How to deploy to Azure Kubernetes AKS a Web API .NET 8

https://github.com/luiscoco/SpringBoot_Sample5-deploy-WebAPI-to-Azure_Kubernetes_AKS/commits/main

0. Prerequisites

Install **Kubectl** command in Windows: https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/

Download and Install **Docker Desktop**: https://docs.docker.com/desktop/install/windows-install/

Install **Azure CLI**: https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-windows

1. Create .NET 8 WebAPI with Visual Studio 2022 Community Edition

Run Visual Studio 2022 Community Edition and create a new .NET 8 WebAPI.

For more infor see: https://github.com/luiscoco/Docker_Create_and_run_Image-_for_dotNET_8_Web_API

VERY IMPORTANT! Do not forget to enable Docker support

2. Create Azure Container Registry (ACR)

```
az account set --subscription "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

2.1. Login in to Azure:

```
az login
```

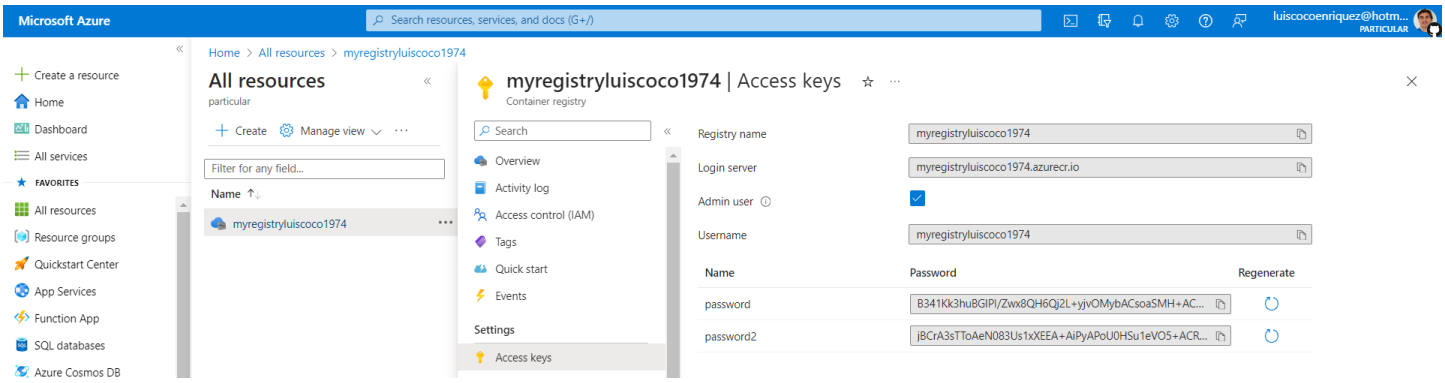
2.2. Create a ResourceGroup:

```
az group create --name myRG --location westeurope
```

2.3. Create an ACR instance (Note: only use lowercase letters for the ACR name):

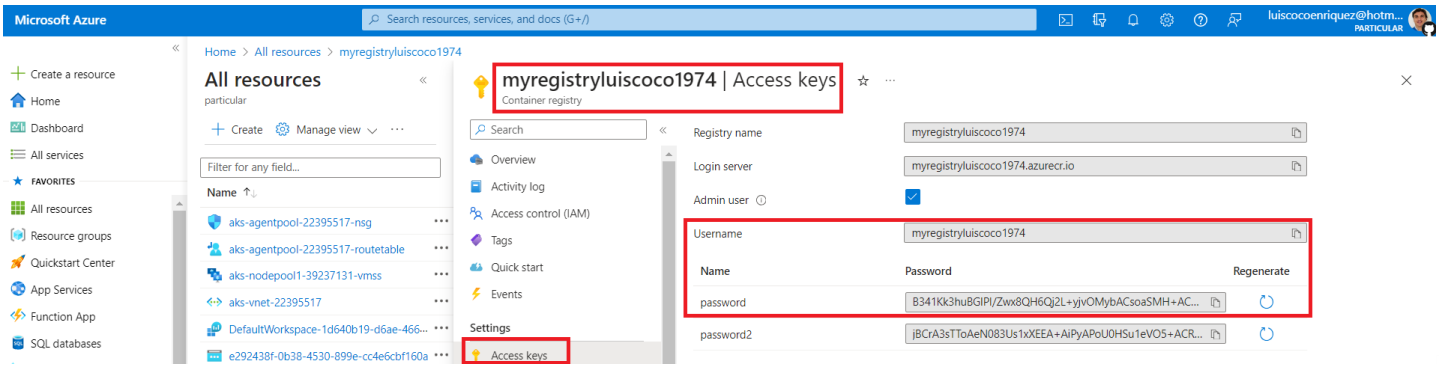
```
az acr create --resource-group myRG --name myregistryluiscoco1974 --sku Basic --location weste
```

2.4. Set the Admin user in the ACR and copy the username and password:



Verify you can log in to the Azure ACR with the Admin User credentials

