# Lab 4: Deploy and run Azure Kubernetes Services (AKS);

## What you will learn

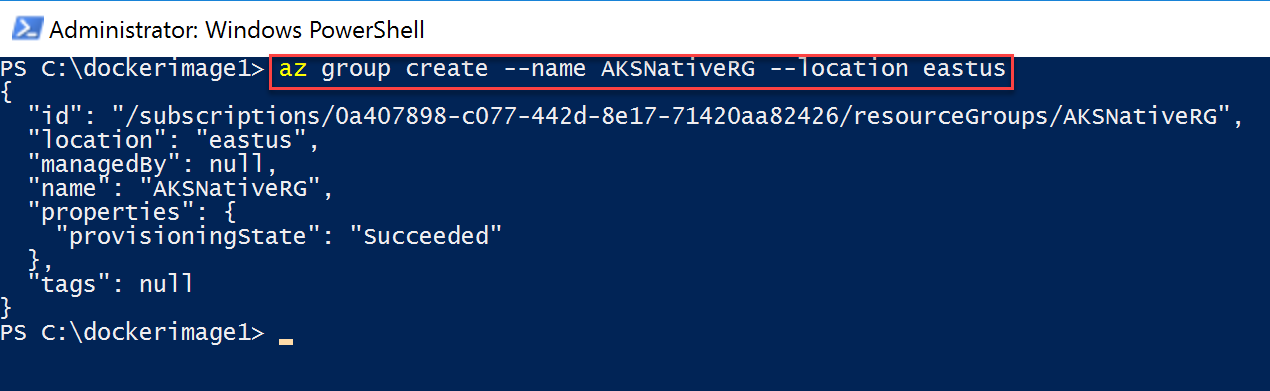
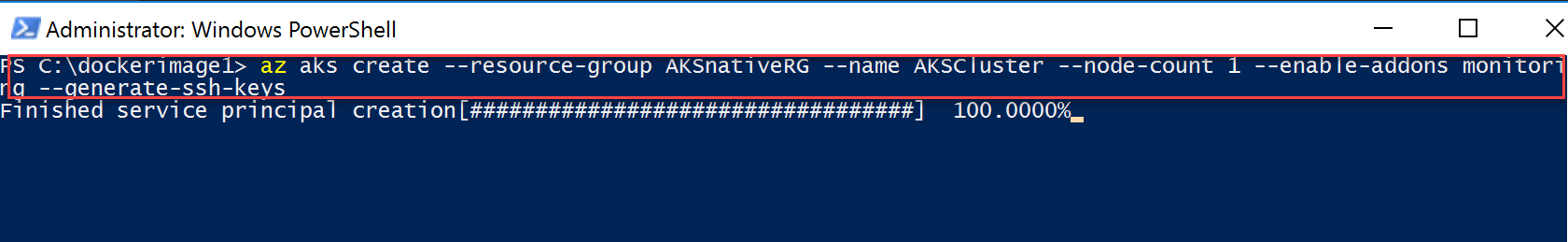
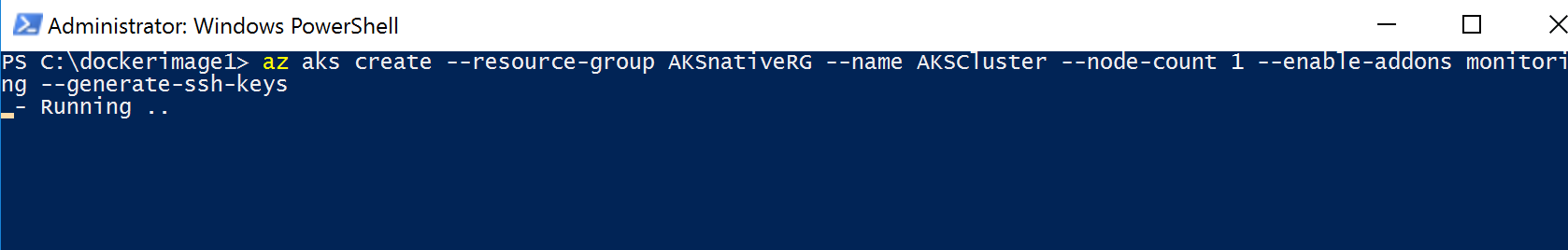
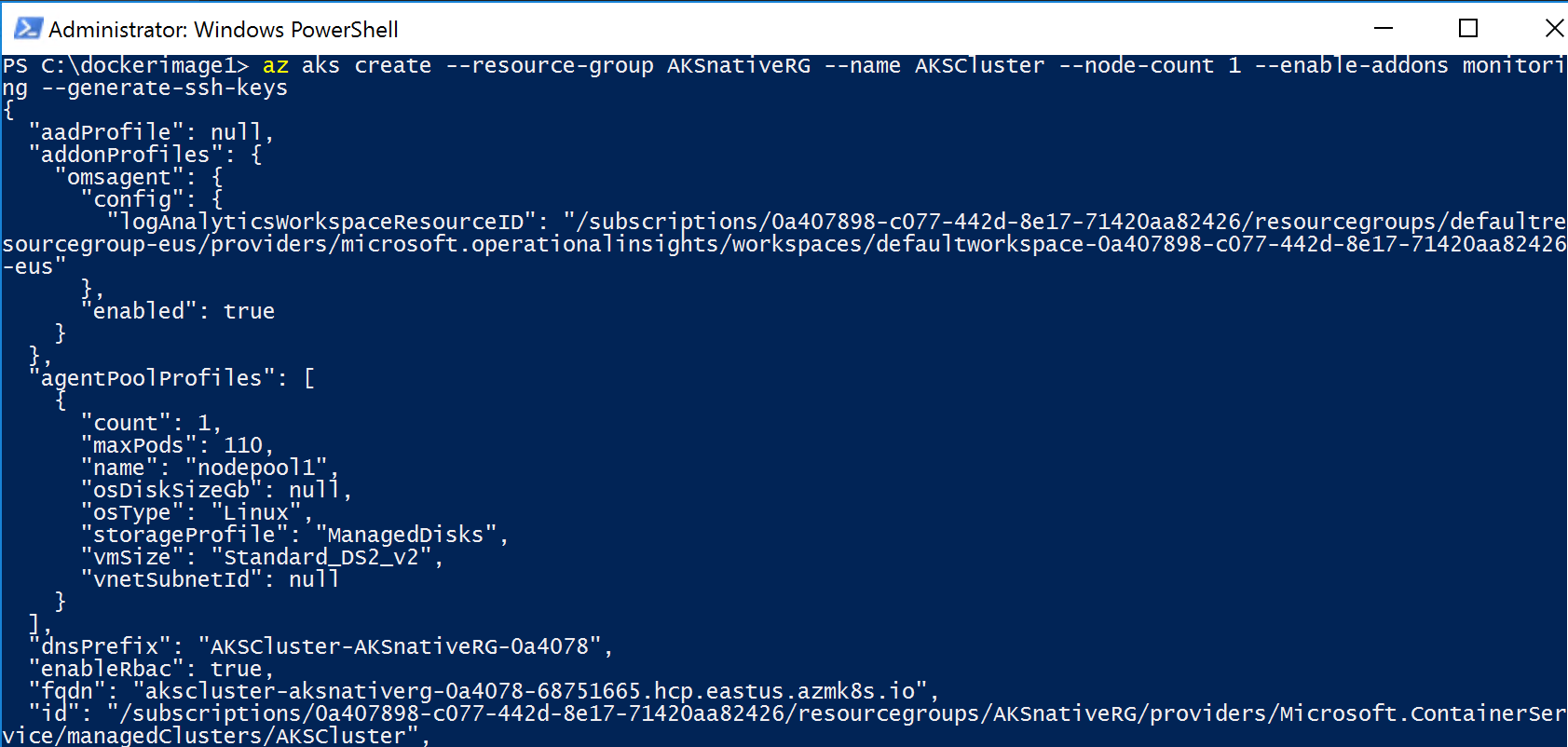
