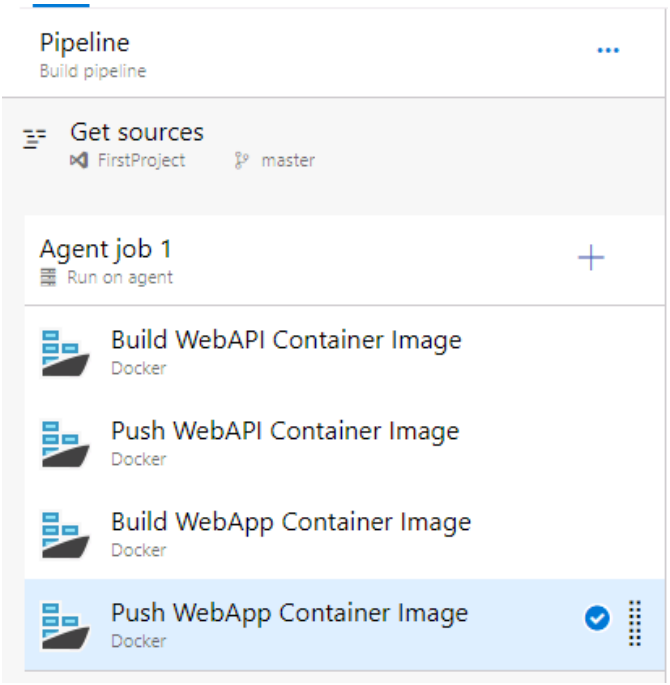


14. Change the **Push WebApi Container Image copy** task to the following attributes:

    - **Display Name**: Push WebApp Container Image
    - **Image Name**: demo-webapp:$(Build.BuildId)



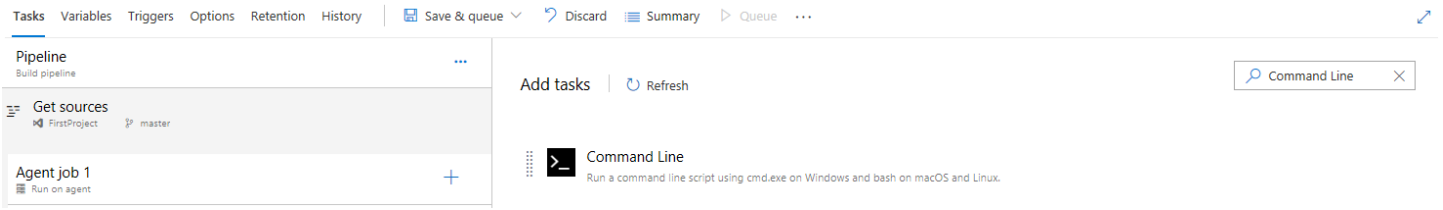
15. Verify you have the following tasks so far:



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.

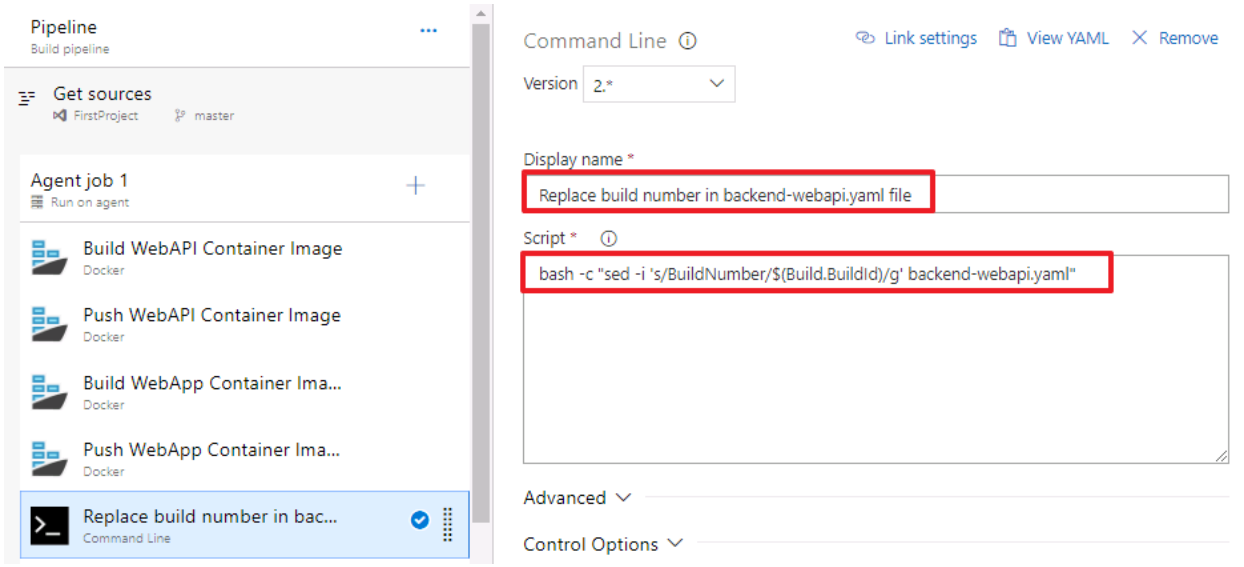
To pick these images up from ACR and deploy to Kubernetes container orchestrator, you will need to use the same build number in release pipeline. Therefore you need to place it in Kubernetes deployment configuration files: ***backend-webapi.yaml*** *and* ***frontend-webapp.yaml****.*

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



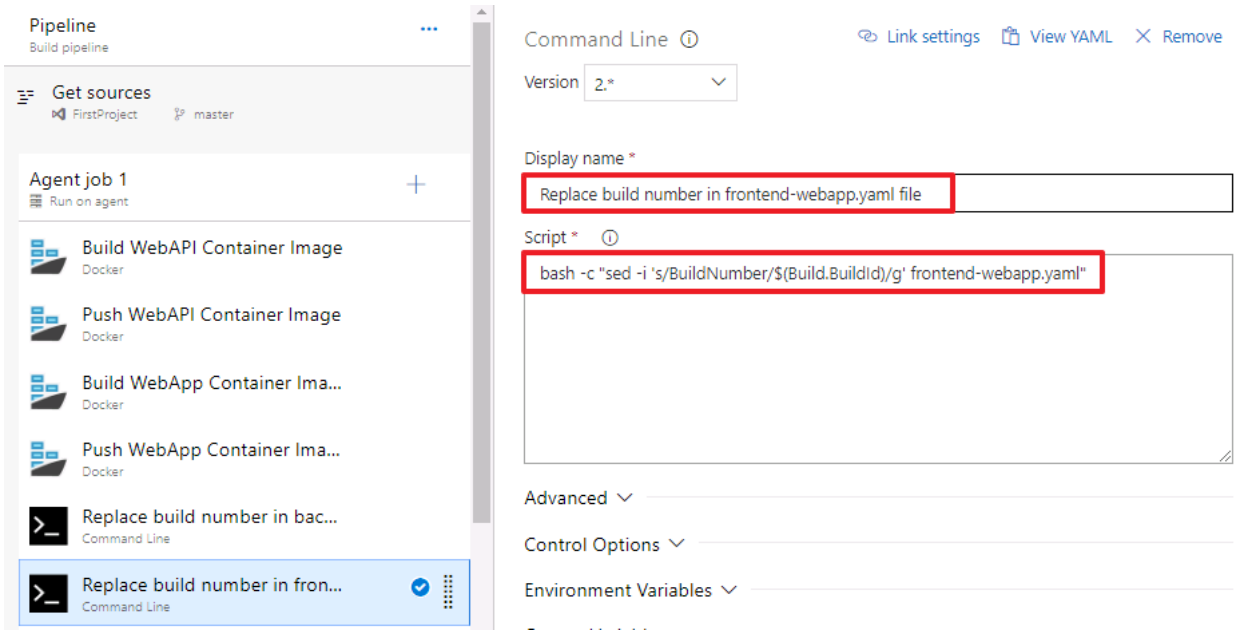
Configure the first **Command Line task** as follows:

- **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)



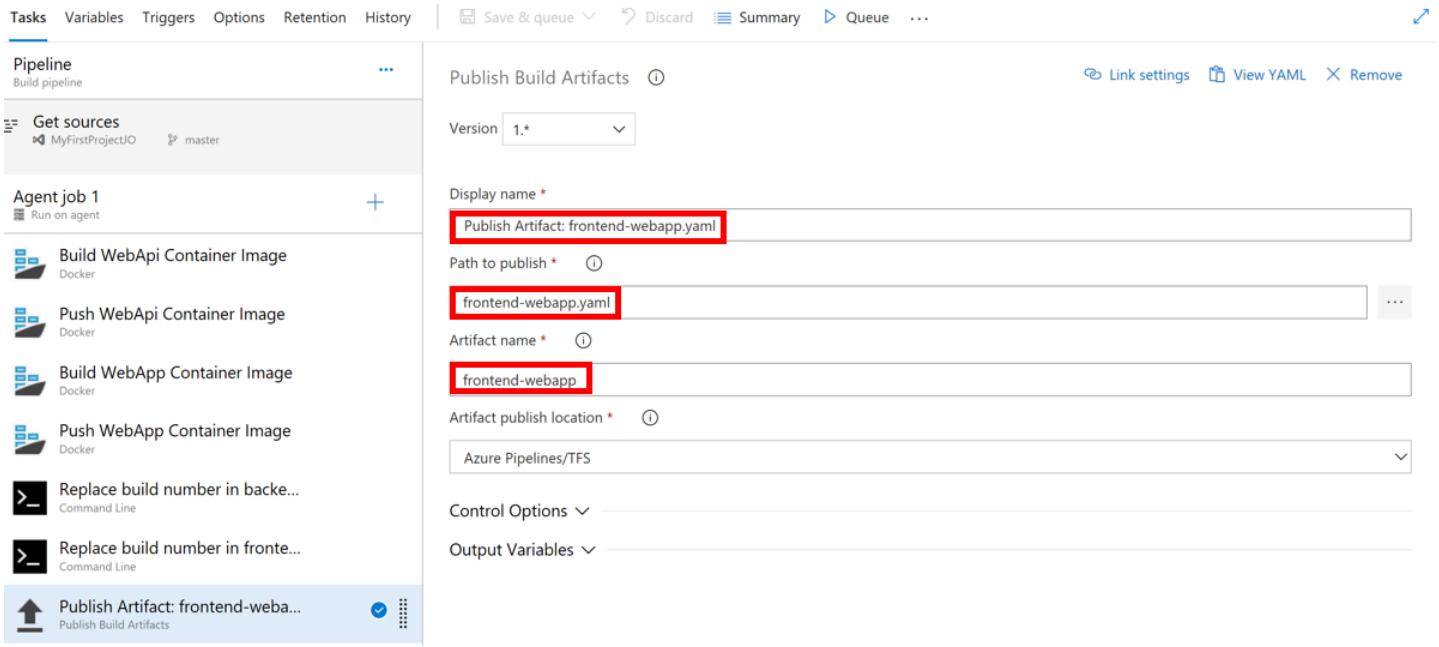
17. Clone your first command line task, and configure the second **Command Line task** as follows:

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

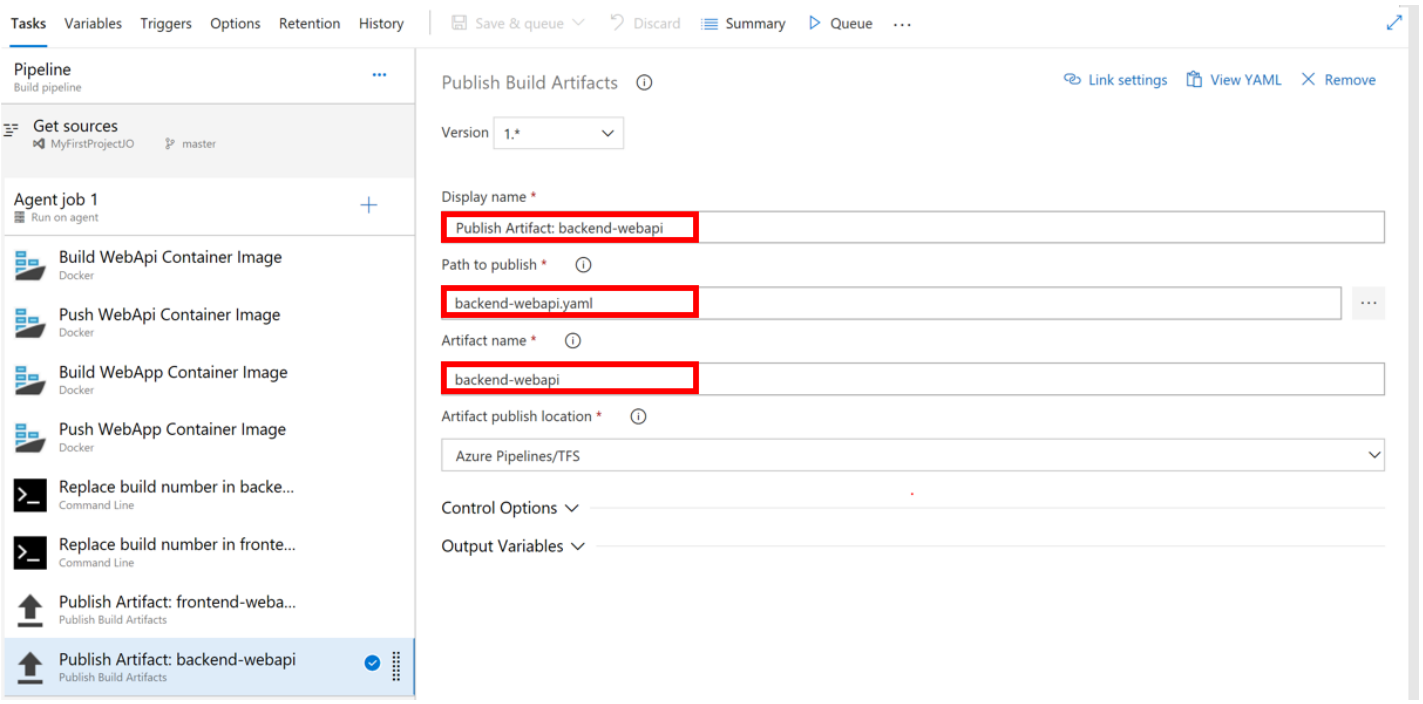


The previous task ensures that the Yaml file's content is updated with build number, now it is time to upload it as build artifact to be used in release pipeline later.