```
docker login myregistryluiscoco1974.azurecr.io
```



3. Build and Push Docker image

3.1. Navigate to your project

```
cd /to/your/project
```

3.2. Log in to ACR:

```
az acr login --name myregistryluiscoco1974
```

NOTE: if you cannot enter with this command run again "az login" and try again running the command "az acr login --name myregistryluiscoco1974"

3.3. Build your Docker image:

Create a Dockerfile image:

```
#See https://aka.ms/customizecontainer to learn how to customize your debug container and how

FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER app
WORKDIR /app
EXPOSE 8080
EXPOSE 8081

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY [".NET 8 Web API Kubernetes.csproj", "."]
RUN dotnet restore "././.NET 8 Web API Kubernetes.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "./.NET 8 Web API Kubernetes.csproj" -c $BUILD_CONFIGURATION -o /app/build

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "./.NET 8 Web API Kubernetes.csproj" -c $BUILD_CONFIGURATION -o /app/publis

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", ".NET 8 Web API Kubernetes.dll"]
```

```
docker build -t myregistryluiscoco1974.azurecr.io/mywebapi:v1 .
```

3.4. Push the Image to ACR:

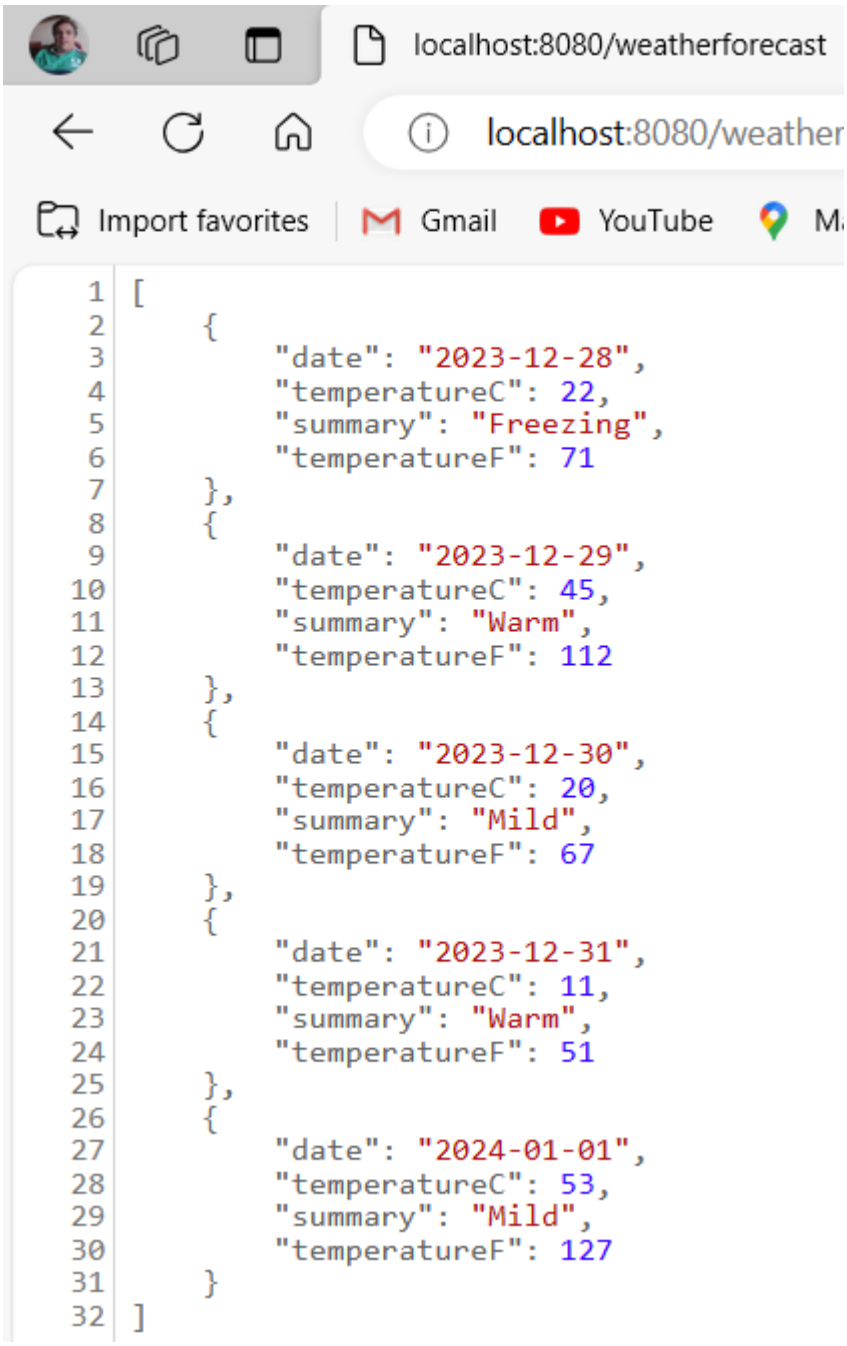
```
docker push myregistryluiscoco1974.azurecr.io/mywebapi:v1
