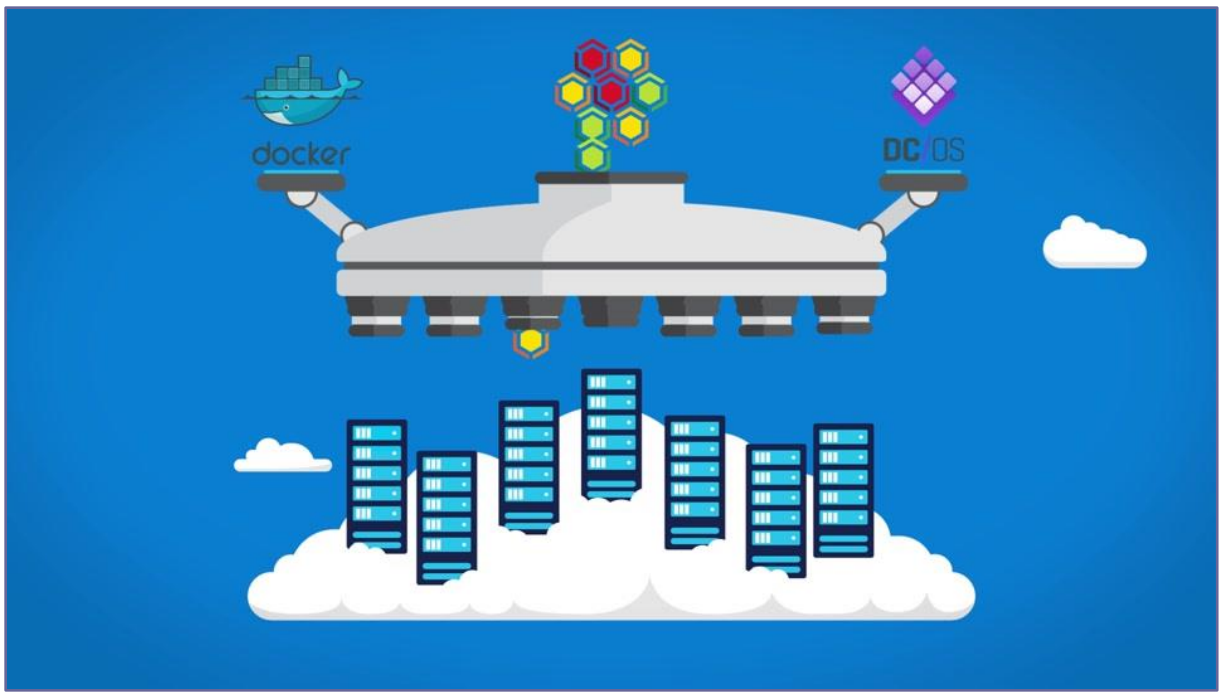


Azure Developer Series

Migrating a legacy ASP.NET 2-tier application to Azure using Container Services

Hands-On-Labs step-by-step guides

Prepared by:

Peter De Tender

CEO and Lead Technical Trainer
PDTIT and 007FFFLearning.com

@pdtit @007FFFLearning

Version: April 2019 – 2.0

Contents

Migrating a legacy ASP.NET 2-tiered application to Azure using Container Services - Hands-On-Labs step-by-step

Abstract and Learning Objectives

This workshop enables anyone to learn, understand and build a Proof of Concept, in performing a multi-tiered legacy ASP.NET web application using Microsoft SQL Server database, platform migration to Azure public cloud, leveraging on different Azure Platform Azure A Service (PaaS) and Azure Container Services.

After an introductory module on cloud app migration strategies and patterns, students get introduced to the basics of automating Azure resources deployments using Visual Studio and Azure Resource Manager (ARM) templates. Next, attendees will learn about Microsoft SQL database migration to SQL Azure PaaS, as well as deploying and migrating Azure Web Apps.

After these foundational platform components, the workshop will totally focus on the core concepts and advantages of using containers for running web apps, based on Docker, Azure Container Registry (ACR), Azure Container Instance (ACI), as well as how to enable container cloud-scale using Azure Container Services (ACS) with Kubernetes and Azure Kubernetes Service (AKS).

