Get to know Azure
Explore the world of cloud computing and learn what sets Azure apart

Azure Developer Series

Migrating a dotnetcore 2-tier application to Azure, using different architectures and DevOps best practices

Hands-On-Labs step-by-step guides

Prepared by:

Peter De Tender

CEO and Lead Technical Trainer
PDTIT and 007FFFLearning.com

@pdtit @007FFFLearning

Version: Sept 2019 – 1.0

Contents

Migrating a dotnetcore 2-tiered application to Azure using different architectures and DevOps best practices - Hands-On-Labs step-by-step

You are part of an organization that is running a dotnetcore e-commerce platform application, using Windows Server infrastructure on-premises today, comprising a WebVM running Windows Server 2012 R2 with Internet Information Server (IIS) and a 2nd SQLVM running Windows Server 2012 R2 and SQL Server 2014.

The business has approved a migration of this business-critical workload to Azure, and you are nominated as the cloud solution architect for this project. No decision has been made yet on what the final architecture should or will look like. Your first task is building a Proof-of-Concept in your Azure environment, to test out the different architectures possible:

- Infrastructure as a Service (IAAS)
- Platform as a Service (PAAS)
- Containers as a Service (CaaS)

At the same time, your CIO wants to make use of this project to switch from a more traditional mode of operations, with barriers between IT sysadmin teams and Developer teams, to a DevOps way of working. Therefore, you are tasked to explore Azure DevOps and determine where CI/CD Pipelines can assist in optimizing the deployment and running operations of this e-commerce platform, especially when deploying updates to the application.

As you are new to the continuous changes in Azure, you want to make sure this process goes as smooth as possible, starting from the assessment to migration to day-to-day operations.

Abstract and Learning Objectives

This workshop enables anyone to learn, understand and build a Proof of Concept, in performing a multi-tiered .Net Core web application using Microsoft SQL Server database, platform migration to Azure public cloud, leveraging on different Azure Infrastructure as a Service, Azure Platform as a Service (PaaS) and Azure Container offerings like Azure Container Instance (ACI) and Azure Kubernetes Services (AKS).

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 learn about the importance of performing proper assessments, and what tools Microsoft offers to help in this migration preparation phase. Once the application has been deployed on Azure Virtual Machines, students learn about Microsoft SQL database migration to SQL Azure PaaS, as well as deploying and migrating web applications to Azure Web Apps.

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

In the last part of the workshop, students get introduced to Azure DevOps, the new Microsoft Application Lifecycle environment, helping in building a CI/CD Pipeline to publish workloads using the DevOps principals and concepts, showing the integration with the rest of the already touched on Azure services like Azure Web Apps and Azure Kubernetes Services (AKS), closing the workshop with a module on overall Azure monitoring and operations and what tools Azure has available to assist your IT teams in this challenge.

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 2019;
- Publishing a .NET Core e-commerce application to an Azure Web Virtual Machine and SQL DB Virtual Machine;
- Performing a proper assessment of the as-is Web and SQL infrastructure using Microsoft Assessment Tools;
- Migrating a SQL 2014 database to Azure SQL PaaS (Lift & Shift);
- Migrating a .NET Core web application to Azure Web Apps (Lift & Shift);
- Containerizing a .NET Core web application using Docker, and pushing to Azure Container Registry (ACR);
- Running Azure Container Instance (ACI) and WebApp for Containers;
- Deploy and run Azure Azure Kubernetes Services (AKS);

- Deploying Azure DevOps and building a CI/CD Pipeline for the subject e-commerce application;
- Managing and Monitoring Azure Kubernetes Services (AKS);

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) 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.
b) 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.
c) 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); this could also be Visual Studio 2019 community edition - latest version
- Docker for Windows (updated to latest version)
- Azure CLI 2.0 (updated to latest version)
- Kubernetes CLI (updated to latest version)
- SimplCommerce Open Source e-commerce platform example (http://www.simplcommerce.com)

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 6: Deploying and running Azure Kubernetes Services (AKS)

What you will learn

In this lab, you will learn what it takes to deploy an Azure Kubernetes Service (AKS), creating a Kubernetes YAML deploy file, and running a Docker containerized application within the AKS cluster.

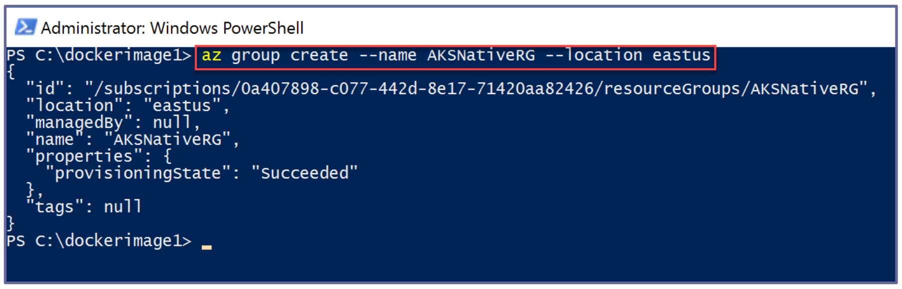
Time Estimate

This lab should take about 45 minutes to complete.