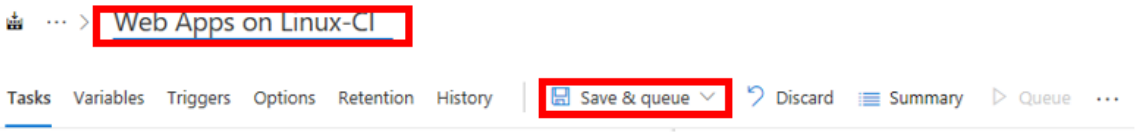
18. Add a new **Publish Build Artifacts** task and configure as follows:
    - **Display Name**: Publish Artifact: frontend-webapp.yaml
    - **Path to Publish**: frontend-webapp.yaml
    - **Artifact Name**: frontend-webapp



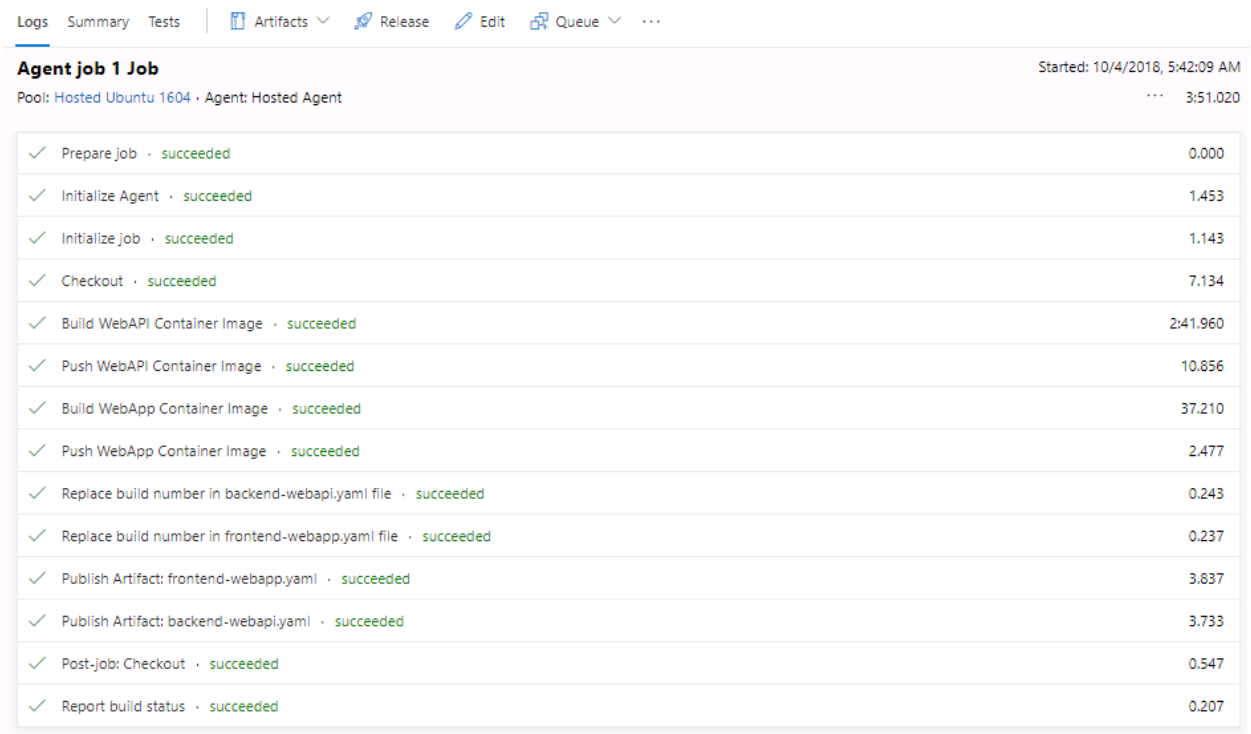
19. Clone your Publish Build Artifact task and modify the second one to be configured as follows:
    - **Display Name**: Publish Artifact: backend-webapi.yaml
    - **Path to Publish**: backend-webapi.yaml
    - **Artifact Name**: backend-webapi



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):

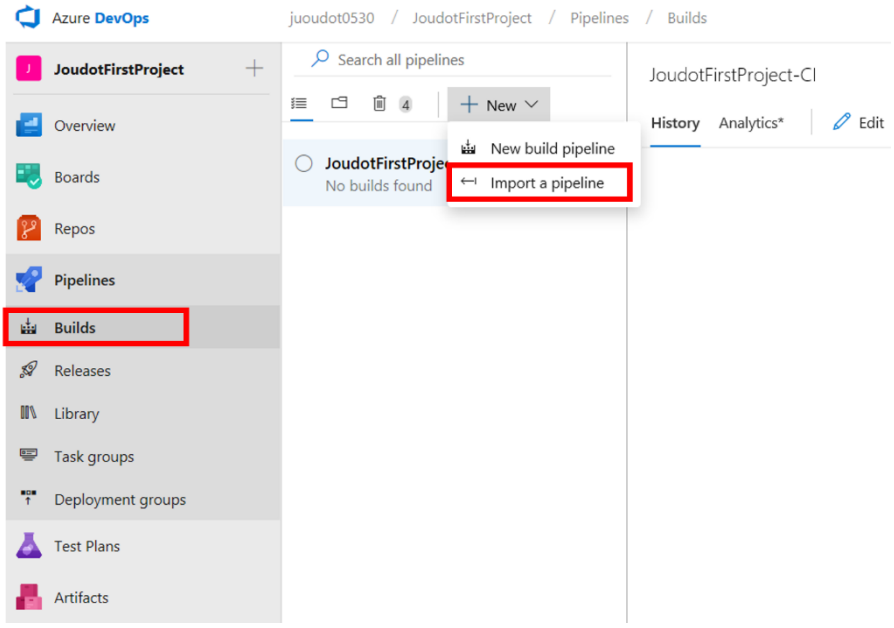


Optional: Import a build pipeline

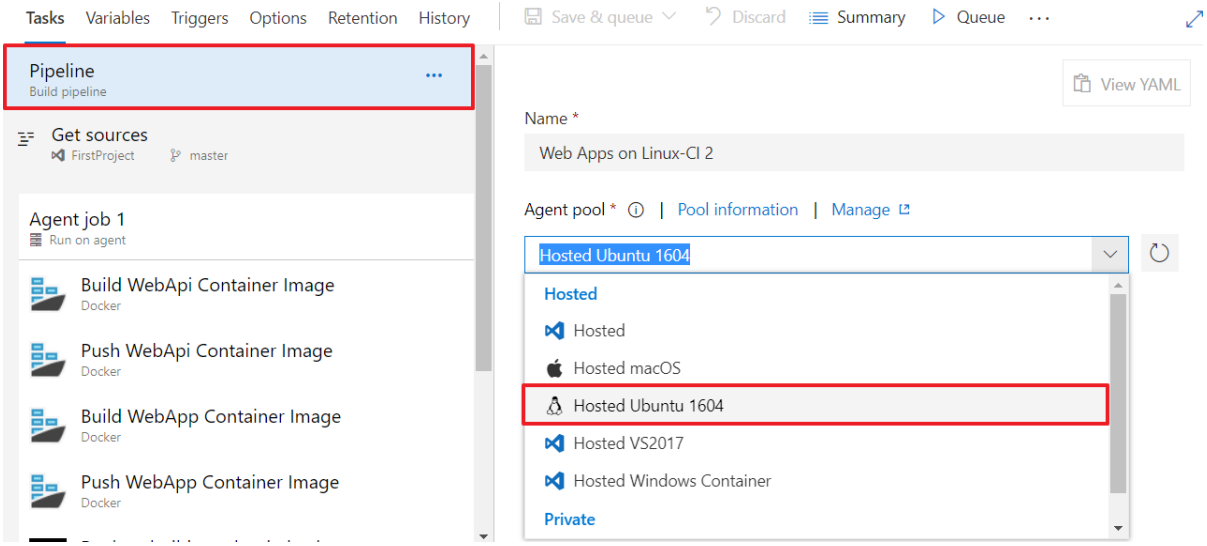
> Skip this optional task if you already created the build pipeline.

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

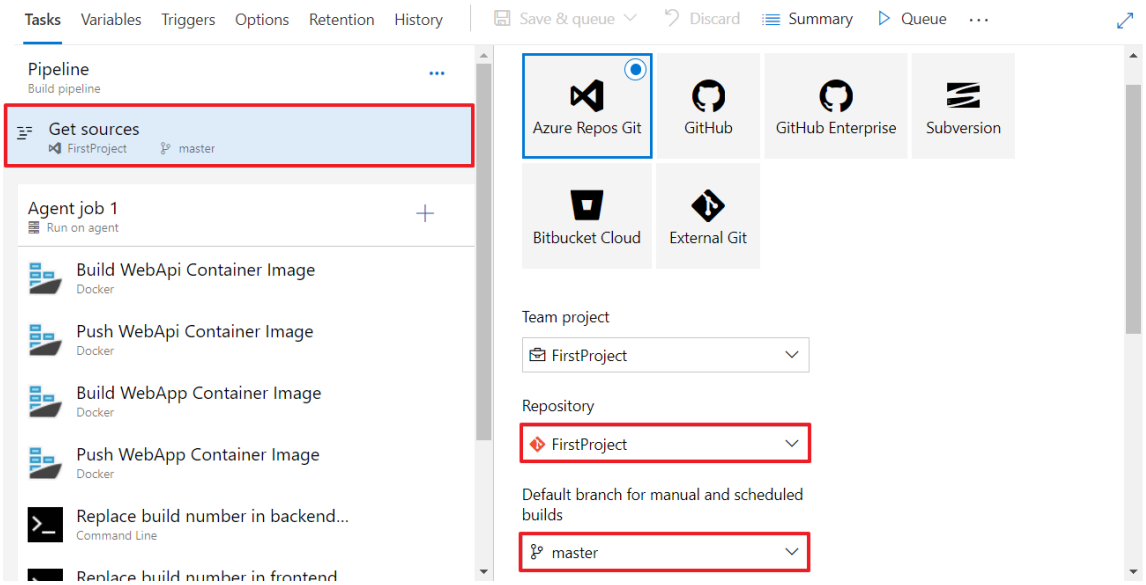
Note: Currently, we can only do this when at least one pipeline exists. So, you will need to create an empty pipeline first.



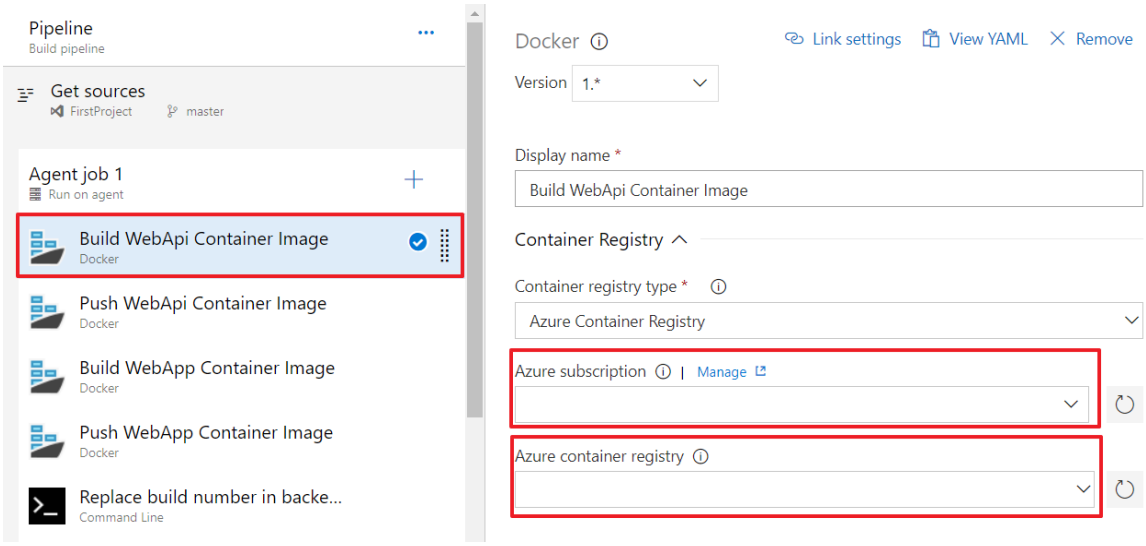
2. Select **C:\labs\module6-ext\Web Apps on Linux-CI.json** and click **Import**
3. Select **Hosted Ubuntu 1604** as an agent pool



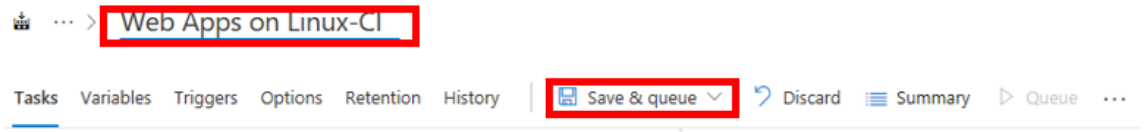
4. Click on **Get sources** and make sure that correct Git repository and branch is selected.



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.

