**Task : Deployment of ASP.NET Core API on AKS**

In this ASP.NET Core Kubernetes tutorial, I’ll walk through these steps:

## Creating the ASP.NET Core Project using Visual Studio 2022

* To create the sample project in Visual Studio, select **File** > **New** > **Project**, select the **Web** project type and then the **ASP.NET Core Web Api** template. You can also search for the template if you need it.
* Build a Docker image and push it to Docker hub.
* Build an ASP.NET application Image:

1. For building ASP .NET image we have to write Dockerfile for that.

* FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
* WORKDIR /app
* # Copy csproj and restore as distinct layers
* COPY SampleWebApp/\*.csproj .
* RUN dotnet restore
* # Copy everything else and build website
* COPY SampleWebApp/. .
* RUN dotnet publish -c release -o /WebApp --no-restore
* # Final stage / image
* FROM mcr.microsoft.com/dotnet/aspnet:5.0
* WORKDIR /WebApp
* COPY --from=build /WebApp ./
* ENTRYPOINT ["dotnet", "SampleWebApp.dll"]

1. After done writing a dockerfile we use docker cli commands to build it .

docker build –t name\_of\_image:tag {path\_of \_file}

1. After the build is done , its time to test working of this file, so for that follow this docker cli command.

docker run -p 5106:5106 ASPNETCORE\_URLS=http://+:5106 image\_name:tag

* Pushing this image to docker Hub:

.

1. In this tutorial I am using docker hub which is open source and free container image registry but we can also use azure container registry or Elastic container registry
2. For pushing our image to dockerhub we have to first tag this our image to our registry and then push to registry . so for that we use this command .

docker tag iamge\_name:tag dockerhub\_name/image\_name:tag

docker push dockerhub\_name/image\_name:tag

* Deployment to AKS:
* Create a resource group-

Create a resource group using the [az group create](https://learn.microsoft.com/en-us/cli/azure/group" \l "az-group-create) command.

az group create --name myResourceGroup --location eastus

* Create an AKS cluster-
* Create an AKS cluster using th e [az aks create](https://learn.microsoft.com/en-us/cli/azure/aks" \l "az-aks-create) command

az aks create -g myResourceGroup -n myAKSCluster --enable-managed-identity --node-count 1 --enable-addons monitoring --generate-ssh-keys

* Configure kubectl to connect to your Kubernetes cluster using the [az aks get-credentials](https://learn.microsoft.com/en-us/cli/azure/aks" \l "az-aks-get-credentials) command.

az aks get-credentials --resource-group myResourceGroup --name myAKSCluster

Kubectl get nodes. (check the cluster deployment by this command).

* Create a deployment .

1. Create and code a manifest file deployment.yaml file and use Kubectl commands for deployment purpose .
2. A [Kubernetes manifest file](https://learn.microsoft.com/en-us/azure/aks/concepts-clusters-workloads#deployments-and-yaml-manifests) defines a cluster's desired state, such as which container images to run.

Kubectl apply –f deployment.yaml (create deployment)

Kubectl get deployment (check your deployment).

Kubectl get pods (check your pods).

If pod in not working or it incountered an error . then use

kubectl describe pod name\_of\_pod (to see what is happening in the pod)

kubectl logs pod name\_of\_pod (to see the logs of the pod)

* Kubenetes service brief overview :
* A [Kubernetes service manifest file](https://learn.microsoft.com/en-us/azure/aks/concepts-clusters-workloads#deployments-and-yaml-manifests) defines how we going to expose our service to the outside the cluster
* When the application runs, a Kubernetes service exposes the application front end to the internet. This process can take a few minutes to complete
* Use a public standard load balancer in Azure Kubernetes Service (AKS):

A **public** load balancer integrated with AKS serves two purposes:

1. To provide outbound connections to the cluster nodes inside the AKS virtual network by translating the private IP address to a public IP address part of its *Outbound Pool*.
2. To provide access to applications via Kubernetes services of type LoadBalancer, enabling you to easily scale your applications and create highly available services.

* Specify the load balancer IP address

Adding the LoadBalancerIP property to the load balancer YAML manifest file.

* **Set service annotations**: Use service.beta.kubernetes.io/azure-load-balancer-ipv4 for an IPv4 address and service.beta.kubernetes.io/azure-load-balancer-ipv6 for an IPv6 address.
* **Add the LoadBalancerIP property to the load balancer YAML manifest**: Add the Service.Spec.LoadBalancerIP property to the load balancer YAML manifest. This field is deprecating following [upstream Kubernetes](https://github.com/kubernetes/kubernetes/pull/107235), and it can't support dual-stack. Current usage remains the same and existing services are expected to work without modification.
* Deploy the service manifest.

Deploy the public service manifest using [kubectl apply](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands" \l "apply) and specify the name of your YAML manifest.

kubectl apply -f public-svc.yaml

The Azure Load Balancer is configured with a new public IP that fronts the new service. Since the Azure Load Balancer can have multiple frontend IPs, each new service that you deploy gets a new dedicated frontend IP to be uniquely accessed.

Confirm your service is created and the load balancer is configured using the following command.

kubectl get service public-svc (OUTPOT)

NAMESPACE NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

default public-svc LoadBalancer 10.0.39.110 52.156.88.187

For more detailed information about your service, use the following command.

kubectl describe service public-svc