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:

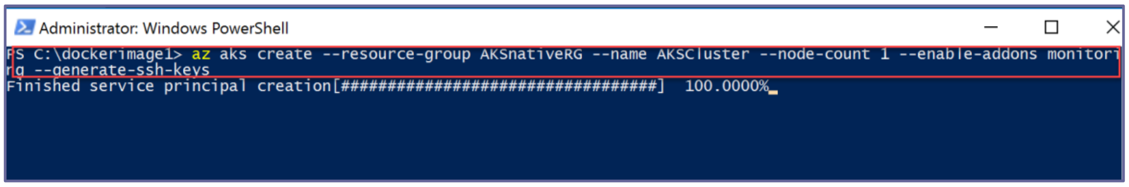
   az group create --name AKSNameofChoiceRG --location eastus

```
PS C:\dockerimage1> az group create --name AKSNativeRG --location eastus
{
  "id": "/subscriptions/0a407898-c077-442d-8e17-71420aa82426/resourceGroups/AKSNativeRG",
  "location": "eastus",
  "managedBy": null,
  "name": "AKSNativeRG",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null
}
PS C:\dockerimage1> _
```

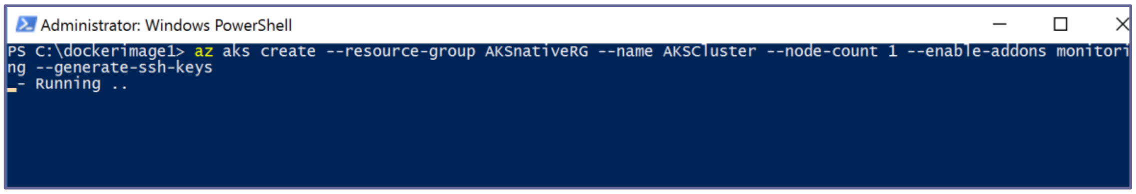
2. Next, run the following command to deploy the actual Azure Kubernetes Services resource:

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

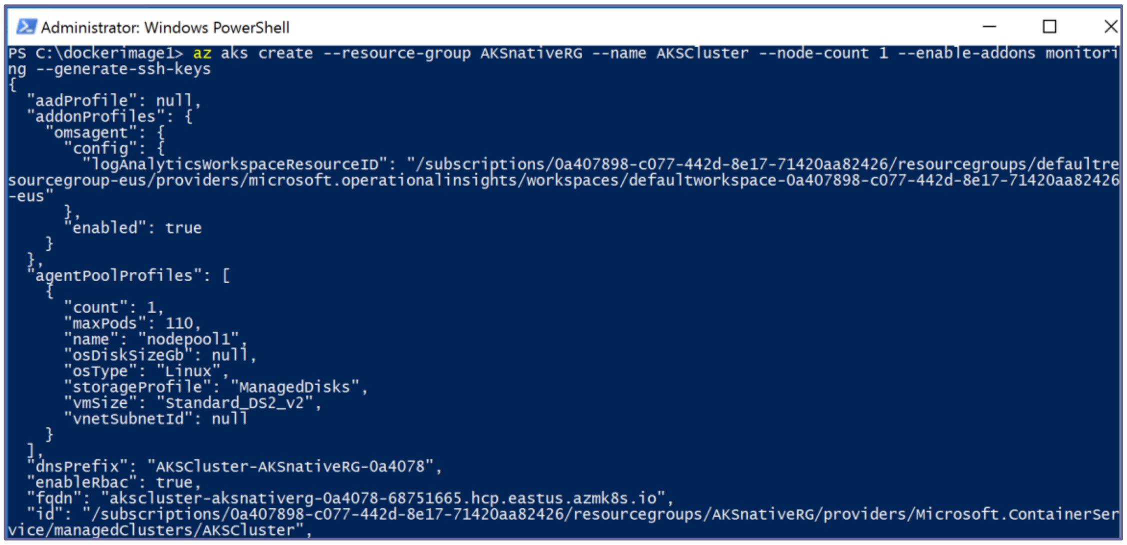
```
PS C:\dockerimage1> az aks create --resource-group AKSnativeRG --name AKSCluster --node-count 1 --enable-addons monitori
ng --generate-ssh-keys
Finished service principal creation[##################################]  100.0000%_
```

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



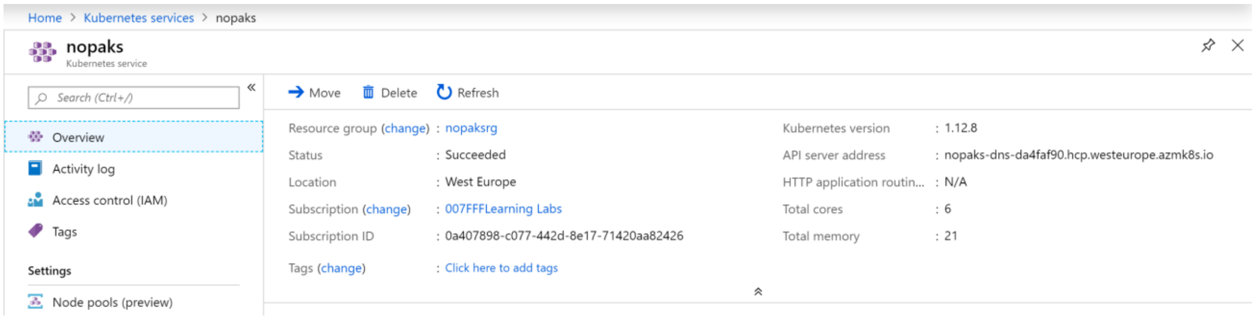
```
PS C:\dockerimage1> az aks create --resource-group AKSnativeRG --name AKSCluster --node-count 1 --enable-addons monitori
ng --generate-ssh-keys
- Running ..
```

