**Q3 - SCENARIO**

A Toy Retail company ToyTrex has it retail application deployed as 3-tier application - Web App(UI), Web API(middle layer) and Database as Azure SQL.

The user load started increasing multiple fold every month and complex programs getting implemented, the application started performing poorly.

As a result, company decided to re-architect the middle layer as microservices using Azure Kubernetes Services.

The new architecture has below design decisions.

*1) The middle layer should be implemented as Microservices using Azure AKS*

*2) The middle layer API should be deployed as containerized application images*

*3) The container images will use Azure Container Repository (ACR) as the private image repository*

*4) The CI/CD pipelines for microservices should be implemented using Azure DevOps services.*

*5) The Azure DevOps should be able to access ACR and download the container images for microservices deployment*

*6) The image should be deployed as templates such as <image\_name>:<build\_id>*

*Explain the DevOps configuration and steps in detail for above requirements*

**Solution:**

Yes we can implement this scenario we need few resources like ACR we need to create an Azure Container Registry and Azure Kubernetes Cluster under Azure subscription.

We have to create a docker file based on the code that we are using for that API and have to pass the instructions correctly in that docker file to build a docker image.

Also we need to create a deployment.yaml file ( descriptor file ) we have to mention all the things like kind, containers, secrets, name of the image and image path ( acr path ) etc that are required to deploy on AKS cluster.

**CI Pipeline:**

We have to add the tasks docker build and docker push have to configure all the required details from azure subscription, Container registry type, Azure container registry name and the command which needs to be executed like build or push and the docker file path and <APIservice>:$(Build.BuildId).

Add a task to **copy the files to the artifact staging directory** that are included in the deployment folder.( have to fill the fields that are required )

Add another task that needs to be published ( have to fill the fields that are required which includes with a artifact name )

Below are few screenshots for reference we can do build and push an image in one task here I have used as two tasks for build and push an docker image and same fields we need to fill for oush an image task.

* **Sample Deployment.yaml file**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: <name>

spec:

  replicas: 1

  selector:

    matchLabels:

      app: <name>

  template:

    metadata:

      labels:

        app: <name>

    spec:

      containers:

        - name: <name>

          image: acrname.azurecr.io/user:$tag

          imagePullPolicy: Always

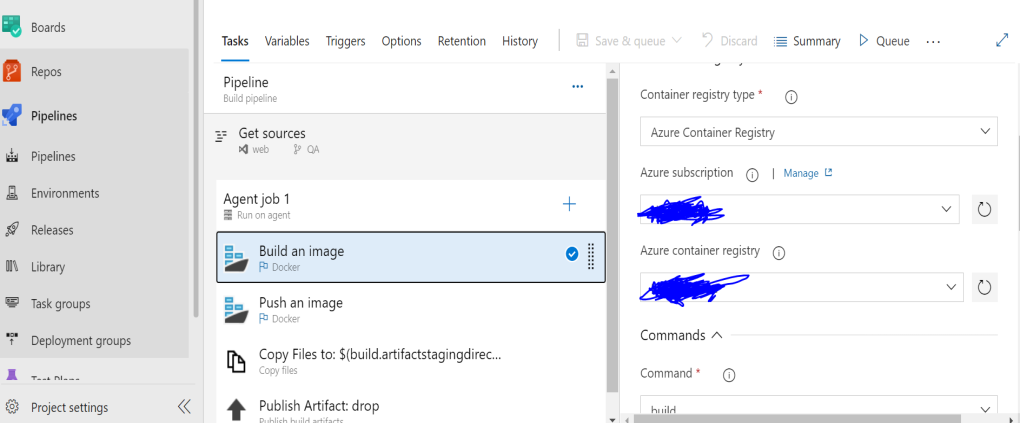
          ports:

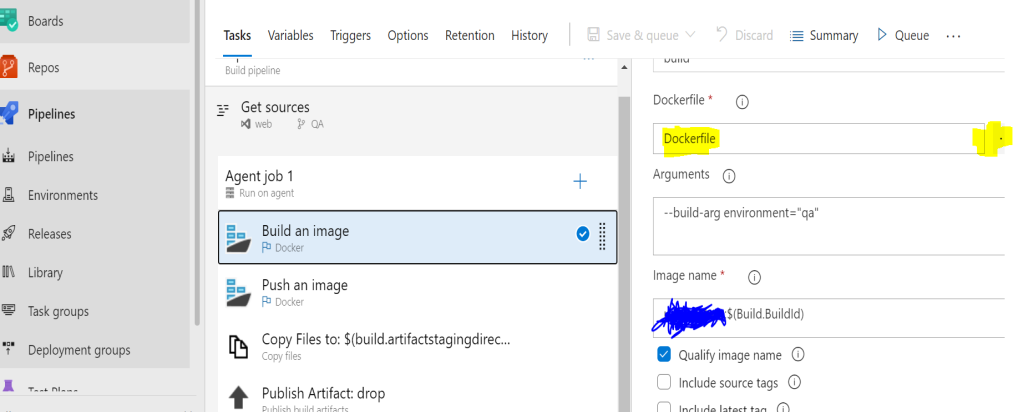
          - containerPort: <portnumber to be opened>

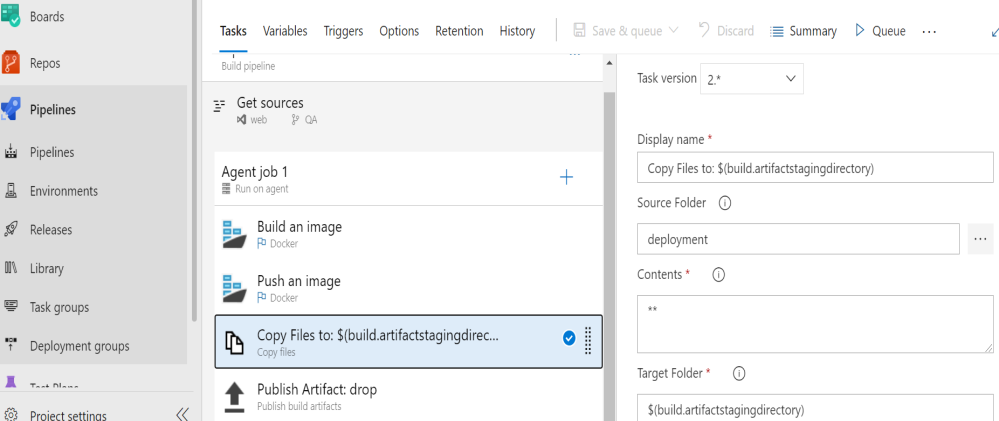
      imagePullSecrets:

        - name: mysecret

Copy the files to the directory and gave the path of deployment folder where the actual deployment.yaml exists and publishing the artifact as drop package. Which will deploy that package in release pipeline ( CD )