The focus of the workshop is having a Hands-On-Labs experience, by going through the following exercises and tasks:

- Deploying a 2-tier Azure Virtual Machine (Webserver and SQL database Server) using ARM-template automation with Visual Studio 2017;
- Migrating a legacy SQL 2012 database to Azure SQL PaaS (Lift & Shift);
- Migrating a legacy ASP.NET web application to Azure Web Apps (Lift & Shift);
- Containerizing a legacy ASP.NET web application using Docker;
- Running Azure Container Instance (ACI) from an Azure Container Registry (ACR) image;
- Deploy and run Azure Container Services (ACS) with Kubernetes;
- Deploy and run Azure Kubernetes Services (AKS);
- Managing and Monitoring Azure Container Services (ACS) and Azure Kubernetes Services (AKS);

Hands-On-Lab Scenario

The baseline of the hands-on-lab scenario is starting from an 8-year-old legacy ASP.NET application, developed around 2010, currently offering a product catalog browsing web page, pulling the product catalog list from a legacy Microsoft SQL Server 2012 database, running on dedicated Windows Server 2012 R2 Virtual Machines. (This could be seen as a subset of a larger application landscape within your organization, think of manufacturing database information, HR solutions, e-commerce platforms,… and alike). Imagine this application is running in our on-premises datacenter, where you want to perform an "application digital migration" to Azure Public cloud. You will use several Azure cloud-native services and solutions, ranging from Virtual Machines, Platform Services and different Container Solutions on Azure.

Requirements

Naming Conventions:

IMPORTANT: Most Azure resources require unique names. Throughout these steps you will see the word **"[SUFFIX]"** as part of resource names. You should replace this with your initials, guaranteeing those resources get uniquely named.

Azure Subscription:

Participants need a "pay-as-you-go", MSDN or other paid Azure subscription

a) Azure Trial subscriptions won't work
b) In one of the Azure Container Services tasks, you are required to create an Azure AD Service Principal, wich typically requires an Azure subscription owner to log in to create this object. If you don't have the owner right in your Azure subscription, you could ask another person to execute this step for you.
c) The Azure subscription must allow you to run enough cores, used by the baseline Virtual Machines, but also later on in the tasks when deploying the Azure Container Services, where ACS agent and master machines are getting set up. If you follow the instructions as written out in the lab guide, you need 12 cores.
d) If you run this lab setup in your personal or corporate Azure payable subscription, using the configuration as described in the lab guide, the estimated Azure consumption costs for running the setups during the 2 days of the workshop is $20.

Other requirements:

Participants need a local client machine, running a recent Operating System, allowing them to:

- browse to https://portal.azure.com from a most-recent browser;
- establish a secured Remote Desktop (RDP) session to a lab-jumpVM running Windows Server 2016;

Alternative Approach:

Where the lab scenario assumes all exercises will be performed from within the lab-jumpVM, (since several tools will be installed on the lab-jumpVM or are already installed by default), participants could also execute (most, if not all…) steps from their local client machine.

The following tools are being used throughout the lab exercises:

- Visual Studio 2017 community edition (updated to latest version)
- Docker for Windows (updated to latest version)
- Azure CLI 2.0 (updated to latest version)
- Kubernetes CLI (updated to latest version)

Make sure you have these tools installed prior to the workshop, if you are not using the lab-jumpVM. You should also have full administrator rights on your machine to execute certain steps within using these tools.

Final Remarks:

VERY IMPORTANT: You should be typing all of the commands as they appear in the guide, except where explicitly stated in this document. Do not try to copy and paste from Word to your command windows or other documents where you are instructed to enter information shown in this document. There can be issues with Copy and Paste from Word or PDF that result in errors, execution of instructions, or creation of file content.

IMPORTANT: Most Azure resources require unique names. Throughout these steps you will see the word "[SUFFIX]" as part of resource names. You should replace this with your initials, guaranteeing those resources get uniquely named.

Lab 7: Deploy and run Azure Kubernetes Services (AKS);

What you will learn

In this lab, you will learn what it takes to deploy an Azure Kubernetes Service (AKS), and noticing the differences compared to Azure Container Services (ACS) for Kubernetes.

Time Estimate

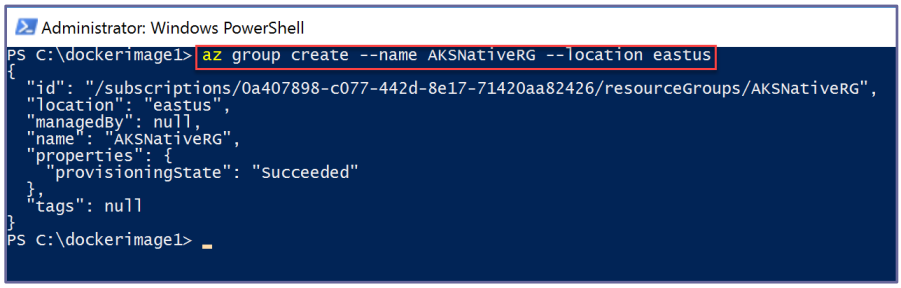
This lab should take about 30 minutes to complete.

