

**Module 7 Lab: Kubernetes CI/CD to Azure**

Template Version: 2.0

**Estimated Time**

60 minutes

**Objectives**

At the end of this lab, you will be able to:

* Complete a CI/CD in VSTS for AKS

Table of Contents

[Lab: Module 6 - DevOps with Containers 3](#_Toc523593750)

[Exercise 1: Setup Project and Azure Container Registry (ACR) 3](#_Toc523593751)

[Exercise 2: Create Build Definition for Linux Containers 5](#_Toc523593752)

[Exercise 3: Create Release Definition for AKS Cluster 21](#_Toc523593753)

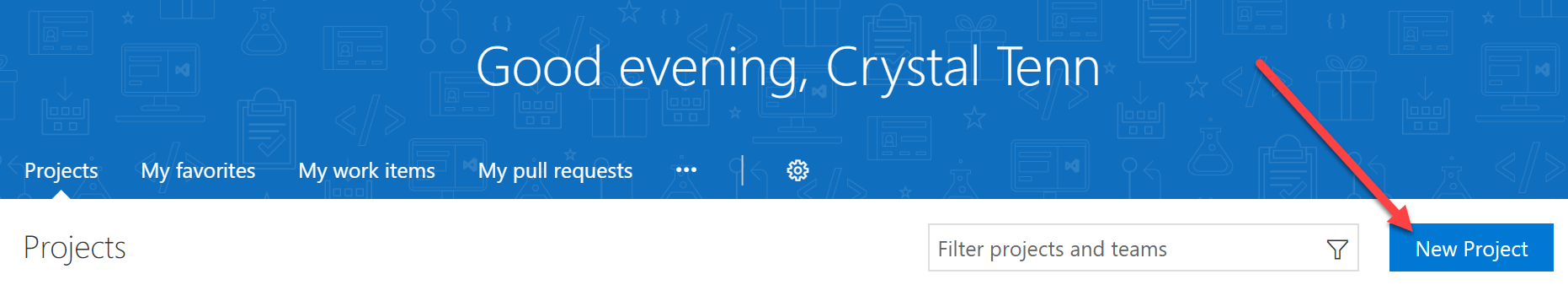
Lab: Module 6 - DevOps with Containers

Exercise 1: Setup Project and Add to VSTS

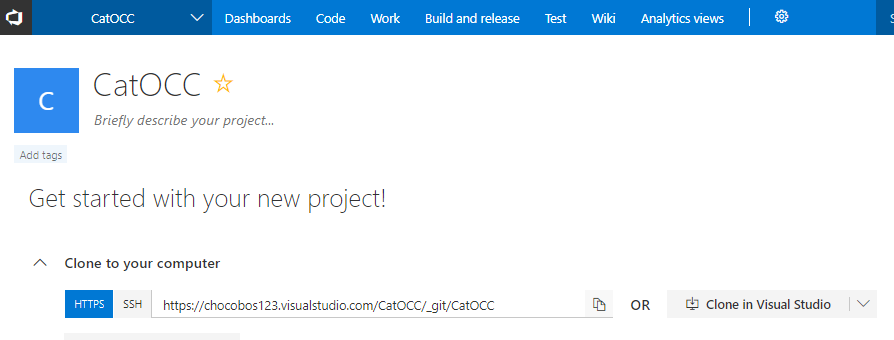
In this exercise, you are going to complete first part of CI/CD pipeline by pushing the code artifacts to VSTS which includes sample web application and web API. You will also create an Azure Container Registry (ACR) to store Docker images in the private repository.

Tasks

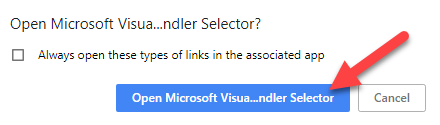
1. Edit yaml files
2. RDP into your VM and use VM for all steps on local machine (VSTS steps can be done either on VM or your local computer).
3. Go to C:/labs/day2
4. Open the backend-webapi.yaml file and change text for image from **yogurtthecat.azurecr.io/demo-webapi** to **initialsnewregistry.azurecr.io/demo-webapi:BuildNumber**
5. Open the frontend-webapp.yaml file and change text for image from **yogurtthecat.azurecr.io/demo-webapi** to **initialsnewregistry.azurecr.io/demo-webapi:BuildNumber**
6. Add Solution to Source Control (Git and VSTS)
7. Navigate to initials-occ.visualstudio.com
8. Sign in if needed.
9. Click New Project.



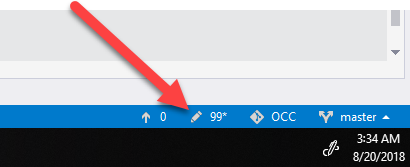
1. Name it initials-OCC-containers
2. Leave all other settings as default, Git and Scrum. Click Create.
3. After the Create step, you will see a landing page like below. Click Clone in Visual Studio.



1. Hit Open Microsoft Visual Studio… Selector



1. Wait a couple of minutes for Visual Studio to open and for a modal to pop-up like below. Click the “3 dots” to change the local path.
2. Add a new OCC-containers folder under the C:\ drive. Change the local path to the C:\OCC-containers drive. Click Clone.
3. Copy all the files from the C:\labs\day2 folder into the C:\OCC-containers folder.
4. Click on the pencil icon on the bottom of Visual Studio with a # next to it.



1. Write a commit message: added initial project to VSTS
2. Commit the code.
3. Push the code to VSTS.
4. Verify the code is in VSTS.

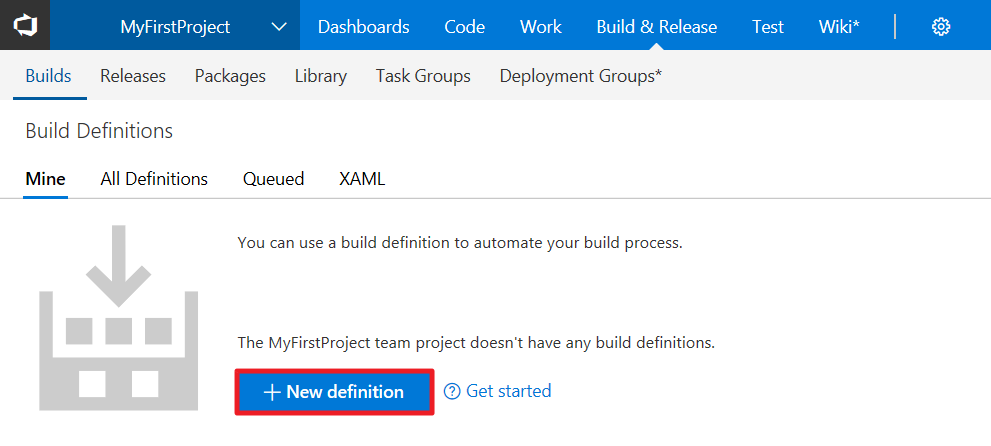
Exercise 2: Create Build Definition for Linux Containers

Tasks

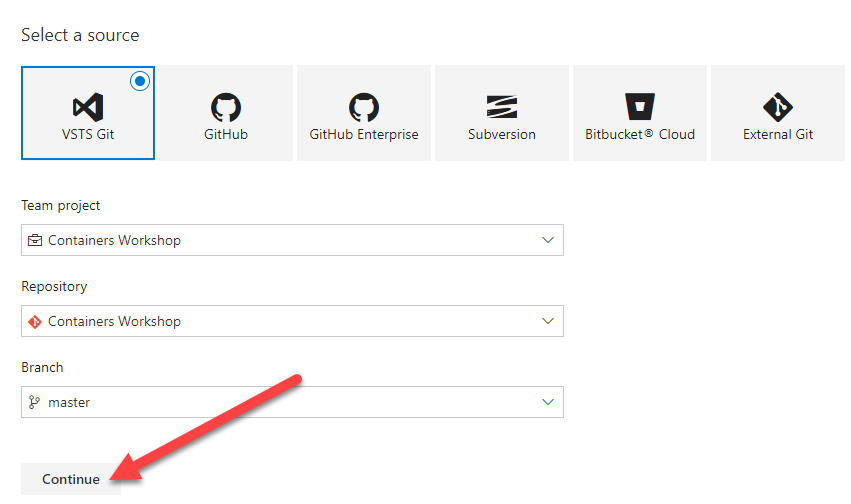
1. Create new build definition

In this task, you are going to create a build definition 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).

