**Agenda: Working with AKS**

- Creating an AKS Cluster
- Writing Deployment and Service YAML files
- Deploying the Application using Kubectl
- Building a CI and CD Pipeline for Deploying to Kubernetes Cluster.

## Azure Kubernetes Service (AKS)

- AKS makes it quick and easy to deploy and manage containerized applications without container orchestration expertise.
- AKS reduces the complexity and operational overhead of managing Kubernetes by offloading much of that responsibility to Azure.
- As a hosted Kubernetes service, Azure handles critical tasks like provisioning, upgrading, scaling, health monitoring and maintenance for you.
- In addition, the service is free, you only pay for the agent nodes within your clusters, not for the masters.

**Steps Required for Running an Application in Kubernetes**

1. Package your Application in to one or more containers/images
2. Push those images to an image registry like Docker Hub or ACR
3. Post App Descriptor (YAML) to the Kubernetes API Server
    a. Scheduler schedules containers on available Worker Nodes
    b. Kubelet instructs Nodes to download Container Images
    c. Kubelet instructs Nodes to run the Containers

**Walkthrough:**

Step 1: Develop and Test the application locally

Step2: Deploy the docker images in ACR

Step3: Create Kubernetes Cluster

Step4: Run the application developed in step 1

**Step 1: Develop and Test the application locally**

1. Execute the following commands to create an ASP.NET Core MVC project.

```
D:\>md HelloWebApp
D:\>cd HelloWebApp
D:\HelloWebApp:\> dotnet new mvc
D:\HelloWebApp:\> Code .
```

2. Add the following docker file to the project

```
FROM mcr.microsoft.com/dotnet/core/aspnet:3.1-buster-slim AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/core/sdk:3.1-buster AS build
WORKDIR /src
COPY ["HelloWebApp.csproj", ""]
RUN dotnet restore "./HelloWebApp.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "HelloWebApp.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "HelloWebApp.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "HelloWebApp.dll"]
```

3. Build the docker image

   D:\HelloWebApp>**docker build** -t sandeepsoni/hellowebapp .

4. Test the image locally

   D:\HelloWebApp>**docker run** -p "8080:80" sandeepsoni/hellowebapp

   Visit: http://localhost:8080

**Step2: Create Container Registry and Push the Image**

5. Login to Azure

```
az login
az account set --subscription "Visual Studio Enterprise - SS1"
```

6. **Create Resource Group**

```
az group create --name DemoKRG --location eastus
```

7. **Create Azure Container Registry**

```
az acr create --resource-group DemoKRG --name dssaprregistry --sku Standard --location eastus
```

8. Goto ACR → Under **Admin user**, select **Enable**. Take note of the following values:

- Login server
- Username
- password

9. Local Machine: Login to ACR

```
docker login --username dssaprregistry --password oZ2+N4QW+DmhwDvLe5pDFF/9oMB0PS/r
dssaprregistry.azurecr.io
```

10. Tag Local Images

```
docker image tag sandeepsoni/hellowebapp dssaprregistry.azurecr.io/hellowebapp
```

11. Push images to ACR

```
docker push dssaprregistry.azurecr.io/hellowebapp
```

**Step3: Create Azure Kubernetes (AKS) Cluster**

12. Create **Azure Kubernetes Service** with associated **Service Principal**

```
az aks create --resource-group DemoKRG --name dssdemocluster --node-count 2 --generate-ssh-keys
```

After a few minutes, the command completes and returns JSON-formatted information about the cluster.

13. **Create Service Principal in Azure Active Directory. This will be used by Kubernetes Secret to pull images from ACR.**

    Azure Active Directory → App Registration → + Register App

    Name = **KubernetesServicePrincipal,**

    OK.

    Note the service principal ClientID and Secret.

14. Assign Role to Service Principal to read images from Azure Container Registry

    Go to Azure Contaner Registry → Access Control (IAM) → Add Role Assignment

    Role = AcrPull

    Service Principal = KubernetesServicePrincipal

    Assign.

15. **Connect to Kubernetes Cluster**

```
az aks get-credentials --resource-group DemoKRG --name dssdemocluster
```