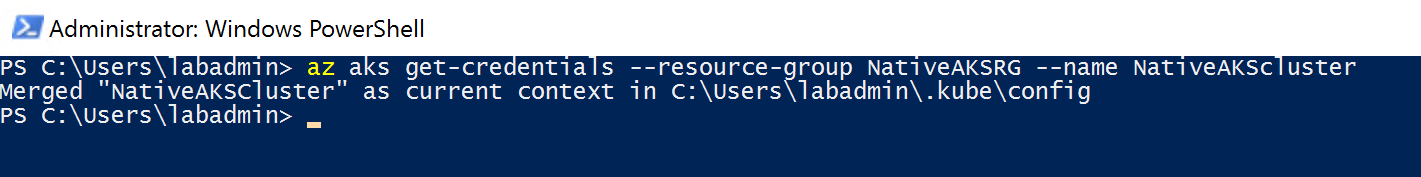
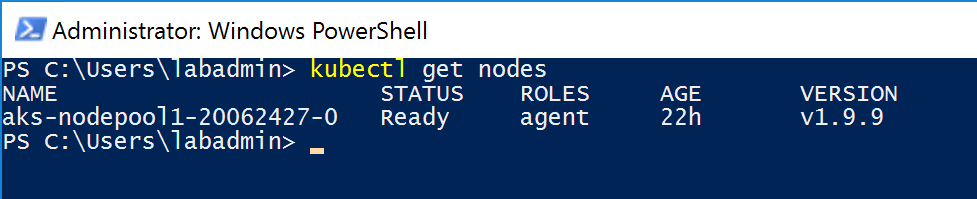
In this lab, you will learn what it takes to deploy an Azure Kubernetes Service (AKS), and noticing the differences compared to Azure Container Services (ACS) for Kubernetes.