Note, since Azure Kubernetes Services doesn't support Windows-based containers yet (only available in preview for now), we cannot demonstrate the cloudshop application running inside it, as that one is based on a Windows container. But it shouldn't block you from running this lab.

Task 1: Deploying Azure Kubernetes Service using Azure CLI 2.0

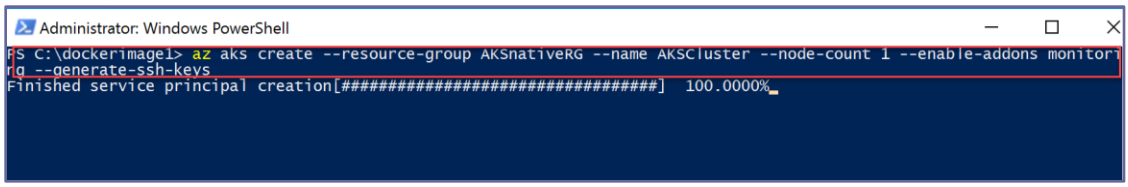
1. From the lab-jumpVM, Open PowerShell and run the following command to create a new Azure Resource Group:

   az group create --name AKSNativeRG --location eastus



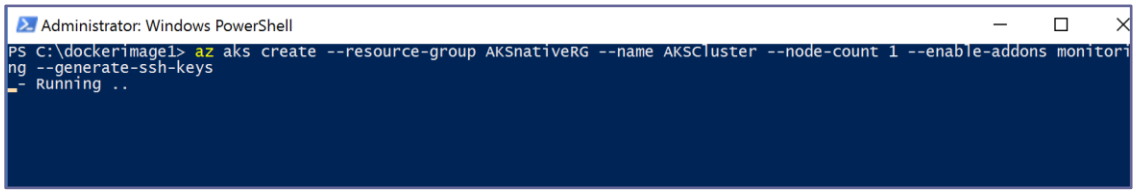
2. Next, run the following command to deploy the actual Azure Kubernetes Services resource:

   az aks create --resource-group AKSnativeRG --name AKSCluster --node-count 1 --enable-addons monitoring --generate-ssh-keys

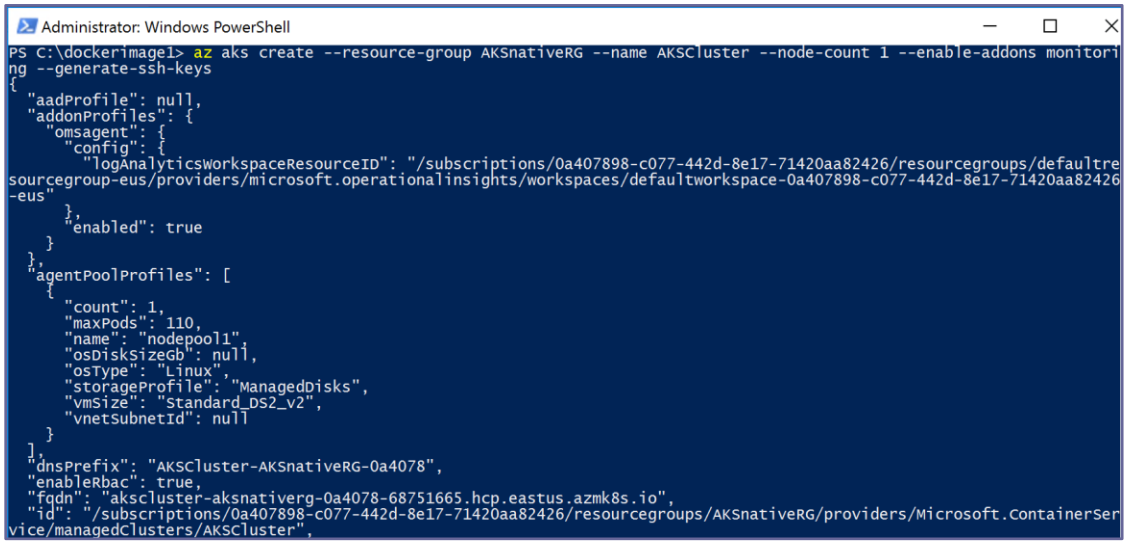


   where it first starts with creating the service principal, and moving on with the actual AKS

deployment:

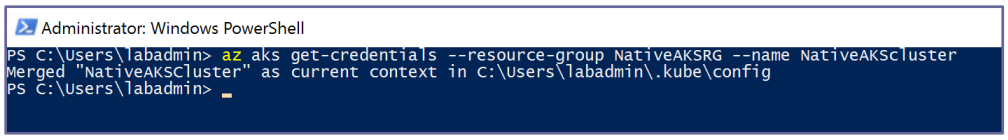


3. After about **10 minutes**, the AKS resource has been created, **as you can notice** from the PowerShell Azure CLI window, JSON output once the deployment is completed successfully:



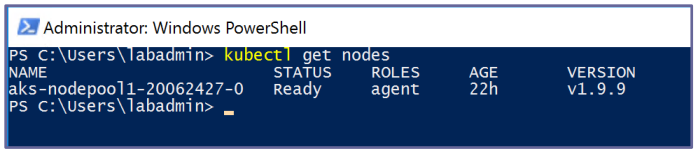
4. Now we have the Kubernetes Cluster up and running, let us start with **connecting to the Kubernetes environment and validating** it is running ok, by **performing the following steps:**

```
az aks get-credentials --resource-group [SUFFIX]AKSRG --name
[SUFFIX]AKSCluster
```



5. **Next, validate the functioning by checking the nodes:**

kubectl get nodes



Similar to how we integrated the docker application image from Azure Container Registry (ACR) into Azure Container Services (ACS), we can have Azure Kubernetes Services connect to different container registries (Docker Public and Private, Azure Container Registry, AWS and Google).

Task 2: Running a Docker public image in an Azure Kubernetes Service

1. Since we cannot run Windows container-images in Azure Kubernetes Service for now, we cannot use our webshop image. However, we could pull an image from a public Docker Hub instead. This information is defined in a new kubernetes2.yml file we will create.

2. On the lab-jumpVM, open Visual Studio Code. Copy in the following lines of code: (although the layout is copied and might look like a screenshot, you can actually copy these lines)

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: akshelloworld
spec:
  replicas: 1
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  minReadySeconds: 5
  template:
    metadata:
      labels:
        app: akshelloworld
    spec:
      containers:
      - name: adsacr
        image: docker.io/microsoft/aci-helloworld
        ports:
        - containerPort: 80
      imagePullSecrets:
      - name: adsacr-auth
```

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: akshelloworld
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: akshelloworld
```

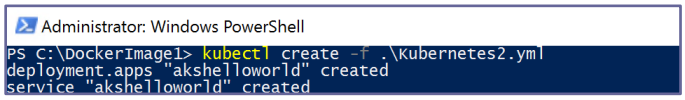
Note the parameter `image: docker.io/microsoft/aci-helloworld` is the pointer to the public image in Docker Hub.

1. Replace the "akshelloworld" names and variables with [SUFFIX]helloworld.

2. **Save** the files as Kubernetes2.yml in the C:\Dockerimage1 folder (the one we already used in previous labs).

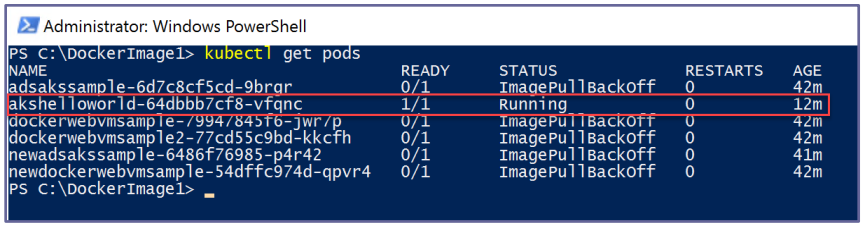
3. Next, **run the deployment of this Kubernetes service, by using the following command:**

   `kubectl create -f "path to kubernetes2.yml file here"`



4. **Validate** if the image is being pushed into the Kubernetes Service, by **checking the pods** again:

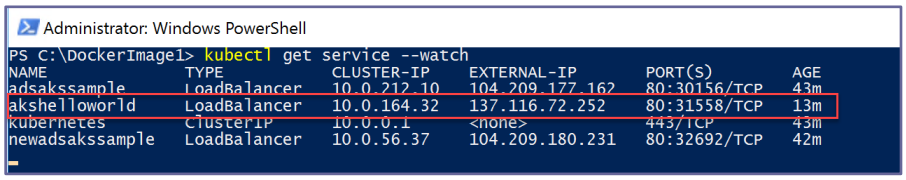
   `kubectl get pods`



5.