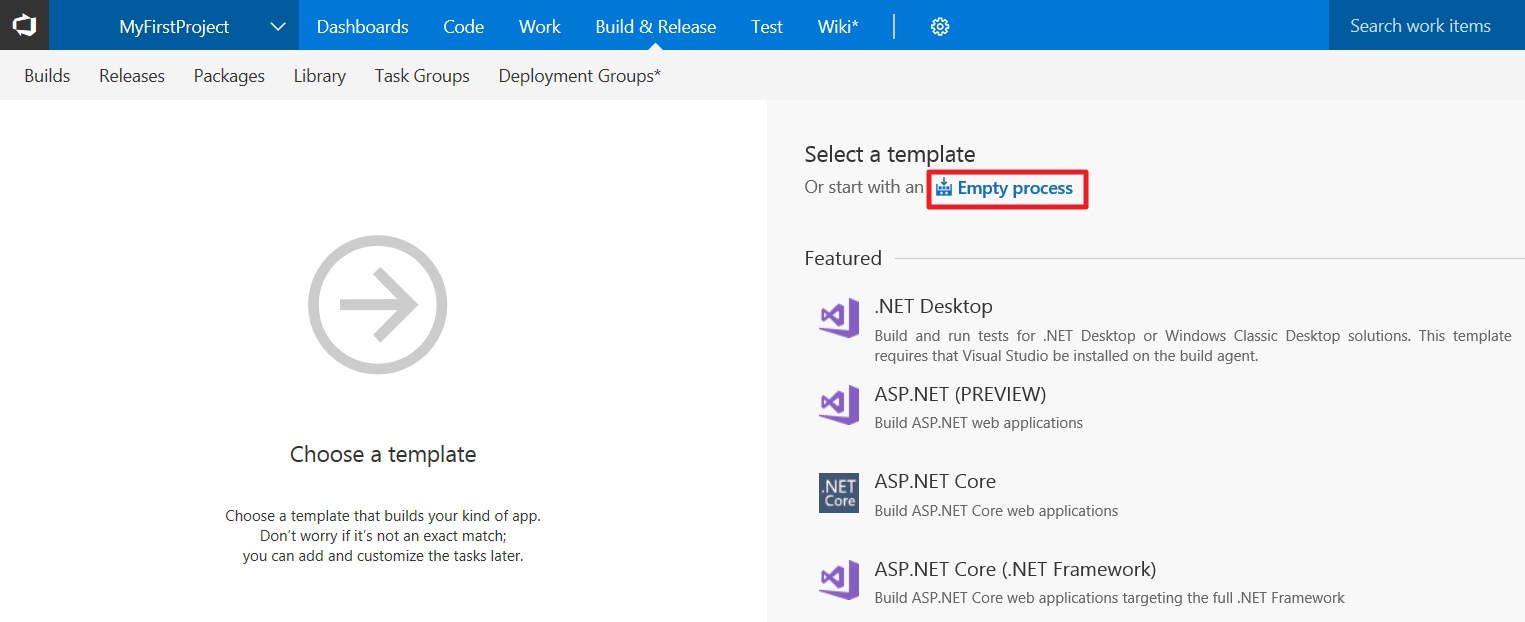
1. On VSTS portal go to **Builds** page and click **New definition**.



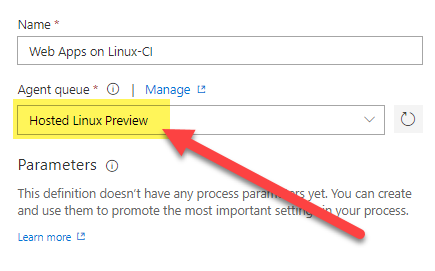
1. Choose the correct team project and repository, should be defaulted to the right one. Click Continue.



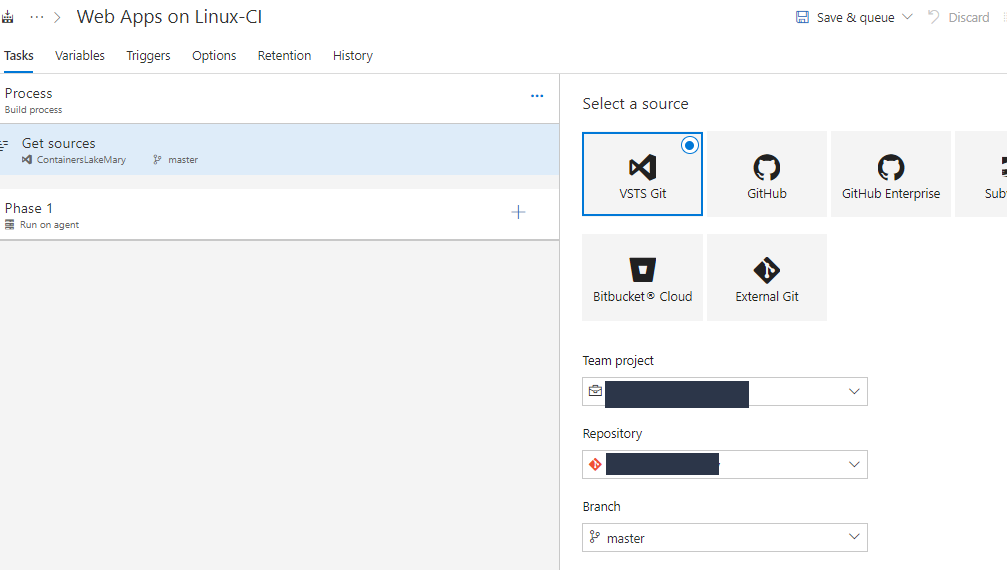
1. Create an empty build definition by clicking on **Empty process / Empty pipeline**.



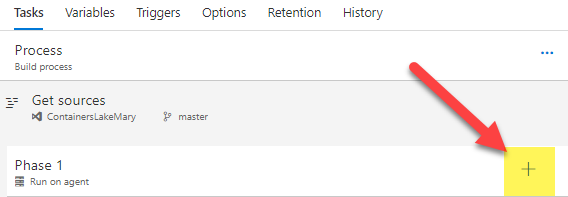
1. Before adding build tasks to the process, you are going to select an agent queue containing agents which will be assigned to handle build requests. Select **Hosted Linux Preview** (has to say Hosted Linux, may or may not say Preview, if you miss this step you will get an error later) agent queue which contains agents installed on Linux OS.



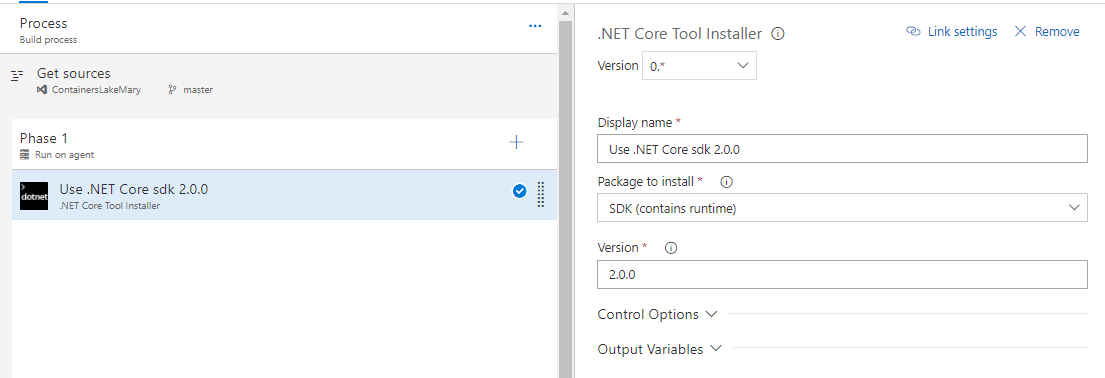
1. Click on **Get sources** task and leave default options.