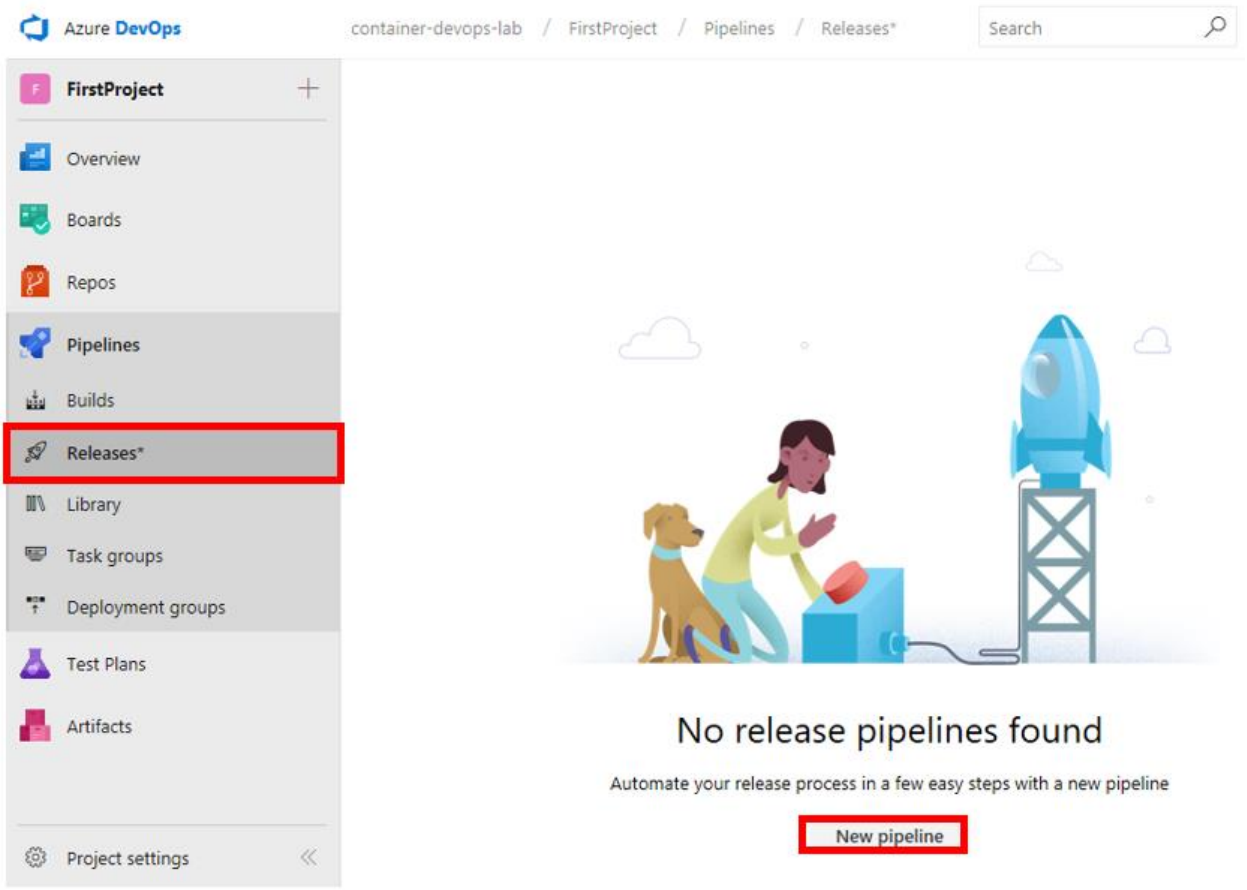
Tasks

> 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**

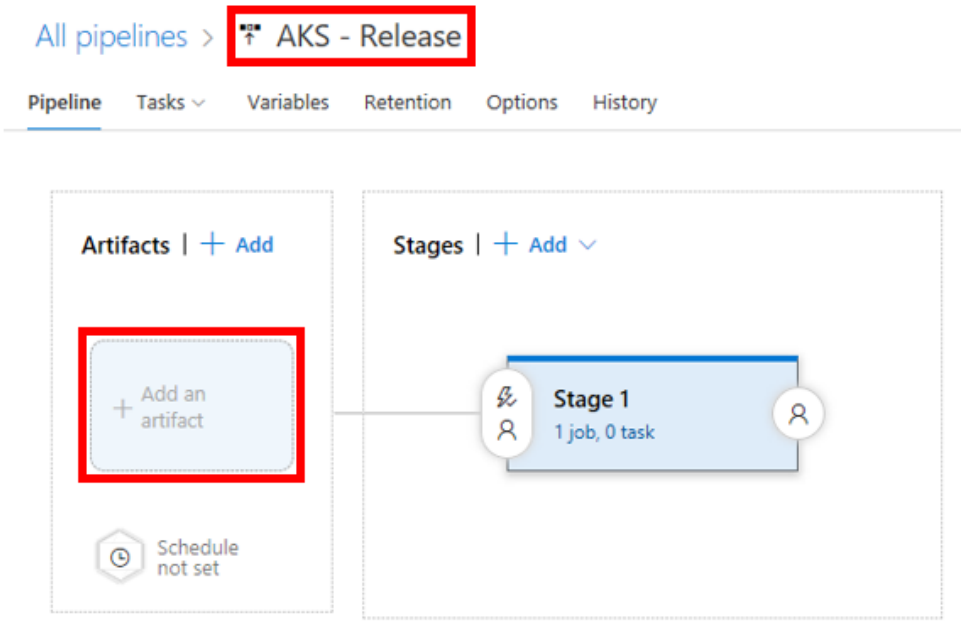
   Build pipeline is used for compiling the application, building a container image, and then pushing container images to container registry. Release pipeline, on the other hand, is used to pull the container image from registry and then deploy to container cluster.

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

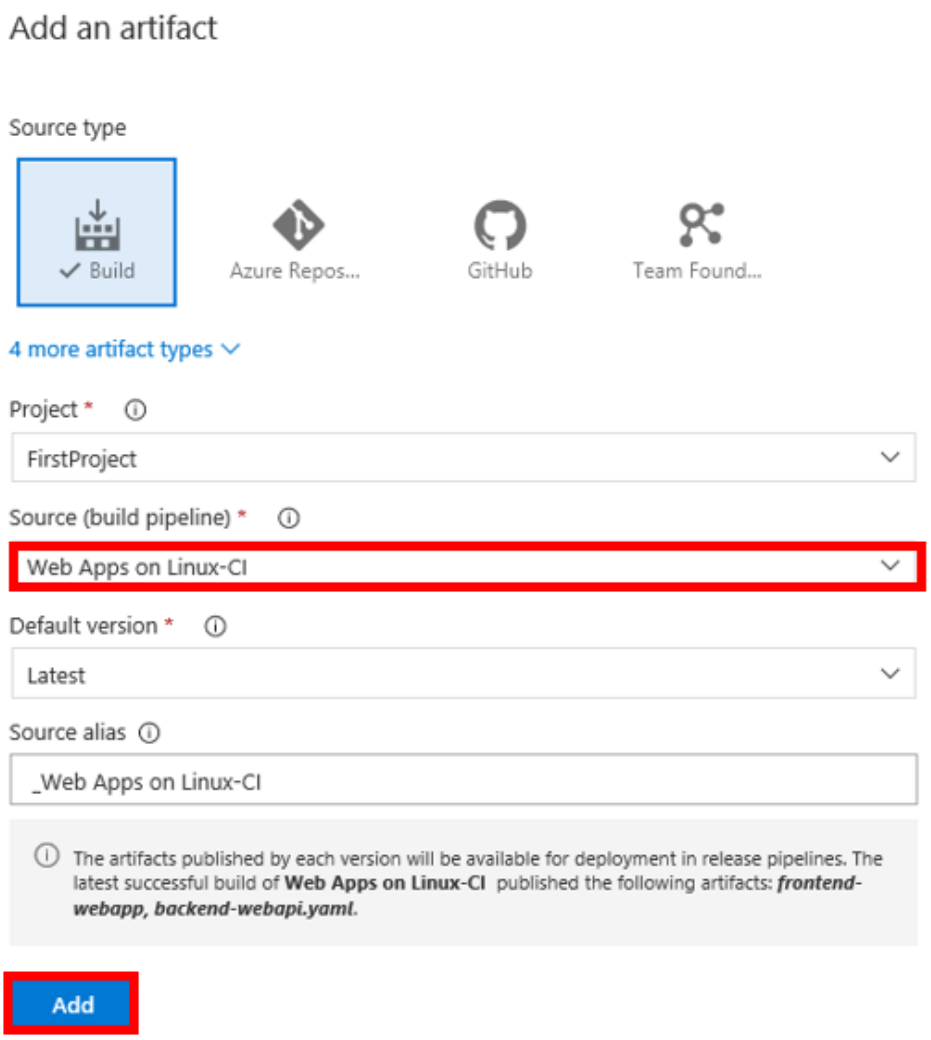


2. Although there are templates about Kubernetes deployment, you are going to start with empty template. Click **Empty job** to create an empty release pipeline.

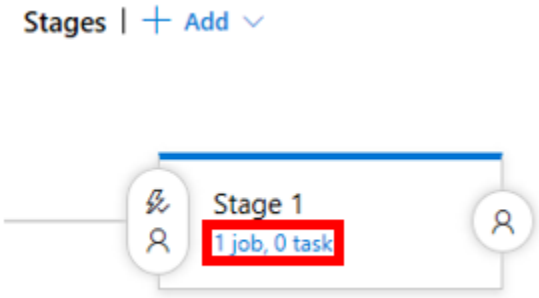
3. Click on release pipeline name and change it to **AKS–Release**. Then click on **Add an artifact** to link this release pipeline with build artifacts.



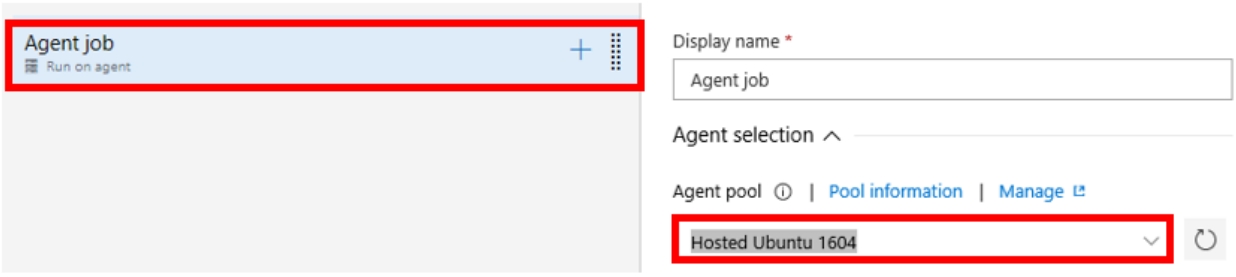
4. On the **Add artifact** popup, select **Web Apps on Linux -CI** build pipeline. Finally select **Add**.



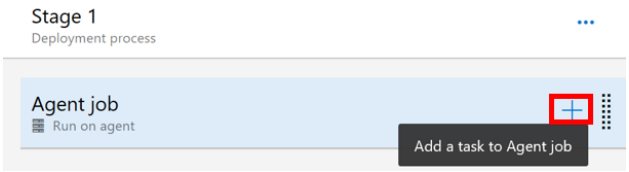
5. Click "**1 job, 0 task**" link available inside the Stage 1.



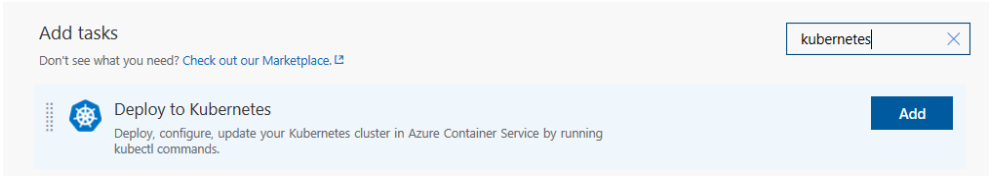
6. Click on the **Agent job** and then select Hosted Ubuntu 1604 from the Agent pool dropdown.



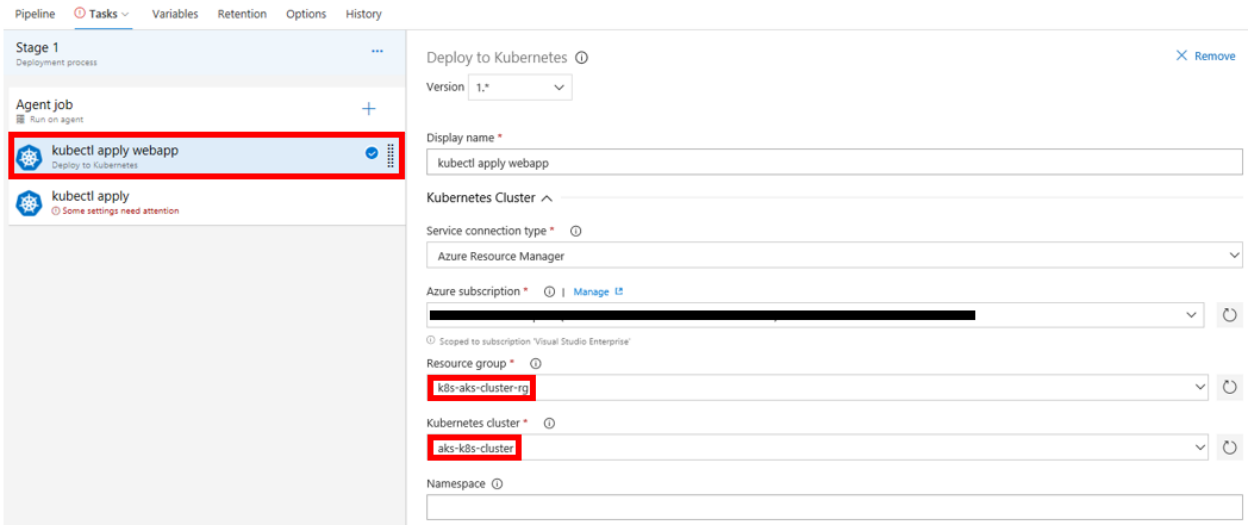
7. Now you are ready to add tasks to deploy your containers to Kubernetes cluster. Click plus button on **Agent phase**, and then find **Deploy to Kubernetes** task.



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

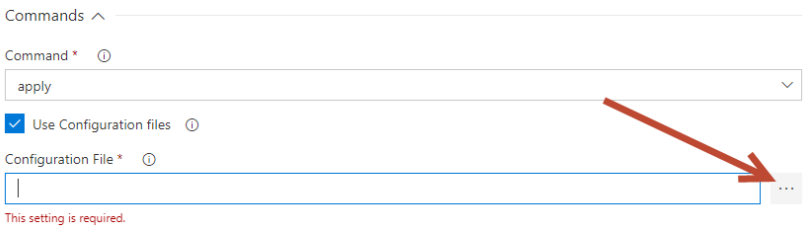


8. Change the first task name to **kubectl apply webapp**. Then to add a connection to your AKS Kubernetes cluster by selecting the Azure subscription, resource group and cluster.