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:



```
PS C:\dockerimage1> az aks create --resource-group AKSnativeRG --name AKSCluster --node-count 1 --enable-addons monitori
ng --generate-ssh-keys
{
  "aadProfile": null,
  "addonProfiles": {
    "omsagent": {
      "config": {
        "logAnalyticsWorkspaceResourceID": "/subscriptions/0a407898-c077-442d-8e17-71420aa82426/resourcegroups/defaultre
sourcegroup-eus/providers/microsoft.operationalinsights/workspaces/defaultworkspace-0a407898-c077-442d-8e17-71420aa82426
-eus"
      },
      "enabled": true
    }
  },
  "agentPoolProfiles": [
    {
      "count": 1,
      "maxPods": 110,
      "name": "nodepool1",
      "osDiskSizeGb": null,
      "osType": "Linux",
      "storageProfile": "ManagedDisks",
      "vmSize": "Standard_DS2_v2",
      "vnetSubnetId": null
    }
  ],
  "dnsPrefix": "AKSCluster-AKSnativeRG-0a4078",
  "enableRbac": true,
  "fqdn": "akscluster-aksnativerg-0a4078-68751665.hcp.eastus.azmk8s.io",
  "id": "/subscriptions/0a407898-c077-442d-8e17-71420aa82426/resourcegroups/AKSnativeRG/providers/Microsoft.ContainerSer
vice/managedClusters/AKSCluster",
```

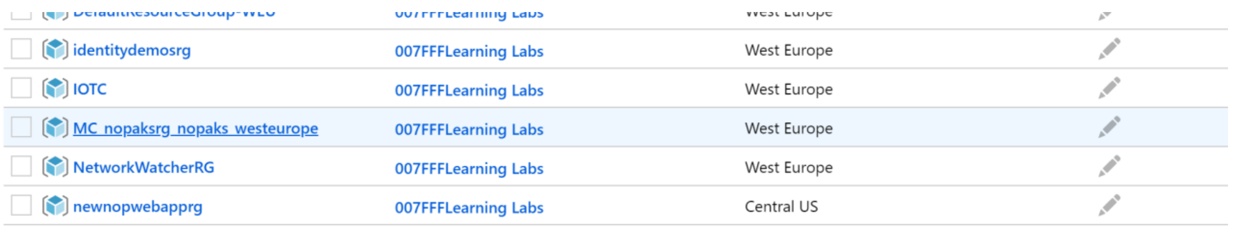
4. Validating the deployment from the Azure Portal, shows the following:



Resource group (change)	: nopaksrg	Kubernetes version	: 1.12.8
Status	: Succeeded	API server address	: nopaks-dns-da4faf90.hcp.westeurope.azmk8s.io
Location	: West Europe	HTTP application routin...	: N/A
Subscription (change)	: 007FFFLearning Labs	Total cores	: 6
Subscription ID	: 0a407898-c077-442d-8e17-71420aa82426	Total memory	: 21
Tags (change)	: Click here to add tags		

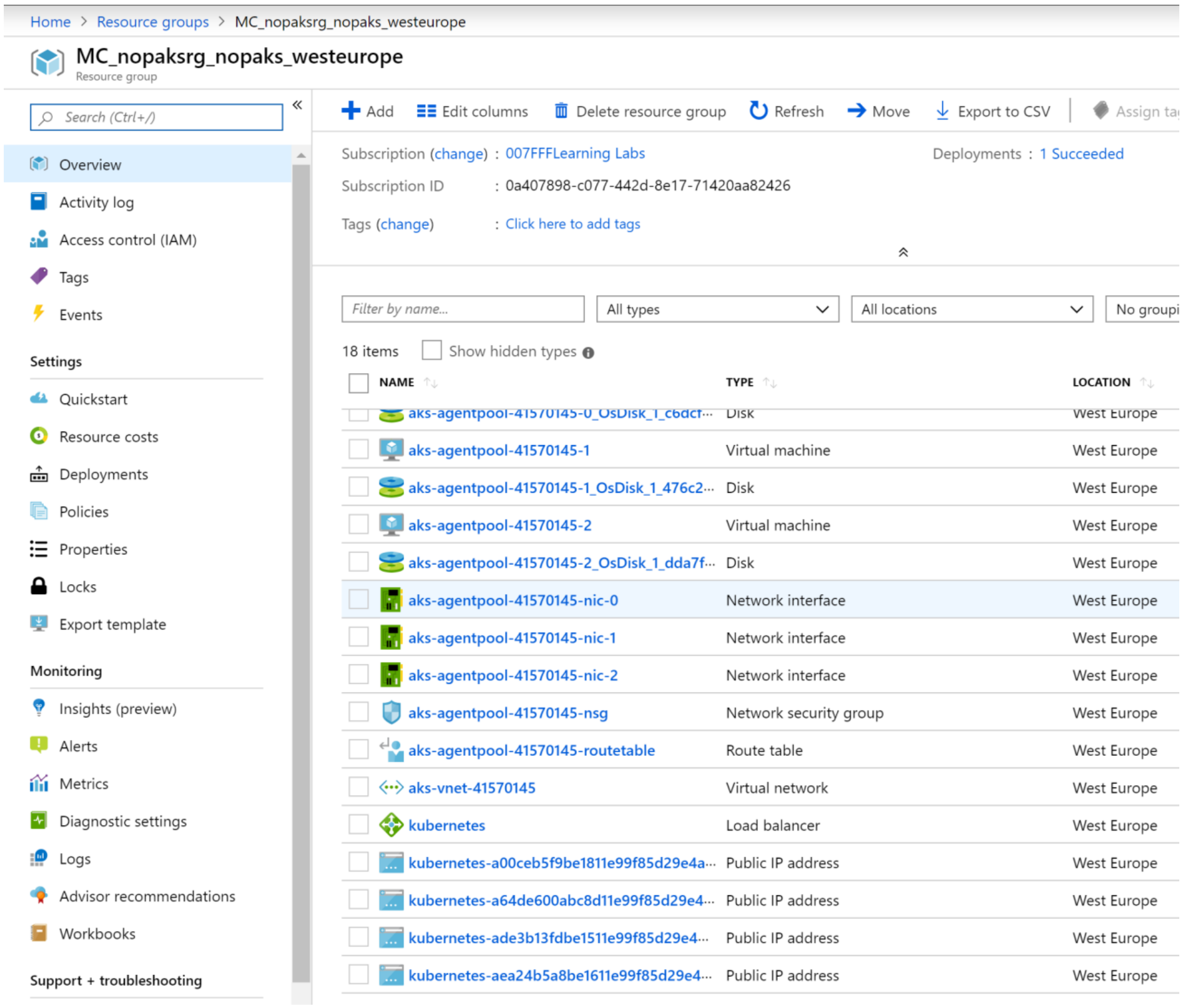
5. While AKS runs "as a service", it relies on Azure infrastructure for most of its service. And obviously the intent is not to start managing the different cluster nodes yourself, but nevertheless interesting to see is how the AKS cluster is technically built up. To get a view on that, check for a Resource Group, reflecting the name of the AKS cluster as follows:

   MC_<aksclusterRG>_aksclustername_region



DefaultResourceGroup-WEU	007FFFLearning Labs	West Europe
identitydemosrg	007FFFLearning Labs	West Europe
IOTC	007FFFLearning Labs	West Europe
MC_nopaksrg_nopaks_westeurope	007FFFLearning Labs	West Europe
NetworkWatcherRG	007FFFLearning Labs	West Europe
newnopwebapprg	007FFFLearning Labs	Central US

6. Containing all required Azure IAAS resources to build up the AKS cluster (Virtual machines for the nodes, virtual network, storage, load balancer,…)



Home > Resource groups > MC_nopaksrg_nopaks_westeurope

MC_nopaksrg_nopaks_westeurope
Resource group

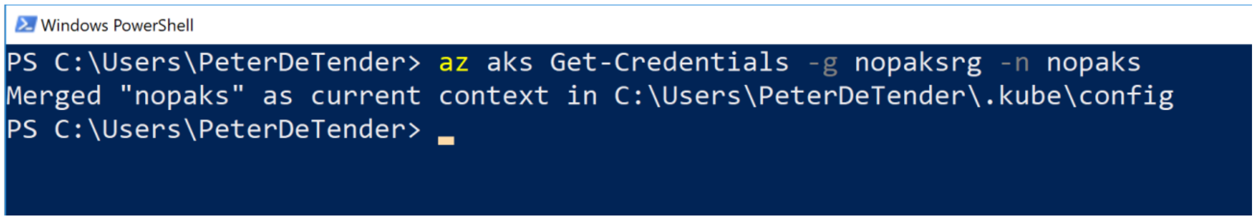
Subscription (change) : 007FFFLearning Labs
Deployments : 1 Succeeded
Subscription ID : 0a407898-c077-442d-8e17-71420aa82426
Tags (change) : Click here to add tags

18 items Show hidden types

NAME	TYPE	LOCATION
aks-agentpool-41570145-0_OsDisk_1_c6dcf…	Disk	West Europe
aks-agentpool-41570145-1	Virtual machine	West Europe
aks-agentpool-41570145-1_OsDisk_1_476c2…	Disk	West Europe
aks-agentpool-41570145-2	Virtual machine	West Europe
aks-agentpool-41570145-2_OsDisk_1_dda7f…	Disk	West Europe
aks-agentpool-41570145-nic-0	Network interface	West Europe
aks-agentpool-41570145-nic-1	Network interface	West Europe
aks-agentpool-41570145-nic-2	Network interface	West Europe
aks-agentpool-41570145-nsg	Network security group	West Europe
aks-agentpool-41570145-routetable	Route table	West Europe
aks-vnet-41570145	Virtual network	West Europe
kubernetes	Load balancer	West Europe
kubernetes-a00ceb5f9be1811e99f85d29e4a…	Public IP address	West Europe
kubernetes-a64de600abc8d11e99f85d29e4…	Public IP address	West Europe
kubernetes-ade3b13fdbe1511e99f85d29e4…	Public IP address	West Europe
kubernetes-aea24b5a8be1611e99f85d29e4…	Public IP address	West Europe

7. 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:**

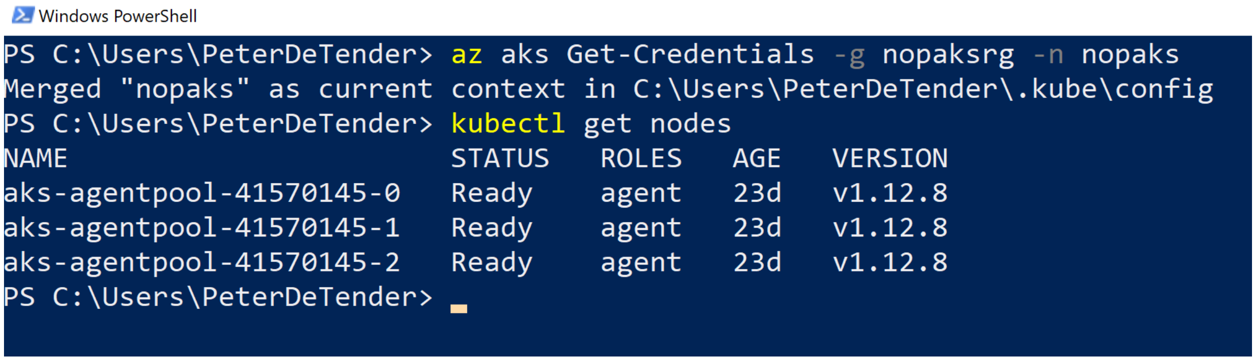
```
az aks get-credentials --resource-group [SUFFIX]AKSRG --name
[SUFFIX]AKSCluster
```