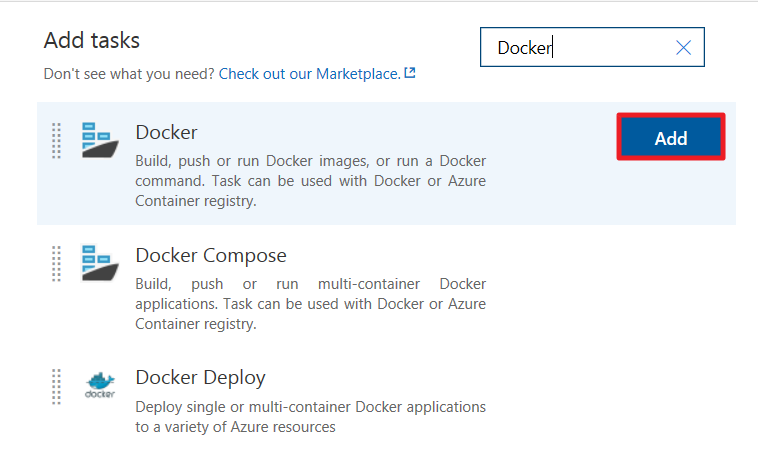
1. Now we can start adding necessary tasks to build the web application and web API. You will click the + sign to add new tasks:



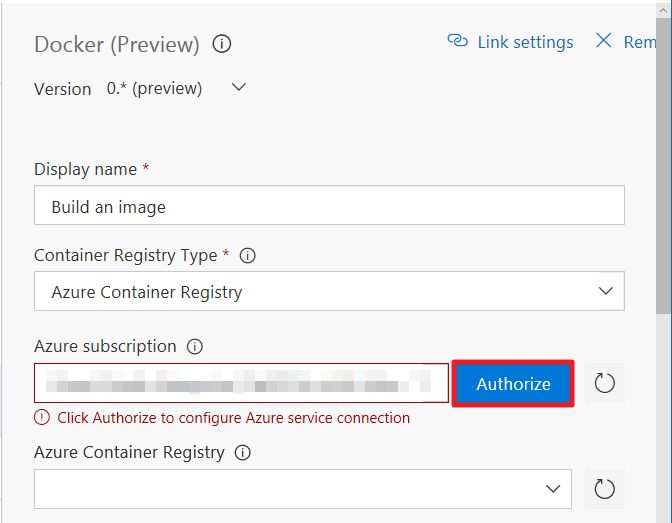
1. Add one **.NET Core** **Tool Installer** task (this will download and install the correct version of the .NET Core SDK for your applications). Set **Package to install** to **SDK** and set **Version** to **2.0.0**.



1. 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 definition.
2. Add a **Docker** task to build container image for web API application.

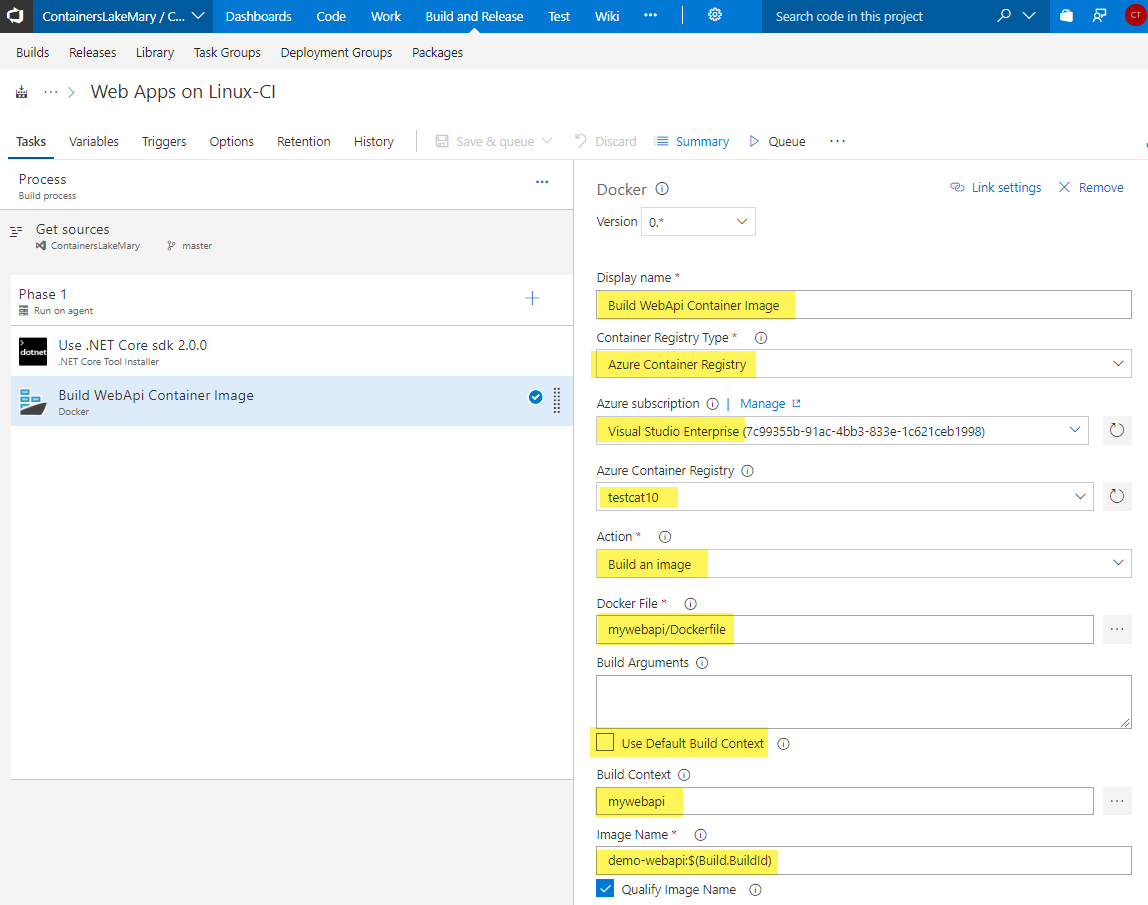


1. Select and Azure subscription and click **Authorize** to allow VSTS account access to the resources in the subscription.



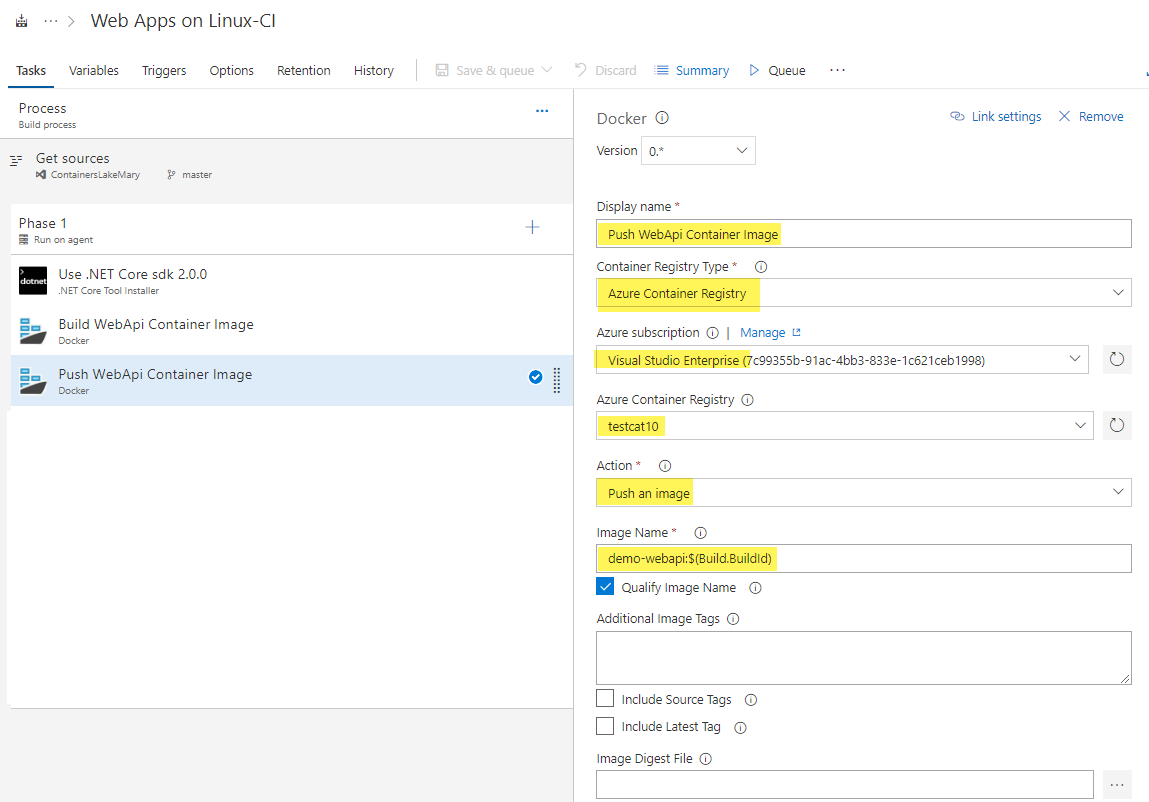
Now configure this task’s other properties as follows:

* **Version**: 0\*
* **Display Name**: Build WebApi Container Image
* **Container Registry Type**: Azure Container Registry
* **Azure Container Registry**: Select the container registry you have created in Task 3 of this exercise.
* **Action**: Build an image
* **Docker File**: mywebapi/Dockerfile
* Unselect **Use Default Build Context**
* **Build Context**: mywebapi
* **Image Name**: demo-webapi:$(Build.BuildId)

