

**Module 6 Lab: Kubernetes**

Template Version: 2.0

**Estimated Time**

60 minutes

**Objectives**

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

* Create Kubernetes .yml / .yaml manifest files (\*Note yml and yaml are equivalent, you can use either equally)
* Deploy an application to Kubernetes

**Logon Information**

Please use the Azure Pass provided to you for this lab.

* Username: Your personal Outlook email address.
* Password: P@ssw0rd123!

Table of Contents

[Module 6 Lab: AKS 3](#_Toc523692256)

[Exercise 1: Create a Kubernetes Cluster 3](#_Toc523692257)

[Exercise 2: Deploy your Application to AKS Manually 5](#_Toc523692258)

[Exercise 3: Rolling Deploy + Rollback 8](#_Toc523692259)

Module 6 Lab: AKS

Exercise 1: Create a Kubernetes Cluster

Tasks

1. Login and Deploy Cluster using the Azure CLI
2. RDP into your VM.
3. In your VM, open a PowerShell window. First, we need to login to the Azure CLI. Run the following command and follow the instructions. Login with your personal account that you have been using for the Azure Pass account.

az login

1. Run the following to set your subscription to the Azure Pass account:

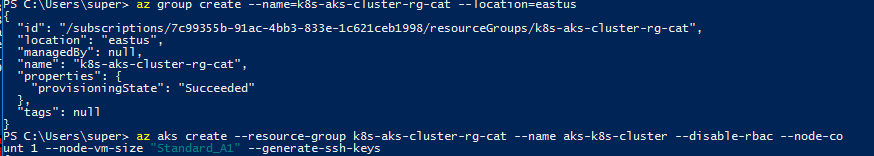
az account set --subscription “Azure Pass”

1. Run the following command to create a new empty resource group in Azure to hold your AKS cluster. As always, replace the highlighted **INITIALS** with your own initials to make the name unique.

az group create --name=k8s-aks-cluster-rg-INITIALS --location=eastus

1. Run the following command to create an AKS cluster called aks-k8s-cluster inside of the new resource group that you just created. This command will take **15 minutes** because the cluster must provision VMs in Azure for the master node and worker nodes, so please be patient with this step.

az aks create --resource-group k8s-aks-cluster-rg-INITIALS --name aks-k8s-cluster --disable-rbac --node-count 1 --node-vm-size "Standard\_A1" --generate-ssh-keys



1. **Set your local environment to your AKS cluster**
2. Once the above command to deploy AKS completes, run the following to get credentials for your AKS cluster onto your VM.

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

1. Run the following to set your context to you current AKS cluster, you can have multiple AKS clusters you work with, so it sets the name of the cluster you want to work with now.

kubectl config set-context aks-k8s-cluster

Exercise 1 has been completed

Exercise 2: Deploy your Application to AKS Manually

Tasks

1. **View the Kubernetes UI and setup a secret.**
2. Create a secret. A secret will allow you to use a private Azure Container Registry with Kubernetes. You will create the secret in conjunction with my ACR because mine has the Linux container in it.

kubectl create secret docker-registry acr --docker-server yogurtthecat.azurecr.io --docker-username yogurtthecat --docker-password UUv1JmUZQsRrAoZcrqnY+J=hyr7=k/wA --docker-email email@email.com

1. Run the following to open a Kubernetes UI. **It will open a window in IE, make sure to copy paste the URL to Chrome as the Kubernetes UI does not support IE or Edge.** This is available in general for Kubernetes, so you will get this view using either ACS or AKS.

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

1. **Setup the yaml files and deploy them to AKS manually**

**Important Note**: We are going to do something interesting here for lab purposes only.

Typically, when you do a “docker pull”, this will signal the image repo to check and see your OS type. If it sees a Windows type, it will see if it has a compatible image for you, if it does, then it lets you pull. Same for Linux. If you are Windows and ask for an image with only a Linux image available, you will get an error on the “docker pull” step.

On this particular Windows Server 2016 VM, you can only create Windows images. Sometimes you can create Linux images on a Windows machine using a Hyper-VM with a Linux OS and special Docker versions/features/add ons.

The base image we use for .NET Core has **both** a Windows and Linux version, so no matter which OS you run the “docker pull” from, you will get the corresponding image.

AKS is Linux only currently (9/2018), although in the future it is promised to work with Windows.

The lab is using a Windows Server 2016 which cannot easily make a Linux image. We need a Linux image of our app to run it on AKS. We cannot run a Windows image of our web app and API app on AKS. I have created a Linux image using a Linux OS for you to use in the lab to work on AKS for this and the next lab.