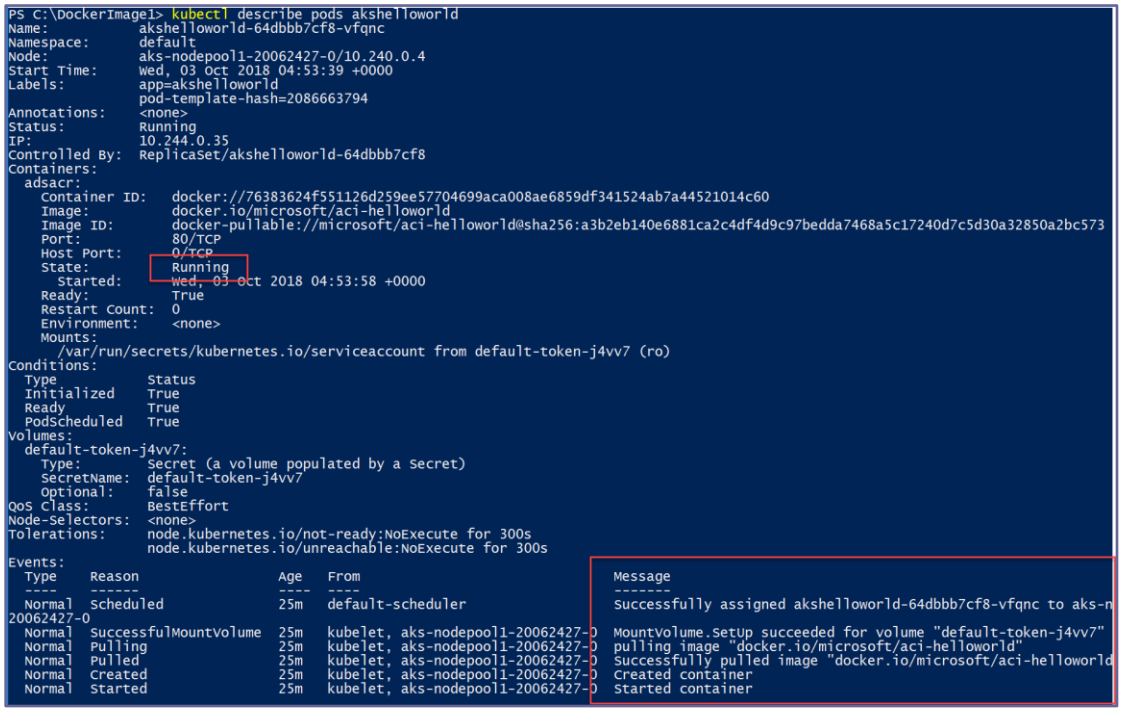
6. Or checking the actual container service, by running the following command:

```
kubectl get service dockerwebvmsample --watch
```

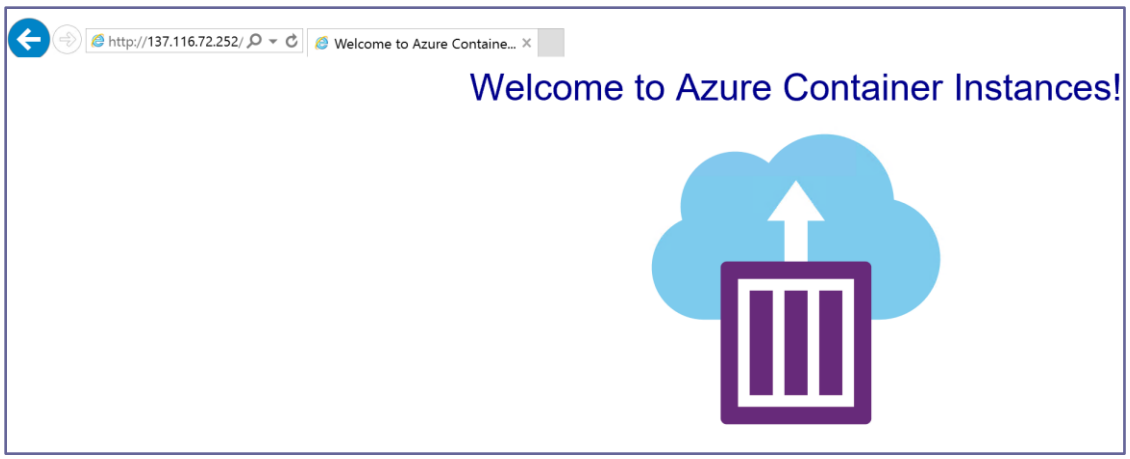


7. Another useful command you can use is "kubectl describe", which gives you detailed information regarding a running (or failing) pod.

   kubect describe akshelloworld



8. Give it another 2-3 minutes, then open your internet browser, and connect to the EXTERNAL-IP of the akshelloworld service:



9. If you see the Azure Container Instances welcome page, it means the container is running successfully (the docker.io/microsoft/aci-helloworld is a sample Docker container image with a Node.JS app, that just does this, showing a welcome page).

10. This completes the lab.

Task 3: Running a Docker Drupal web app public image in AKS

1. We can imagine just having a static image up in a container doesn't get you convinced about the power of running containers, in Kubernetes on Azure. So let's try to make it a bit more dynamic, to proof how cool this actually is.

