Lab 08 – Deploy the model to K8s

Lab 08

In this lab we will use the private repo in Azure which now holds our image and deploy it using Kubernetes.

Step 1

Create a new instance of Azure Kubernetes Service (AKS)

1. Pick a name for your instance of K8s (mlworkshopaks)
2. Run the following code (amend to your settings)

```
az aks create --resource-group mlworkshoprg --name mlworkshopaks --node-count 1 --generate-ssh-keys
```

Right this is going to take a few minutes. Ask me some questions, take a comfort break, check some emails.... **It will take 15 minutes**

You could play this game to kill some time. http://guessthecorrelation.com/

So, in the background what this is doing is building all the infrastructure required to deploy any container to Kubernetes. The old version of AKS ACS used to show you all the stuff it made. It was confusing. Now we just see one little resource. Much better.

Ok, that has finished creating.

Step 2

If you followed the instructions which were emailed out, you installed the wrong command line. You needed AKS... Sorry about that. Let's fix that.

3. Open a command prompt
4. Run the following (amended to your settings)

```
az aks install-cli
```

Step 3

We need to authenticate of Azure CLI with the new AKS service.

Lab 08 – Deploy the model to K8s

5. Run the following (amended to your settings)

```
az aks get-credentials --resource-group mlworkshoprg --name mlworkshopaks
```

Step 4

Let's look at what is running on our Kubernetes cluster. If you run the following you will see a list of all the nodes.

6. Run the following

```
kubectl get nodes
```

This will show the one and only node we created. If we had scaled this deployment to multiple nodes, we would see that here. A node can contain multiple pods. What we are deploying is pods. We can run loads of pods on the same node, but they will share resources.

Step 5

We need to write some YAML. YAML is the slimmer version of Json (which is the slimmer version of XML).

YAML is what Kubernetes requires in order to work out what needs to go in each POD.

7. Create a new folder called "Kubernetes"

8. In that folder create a new file called diabetesproduction.yml

9. Add the following (amending for your settings)

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: productiondiabetes
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: productiondiabetes
    spec:
      containers:
      - name: productiondiabetes
        image: <your version>.azurecr.io/productiondiabetes:v1
        ports:
        - containerPort: 5071
          name: dockerport
```

Lab 08 – Deploy the model to K8s

```
---
apiVersion: v1
kind: Service
metadata:
  name: productiondiabetes
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: dockerport
    protocol: TCP
  selector:
    app: productiondiabetes
```

10. Save your changes and close the file.

Step 6

Use the Kubectl command apply to install our container.

11. Run the following

```
kubectl apply -f G:\GitHubProjects\workshop-
ModelManagement\MachineLearningFromModelToProduction\Kubernetes\productiondiabetes
.yml
```

2 things should have been created. A deployment and a service

12. Check to see if that has built yet.

13. Run the following

```
kubectl get service productiondiabetes
```

If this returns you an IP address then there is a good chance it has worked. To make sure we can look at the K8s dashboard.

Step 7

Check the dashboard.

14. Run the following (amending for your settings)

```
az aks browse --resource-group Mlworkshoprg --name Mlworkshopaks
```

Now there is currently a bug connecting ACR and AKS. It is to do with the Service Principal which is created. We can resolve this by doing the following.

15. Run the following and copy the result.

Lab 08 – Deploy the model to K8s

```
az aks show --resource-group Mlworkshop --name Mlworkshop --query
"servicePrincipalProfile.clientId" --output table
```

c0f34e8c-3699-4759-b564-ec5c2e8949e7

16. Run the following and copy the result

```
az acr show --name MlworkshopTerry --resource-group Mlworkshop --query "id" --
output table
```

/subscriptions/599eb5a3-0fc5-4e18-a220-39ebaa74f940/resourceGroups/Mlworkshop/providers/Microsoft.ContainerRegistry/registries/MlworkshopTerry

17. Combine the two as follows and execute.

```
az role assignment create --assignee c0f34e8c-3699-4759-b564-ec5c2e8949e7 --role
Reader --scope /subscriptions/599eb5a3-0fc5-4e18-a220-
39ebaa74f940/resourceGroups/Mlworkshop/providers/Microsoft.ContainerRegistry/regis
tries/MlworkshopTerry
```

Ok that should have resolved the security problem.

Let's remove the old deployment and go again.

18. Run the following

```
kubectl delete -f G:\GitHubProjects\workshop-
ModelManagement\MachineLearningFromModelToProduction\Kubernetes\productiondiabetes
.yml
```

Then apply

19. Run the following

```
kubectl apply -f G:\GitHubProjects\workshop-
ModelManagement\MachineLearningFromModelToProduction\Kubernetes\productiondiabetes
.yml
```

20. Open a new command prompt and run the following again:

```
az aks browse --resource-group Mlworkshop --name Mlworkshop
```

You will now see that there are no errors.

Step 8 Test our model.

21. Run the following

```
kubectl get services
```

Copy the IP address from the load balancer. Make sure you copy the external ip address.

22. Navigate to the following IP address.

Lab 08 – Deploy the model to K8s

```
http://40.113.78.55/train
http://40.113.78.55/score?var1=1
```

Lab complete. You have now created a model and deployed it to a container running on Kubernetes.

IF YOU HAVE 14 ERRORS run this:

```
kubectl create clusterrolebinding kubernetes-dashboard -n kube-system --
clusterrole=cluster-admin --serviceaccount=kube-system:kubernetes-dashboard
```