Windows PowerShell

```
PS C:\Users\PeterDeTender> az aks Get-Credentials -g nopaksrg -n nopaks
Merged "nopaks" as current context in C:\Users\PeterDeTender\.kube\config
PS C:\Users\PeterDeTender> _
```

8. Next, validate the functioning by checking the nodes:

kubectl get nodes


Windows PowerShell

```
PS C:\Users\PeterDeTender> az aks Get-Credentials -g nopaksrg -n nopaks
Merged "nopaks" as current context in C:\Users\PeterDeTender\.kube\config
PS C:\Users\PeterDeTender> kubectl get nodes
NAME                       STATUS   ROLES   AGE   VERSION
aks-agentpool-41570145-0   Ready    agent   23d   v1.12.8
aks-agentpool-41570145-1   Ready    agent   23d   v1.12.8
aks-agentpool-41570145-2   Ready    agent   23d   v1.12.8
PS C:\Users\PeterDeTender> _
```

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

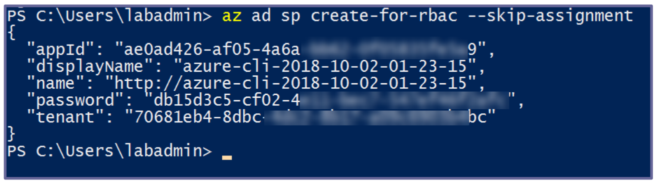
Task 2: Configuring RBAC for managing Azure Kubernetes Services and ACR integration

In the previous step, you deployed the AKS infrastructure and the AKS as a Service resource in Azure. Using the **kubectl get nodes**, you validated the underlying Kubernetes infrastructure is up and running.

Before we can have Kubernetes picking up Docker images from the Azure Container Registry we deployed earlier, we need to define RBAC access for the Kubernetes resource to allow this.

1. To allow Kubernetes to pull our Docker image from the Azure Container Registry, **we need to define an Azure Service Principal object**, which integrates with RBAC. **Create the Service Principal** as follows, from within your **PowerShell window**:

   ```
   az ad sp create-for-rbac --skip-assignment
   ```

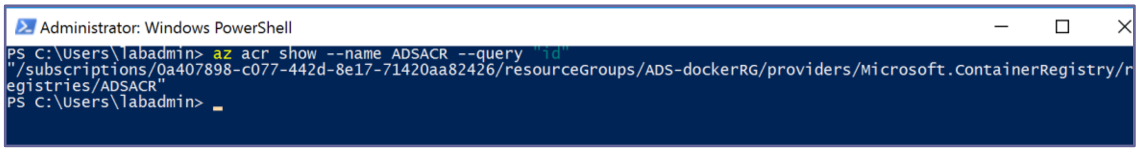
   ```
   PS C:\Users\labadmin> az ad sp create-for-rbac --skip-assignment
   {
     "appId": "ae0ad426-af05-4a6a[redacted]9",
     "displayName": "azure-cli-2018-10-02-01-23-15",
     "name": "http://azure-cli-2018-10-02-01-23-15",
     "password": "db15d3c5-cf02-4[redacted]",
     "tenant": "70681eb4-8dbc[redacted]bc"
   }
   PS C:\Users\labadmin> _
   ```

   Since we need parts of this information later on, might be good to **copy this to a Notepad doc** for easy retrieval.

2. This command creates an applicationID, provides displayname and tenant information that you need later on in the Kubernetes YAML-file (similar to the Dockerfile we used earlier, but for Kubernetes deployments).

3. Next item information we need is the **full Azure Resource ID for our Azure Container Registry**. This information **can be retrieved** using the following command:

   ```
   az acr show --name [SUFFIX]ACR --query "id" --output table
   ```

   ```
   PS C:\Users\labadmin> az acr show --name ADSACR --query "id"
   "/subscriptions/0a407898-c077-442d-8e17-71420aa82426/resourceGroups/ADS-dockerRG/providers/Microsoft.ContainerRegistry/r
   egistries/ADSACR"
   PS C:\Users\labadmin> _
   ```

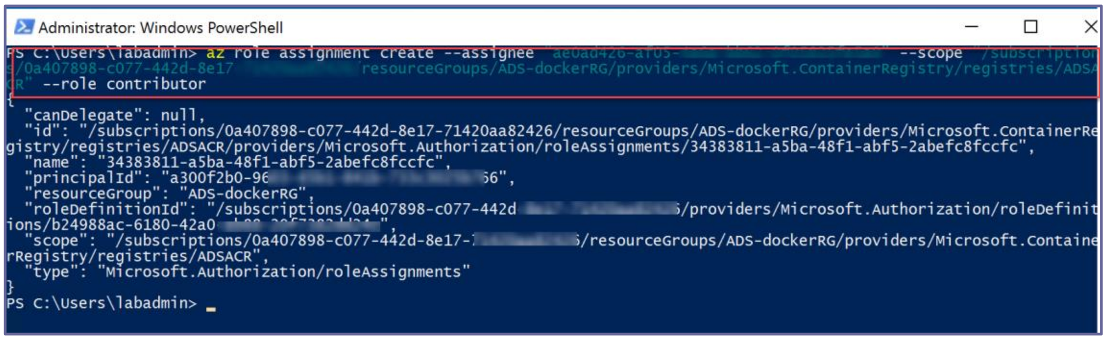
   Since we need parts of this information later on, might be good to **copy this to a Notepad doc** for easy retrieval.