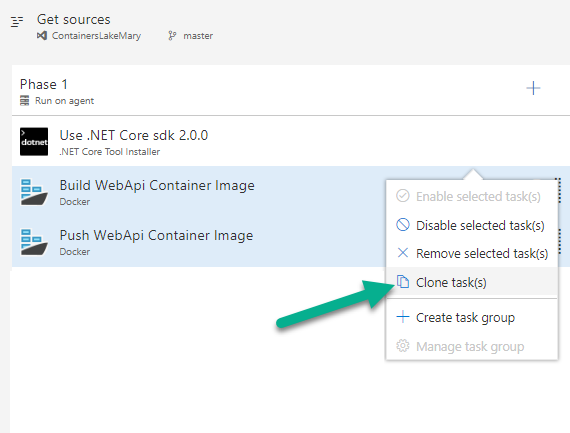


1. Now add another **Docker** task to push the image to ACR.

* Version: 0\*
* **Display Name**: Push WebApi Container Image
* **Container Registry Type**: Azure Container Registry
* **Azure Container Registry**: Select the container registry you have created in Task 3 of this exercise.
* **Action**: Push an image
* **Image Name**: demo-webapi:$(Build.BuildId)

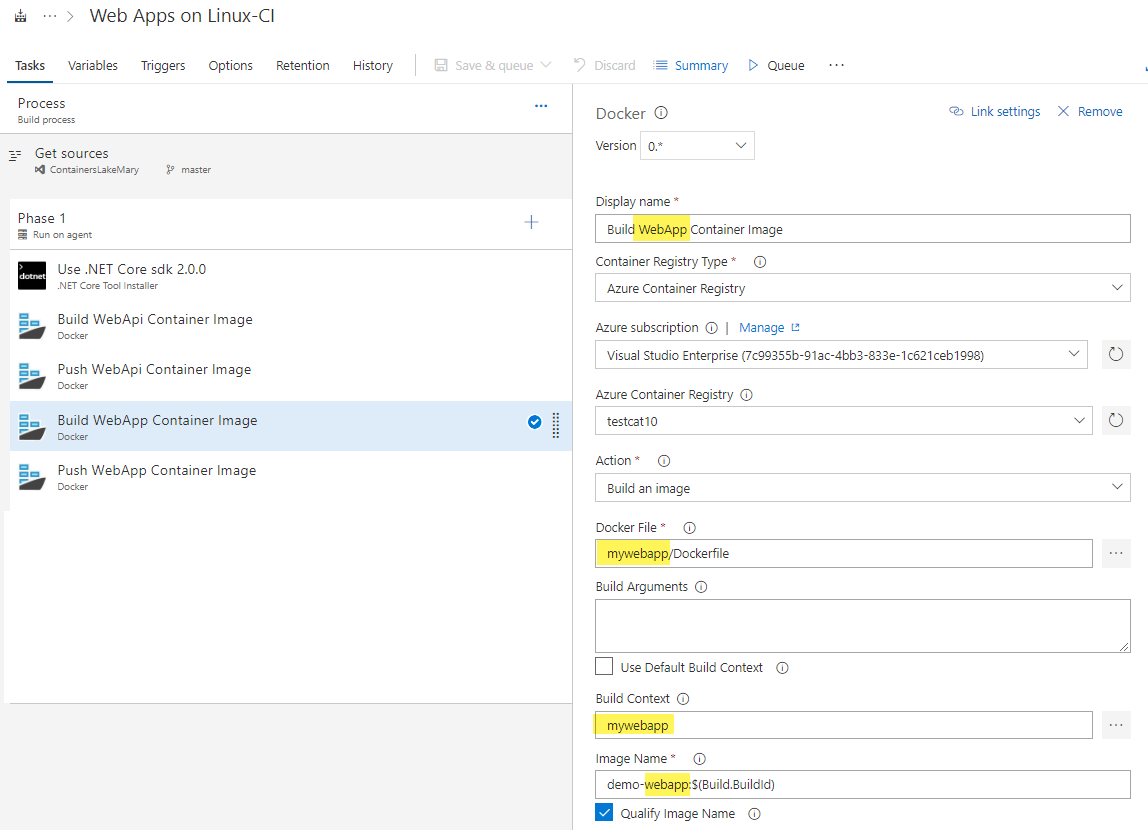


1. You added two Docker tasks for webapi project. Now you will add another two Docker tasks for webapp project. Highlight your Build and Push Docker tasks with **select** and clicking, then **right** click and choose **Clone task(s).**



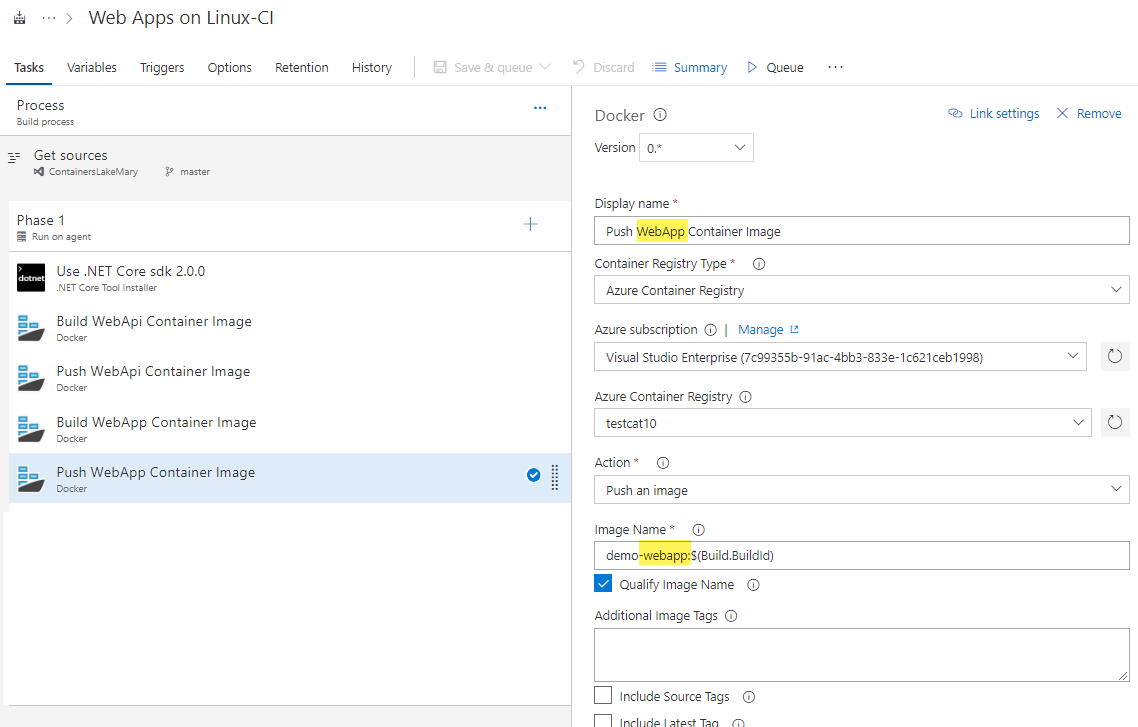
1. Change the **Build WebApi Container Image copy** task to the following attributes:

* Version: 0\*
* **Display Name**: Build WebApp Container Image
* **Docker File**: mywebapp/Dockerfile
* **Build Context**: mywebapp
* **Image Name**: demo-webapp:$(Build.BuildId)

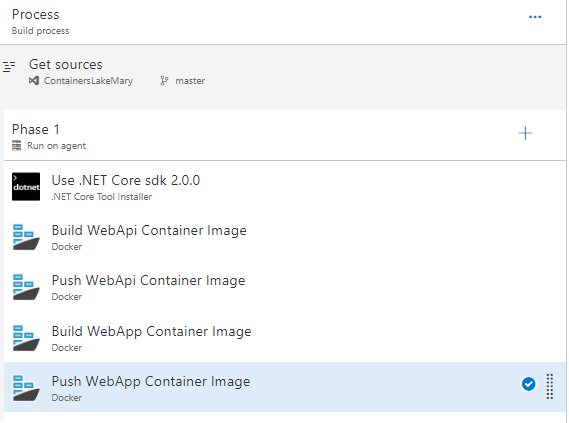


1. Change the **Push WebApi Container Image copy** task to the following attributes:

* Version: 0\*
* **Display Name**: Push WebApp Container Image
* **Image Name**: demo-webapp:$(Build.BuildId)