2. On the lab-jumpVM, open Visual Studio Code. Copy in the following lines of code: (although the layout is copied and might look like a screenshot, you can actually copy these lines)

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: drupalcntr
spec:
  replicas: 1
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  minReadySeconds: 5
  template:
    metadata:
      labels:
```

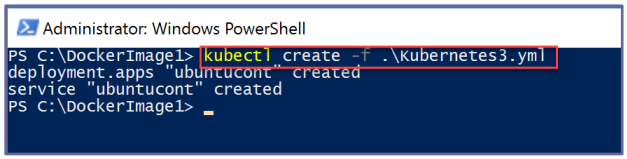
```
        app: drupalcntr
    spec:
      containers:
      - name: adsacr
        image: docker.io/drupal
        ports:
        - containerPort: 80
      imagePullSecrets:
      - name: adsacr-auth


---
apiVersion: v1
kind: Service
metadata:
  name: drupalcntr
  spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: drupalcntr
```

Note the parameter `image: docker.io/drupal` is the pointer to the public image in Docker Hub. Drupal is based on the open-source Linux Alpine platform, and offering a full workable web server and web site environment. Directly running from within a container.

3. Save the files as Kubernetes3.yml in the C:\Dockerimage1 folder (the one we already used in previous labs).

4. Next, run the deployment of this Kubernetes service, by using the following command:

   `kubectl create -f "path to kubernetes3.yml file here"`



5. Validate if the image is being pushed into the Kubernetes Service, by checking the pods again:

```
kubectl get pods

kubectl get services --watch
```

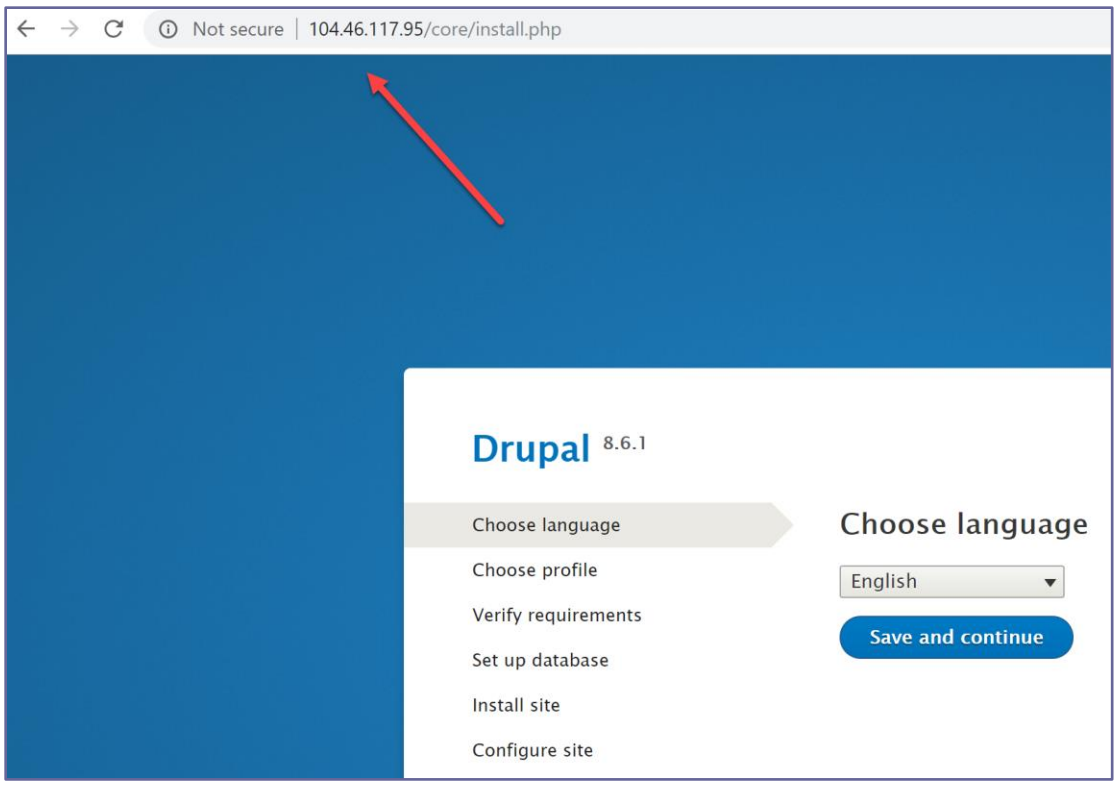
```
PS C:\DockerImage1> kubectl get pods
NAME                                READY     STATUS             RESTARTS   AGE
adsakssample-6d7c8cf5cd-9brgr       0/1       ImagePullBackoff   0          21h
akshelloworld-64dbbb7cf8-vfqnc      1/1       Running            1          21h
dockerwebvmsample-79947845f6-jwr7p  0/1       ErrImagePull       0          21h
dockerwebvmsample2-77cd55c9bd-kkcfh 0/1       ImagePullBackoff   0          21h
drupalcntr-5fff4774bf-zm8lk         1/1       Running            0          1m
newadsakssample-6486f76985-p4r42    0/1       ImagePullBackoff   0          21h
newdockerwebvmsample-54dffc974d-qpvr4   0/1   ImagePullBackoff   0          21h
ubuntucont-6f555d84d8-xs7vl         0/1       Completed          6          5m
PS C:\DockerImage1> kubectl get services --watch
NAME              TYPE           CLUSTER-IP     EXTERNAL-IP       PORT(S)        AGE
adsakssample      LoadBalancer   10.0.212.10    104.209.177.162   80:30156/TCP   21h
akshelloworld     LoadBalancer   10.0.164.32    137.116.72.252    80:31558/TCP   21h
drupalcntr        LoadBalancer   10.0.74.211    104.46.117.95     80:30750/TCP   1m
kubernetes        ClusterIP      10.0.0.1       <none>            443/TCP        21h
newadsakssample   LoadBalancer   10.0.56.37     104.209.180.231   80:32692/TCP   21h
ubuntucont        LoadBalancer   10.0.254.169   104.210.11.189    80:31412/TCP   6m
```

6. Or inspecting the full container deployment process again, by running

   kubectl describe pods drupalcntr