9. In the **Commands** section:

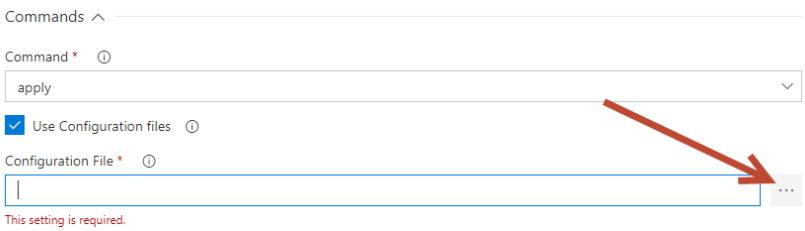
- Check the box for **Use configuration files** and click on the 3 dots to the right of the input box to explore files



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

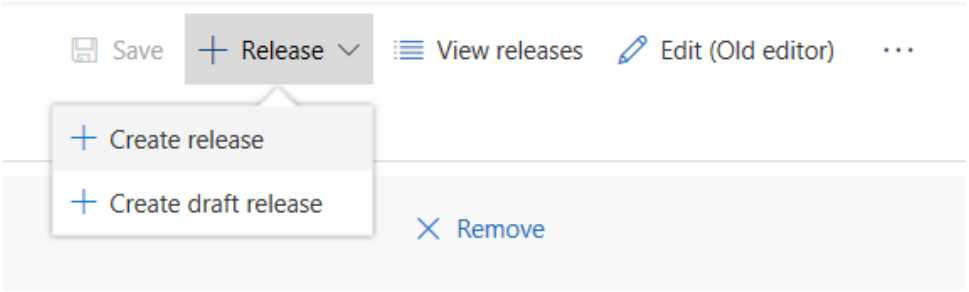
10. Now fill other parameters in the **Secrets** section of the task:
    - **Type of secret**: DockerRegistry
    - **Container Registry type**: Azure Container Registry
    - **Azure Container Registry**: devopsLabACR (this registry should be the same as the one you have used in your build pipeline, 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)

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:

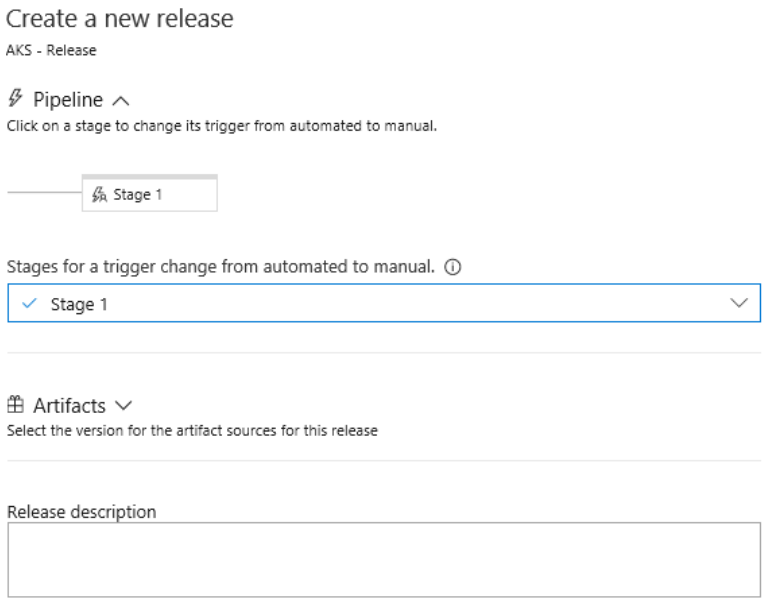


    - Click on the 3 dots to the right of the input box to explore files, select the main folder, select backend-webapi folder, select **backend-webapi.yaml**, then hit OK.
    - **Container Registry type**: Azure Container Registry
    - **Azure Container Registry**: devopsLabACR (this registry should be 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)

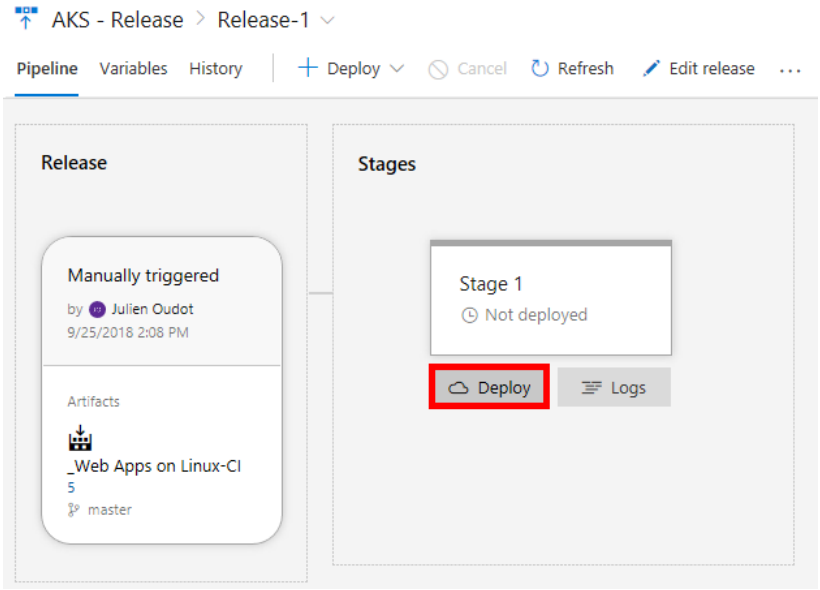
12. Look at the top right of your page and you will see the buttons as in the screenshot below. First, Save the release pipeline. Then click on the Release button and choose "+ Create Release" to trigger a release.



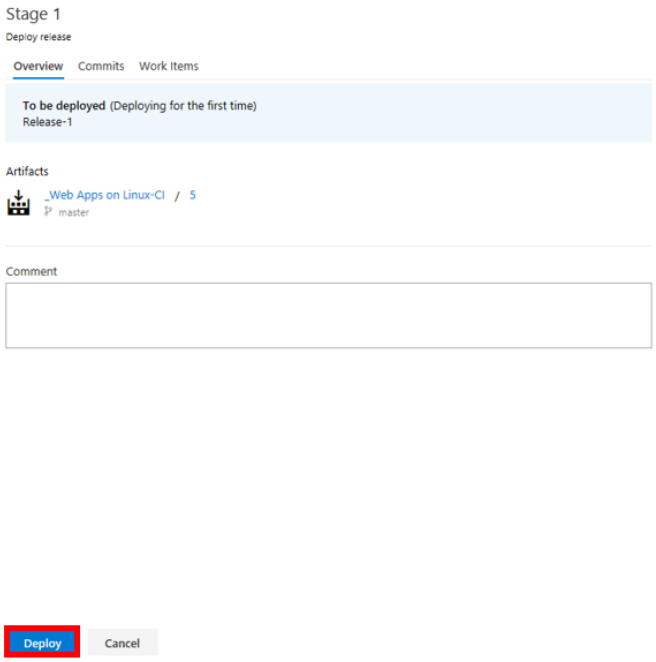
On **Create a new release** window select the Stage 1 from the dropdown. Keep all defaults and press **Create**.



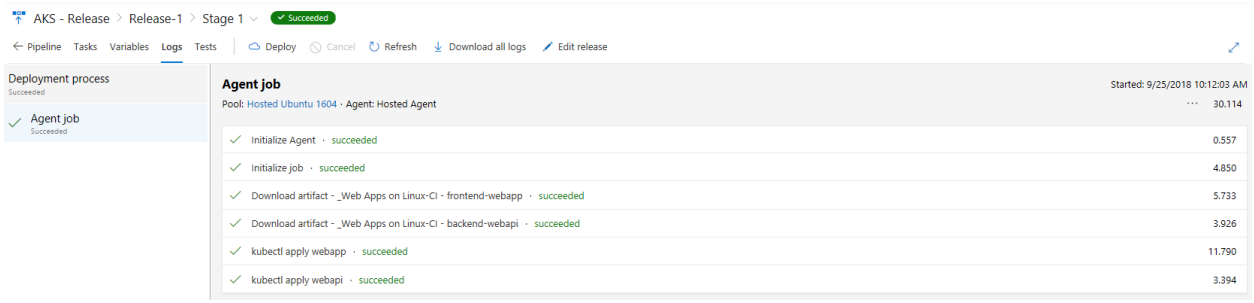
Go back to the Release page by clicking on the name of the created release. Click **Deploy**.



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



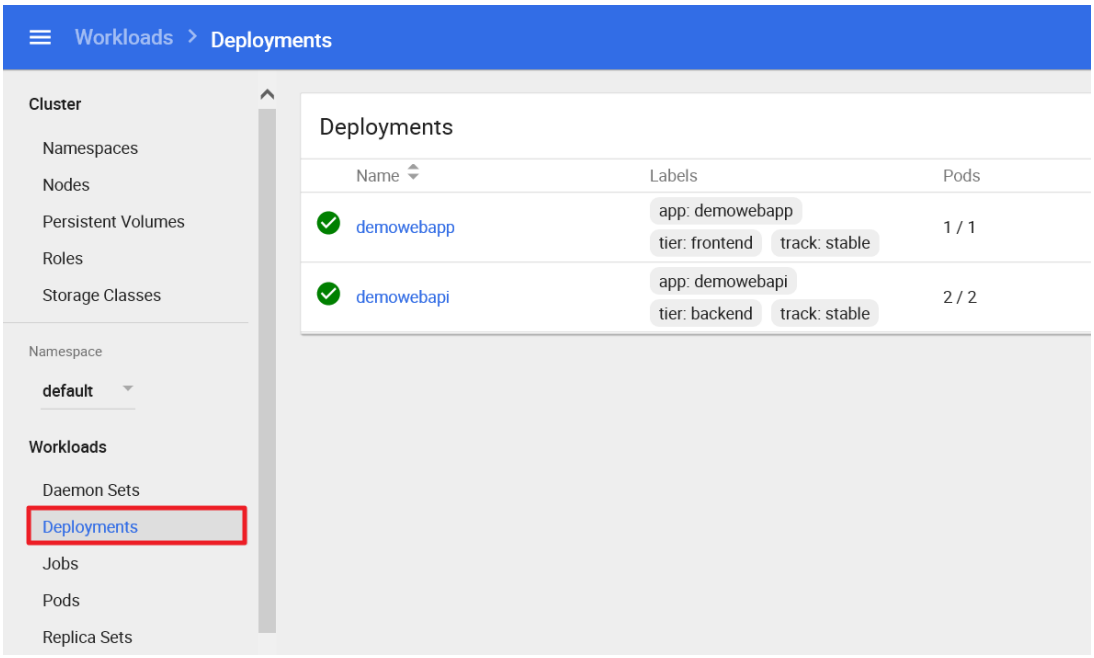
When it succeeds you will see a similar log as below:



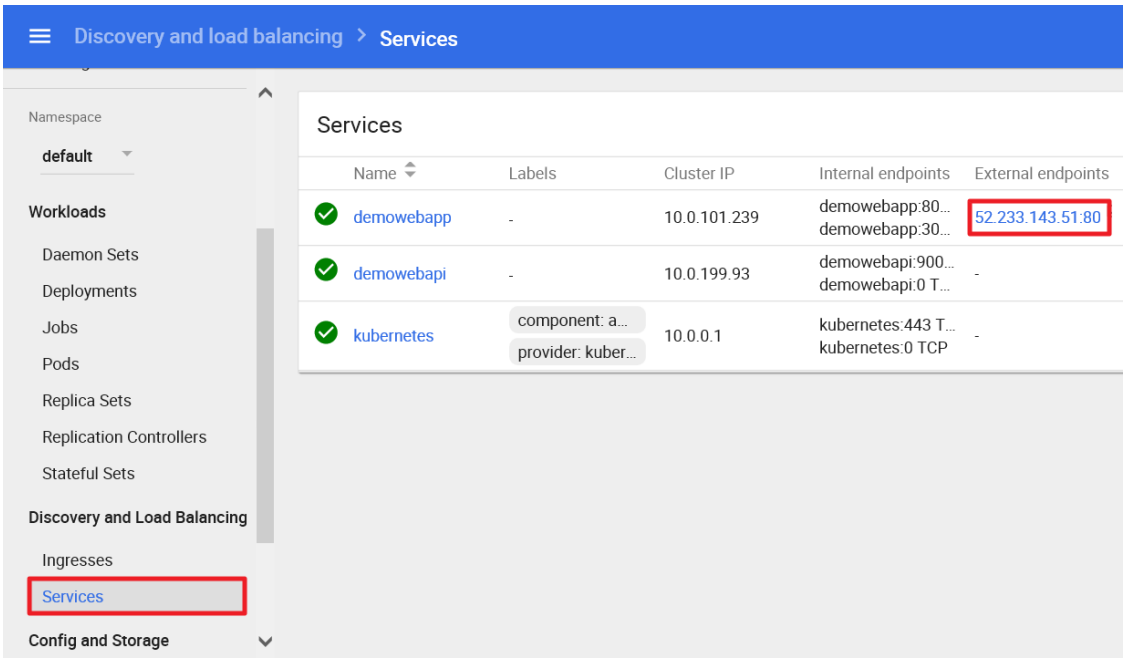
13. Now your deployment in Azure Pipelines is completed, but the deployment in Kubernetes cluster is just beginning. Go to Kubernetes cluster dashboard and check deployment status. After waiting for a while, you will see that the deployment is completed.

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

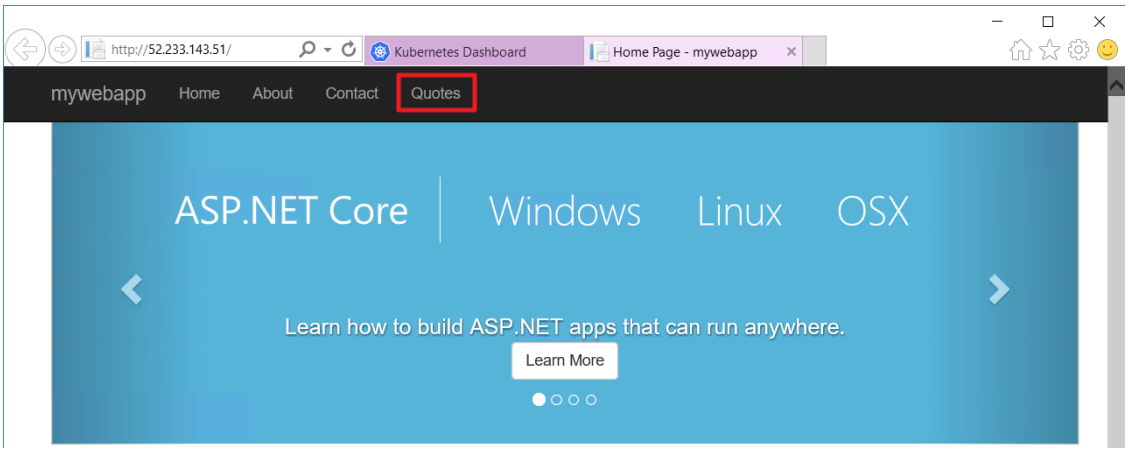
    ```
    az aks browse --resource-group k8s-aks-cluster-rg-INITIALS --name aks-k8s-cluster
    ```



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.