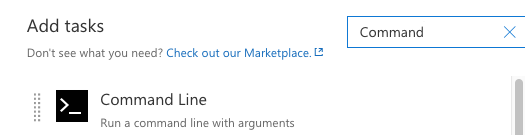
1. Verify you have the following tasks so far:



*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.*

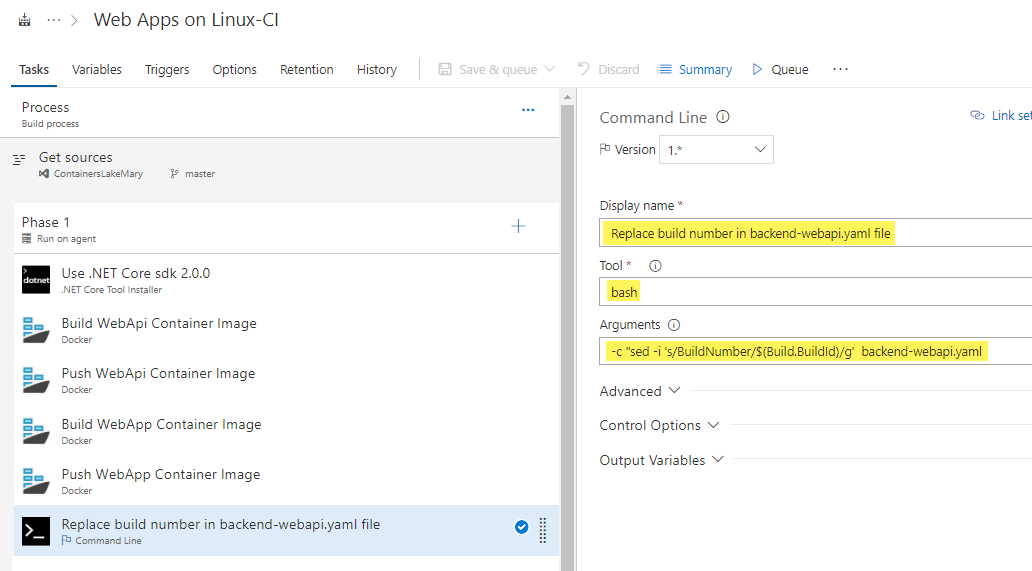
*To pick these images up from ACR and deploy to Kubernetes container orchestrator, you will need to use the same build number in release definition. Therefore you need to place it in Kubernetes deployment configuration files:* ***backend-webapi.yaml*** *and* ***frontend-webapp.yaml****.*

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



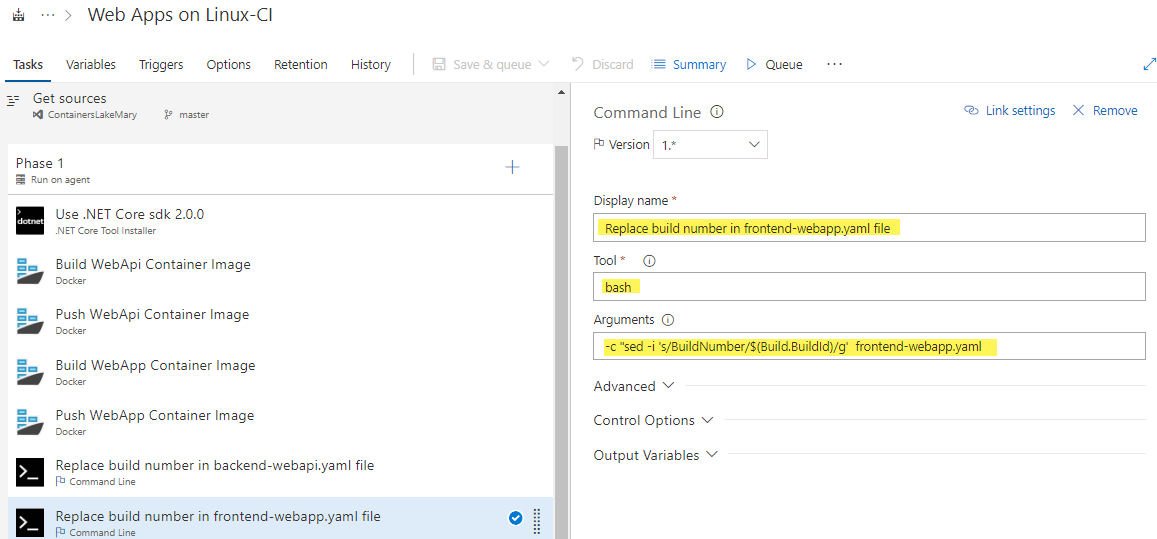
Configure the first **Command Line task** as follows:

* **Display Name**: Replace build number in backend-webapi.yaml file
* **Tool**: bash
* **Arguments**: -c "sed -i 's/BuildNumber/$(Build.BuildId)/g' backend-webapi.yaml



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

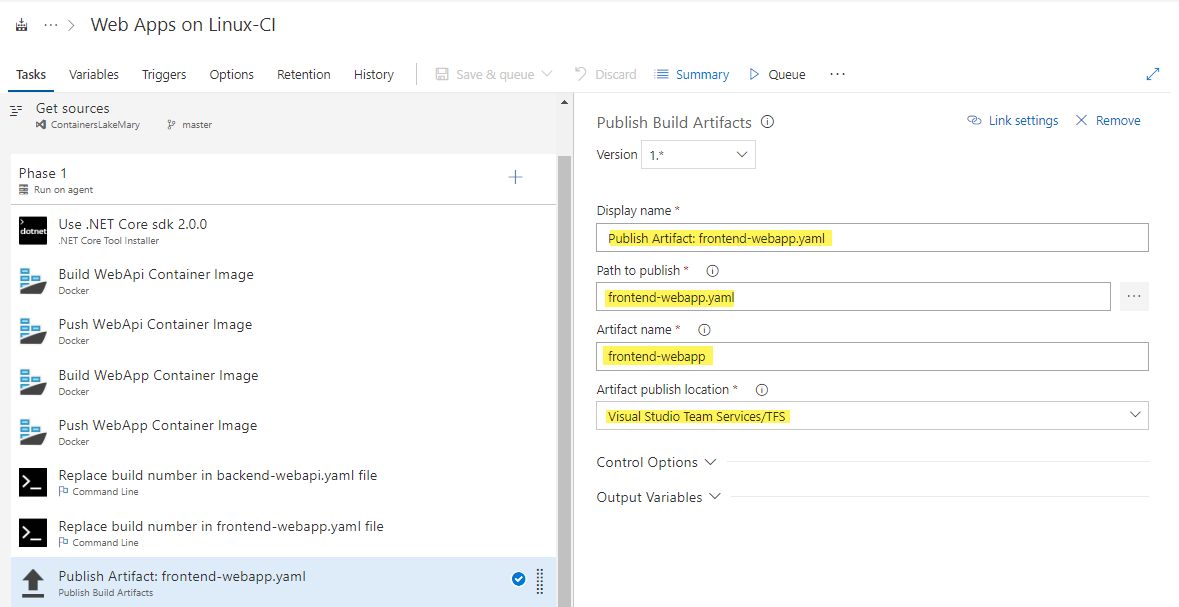
* **Display Name**: Replace build number in frontend-webapp.yaml file
* **Tool**: bash
* **Arguments**: -c "sed -i 's/BuildNumber/$(Build.BuildId)/g' frontend-webapp.yaml



*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 definition later.*

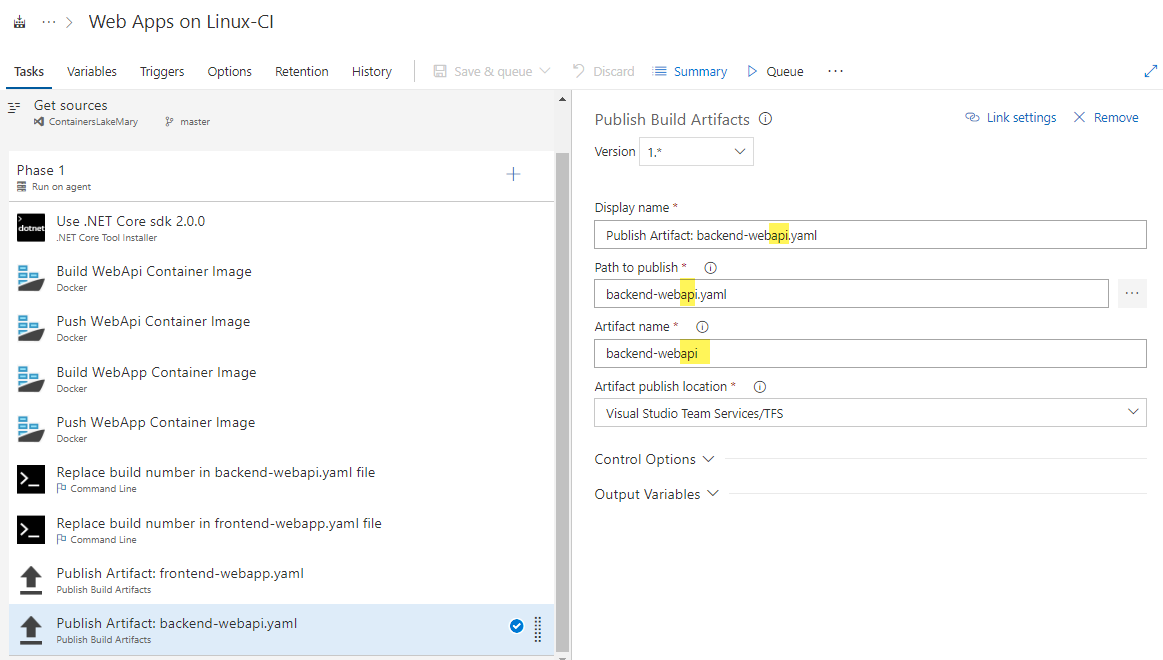
1. Add a new **Publish Build Artifacts** task and configure as follows:

* **Display Name**: Publish Artifact: frontend-webapp.yaml
* **Path to Publish**: frontend-webapp.yaml
* **Artifact Name**: frontend-webapp
* **Artifact publish location:** Visual Studio Team Services/TFS



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

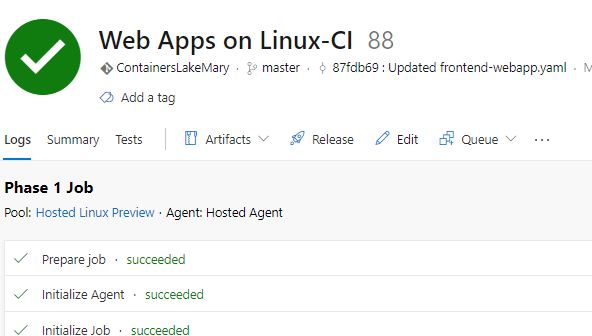
* **Display Name**: Publish Artifact: backend-webapi.yaml
* **Path to Publish**: backend-webapi.yaml
* **Artifact Name**: backend-webapi



Finally change the build definition’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).



1. 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):



Exercise 3: Create Release Definition for AKS Cluster

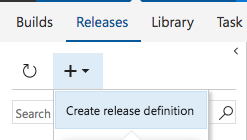
In this exercise, you will create a Release Definition which will pull Windows container images from ACR (result of a previous build task), and then deploy these containers to Kubernetes cluster on AKS, managed Kubernetes service.

Tasks

1. Create Release Definition for Windows Kubernetes Deployment

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

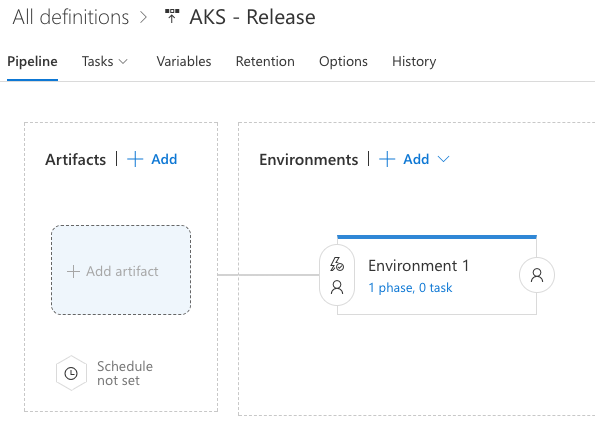
1. Open VSTS portal and then go to Release page. Now click on “+” icon and select “Create release button”.



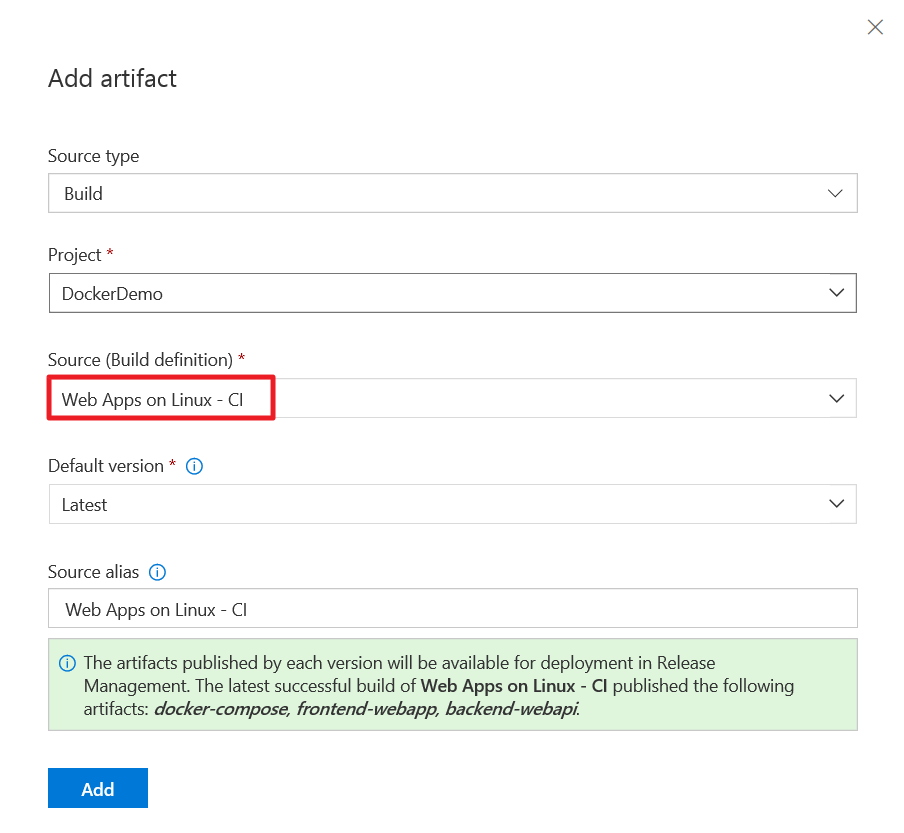
1. Although there are templates about Kubernetes deployment, you are going to start with empty template. Click Empty process to create an empty release definition.



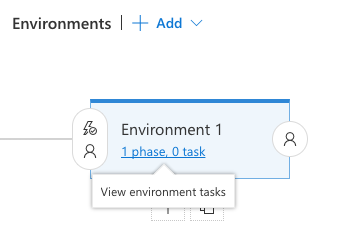
1. Click on release definition name and change it to AKS–Release. Then click on Add artifact to link this release definition with build artifacts.



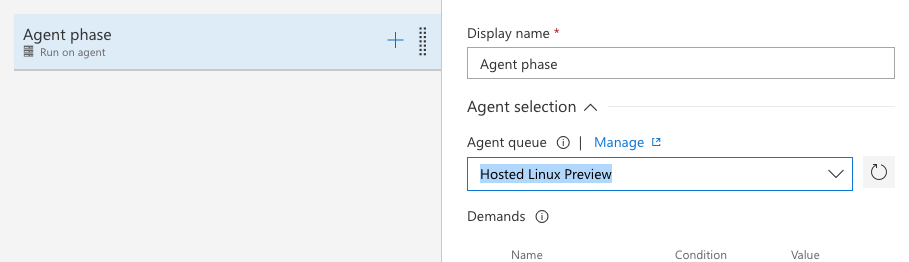
1. On the Add artifact popup, select Web Apps on Linux -CI build definition. Finally select Add.



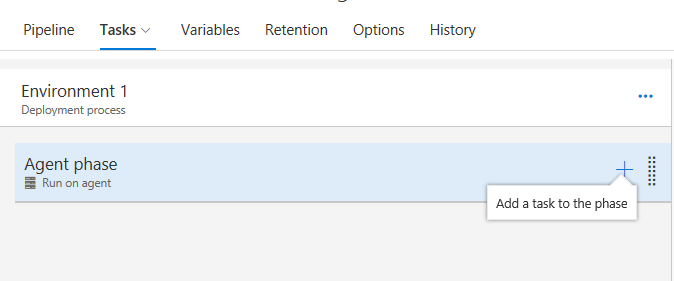
1. Click “1 phase, 0 task” link available inside the Environment 1.