4. Next, **assign the contributor role** for the previously created "appid", to this ACR object, by executing the following command:

```
az role assignment create --assignee "appid" --scope "ACRid" --
role contributor
```

which maps like this in my environment (replaced some characters for security reasons):

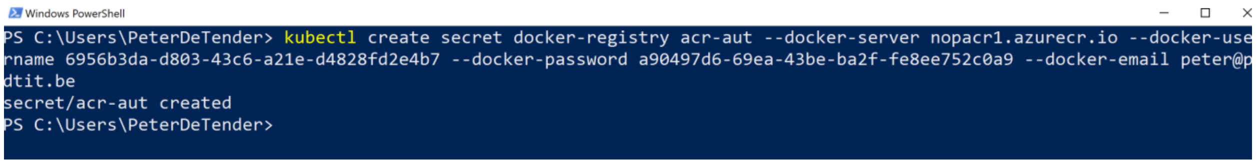
```
az role assignment create --assignee "ae0ad426-af05-4a6a-0000-
00000000" --scope "/subscriptions/0a407898-c077-0000-0000-
7142000000000/resourceGroups/ADS-
dockerRG/providers/Microsoft.ContainerRegistry/registries/ADSACR"
--role contributor
```



```
PS C:\Users\labadmin> az role assignment create --assignee "ae0ad426-af05-[redacted]" --scope "/subscription
s/0a407898-c077-442d-8e17[redacted]resourceGroups/ADS-dockerRG/providers/Microsoft.ContainerRegistry/registries/ADSA
CR" --role contributor
{
  "canDelegate": null,
  "id": "/subscriptions/0a407898-c077-442d-8e17-71420aa82426/resourceGroups/ADS-dockerRG/providers/Microsoft.ContainerRe
gistry/registries/ADSACR/providers/Microsoft.Authorization/roleAssignments/34383811-a5ba-48f1-abf5-2abefc8fccfc",
  "name": "34383811-a5ba-48f1-abf5-2abefc8fccfc",
  "principalId": "a300f2b0-96[redacted]6",
  "resourceGroup": "ADS-dockerRG",
  "roleDefinitionId": "/subscriptions/0a407898-c077-442d[redacted]/providers/Microsoft.Authorization/roleDefinit
ions/b24988ac-6180-42a0[redacted]",
  "scope": "/subscriptions/0a407898-c077-442d-8e17-[redacted]/resourceGroups/ADS-dockerRG/providers/Microsoft.Containe
rRegistry/registries/ADSACR",
  "type": "Microsoft.Authorization/roleAssignments"
}
PS C:\Users\labadmin> _
```

5. We also will instruct **kubectl** (the Kubernetes cluster actually, by using kubectl…), to use a secret, which will be used to get access to the Azure Container Registry, using the following command:

```
kubectl create secret docker-registry acr-auth --docker-server
<yourACR>.azurecr.io --docker-username 6956b3da-0000000 (Appid
here) --docker-password a90497d6-69ea-000000 <app password here>
--docker-email <your docker account email>
```



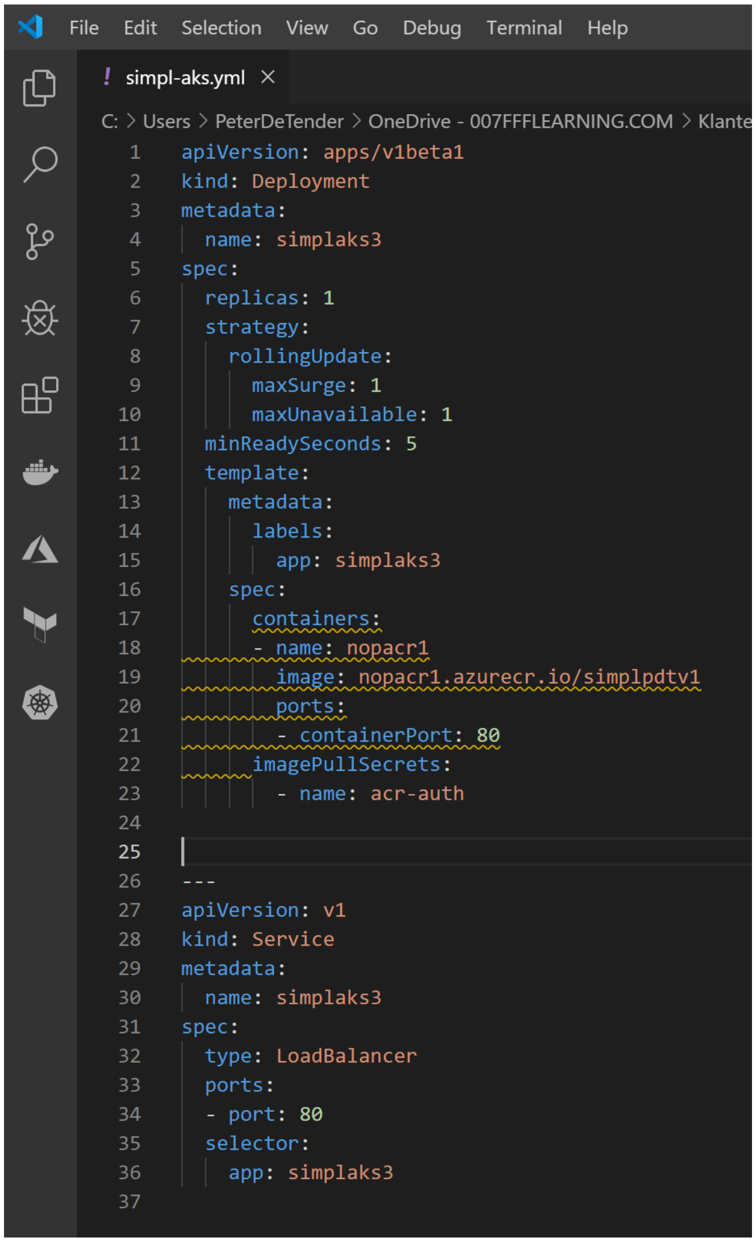
```
PS C:\Users\PeterDeTender> kubectl create secret docker-registry acr-aut --docker-server nopacr1.azurecr.io --docker-use
rname 6956b3da-d803-43c6-a21e-d4828fd2e4b7 --docker-password a90497d6-69ea-43be-ba2f-fe8ee752c0a9 --docker-email peter@p
dtit.be
secret/acr-aut created
PS C:\Users\PeterDeTender>
```

With all the back-end information and RBAC Service Principals in place, we **can build our YAML-deployment file for Kubernetes**. Key information in here is the name of your Azure Container Registry, the Docker container file that you want to push to the Kubernetes cluster, and what port the container should run on.