```

3.5. How to run the Docker container in your local laptop

To run your WebAPI docker image stored in Azure ACR type this command:

```
docker run -d -p 8080:8080 myregistryluiscoco1974.azurecr.io/mywebapi:v1
```

Navigate to the WebAPI URL: http://localhost:8080/weatherforecast



4. Create Azure Kubernetes AKS Cluster

```
az aks create ^
--resource-group myRG ^
--name myAKSClusterluiscoco1974 ^
--node-count 1 ^
--enable-addons monitoring ^
--generate-ssh-keys ^
--attach-acr myregistryluiscoco1974 ^
--location westeurope
```

5. How to deploy.NET 8 WebAPI Docker image deploy to Azure AKS

Authenticate with Azure: Make sure you are logged in to Azure CLI and have access to the subscription and resources.

```
az login
```

Set the context to your AKS cluster: You need to get credentials for your AKS cluster and set the current context of kubectl to your cluster.

```
az aks get-credentials --resource-group <YourResourceGroup> --name <YourAKSClusterName>
```

```
az aks get-credentials --resource-group myRG --name myAKSClusterluiscoco1974
```

Replace and with your AKS resource group name and AKS cluster name, respectively.

Create a Kubernetes Secret for ACR authentication: This step is crucial for allowing your AKS cluster to pull images from your private Azure Container Registry.