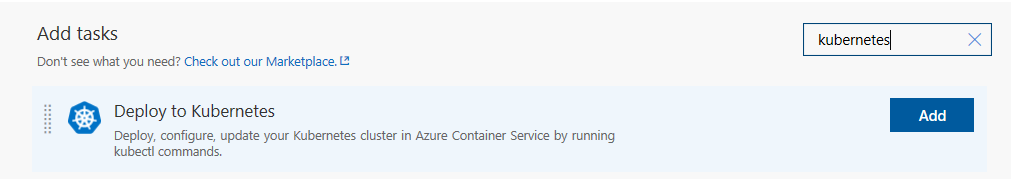
1. Click on the Agent phase and then select Hosted Linux Preview from the Agent queue dropdown.



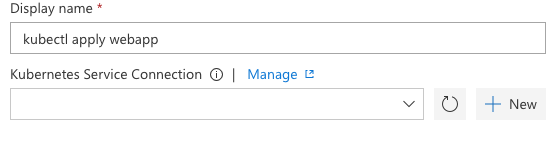
1. Now you are ready to add tasks to deploy your containers to Kubernetes cluster. Click plus button on Agent phase, and then find Deploy to Kubernetes task.



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



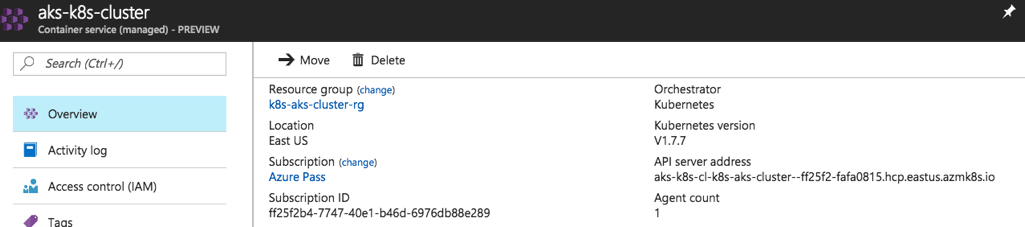
1. Change the first task name to kubectl apply webapp. Then to add a connection to your AKS Kubernetes cluster by clicking the ‘+ New’ button.



1. On the popup enter following parameters:

* Connection name: AKS-Connection
* Server URL: This is the FDQN you have captured in previous lab, make sure it has the https:// at the beginning. (E.g. https://aks-k8s-cl-k8s-aks-cluster--ff25f2-fafa0815.hcp.eastus.azmk8s.io).

If you are not sure of server URL, you can check API server address parameter of your AKS Kubernetes cluster on Azure Portal.

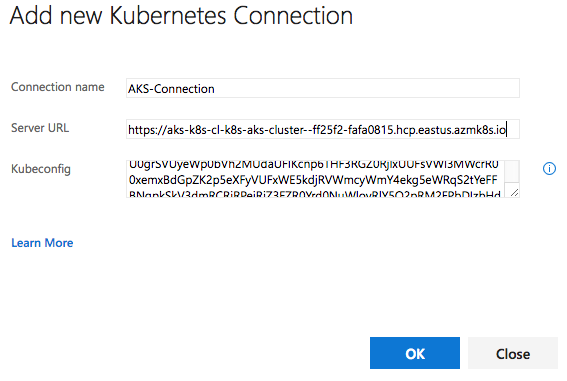


* Kubeconfig: Go back to your PowerShell command line. Run the following:

az aks get-credentials --resource-group=k8s-aks-cluster-rg-INITIALS --name=aks-k8s-cluster --file=aks-k8s-cluster

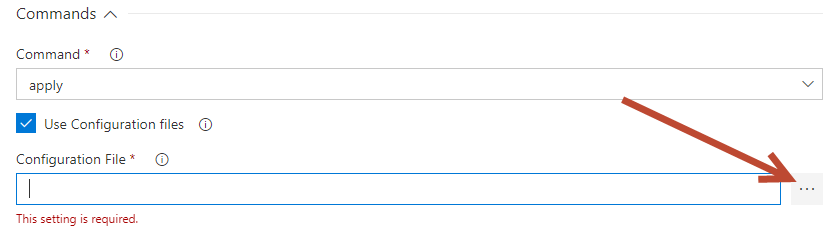
* Copy the content of Kubernetes config file called aks-k8s-cluster that you just created (it will be created wherever your command line path is currently) and paste it in the textbox.

Finally, select OK to save the changes.



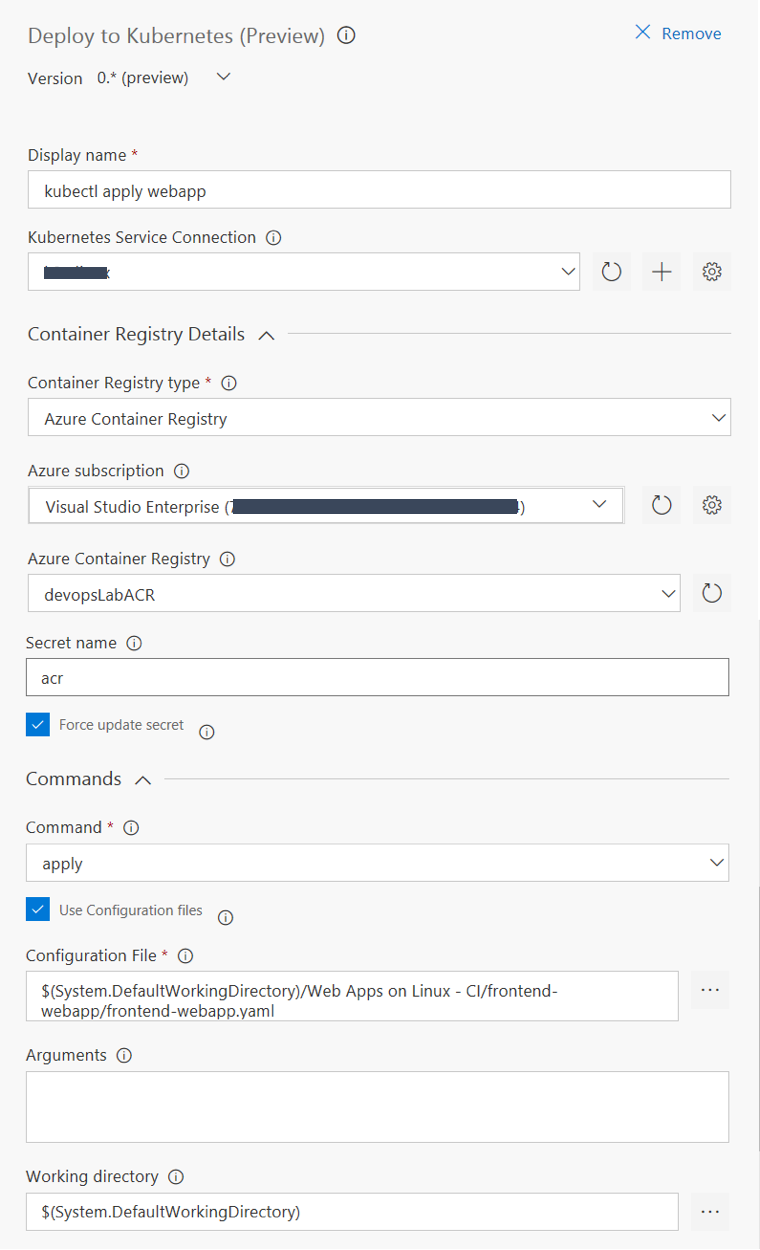
1. Now fill other parameters of the task:

* 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 definition, since container images will be pulled from this registry)
* Check the box for **Use Configuration files**, and then the below input box will open:
  + - * **Configuration File**:



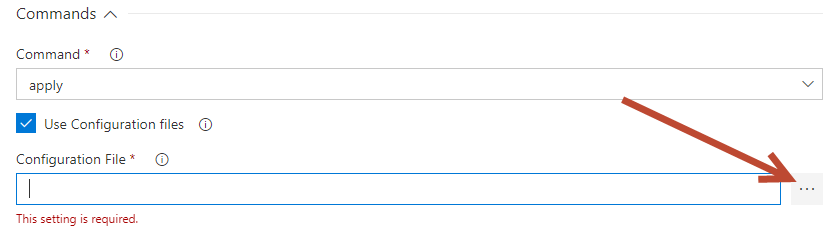
Click on the 3 dots to the right of the input box to explore files, select the main folder, select frontend-webapp folder, select **frontend-webapp.yaml**, then hit OK.