Task 3: Running a Docker Container image from Azure Container Registry in an Azure Kubernetes Service

1. First step in this Docker image to Kubernetes integration, is authoring a **Kubernetes.YML** file. Think of this as a technical script, in which you define what Docker image should be used and started up on the Kubernetes cluster, what port the running container should use etc.

2. On the **lab-jumpVM, open Visual Studio Code**. Browse to the sources folder you used before, containing the sample e-commerce source code, and check for a file **simpl-aks.yml**
   The content looks like the following:


File
Edit
Selection
View
Go
Debug
Terminal
Help
simpl-aks.yml
C: > Users > PeterDeTender > OneDrive - 007FFFLEARNING.COM > Klante

```yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: simplaks3
spec:
  replicas: 1
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  minReadySeconds: 5
  template:
    metadata:
      labels:
        app: simplaks3
    spec:
      containers:
      - name: nopacr1
        image: nopacr1.azurecr.io/simplpdtv1
        ports:
        - containerPort: 80
      imagePullSecrets:
        - name: acr-auth

---
apiVersion: v1
kind: Service
metadata:
  name: simplaks3
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: simplaks3
```

Note the parameters

- **name:** `simplaks3` (this is just a random name you can decide on for the POD in AKS)
- `image: nopcr1.azurecr.io/simplpdtv1` (this the pointer to the Azure Container Registry and Docker images we pushed earlier. (and the same one we ran in Azure Container Instance)

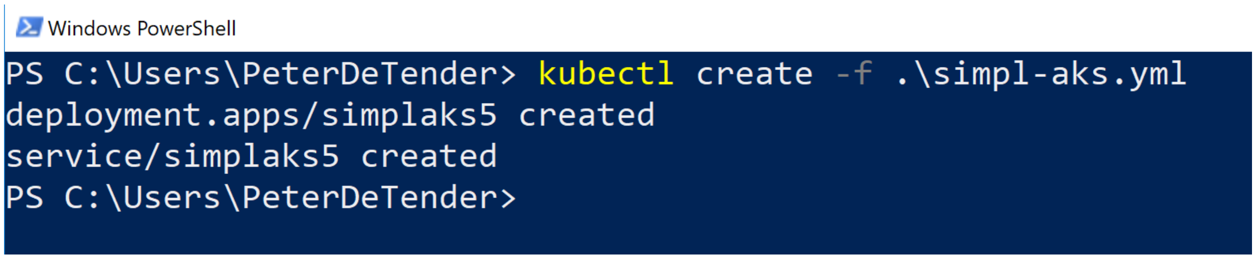
1. Replace the "simplaks3" names and other similar named variables with [SUFFIX]simplaks

2. Replace the "nopacr1.azurecr.io/simplpdtv1" name with the correct name of your ACR and Docker image name

3. Save the updated file.

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

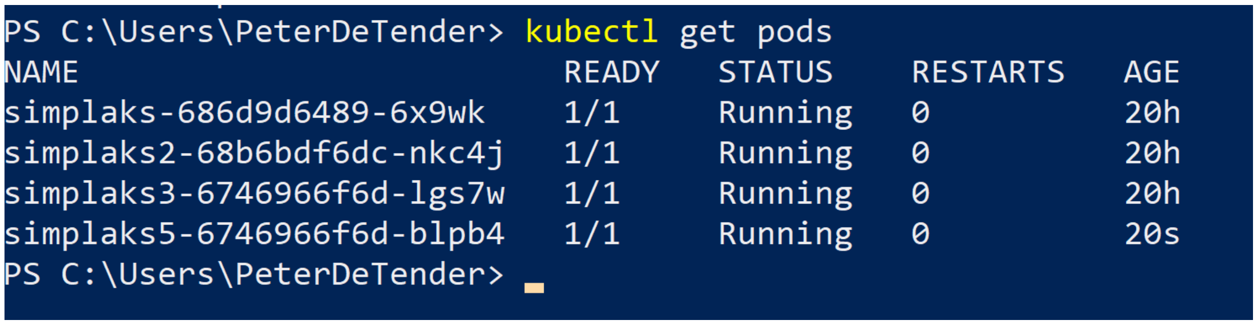
```
kubectl create -f "path to simp-aks.yml file here"
```



```
PS C:\Users\PeterDeTender> kubectl create -f .\simpl-aks.yml
deployment.apps/simplaks5 created
service/simplaks5 created
PS C:\Users\PeterDeTender>
```

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

```
kubectl get pods
```



```
PS C:\Users\PeterDeTender> kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
simplaks-686d9d6489-6x9wk   1/1     Running   0          20h
simplaks2-68b6bdf6dc-nkc4j  1/1     Running   0          20h
simplaks3-6746966f6d-lgs7w  1/1     Running   0          20h
simplaks5-6746966f6d-blpb4  1/1     Running   0          20s
PS C:\Users\PeterDeTender>
```