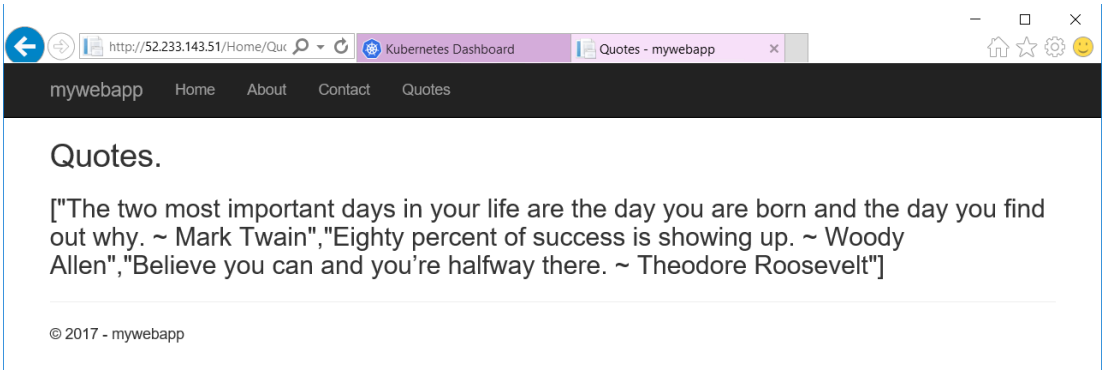
15. When the web site is loaded, click on **Quotes** link.



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.

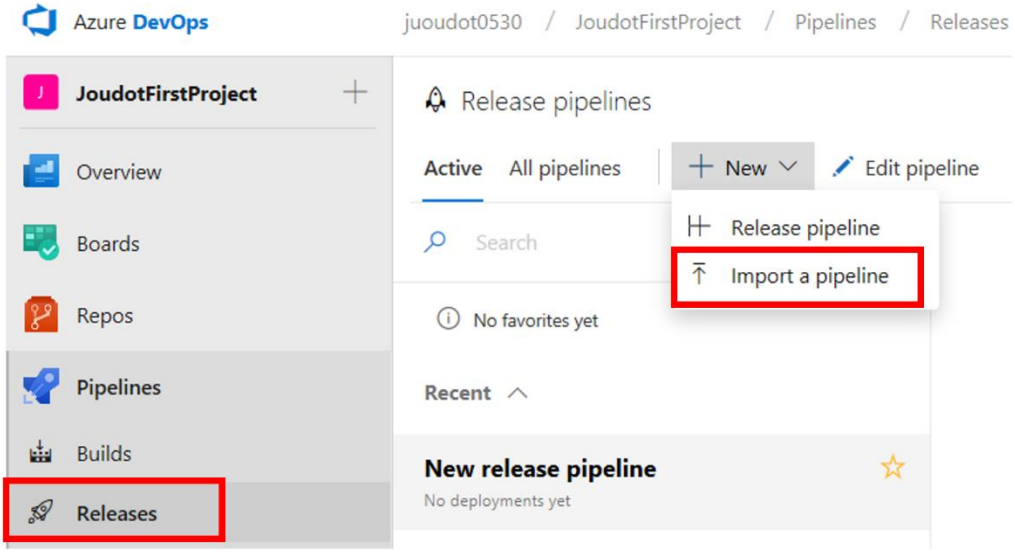


Optional: Import a release pipeline

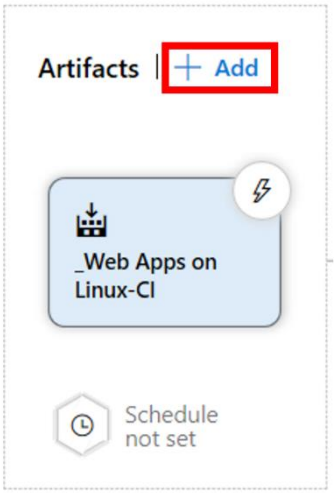
> Skip this optional task if you already created the release pipeline.

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

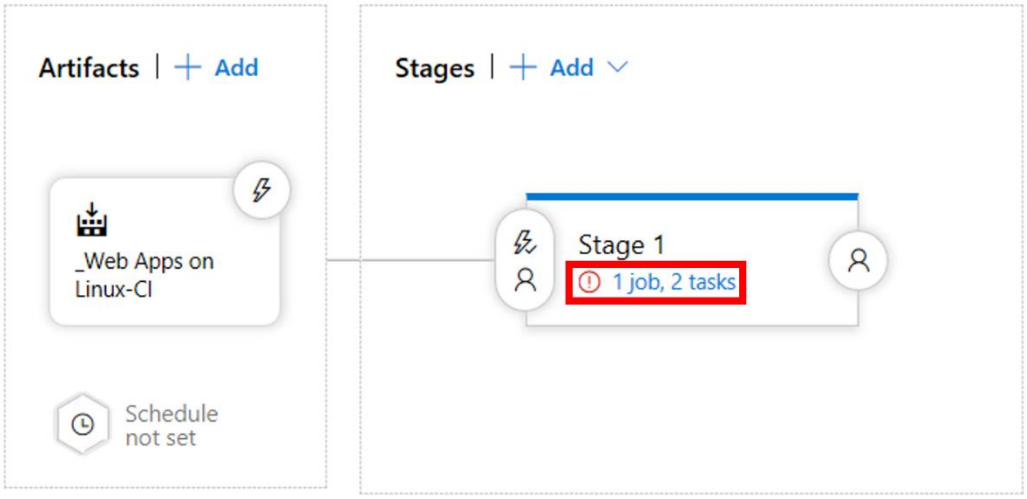
Note: it seems that we can only do that when at least one pipeline exists. So you will need to create an empty pipeline first.



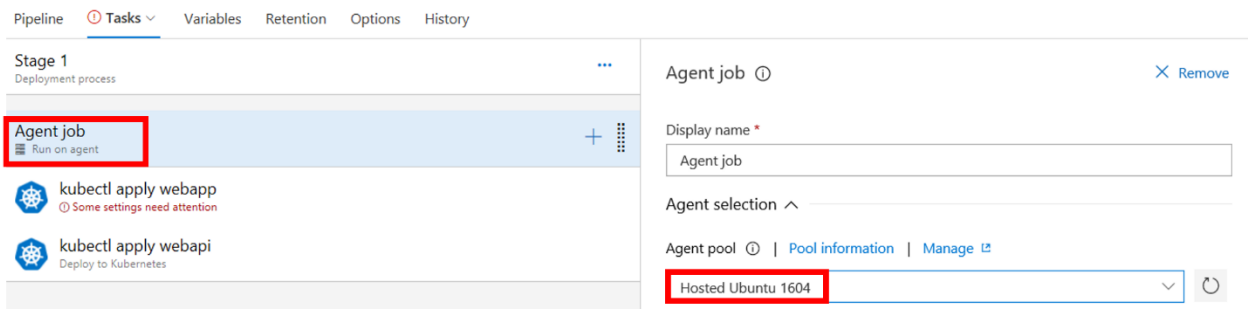
1. Select **C:\labs\module6-ext\AKS - Release-CD.json** and click **Import**
2. Make sure that the input Artifact is valid by deleting the default one: click on the Artifact Name and then **Delete**. Finally, click on **Add an Artifact** to add the artifact coming from your new build.



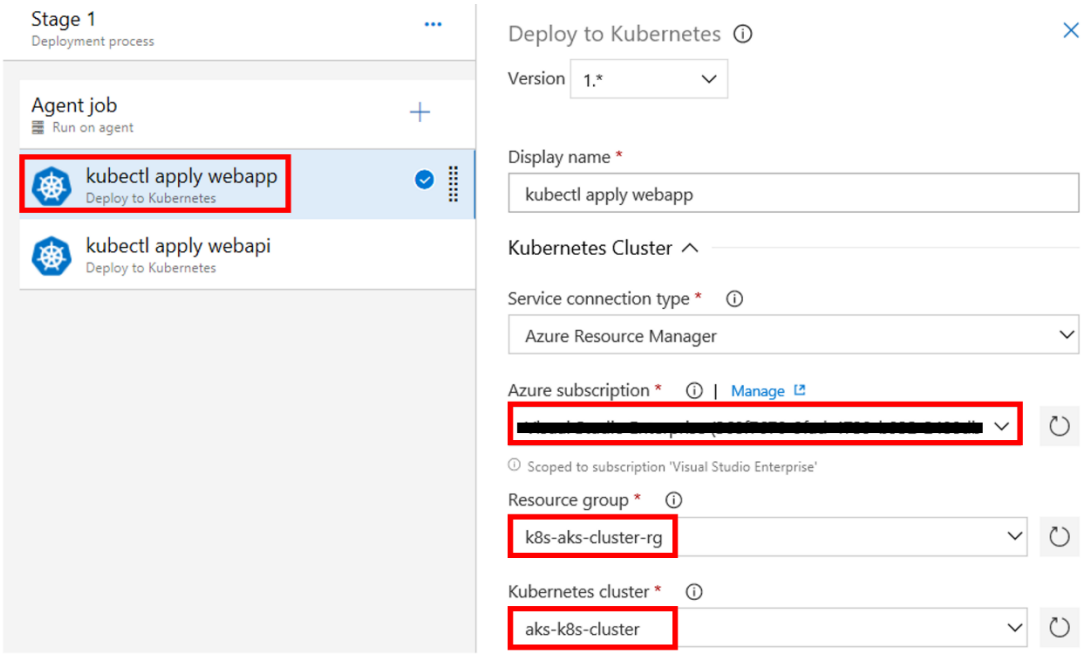
3. Click on the list of tasks under **Stage 1**

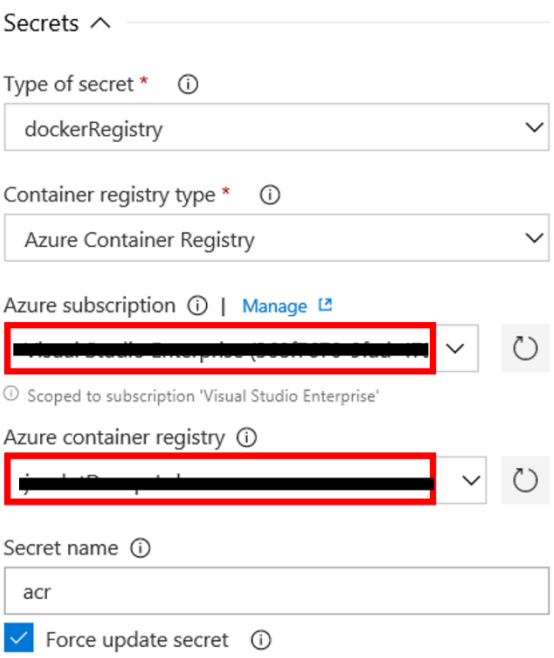


4. Under agent job, select **Hosted Ubuntu 1604** as an agent pool



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





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.

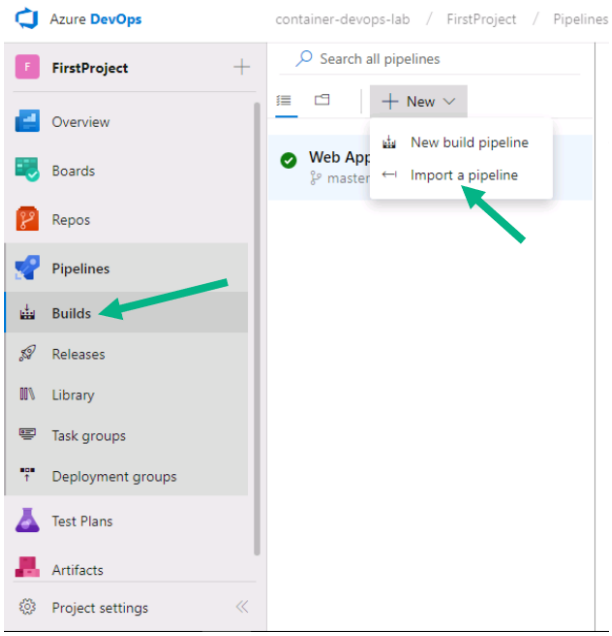
Exercise 5: Create Build Pipeline for Windows Containers

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.

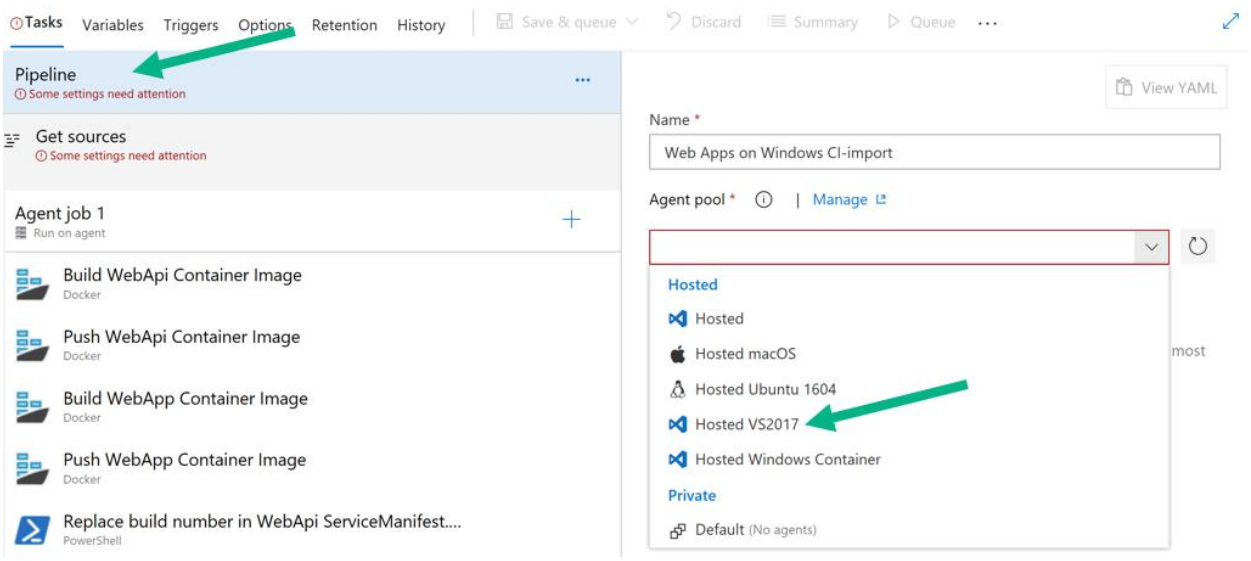
Tasks

Import a build pipeline

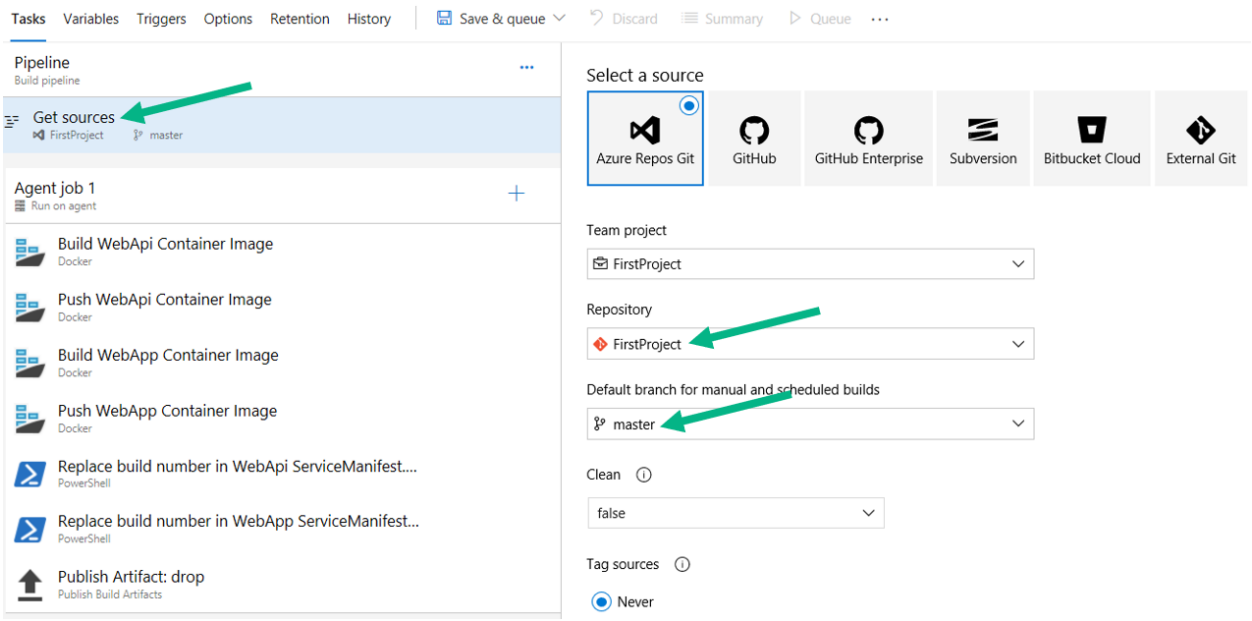
1. Navigate to **Pipelines – Builds** and click **New – Import a pipeline**.