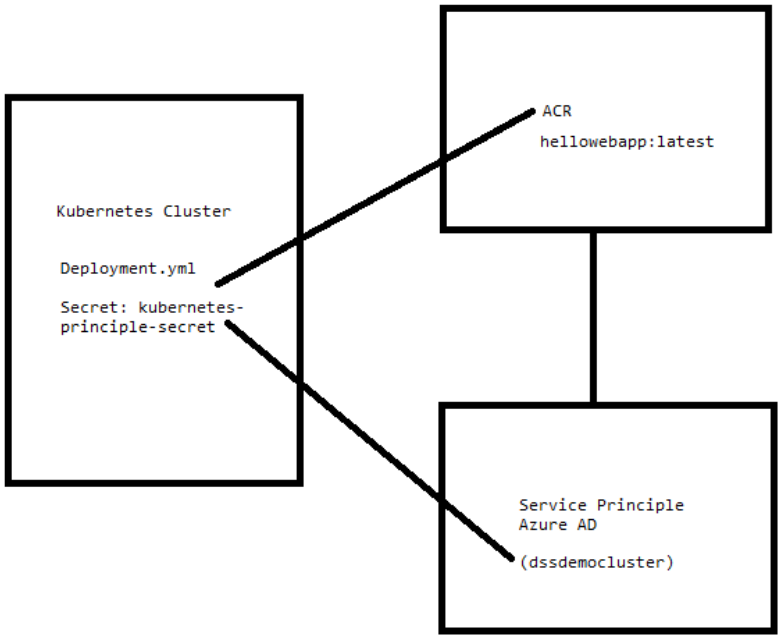
16. **Configure Kubernetes to use your ACR**

When creating deployments, replicasets or pods, kubernetes will try to use docker images already stored locally or pull them from the public docker hub. To change this, you need to specify the custom docker registry as part of your k8s object configuration (yaml or json).

Instead of specifying this directly in your configuration, we'll use the concept of k8s **secrets**. You decouple the k8s object from the registry configuration by just referencing the secret by it's name. But first, let's create a new k8s secret.

kubectrl create secret docker-registry <SECRET_NAME> --docker-server <REGISTRY_NAME>.azurecr.io --docker-email <YOUR_MAIL> --docker-username=<SERVICE_PRINCIPAL_ID> --docker-password <SERVICE_PRINCIPAL_SECRET>

**kubectl create secret** docker-registry **kubernetes-principle-secret** --docker-server dssaprregistry**.azurecr.io** --docker-username="<Service Principal CLIENTID>" --docker-password "<Service Pricipal Secret>"




Kubernetes manifest files define a desired state for a cluster, including what container images should be running.

**17. Create a file DeploymentAndService.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: hellowebapp
 labels:
  app: hellowebapp
spec:
 replicas: 1
 selector:
   matchLabels:
    app: hellowebapp
 template:
  metadata:
   labels:
```

```
      app: hellowebapp
  spec:
    containers:
    - name: hellowebapp
      image: dssdemo.azurecr.io/hellowebapp:latest
      imagePullPolicy: Always
      ports:
      - containerPort: 80
    imagePullSecrets:
    - name: kubernetes-principle-secret

---

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

18. Use the kubectl create command to run the application.

```
kubectl apply -f deployment.yaml
```

**Test the application**

19. To monitor progress, use the the below command.

    `kubectl get service azure-vote-front –watch`

    Once the *EXTERNAL-IP* address has changed from *pending* to an *IP address*, use `CTRL-C` to stop the kubectl watch process.

20. Open a web browser to the external IP address of your service

**Solution 1: Publish to Azure Kubernetes Service using Kubernetes**

1. Add the YML file to manifests folder of the Azure Repository

**/deployment.yaml**

**Edit the deployment.yaml file update the containers section as below**

```
- name: hellowebapp
  image: dssaprregistry.azurecr.io/helloworldapp:~~tag~~
  imagePullPolicy: Always
```

2. Create Service Connections
    a. Azure Kubernetes Cluster
        i. Authentication Method = Azure Subscription
        ii. Name=default
    b. Azure Container Registery