**Tl;dr: You are pushing a Windows docker image of the app to your ACR for practice and completeness to understand the process. But you will be pulling my equivalent but Linux docker image app for the sake of the AKS lab which only allows Linux images.**

1. In PowerShell, navigate to your yaml files.

cd C:/labs/day2

1. Run the following to run the backend web API project:

kubectl apply -f backend-webapi.yaml

1. Run the following to deploy the front-end web app:

kubectl apply -f frontend-webapp.yaml

1. As the pods are creating, take some time to read the yaml files in Notepad and understand what they do. If you want to use command line to see which pods are deployed to your AKS cluster, run the following:

kubectl get pods

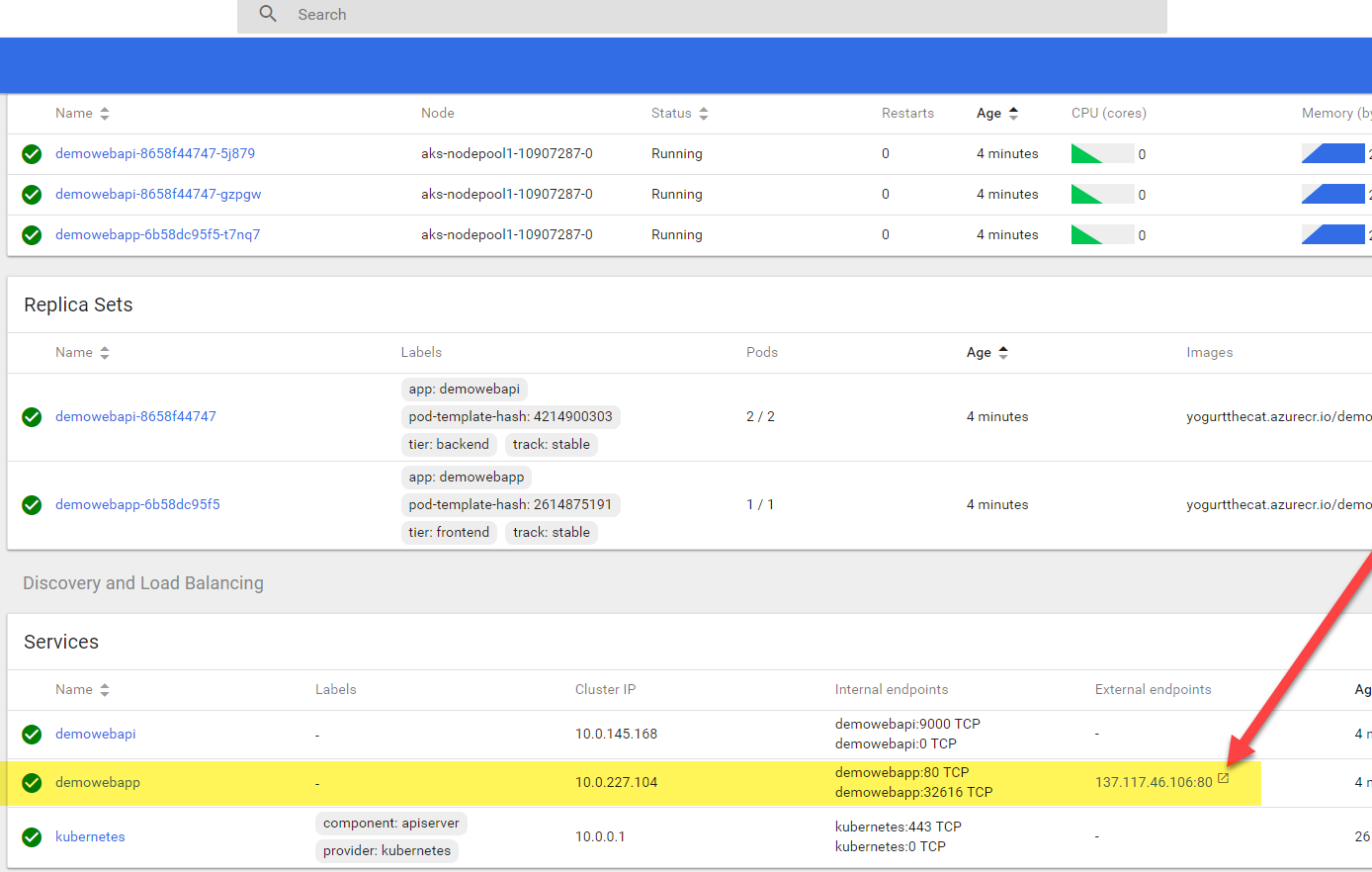
1. If you want to use command line to see which services are deployed to your AKS cluster, run the following:

kubectl get svc

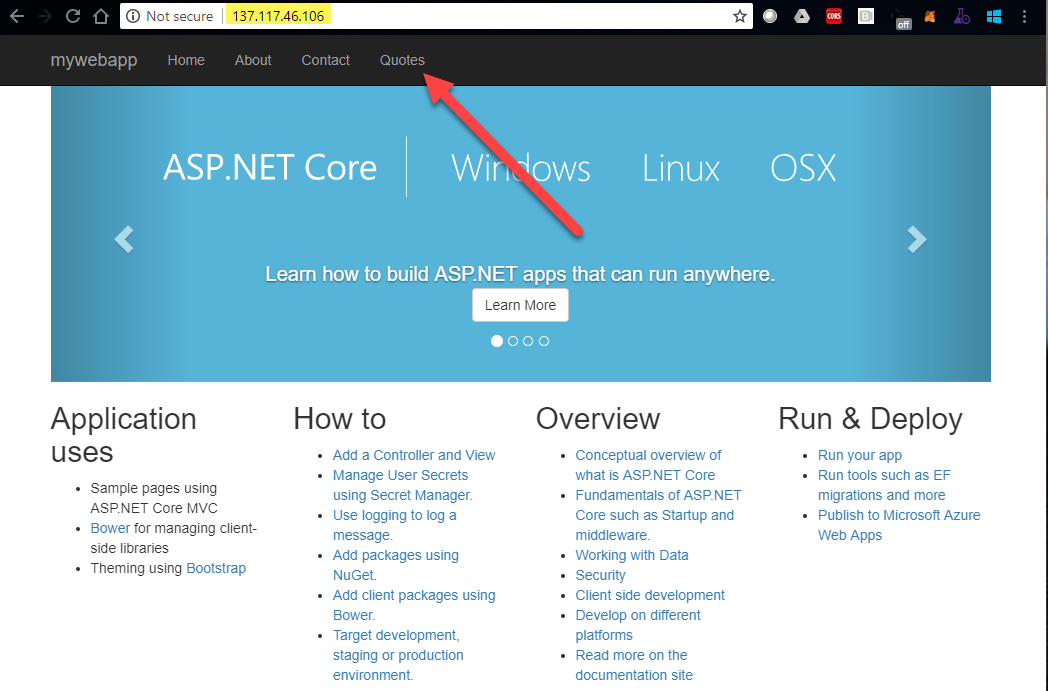
1. Run the following to open a Kubernetes UI. This is available in general for Kubernetes, so you will get this view using either ACS or AKS. If the browser opens using IE, copy and paste the URL into Chrome because the Kubernetes UI is not compatible with IE and Edge browsers:

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

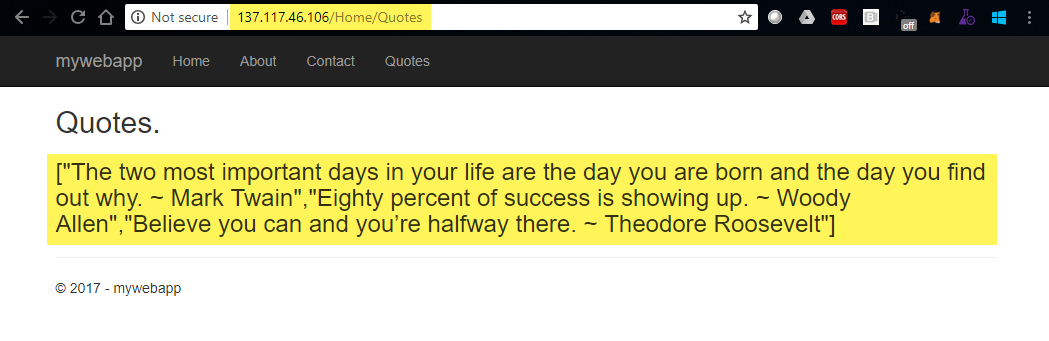
1. Find the Services section and the External endpoint for the demowebapp. If you do not see the IP address, then wait a few minutes (will take 2-3 mins) and refresh until it shows up. The command “kubectl get svc” is equivalent and you can run that to get the IP address also.



1. Go to that IP address in your browser. You should see a .NET Core intro screen like the following screenshot. Click on the Quotes link.



1. If you see a Mark Twain quote, then your web app is successfully hitting your API app in the AKS containers/pods!!! Great job!