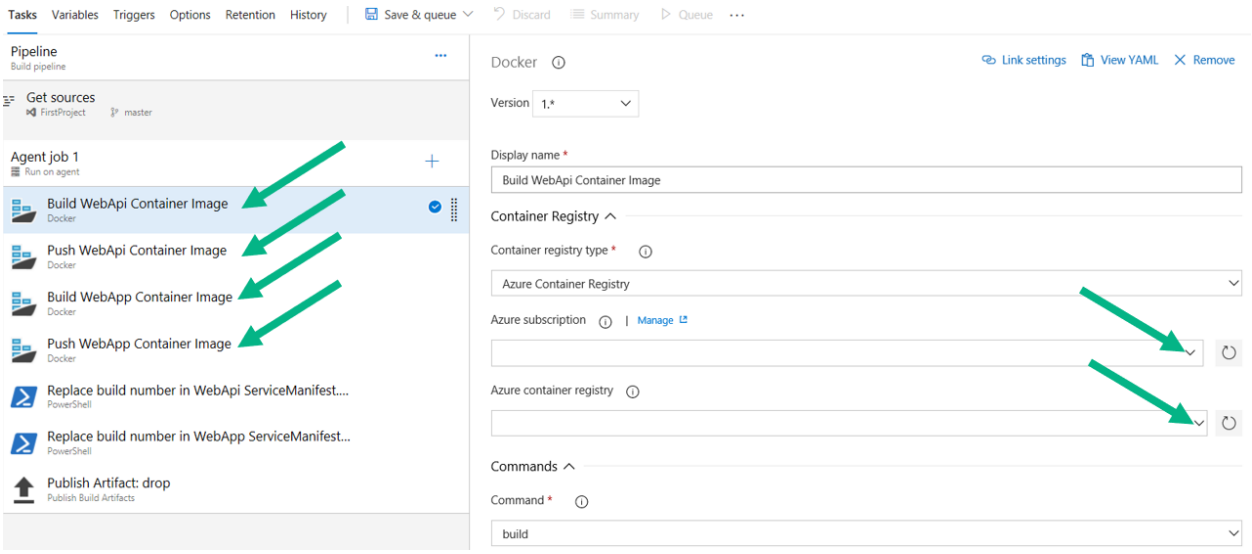
2. Select **C:\labs\module6-ext\Web Apps on Windows-CI.json** and click **Import**
3. Select **Hosted VS2017** as an agent pool. This is a Windows based agent.



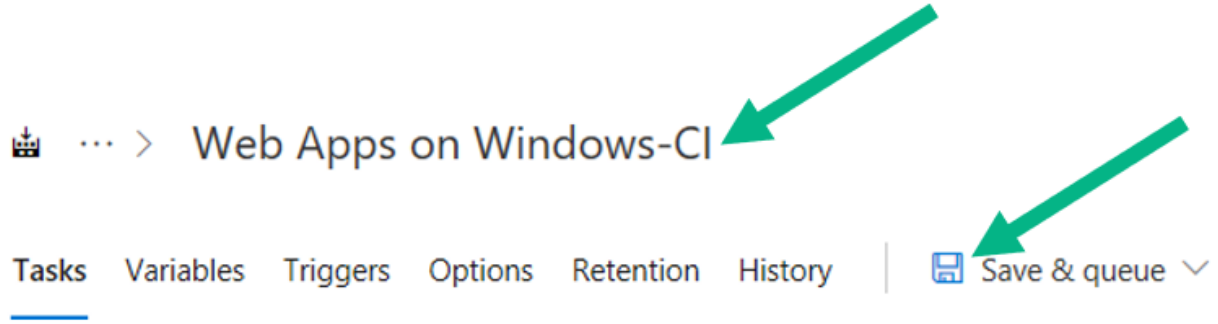
4. Click on **Get sources** and make sure that correct Git repository and branch is selected.



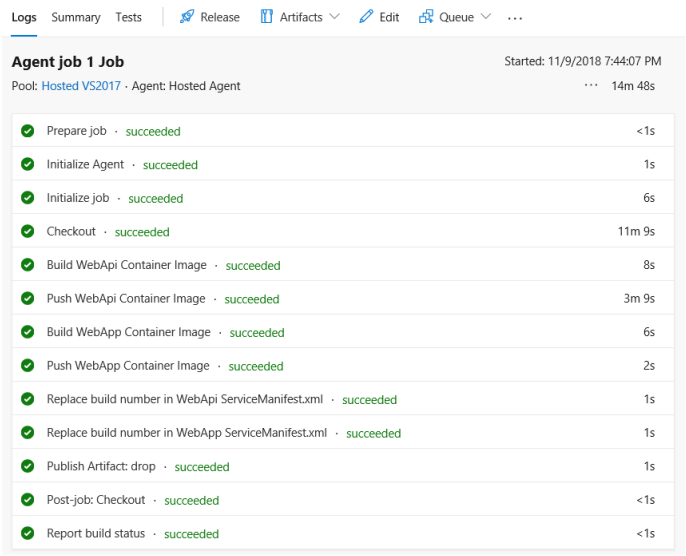
5. Select **all docker tasks** one by one and enter the **subscription and ACR** information.



6. Make sure the Build pipeline is named **Web Apps on Windows-CI** to stay aligned with the following screenshots. Click **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
Started: 11/9/2018 7:44:07 PM
Pool: Hosted VS2017 · Agent: Hosted Agent
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

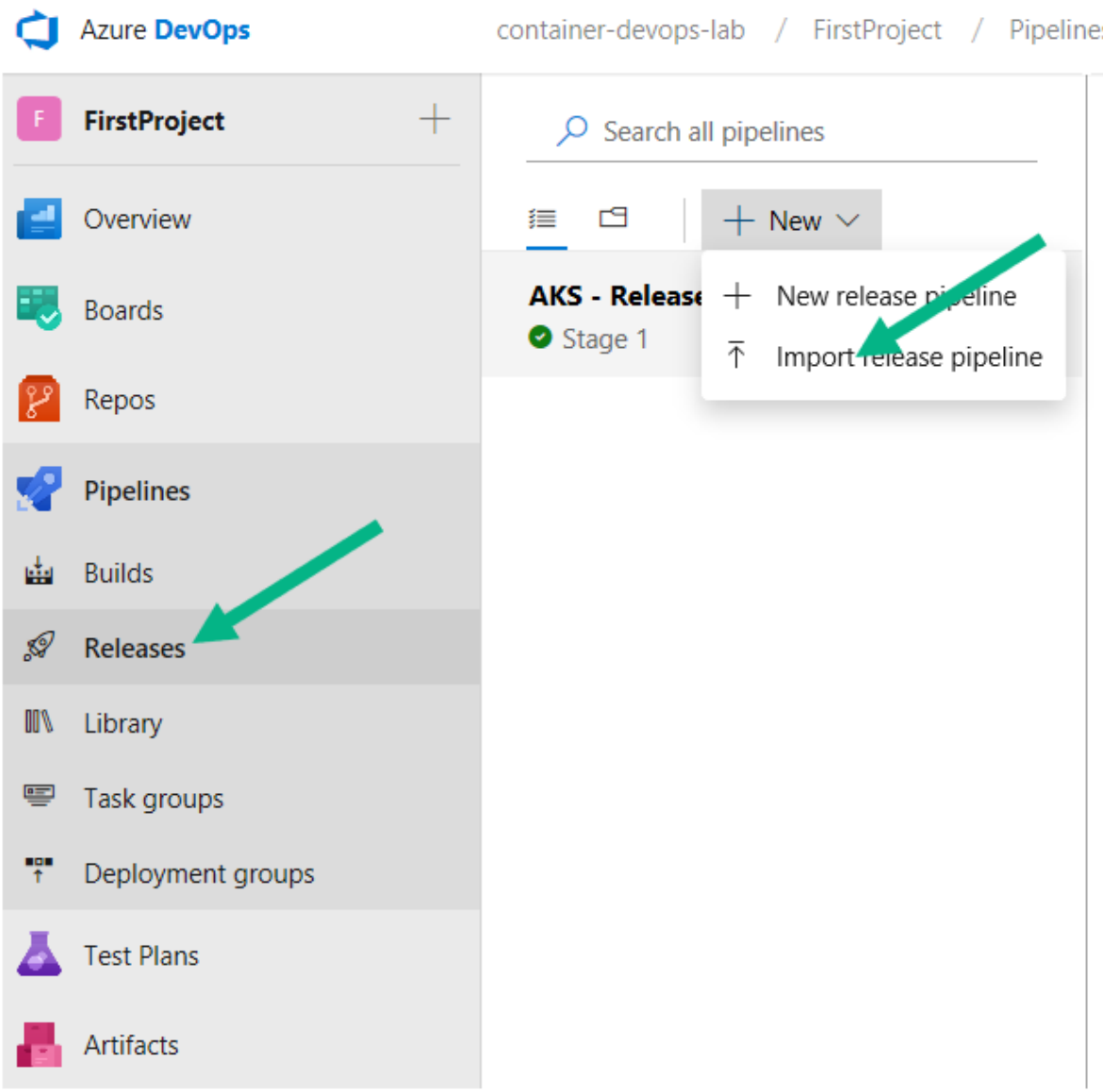
In this exercise, you will create a Release Pipeline which will pull container images from ACR (result of a previous build task), and then deploy these containers to Windows cluster on Service Fabric.

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.

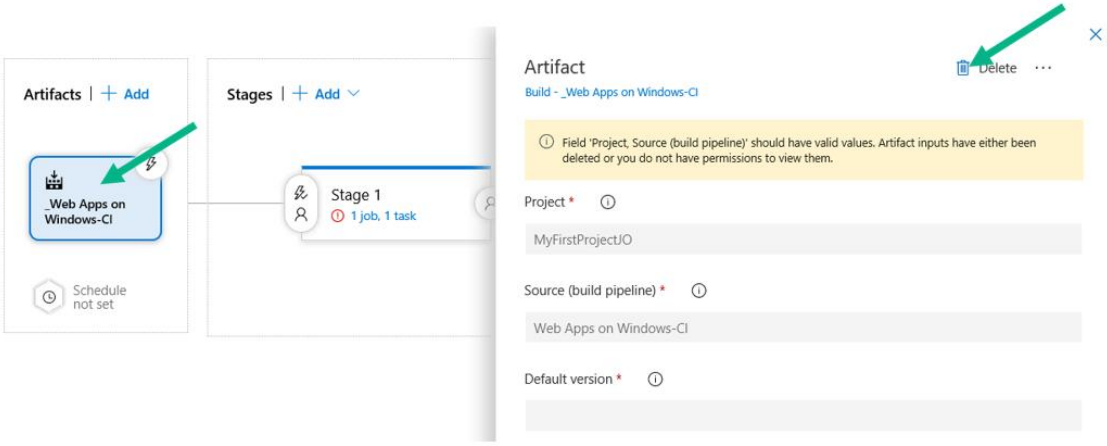
Tasks

Import a release pipeline

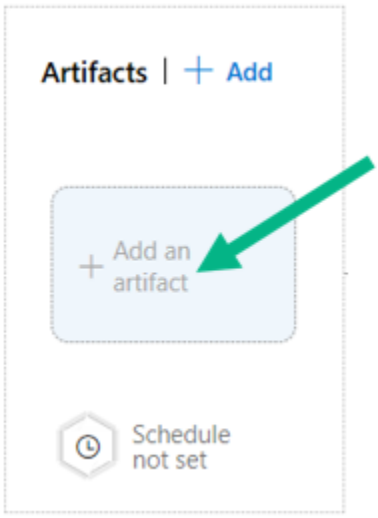
1. Navigate to **Pipelines** – **Releases** and click **New** – **Import a pipeline**.



