GENERAL NOTES:

This guideline uses Azure CLI to create resources in Azure

If haven’t installed the Azure CLI yet, visit:

<https://docs.microsoft.com/es-es/cli/azure/install-azure-cli?view=azure-cli-latest>

For Azure CLI commands in this guideline, when you see values between <>, replace this “tag” with the correct value; for example, replace <resourceGrouName> with Ibercaja if you’ve created a resource group called Ibercaja

Steps to deploy an asp.net core app into an AKS cluster:

1-To add Docker support after project creation, in Visual Studio 2017, right-click on the project node in Solution Explorer and select **Add** > **Docker support** on the context menu.

2-To complete adding Docker support, you can choose Windows or Linux. Select **Linux**, because AKS doesn’t support Windows Containers (it’s preview).

3-After running the project, you can list the images using the “docker images” command. You should see the image created by the automatic deployment of the project with Visual Studio 2017.

4-Upload the image to any Docker registry, like [Azure Container Registry (ACR)](https://azure.microsoft.com/services/container-registry/) or Docker Hub, so the images can be deployed to the AKS cluster from that registry:

4.1-Create an ACR: ac acr create --name <acrName> --resource-group <ResourceGroupName> --sku {Basic, Classic, Premium, Standard}

NOTE: select one kind for sku, for example, --sku Basic

4.2-Log in into the container registry: az acr login -n <registryName>

4.3-Get the login server name (usually <registryName>.azure.io): az acr list -g <resourceGroupName>

4.4-Tag the Docker image you want to push as follows: docker tag <imageName>/<tag1> <loginServerName>/<tag2>

NOTE 1:

To get <tag1>, run “docker images” command: get the tag information shown in the table.

<tag2>: it’s useful to control version of the image to call it the number of the image version alogn with the date it was created.

NTOE 2: You can verify the images now using docker image list.

4.5-Push the Docker image into your registry using: docker push <loginServer>/<tag2>

### 5-Create the image in Release mode: change to Release mode (selecting the option under the toolbar Team option) and run it.

### NOTE: by running “docker images” command, you should see both images: one for debug and another for release

### 6-Create a new Tag for the Image in Release mode, running the command of step 4.4 with the new values.

### NOTE: Each container image needs to be tagged with the loginServer name of the registry, as this tag is used for routing when pushing container images to an image registry.

### 7-Push the new image to ACR:

7.1-Log in into the container registry: az acr login -n <registryName>

### 7.2-Push the Release image to ACR: docker push ssampleacr.azurecr.io/mssampleaksapplication:v1

8-Create the AKS cluste:

8-1-Sign in Azure portal: portal.azure.com

8.2-Select “Create new resource” option

8.3-Select “Containers”, and, in Features, select “Create Kubernetes Service”

8.4-Fill out the basic values:

a) Enter the subscription where you want to create the cluster

b) Create a new resource group to host it or select an existing one

c) Enter a name for you cluster (it will set a default name for DNS)

d) Select the location of your choice

8.5-Select Create + review button, and after it’s been correctly validated, select Create again

9-To deploy the container into AKS cluster:

9.1-Creat a .yml deploy file that contains:

apiVersion: apps/v1beta1

kind: Deployment

metadata:

name: mssamplesbook

spec:

replicas: 1

template:

metadata:

labels:

app: mssample-kub-app

spec:

containers:

- name: mssample-services-app

image: mssampleacr.azurecr.io/mssampleaksapplication:v1

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: mssample-kub-app

spec:

ports:

- name: http-port

port: 80

targetPort: 80

selector:

app: mssample-kub-app

type: LoadBalancer

NOTE: this file content is from a example from the web. Replace its values with yours.

9.2-Get the credentials to the AKS cluster: az aks get-credentials --resource-group <resourceGroupName> --name <aksClusterName>

NOTE: I don’t know exactly why we need this step

9.3-Launch the deployment with Kubectl (command line of Kubernetes): kubectl create -f <deploymentFileName>.yml

NOTE: use Azure CLI to install kubectl, the command line of Kubernetes: az aks install-cli

10-When the deployment completes, you can access the Kubernetes console with a local proxy that you can temporally access with this command:

az aks browse --resource-group <resourceGroupame> --name <aksClusterName>

and accessing the URL <http://127.0.0.1:8001>

11-Now you have your application deployed on Azure, using a Linux Container, and an AKS Kubernetes Cluster. You can access your app browsing to the public IP of your service, which you can get from the Azure portal.

This information has been mainly obtained from the following link:

<https://docs.microsoft.com/en-us/dotnet/standard/containerized-lifecycle-architecture/design-develop-containerized-apps/build-aspnet-core-applications-linux-containers-aks-kubernetes>