## 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

---

apiVersion: v1

kind: Service

metadata:

name: akshelloworld

spec:

type: LoadBalancer

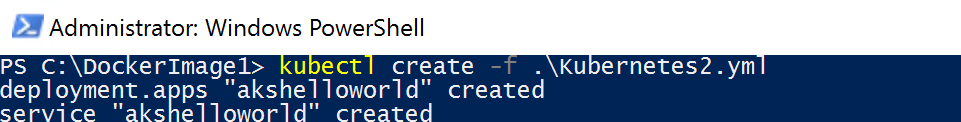
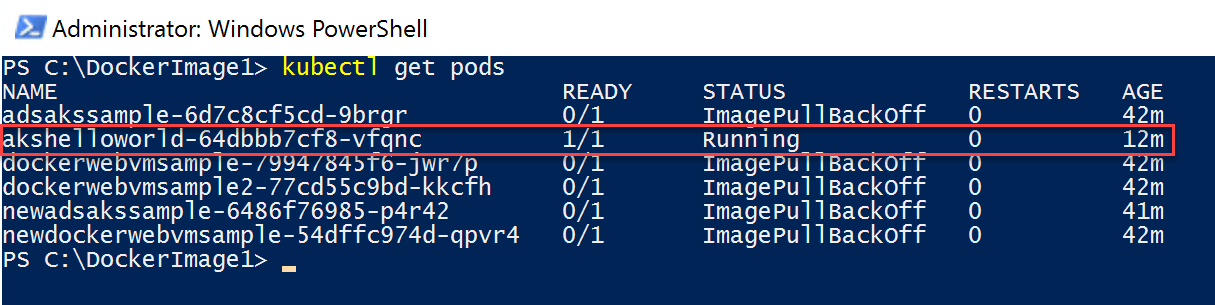
ports:

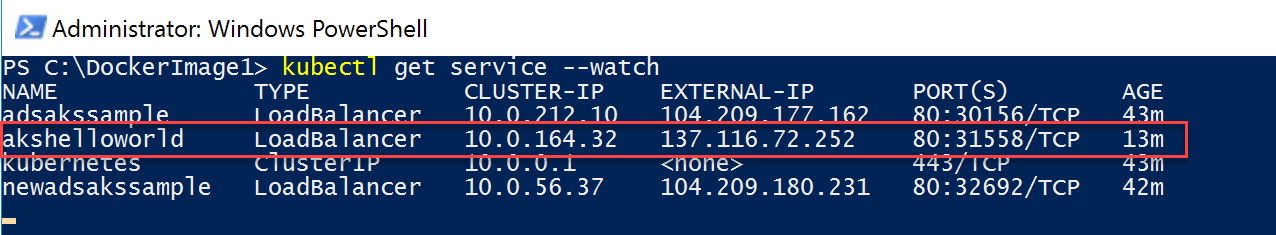
- port: 80

selector:

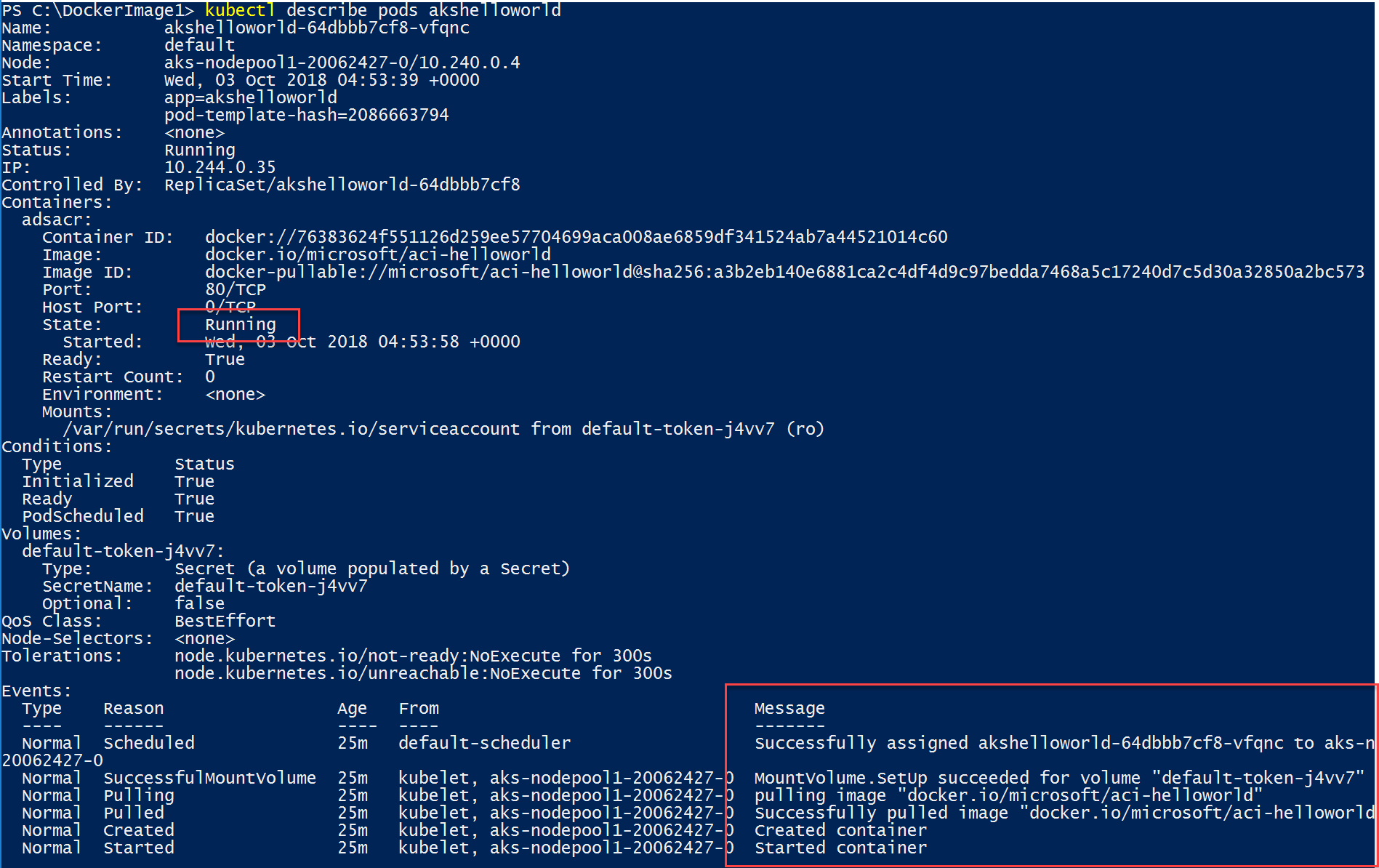
app: akshelloworld

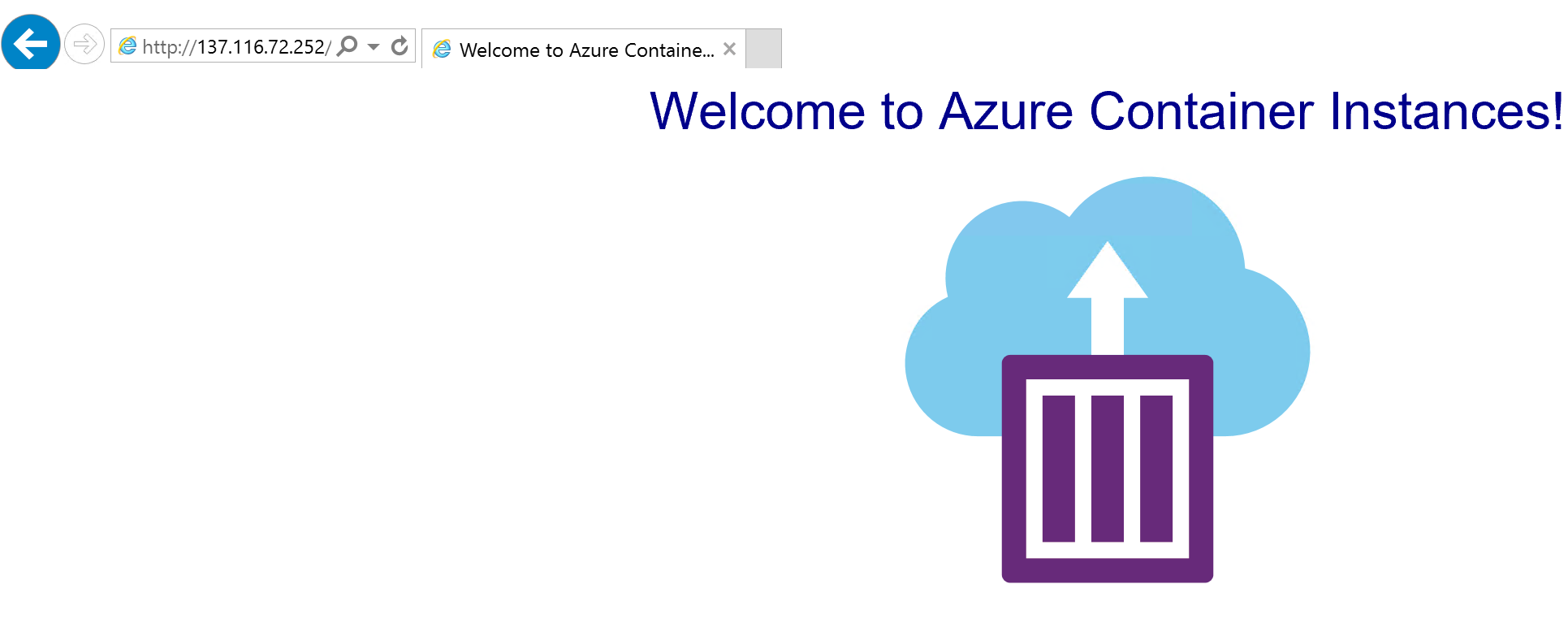
**Note the parameter** image: docker.io/microsoft/aci-helloworld **is the pointer to the public image in Docker Hub.**

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

kubectl get service dockerwebvmsample --watch  
  


1. Another useful command you can use is “kubectl describe”, which gives you detailed information regarding a running (or failing) pod.  
     