Exercise 2 has been completed

Exercise 3: Rolling Deploy + Rollback

Users expect applications to be available all the time and developers are expected to deploy new versions of them several times a day. In Kubernetes this is done with rolling updates. Rolling updates allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones. The new Pods will be scheduled on Nodes with available resources.

Scaling up our application to run multiple instances is a requirement for performing updates without affecting application availability. By default, the maximum number of Pods that can be unavailable during the update and the maximum number of new Pods that can be created, is one. Both options can be configured to either numbers or percentages (of Pods). In Kubernetes, updates are versioned, and any Deployment update can be reverted to previous (stable) version.

Like application Scaling, if a Deployment is exposed publicly, the Service will load-balance the traffic only to available Pods during the update. An available Pod is an instance that is available to the users of the application.

Rolling updates allow the following actions:

* Promote an application from one environment to another (via container image updates)
* Rollback to previous versions
* Continuous Integration and Continuous Delivery of applications with zero downtime

*Reference:* [*https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/*](https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/)

Tasks

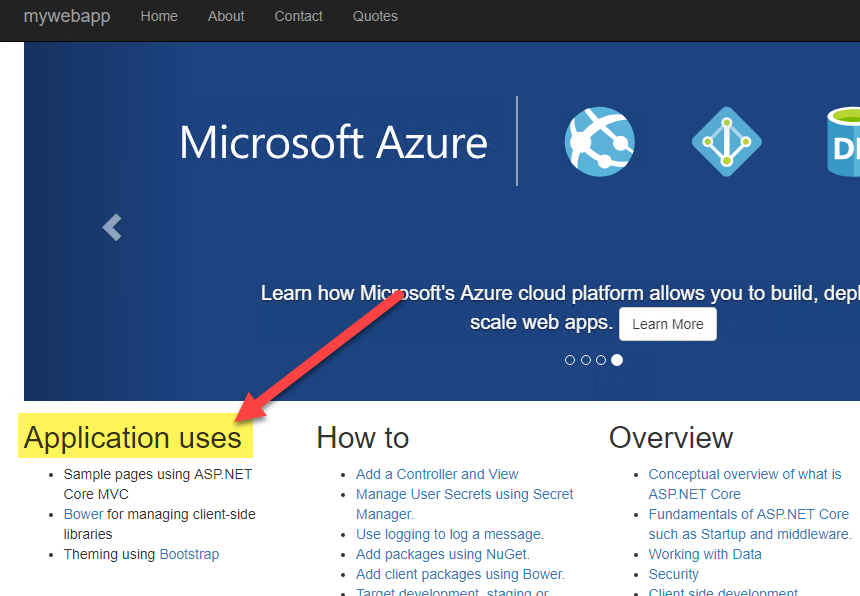
1. Rolling Update using Deployments
2. Let’s look at our new yaml file for the update, notice that the image version has been updated.

Important Note: For the purposes of the lab, I am using two different yaml files for the v1 and v2 version of the Deployment. This is not necessary; instead, you could just update the single deployment manifest (ideally committing it into source control) and apply it.

1. Navigate to your yaml files if you are not already there in PowerShell:

cd C:/labs/day2

1. Look at the page and notice the text here, this will change to “v2” after we rollout the new change:



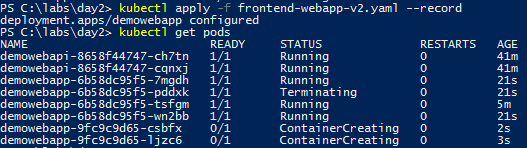
1. Now apply the new yaml file for the backend:

kubectl apply -f frontend-webapp-v2.yml --record

*Note: You can append --record to this command to record the current command in the annotations of the created or updated resource. This is useful for future review, such as investigating which commands were executed in each Deployment revision.*

1. We are only going to apply the front end and, in this case, will say the backend has not changed (Microservices! Different parts of the app can be updated separately on a different schedule). Check the pods, notice there are many in different states of Running and Terminating. Note you can view this either using the command below or by using the Kubernetes UI with the az aks browse command.

kubectl get pods



^ Notice some are running, some are terminating.. the rolling deployment is working! You must do the “get pods” very quickly to see this, it only takes a few seconds for the deployment to complete and setup 4 pods.