**3. Create a YAML Azure DevOps Pipeline**

```yaml
trigger:
- master

resources:
- repo: self

variables:
  # Container registry service connection established during pipeline creation
  dockerRegistryServiceConnection: 'DssAzureContainerRegistry'
  kubernetesServiceConnection: 'KubernetesServiceConnection'
  imageRepository: 'helloworldapp'
  containerRegistry: dssaprregistry.azurecr.io
  dockerfilePath: 'HelloWorldApp.Web/Dockerfile'
  tag: '$(Build.BuildId)'
  secretName: kubernetes-principle-secret
  vmImageName: 'ubuntu-latest'
  azureSubscriptionEndPoint: 'AzureServiceConnection-SS2'
  azureResourceGroup: "DemoKRG"
  kubernetesCluster: dssaprdemocluster1
stages:
- stage: Build
  displayName: Build and push stage
  jobs:
  - job: Build
    displayName: Build
    pool:
      vmImage: $(vmImageName)
    steps:
```

```
    - task: Docker@2
      displayName: Build and push an image to container registry
      inputs:
        containerRegistry: '$(dockerRegistryServiceConnection)'
        repository: '$(imageRepository)'
        command: 'buildAndPush'
        Dockerfile: '**/Dockerfile'
        buildContext: '.'
        tags: '$(tag)'

    - task: replacetokens@3
      inputs:
        targetFiles: 'HelloWorldApp.Web/Kubernetes/deployment.yaml'
        encoding: 'auto'
        writeBOM: true
        actionOnMissing: 'fail'
        keepToken: false
        tokenPrefix: '~~'
        tokenSuffix: '~~'
        useLegacyPattern: false
        enableTransforms: false
        enableTelemetry: true

    - task: CopyFiles@2
      inputs:
        SourceFolder: '.'
        Contents: 'HelloWorldApp.Web/Kubernetes/deployment.yaml'
        TargetFolder: '$(build.artifactstagingdirectory)'
    - task: PublishPipelineArtifact@1
      inputs:
        artifactName: 'manifests'
        path: '$(build.artifactstagingdirectory)'

- stage: Deploy
  displayName: Deploy stage
  dependsOn: Build
  jobs:
  - deployment: Deploy
    displayName: Deploy job
    pool:
      vmImage: $(vmImageName)
    environment: 'deccansoft'
    strategy:
      runOnce:
        deploy:
          steps:
```

```
          - task: DownloadPipelineArtifact@2
            inputs:
              artifactName: 'manifests'
              downloadPath: '$(System.ArtifactsDirectory)/manifests'

          - task: Kubernetes@1
            displayName: kubectl apply using configFile
            inputs:
              connectionType: 'Azure Resource Manager'
              azureSubscriptionEndpoint: $(azureSubscriptionEndPoint)
              azureResourceGroup: $(azureResourceGroup)
              kubernetesCluster: $(kubernetesCluster)
              command: 'apply'
              useConfigurationFile: true
              configuration: '$(System.ArtifactsDirectory)/manifests/HelloWorldAp
p.Web/Kubernetes/deployment.yaml'
              secretType: 'dockerRegistry'
              containerRegistryType: 'Azure Container Registry'
              azureSubscriptionEndpointForSecrets: $(azureSubscriptionEndPoint)
              azureContainerRegistry: $(containerRegistry)
              secretName: $(secretName)
```

**Note: YAML File names are case-sentive.**

## Solution 2: Publish to any Kubernetes Cluster using Kubernetes Service Account and KubernetesManifest Task

**Step 1: Create Kubernetes Service Connection**

Account Type = Service Account

1. Connect to the Kubernetes Cluster
    a. az aks get-credentials --resource-group DemoKRG --name dssaprdemocluster1
2. Server URL = Output of
    a. kubectl config view --minify -o jsonpath={.clusters[0].cluster.server}
3. Secret = Output of
    a. kubectl get serviceAccounts default -n default -o=jsonpath={.secrets[*].name}
4. Run the below command
    a. kubectl get secret <output from step3> -n default -o json
5. Service Connection Name = "Kubernetes Connection"
6. Save

**Step2 : Create a ClusterRoleBinding** which gives the role **cluster-admin** to the ServiceAccount **default in default namespace (default.default)**.

```
kubectl create clusterrolebinding serviceaccounts-cluster-admin -n kube-system --clusterrole=cluster-admin --serviceaccount=default:default
```

**Step 3: Update the YAML file as below:**

```
trigger:
- none

pool:
  vmImage: $(vmImageName)

variables:
  # Container registry service connection established during pipeline creation
  dockerRegistryServiceConnection: 'Docker ACR Connection'
  kubernetesServiceConnection: 'Kubernetes Connection'
  containerRegistry: 'dssdemo1.azurecr.io'
  dockerfilePath: '$(Build.SourcesDirectory)/dockerfile'
  tag: '$(Build.BuildId)'
  secretName: kubernetes-principle-secret
  vmImageName: 'ubuntu-latest'

steps:
- task: Docker@2
  displayName: Build and push an image to container registry
  inputs:
    command: buildAndPush
    repository: helloworldapp
    dockerfile: $(dockerfilePath)
    containerRegistry: $(dockerRegistryServiceConnection)
    tags: |
      $(tag)

- task: replacetokens@3
  inputs:
    targetFiles: 'deployment.yaml'
    encoding: 'auto'
    writeBOM: true
```

```
    actionOnMissing: 'warn'
    keepToken: false
    tokenPrefix: '~~'
    tokenSuffix: '~~'
    useLegacyPattern: false
    enableTelemetry: true

- task: KubernetesManifest@0
  displayName: Create secret
  inputs:
    action: createSecret
    secretType: dockerRegistry
    secretName: $(secretName)
    dockerRegistryEndpoint: $(dockerRegistryServiceConnection)
    kubernetesServiceConnection: $(kubernetesServiceConnection)

- task: KubernetesManifest@0
  displayName: Deploy
  inputs:
    kubernetesServiceConnection: $(kubernetesServiceConnection)
    manifests: deployment.yaml
    containers: |
      $(containerRegistry)/helloworldapp:$(tag)
    imagePullSecrets: |
      $(secretName)
```

Deploying a multi-container application to Azure Kubernetes Services

https://azuredevopslabs.com/labs/vstsextend/kubernetes/