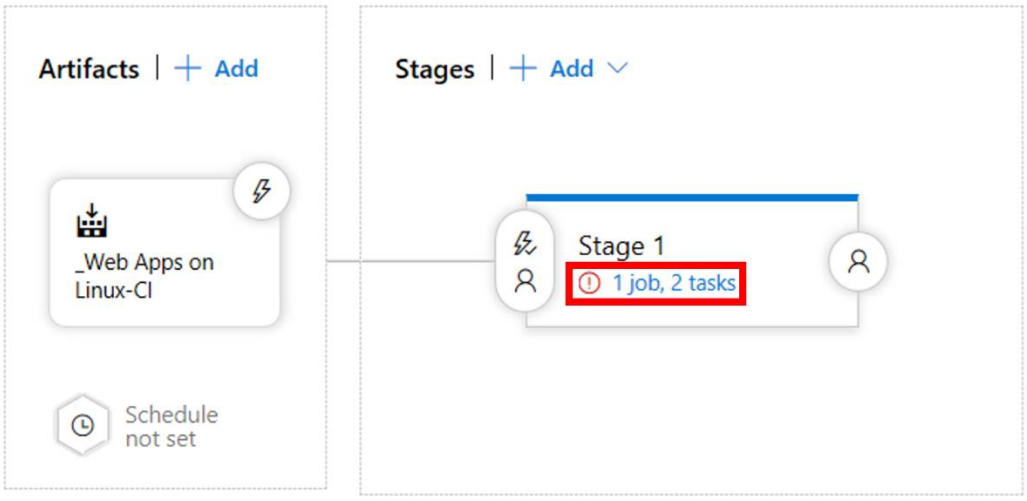
2. Select **C:\labs\module6-ext\SF – Release-CD.json** and click **Import**
3. Make sure that the input Artifact is valid by deleting the default one: click on the Artifact Name and then **Delete**.



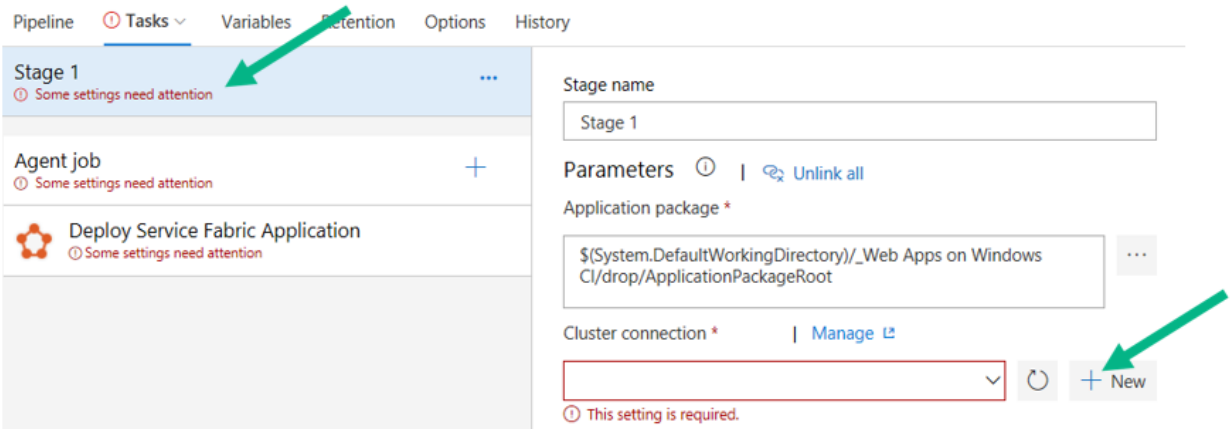
4. Finally, click on **Add an Artifact** to add the artifact coming from your **Web Apps on Windows-CI** build.



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

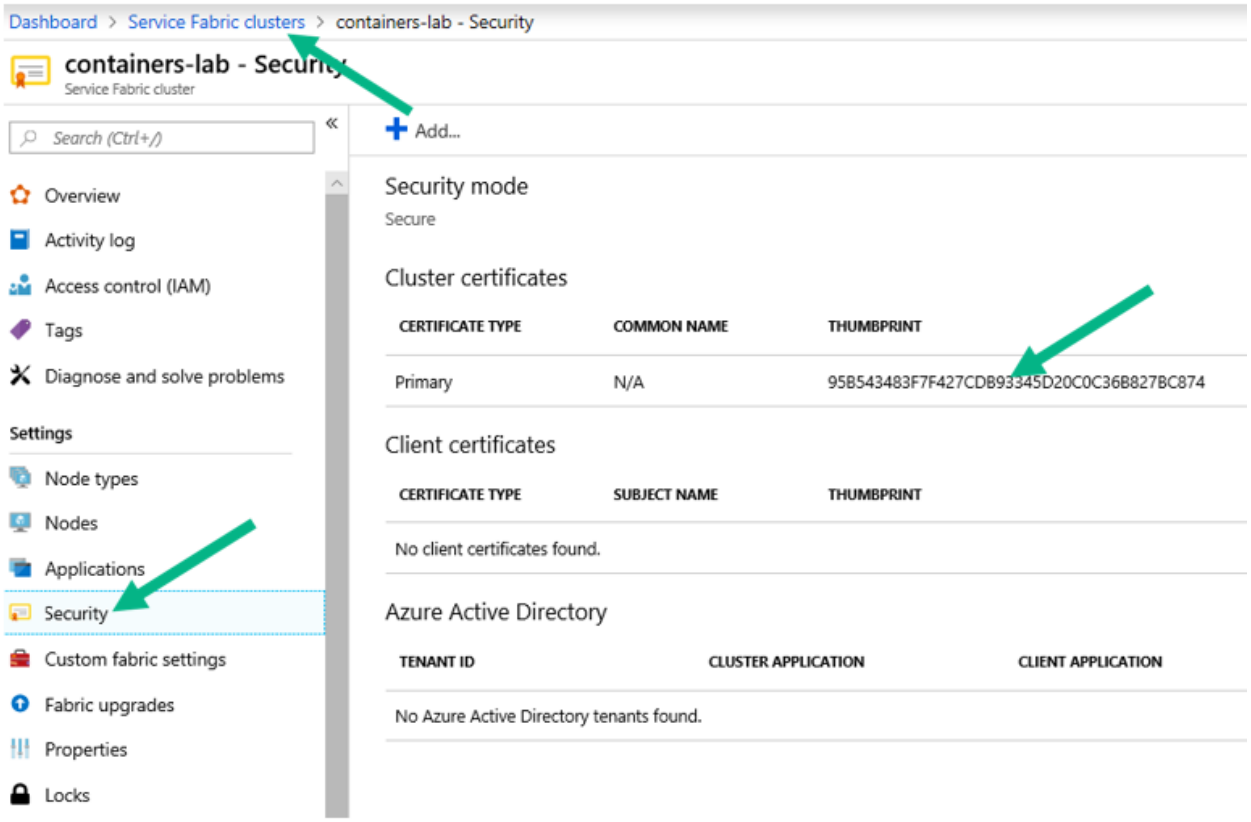


6. In **Stage 1**, click on **New** to add a new cluster connection.



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.

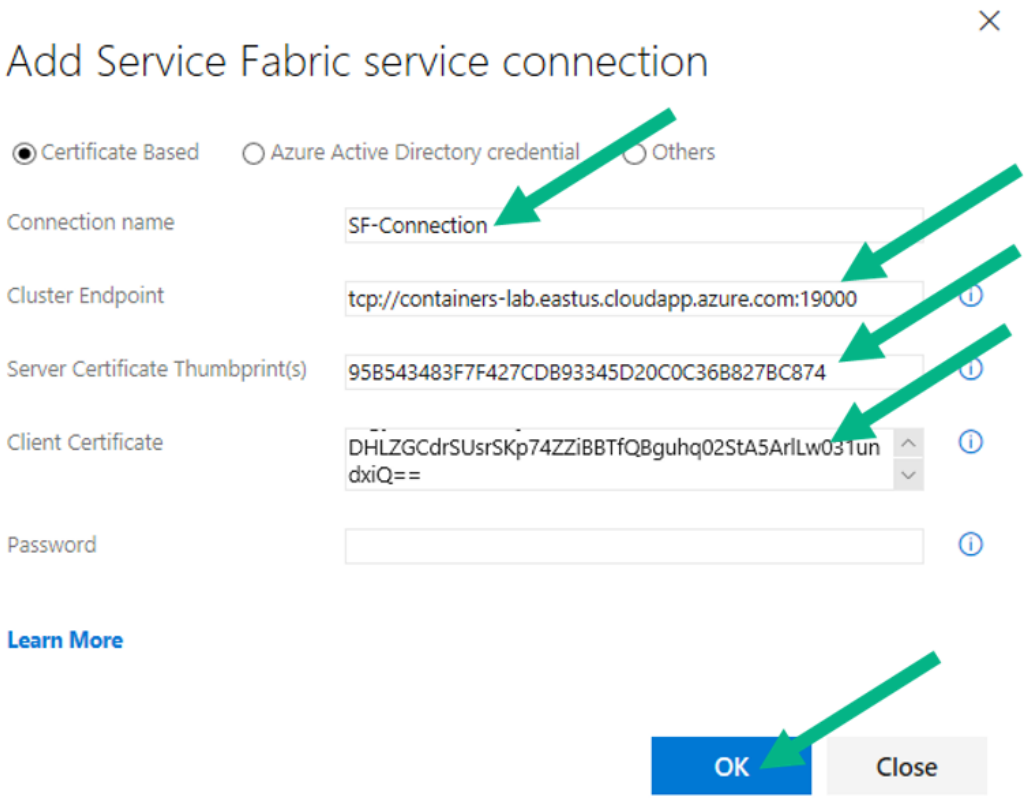


- **Client Certificate**: You can use the following command to obtain a Base64-encoded representation of the client certificate:

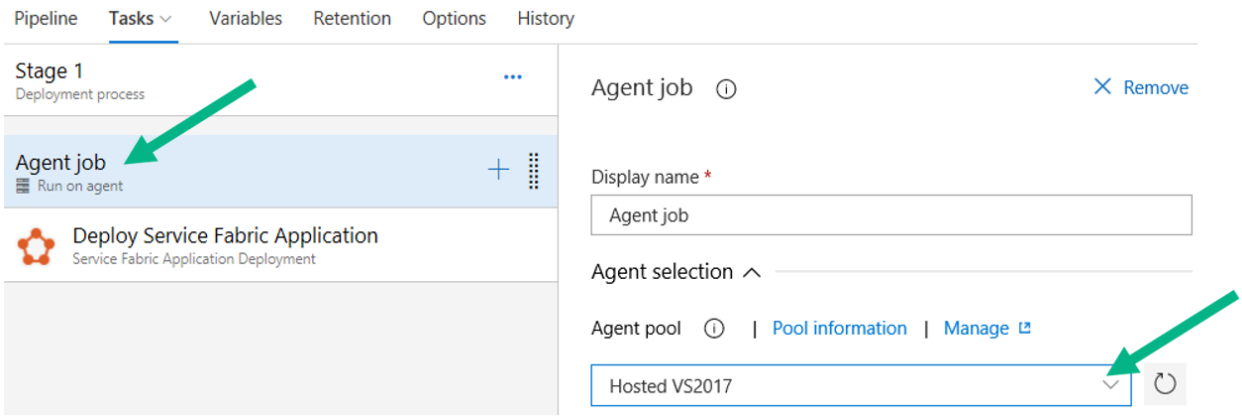
```
[System.Convert]::ToBase64String([System.IO.File]::ReadAllBytes("C:\labs\module5\certificateFile.pfx")) > output.txt
```

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

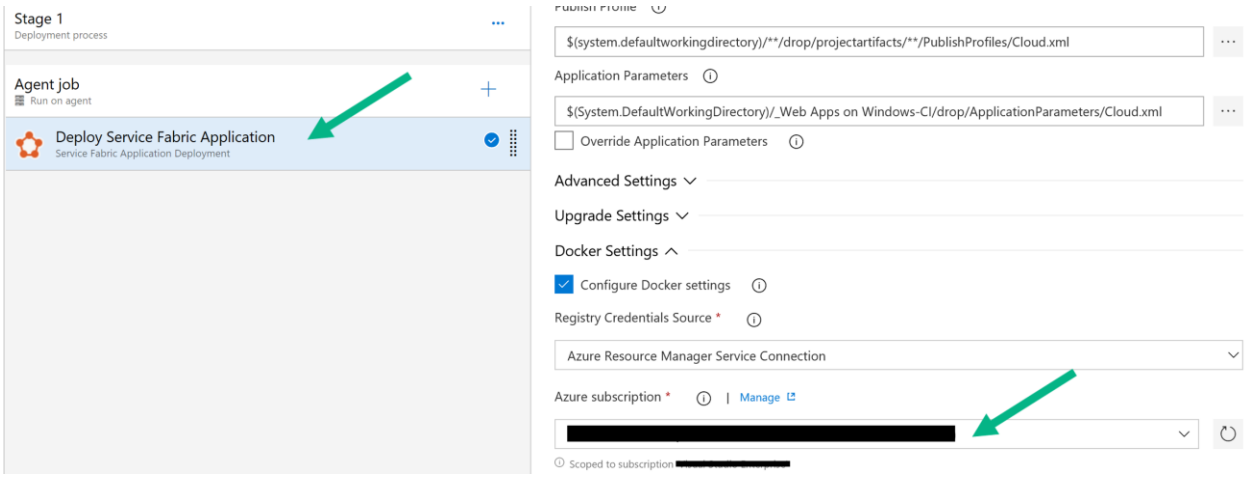
- **Password**: empty (because our certificate does not have any password)



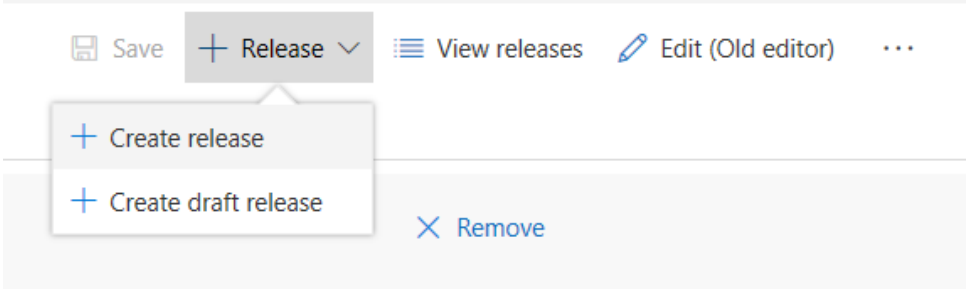
8. Under **Agent job**, select **Hosted VS2017**as an agent pool. This is a Windows agent.