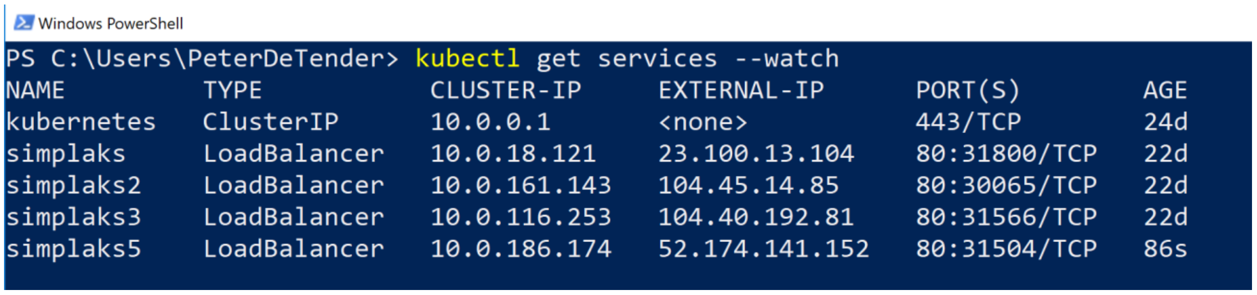
NOTE: if you should see an error message here, it is most probably related to not having defined the ACR authentication correctly

6. One can also **check the actual container service**, by running the following command:

   ```
   kubectl get service <name of POD you used in the yml file> --
   watch
   ```

   or checking for all running services:

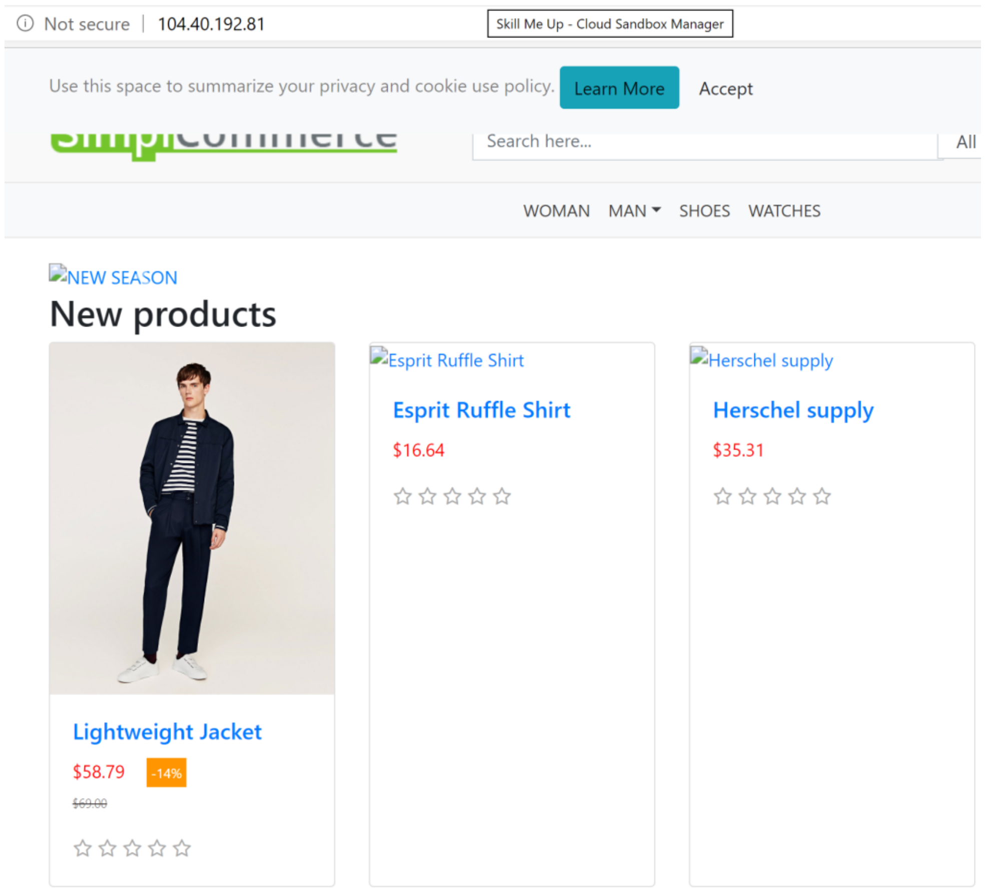
   ```
   kubectl get service –watch
   ```

   ```
   PS C:\Users\PeterDeTender> kubectl get services --watch
   NAME         TYPE          CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
   kubernetes   ClusterIP     10.0.0.1       <none>          443/TCP        24d
   simplaks     LoadBalancer  10.0.18.121    23.100.13.104   80:31800/TCP   22d
   simplaks2    LoadBalancer  10.0.161.143   104.45.14.85    80:30065/TCP   22d
   simplaks3    LoadBalancer  10.0.116.253   104.40.192.81   80:31566/TCP   22d
   simplaks5    LoadBalancer  10.0.186.174   52.174.141.152  80:31504/TCP   86s
   ```

7. **Wait for the service to receive an external-IP** address, which would mean the POD is fully up-and-running in AKS. From here, you could open your browser, and connect to the public IP address, revealing the e-commerce sample application!!

   Note this can take another few minutes before the app is actually fully loaded, no panic if it is not showing up immediately!



(using the Public IP-address for POD simplaks3 here as an example)

This confirms that our AKS service is fully operational, but the Docker container image that we pushed from the YAML-file settings, is also working correctly). Nice job!!

This completes the lab.

Summary

In this lab, you learned how to deploy Azure Kubernetes Services (AKS) using Azure CLI, as well as how to manage and validate the deployment using kubectl Kubernetes command line. Next, you configured RBAC and ACR authentication for a service principal. This was followed by the creation of a kubernetes.yml deployment file, having a pointer to the Azure Container Registry 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.