```
az aks update -n <YourAKSClusterName> -g <YourResourceGroup> --attach-acr <YourACRName>
```

```
az aks update -n myAKSClusterluiscoco1974 -g myRG --attach-acr myregistryluiscoco1974
```

Replace with the name of your Azure Container Registry (without the domain), for example, myregistryluiscoco1974.

Deploy your application: You will need a Kubernetes manifest file to define your deployment and service.

This file typically is a YAML file that specifies the container image, ports, replicas, and other configurations.

Here's an example of a basic deployment YAML file (**deployment.yaml**):

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-dotnet-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-dotnet-app
  template:
    metadata:
```

```
        labels:
          app: my-dotnet-app
      spec:
        containers:
        - name: my-dotnet-app
          image: myregistryluiscoco1974.azurecr.io/mywebapi:v1
          ports:
          - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: my-dotnet-app-service
spec:
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 80  # The port the load balancer listens on
      targetPort: 8080  # The port the container accepts traffic on
  selector:
    app: my-dotnet-app
```

Replace mydotnetapp:latest with your image name and tag.

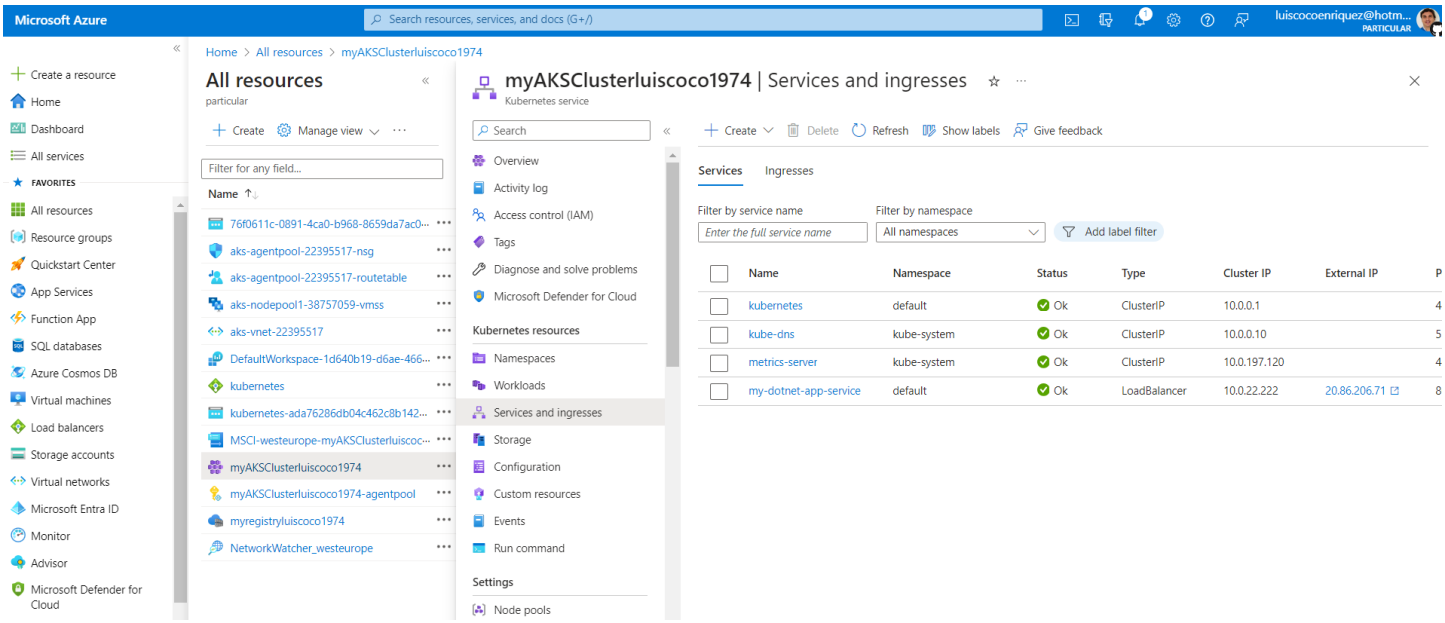
Deploy the application using the following command:

```
kubectl apply -f deployment.yml
```

```
kubectl get all
```

6. Access to the Web API endpoint

We navigate to the ResourceGroup "myRG", and Then we click in the Kubernetes service "myAKSClusterluiscoco1974":



We copy the Load Balancer External IP. In the internet web browser we input the Load Balancer External IP followed by the controller name "weatherforecast":