1. You can check the status of the deployment using the following command (deployment in these cases is “**demowebapp**”):

kubectl rollout status deployment <deployment>

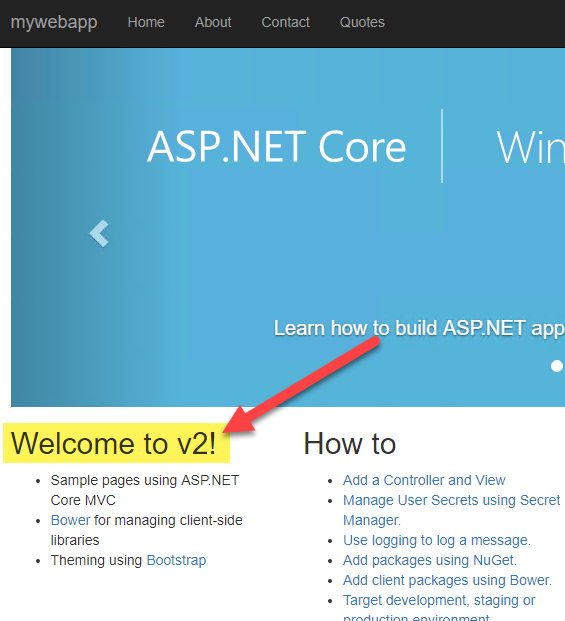
1. You can pause the status of the deployment using the following command:

kubectl rollout pause deployment <deployment>

1. You can resume the status of the deployment using the following command:

kubectl rollout resume deployment <deployment>

1. Refresh the browser tab with the container and see the “v2” change:



1. Rollback

After the image update, your colleague finds the service become unstable you may want to go back to the previous version. Unfortunately, he/she doesn’t remember how the previous config looks like. Well, you don’t need the time machine, just let rollback to do its job.

At previous part, the parameter --record comes with command let the Kubernetes record the command you typed, so that you can distinguish between the revisions.

1. To get the names of the deployments, either see the Kubernetes UI or run:

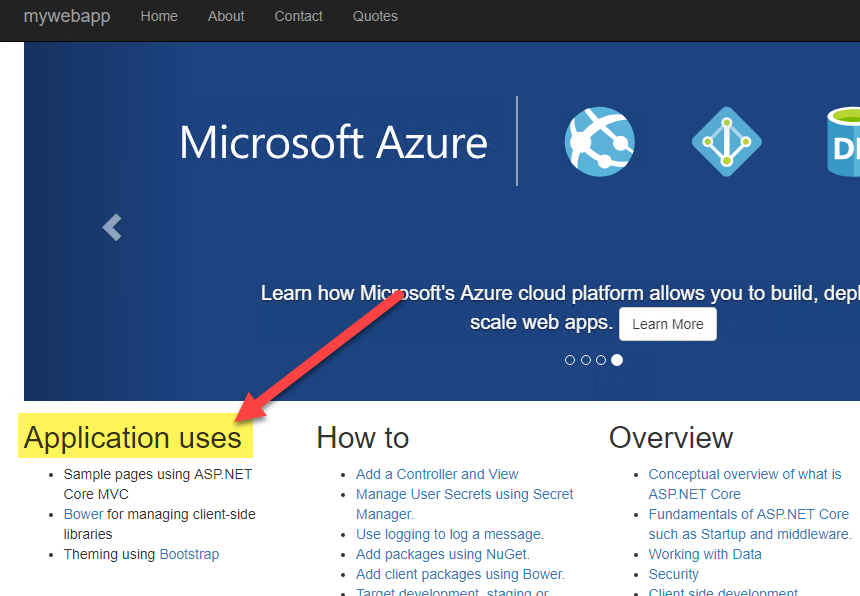
kubectl get deployment

1. Run the following command to undo the latest deployment and go back to what was there last:

kubectl rollout undo deployment demowebapp

Note, the format for the command is: kubectl rollout undo deployment <deployment>

1. Refresh the browser tab with the container app and notice it is back to normal:



Note for future reference, you can run the following to rollback to a specific revision:

**kubectl rollout undo deployment <deployment> --to-revision=<revision>**

Try running either of the commands to rollback and confirm that your pods are terminating the new versions and creating new pods with the old version of the app. Also make sure that once it is complete, that you see the old version of your website at the external endpoint IP address. Notice that there is no downtime for your application!!!

1. Please deploy the v2 back again so you can see the difference when you do the CI/CD deployment in the next lab.
2. Test yourself!
3. Try doing the rolling deploy again while refreshing the .NET Core container page and notice there is no downtime!
4. Then try rolling back again to the older version for practice.
5. Inspect the v2 yaml file and compare to the regular one. Notice the differences!
6. Please deploy the v2 back again so you can see the difference when you do the CI/CD deployment in the next lab.

Exercise 3 has been completed