   kubect describe akshelloworld



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

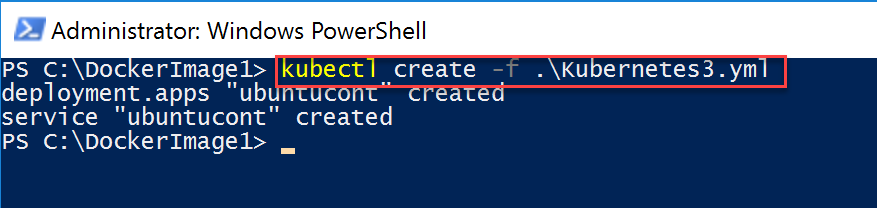
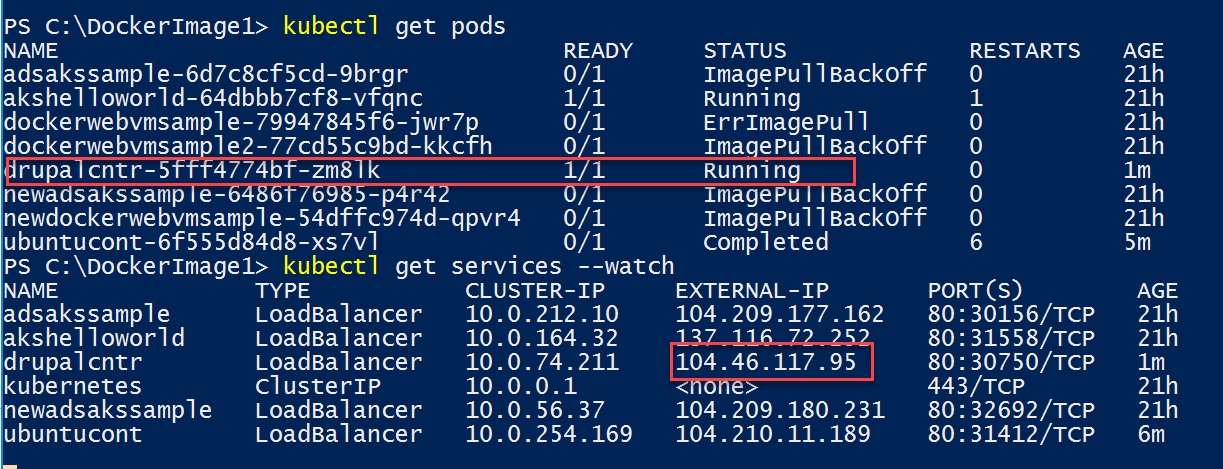
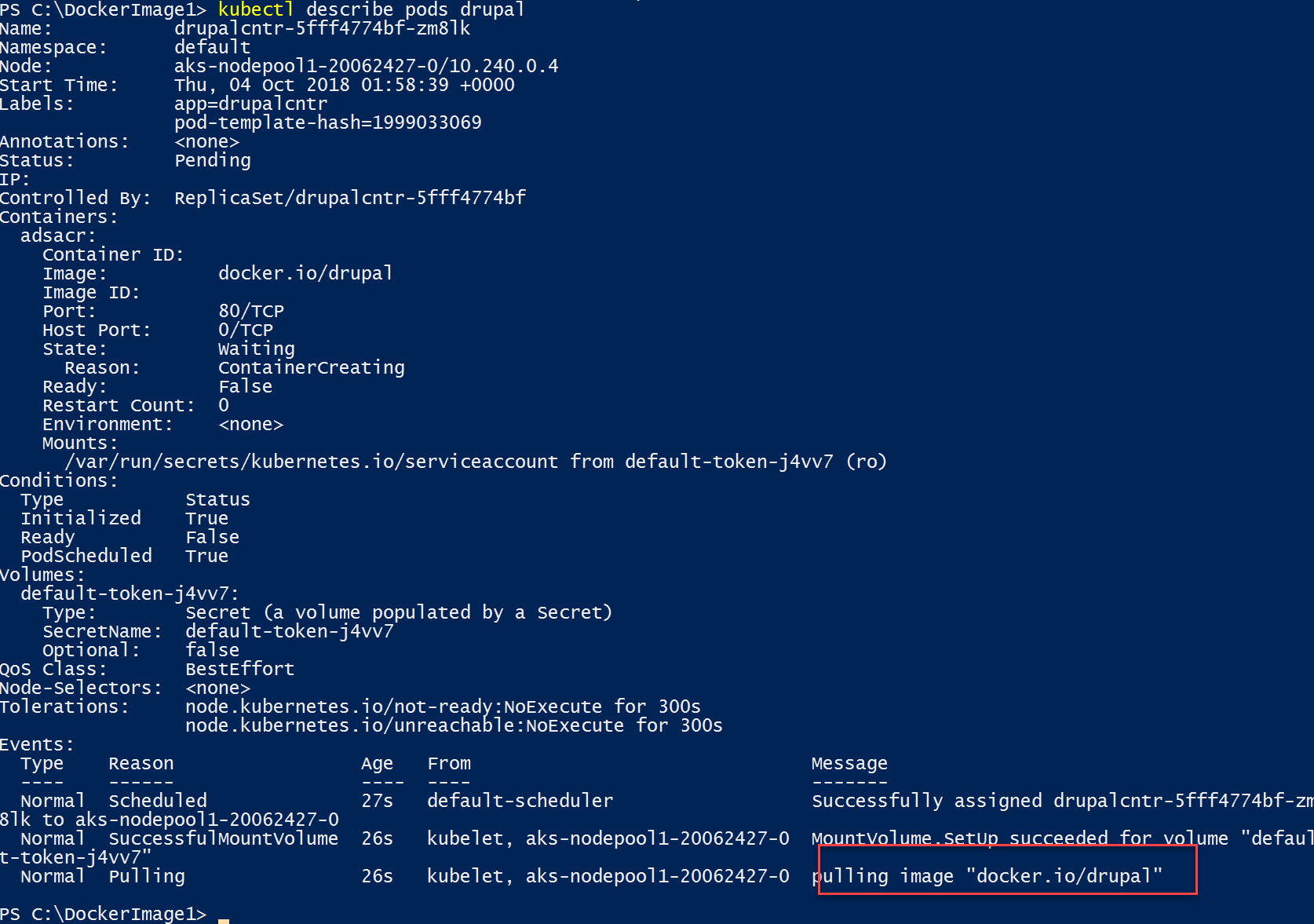
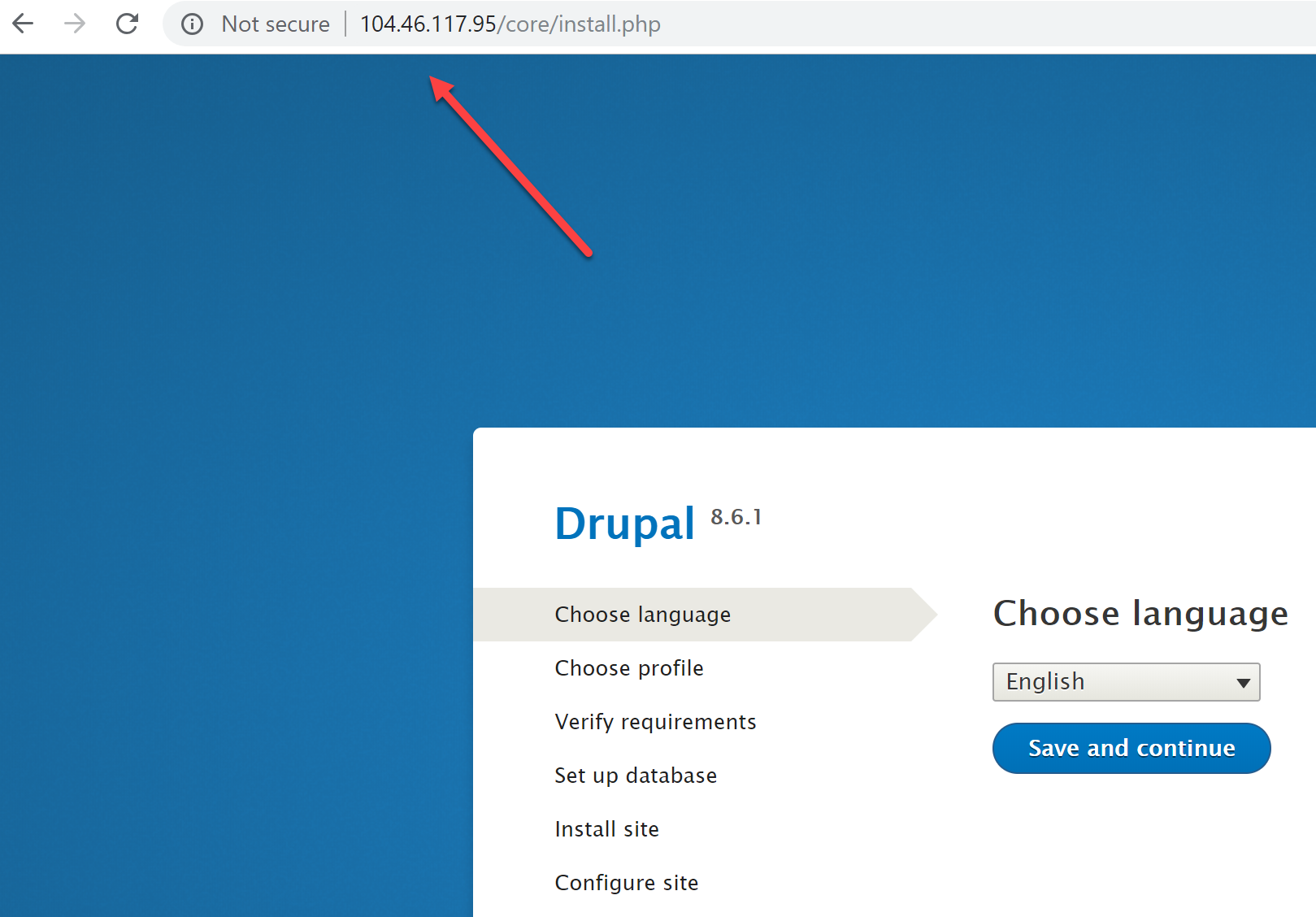
ports:

- port: 80

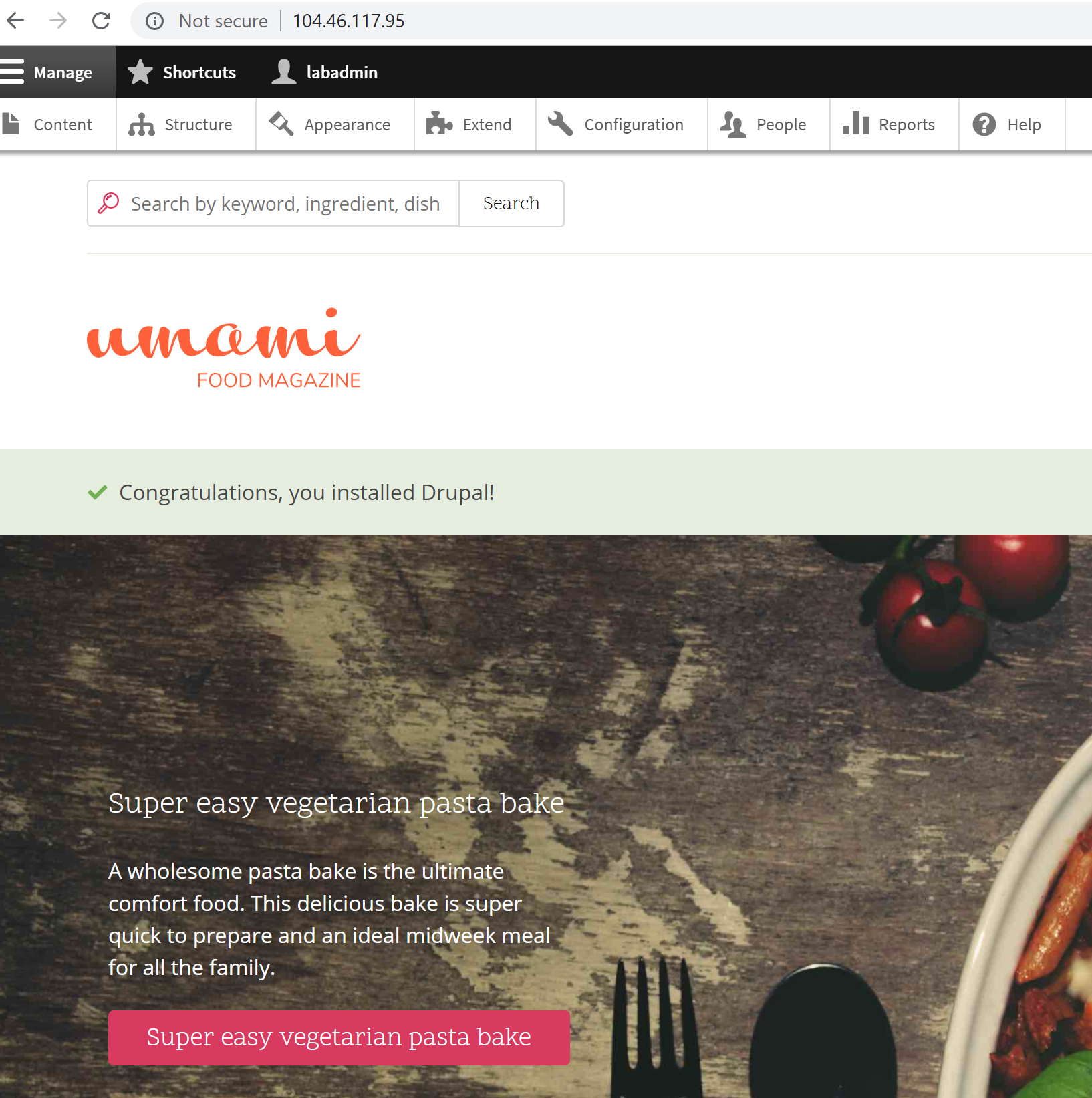
selector:

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.

1. **Save** the files as Kubernetes3.yml in the C:\Dockerimage1 folder (the one we already used in previous labs).
2. Next, **run the deployment of this Kubernetes service, by using the following command:**  
   kubectl create -f “path to kubernetes3.yml file here”  
     
   
3. **Validate** if the image is being pushed into the Kubernetes Service, by **checking the pods** again:  
     
   kubectl get pods   
     
   kubectl get services --watch   
     
   
4. Or inspecting the full container deployment process again, by running  
     
   kubectl describe pods drupalcntr  
     
   
5. **Give it another 2-3 minutes, then open your internet browser**, and **connect** to the EXTERNAL-IP of the drupalcntr service:  
     
   
6. **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**](mailto:L@BadminPa55w.rd)

**Credentials**

1. Once all the steps are completed, **give it a few seconds more**, which will **open up the sample web site**. How cool is that!!  
     
   
2. **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…!!
3. 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.