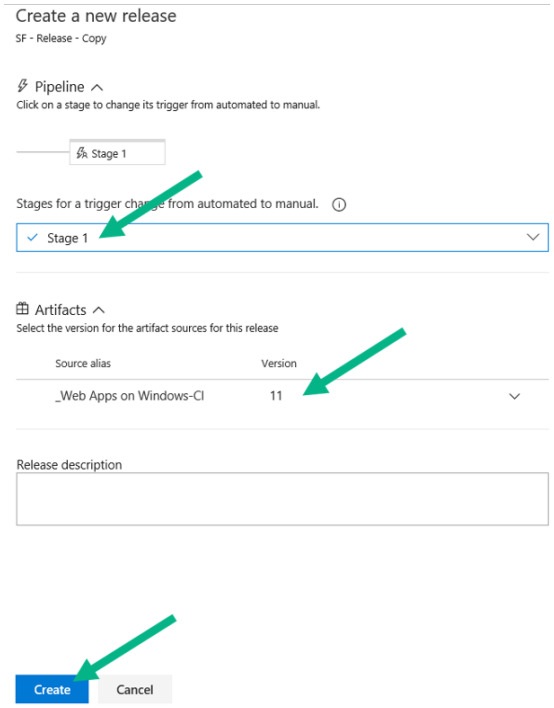
9. Select the **Deploy Service Fabric Application** task and under **Docker Settings** select **Azure Resource Manager Service Connection** and then select your Azure subscription. This will allow the cluster to pull the container images located in the Azure registry.



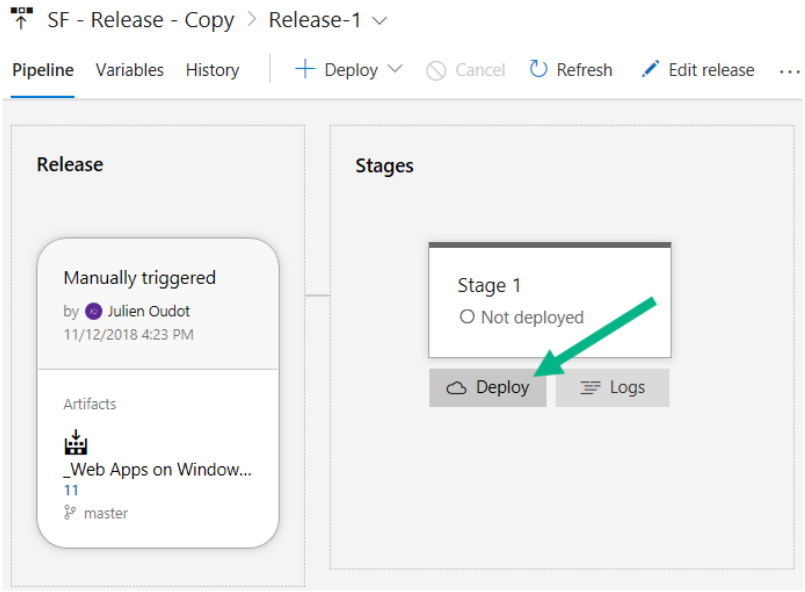
10. Look at the top right of your page and you will see the buttons as in the screenshot below. First, **Save** the release pipeline. Then click on the **Release** button and choose "+ **Create Release**" to trigger a release.



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

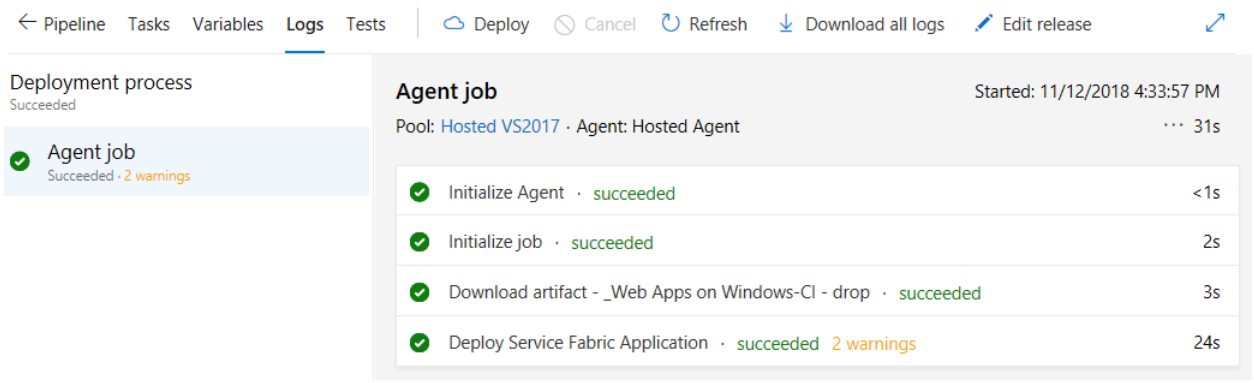


Go back to the Release page by clicking on the name of the created release at the top of the screen. Click **Deploy**.

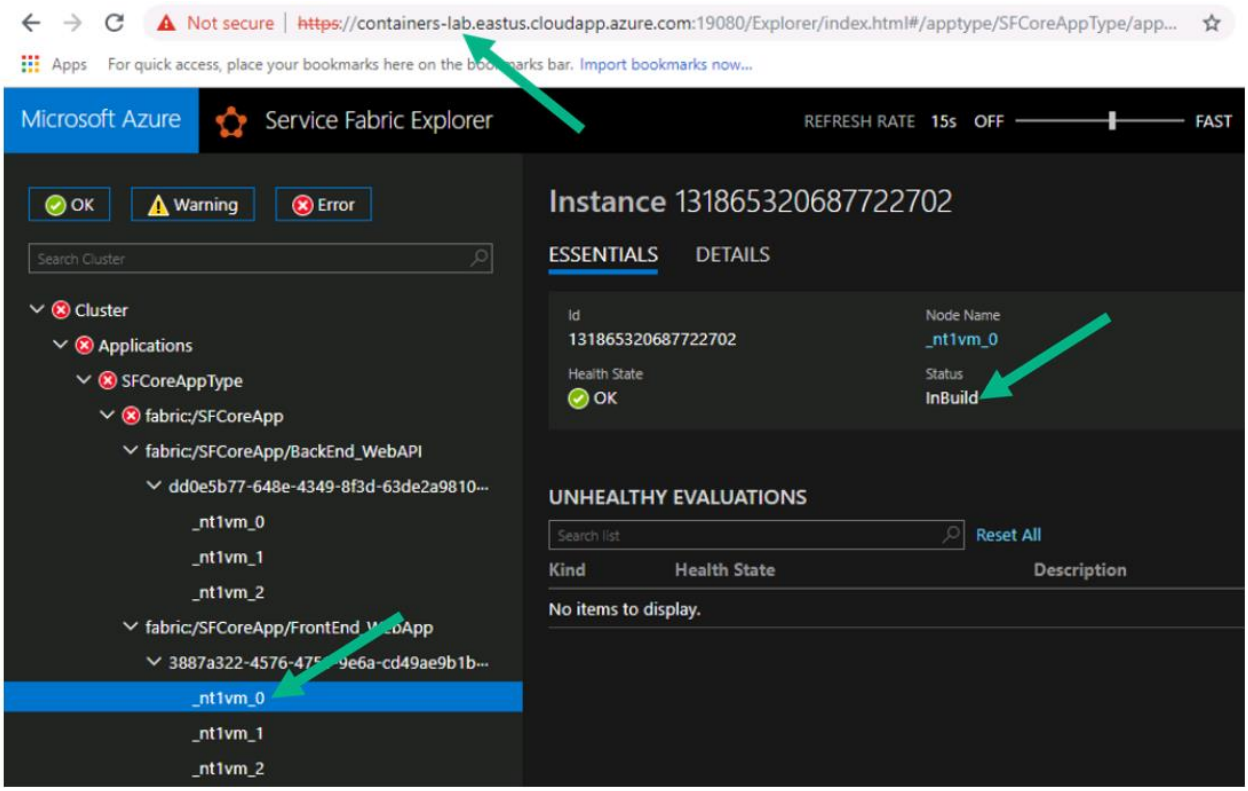


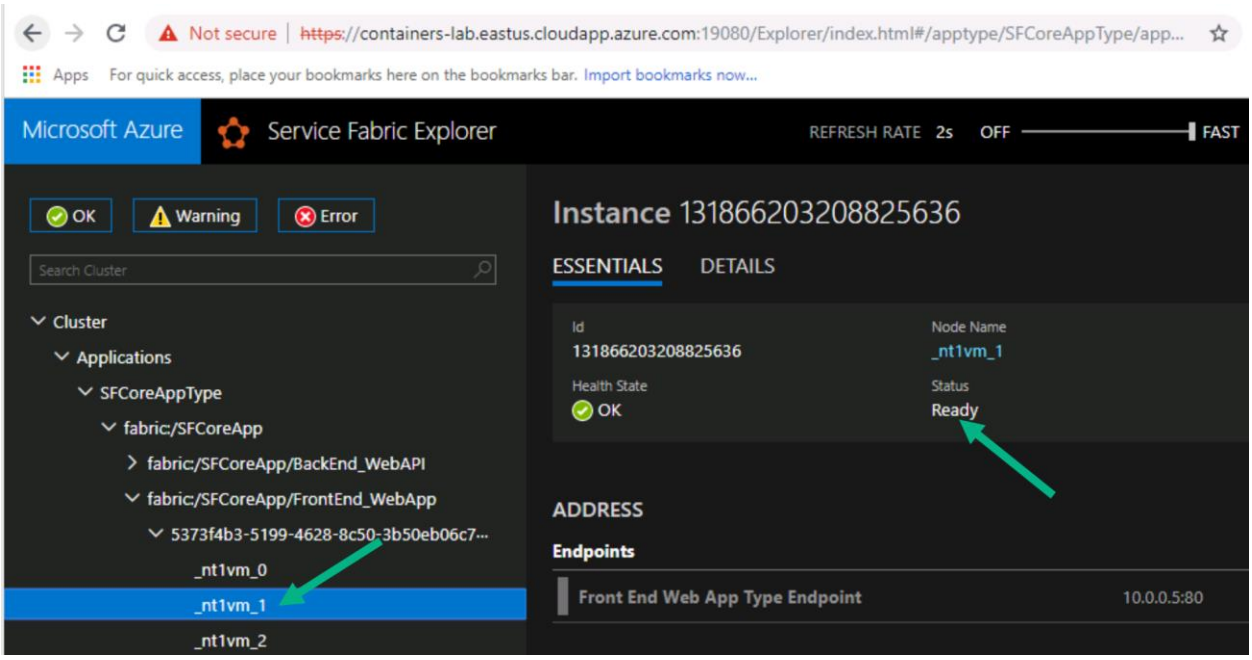
Finally, press **Deploy** to kick off the deployment to Service Fabric cluster.

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

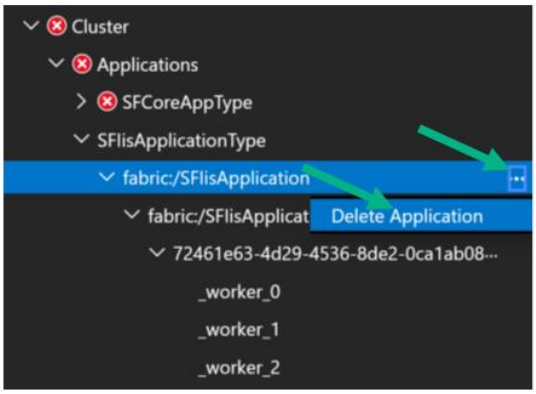


11. Now your deployment in Azure Pipelines is completed, but the deployment in Service Fabric cluster is just beginning. Go to Service Fabric explorer and check the deployment status. After waiting for a while, you will see that the deployment goes from **InBuild** to **Ready** state.

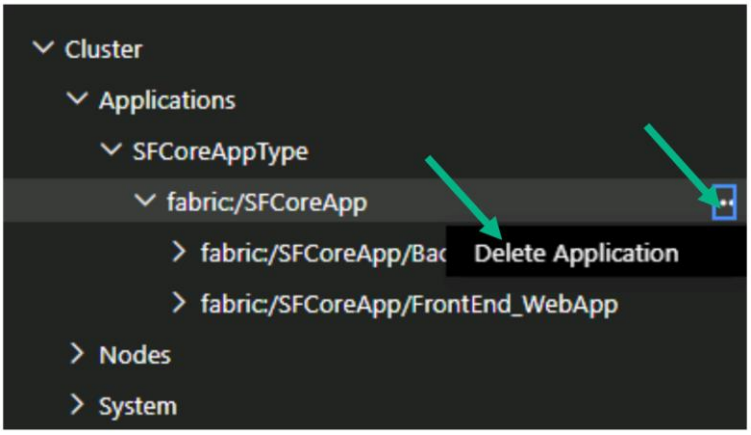




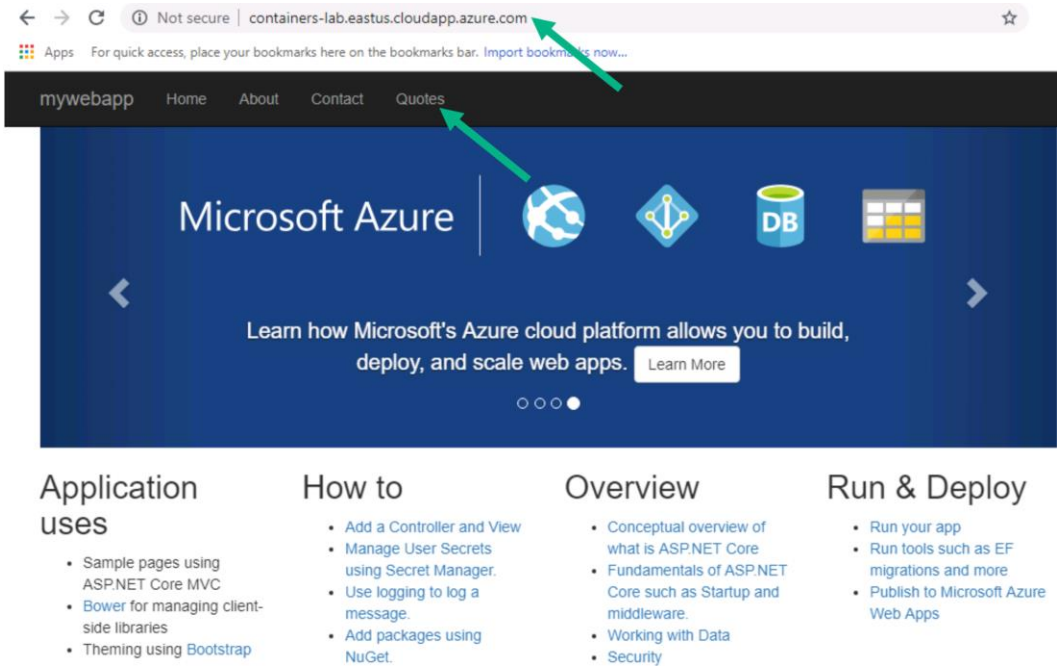
If you still have the IIS application deployed, you will see an error saying that the port 80 is not available. To fix that, click on the thee dots next to the **fabric:/SFIisApplication** and **Delete Application**.



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.



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