```
PS C:\DockerImage1> kubectl describe pods drupal
Name:           drupalcntr-5fff4774bf-zm8lk
Namespace:      default
Node:           aks-nodepool1-20062427-0/10.240.0.4
Start Time:     Thu, 04 Oct 2018 01:58:39 +0000
Labels:         app=drupalcntr
                pod-template-hash=1999033069
Annotations:    <none>
Status:         Pending
IP:
Controlled By:  ReplicaSet/drupalcntr-5fff4774bf
Containers:
  adsacr:
    Container ID:
    Image:          docker.io/drupal
    Image ID:
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Waiting
      Reason:       ContainerCreating
    Ready:          False
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-j4vv7 (ro)
Conditions:
  Type           Status
  Initialized    True
  Ready          False
  PodScheduled   True
Volumes:
  default-token-j4vv7:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-j4vv7
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason                 Age   From                              Message
  ----    ------                 ----  ----                              -------
  Normal  Scheduled              27s   default-scheduler                 Successfully assigned drupalcntr-5fff4774bf-zm
8lk to aks-nodepool1-20062427-0
  Normal  SuccessfulMountVolume  26s   kubelet, aks-nodepool1-20062427-0  MountVolume.SetUp succeeded for volume "defaul
t-token-j4vv7"
  Normal  Pulling                26s   kubelet, aks-nodepool1-20062427-0  pulling image "docker.io/drupal"