* 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)



1. Now configure second Kubernetes task:

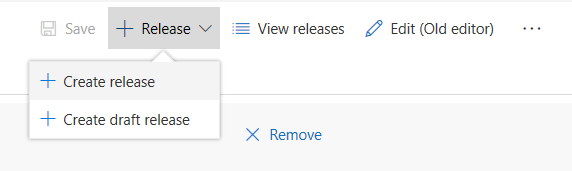
* Display Name: kubectl apply webapi
* Kubernetes Service Connection: AKS-Connection
* Container Registry type: Azure Container Registry
* Azure Container Registry: devopsLabACR (this registry should be the same as the one you have used in your build definition, since container images will be pulled from this registry)
* Check the box for **Use Configuration files**, and then the below input box will open:
  + - * **Configuration File:**



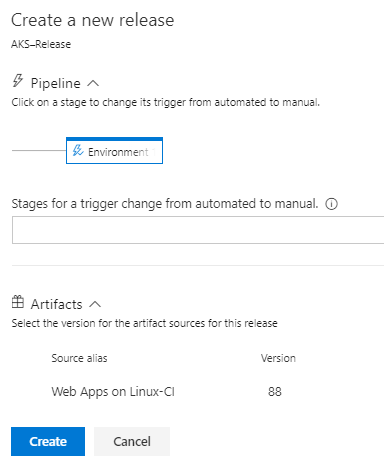
Click on the 3 dots to the right of the input box to explore files, select the main folder, select backend-webapi folder, select backend-webapi.yaml, then hit OK.

* 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)

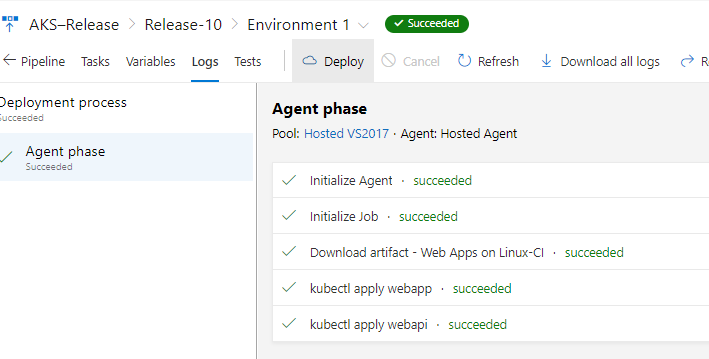
1. Look at the top right of your page and you will see the buttons as in the screenshot below. First, Save the release definition. Then click on the Release button and choose “+ Create Release” to trigger a release.



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



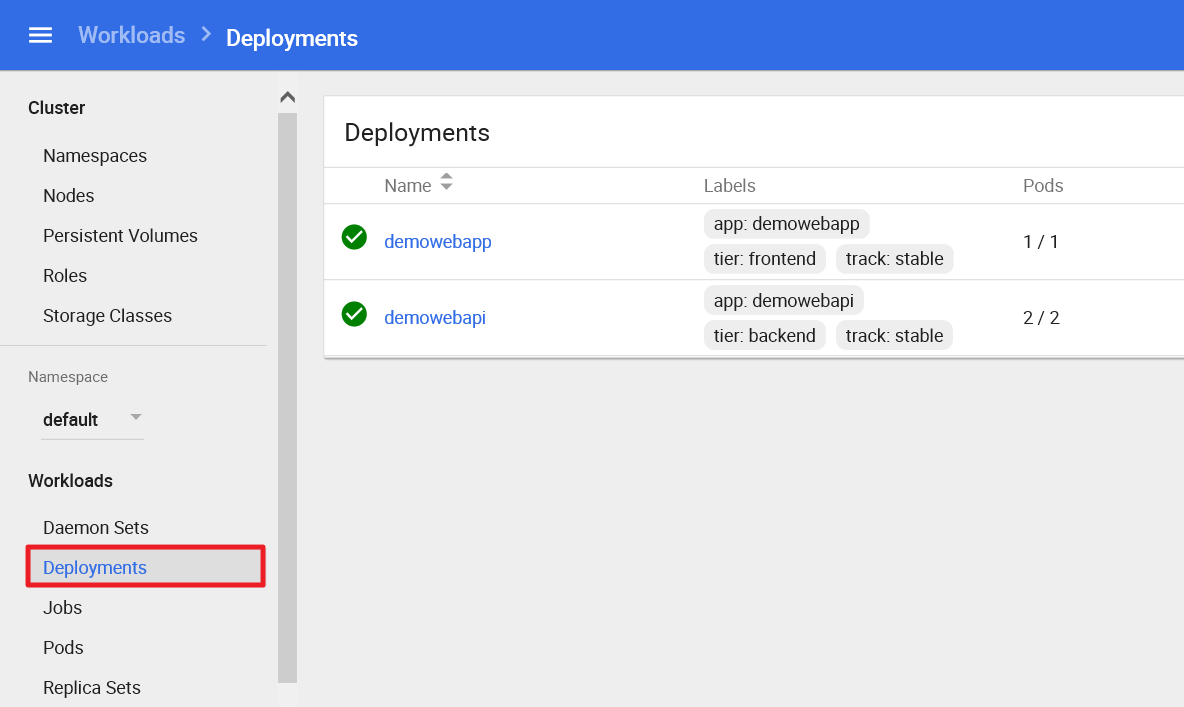
When it succeeds you will see a similar log as below:



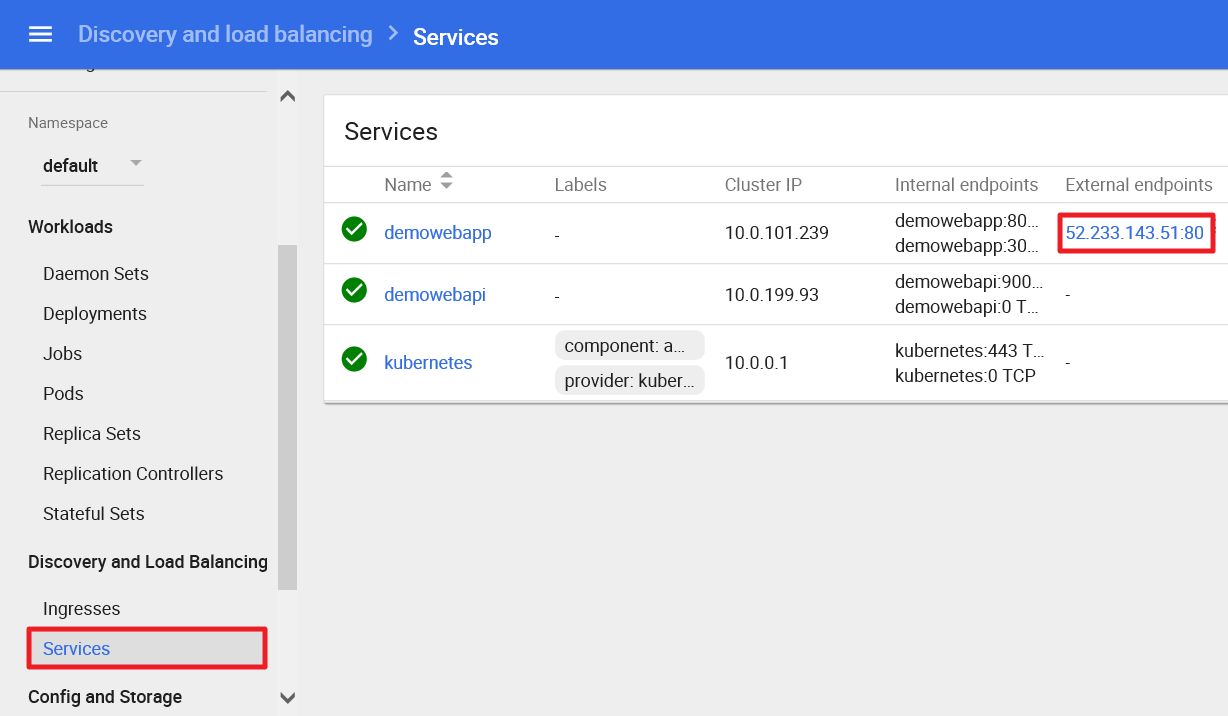
1. Now your deployment in VSTS is completed, but the deployment in Kubernetes cluster is just beginning. Go to Kubernetes cluster dashboard and check deployment status. After waiting for a while, you will see that the deployment is completed.

*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*



1. 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.



1. When the web site is loaded, notice the “v2” from the previous lab is gone. Then, click on Quotes link.
2. Notice that frontend web application is successfully fetching quotes by connecting to backend web API.
3. 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 so you will have enough cores for the next exercise.

You can check HomeController.cs in webapp project that it is connecting to the web API via <http://demowebapi:9000/api/quotes> which is configured in backend-webapi.yaml file.