PS C:\DockerImage1> _
```

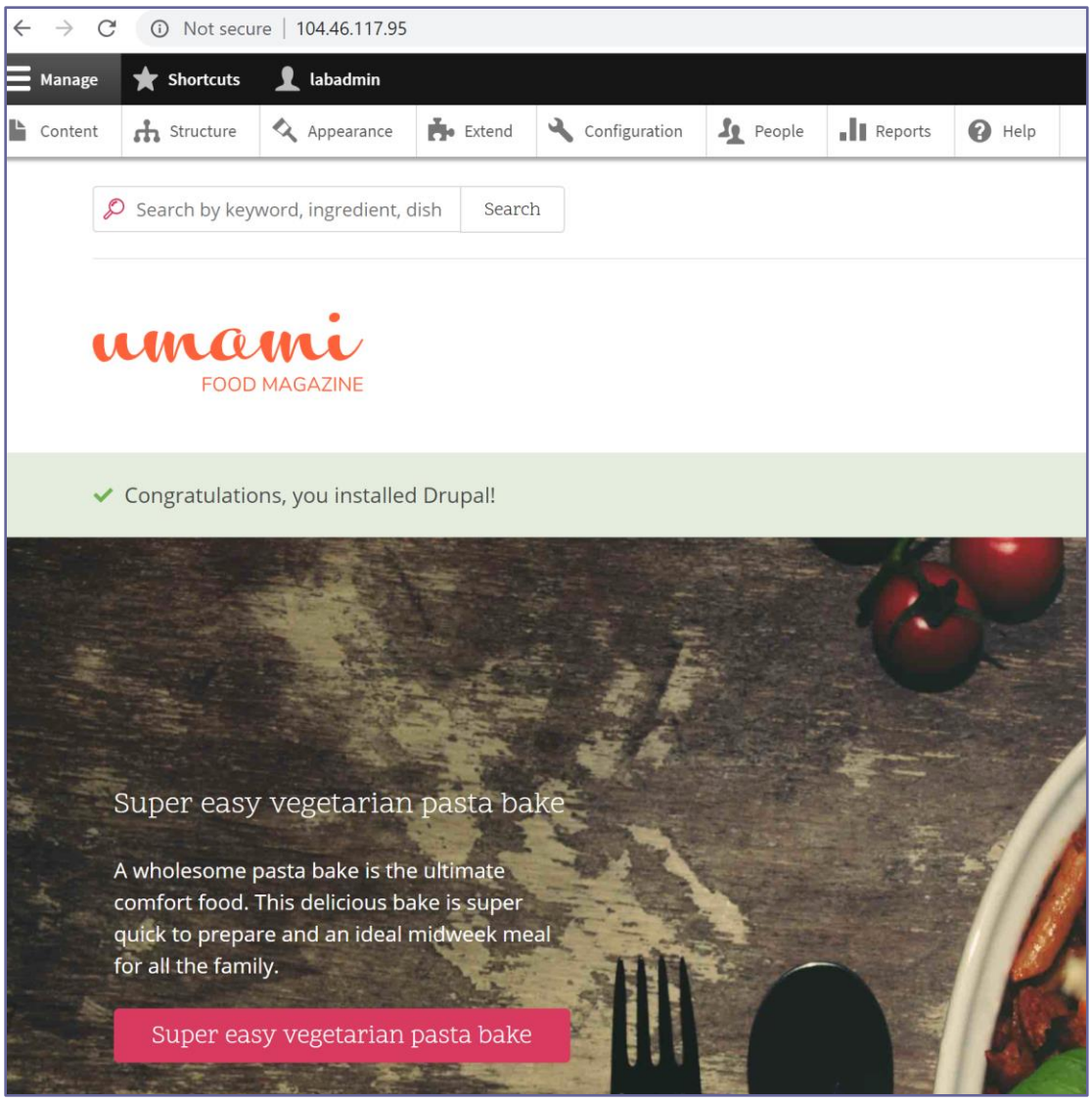
7. Give it another 2-3 minutes, then open your internet browser, and connect to the EXTERNAL-IP of the drupalcntr service:



8. If you see the Drupal Welcome page, it means the container is running successfully. We promised some more dynamics than a static image, so let's quickly walk over the Drupal configuration steps, resulting in having a sample Drupal web site up-and-running in just a few seconds:

- Choose Language:	accept English and Press Save and Continue
- Choose Profile:	Select Demo – Umami Food Magazine | Save & Cont
- Set up database:	choose SQLite | Save and Continue
- Install site:	wait for the process to complete
- Configure site:	provide your email, labadmin L@BadminPa55w.rd Credentials

9. Once all the steps are completed, give it a few seconds more, which will open up the sample web site. How cool is that!!



10. Note again, you are running a stripped-down Linux Operating System, which has the Drupal web engine running, together with a SQLite database, all within that same Docker container. Inside Azure Kubernetes Services. Pretty impressive if you ask me...!!

11. This completes this part of the lab task.

Summary

In this lab, you learned how to deploy Azure Kubernetes Services (AKS) using Azure CLI, as well as how to expand the running AKS cluster with more nodes. Next, you created a kubernetes.yml deployment file, having a pointer to a Docker Hub public repository image to use. After deploying this container image within the AKS cluster, you validated the functioning using the EXTERNAL-IP of the AKS Service as well as checked the pods. In the last task, we made it even more impressive,

creating a new Kubernetes3.yml file, pulling a Drupal image into AKS, and running a dynamic website, including a SQLite database.

For any further information or references, please have a look at the following Microsoft Docs around containers:

https://azure.microsoft.com/en-us/overview/containers/

https://azure.microsoft.com/en-us/services/kubernetes-service/

https://azure.microsoft.com/en-us/services/app-service/containers/

https://azure.microsoft.com/en-us/services/container-registry/

https://azure.microsoft.com/en-us/services/container-instances/

https://docs.microsoft.com/en-us/azure/aks/

Migrating a legacy ASP.NET 2-tier application to Azure using Container Services

Hands-On-Labs step-by-step guides

Prepared by:

Peter De Tender

CEO and Lead Technical Trainer
PDTIT and 007FFFLearning.com

@pdtit @007FFFLearning

Version: April 2019 – 2.0