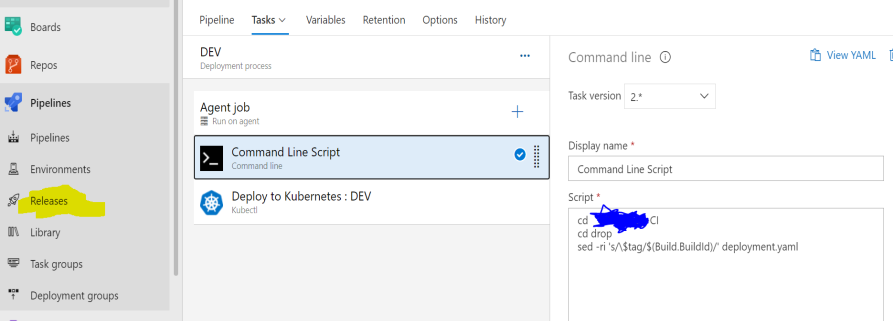


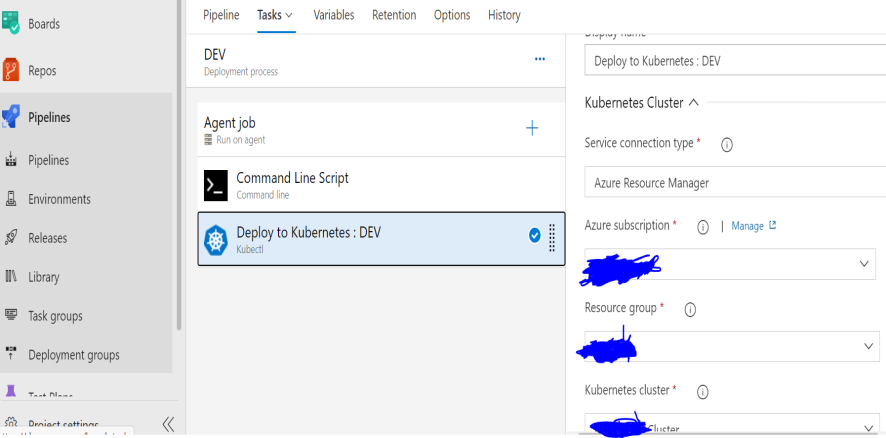
**CD Pipeline:**

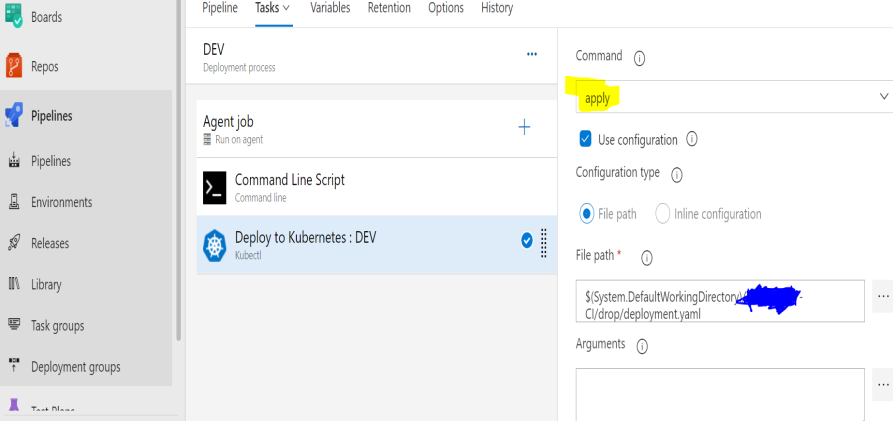
We need to add the artifact that we already published in the CI pipeline.  
We need to add **Kubectl** tasks and add the required fields like Azure subscription, AKS Cluster name, service connection type, resource group, command ( apply ) because we are executing and file path of your package.

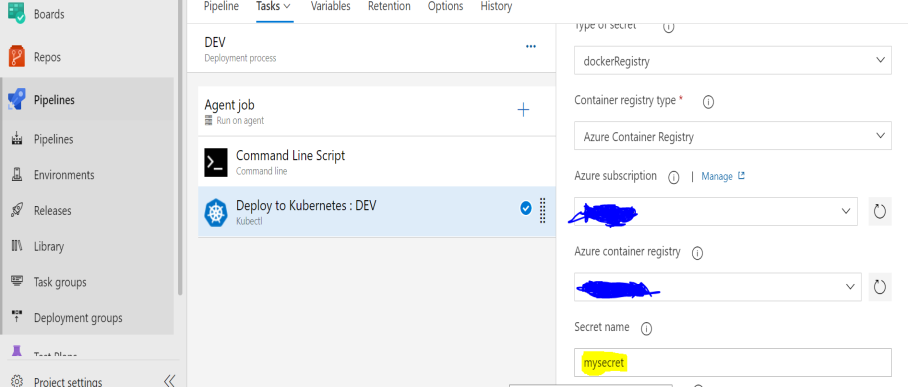
we have to fill the fields under secrets in that task, type of secret, registry type, container registry etc.,

After that we need to expose the service (APIservice) that we mentioned in the deployment descriptor file for that application, then we have to check the pods are running or not and then execute the command to get the service Kubectl get service -- watch. Once if you see the IP then we can check and browse with that ip whether our application got deployed or not.









1. Deploy Kubernetes to Azure, using CLI:

i. Get the latest available Kubernetes version in your preferred region into a bash variable. Replace <region> with the region of your choosing, for example eastus.

version=$(az aks get-versions -l <region> --query 'orchestrators[-1].orchestratorVersion' -o tsv)

ii. Create a Resource Group

az group create --name <name> --location <region>

iii. Create AKS using the latest version available

az aks create --resource-group <name> --name <unique-aks-cluster-name> --enable-addons monitoring --kubernetes-version $version --generate-ssh-keys --location <region>

1. Deploy Azure Container Registry(ACR): Run the below command to create your own private container registry using Azure Container Registry (ACR).

az acr create --resource-group <name> --name <unique-acr-name> --sku Standard --location <region>

1. Grant AKS-generated Service Principal access to ACR : Authorize the AKS cluster to connect to the Azure Container Registry using the AKS generated Service Principal. Replace the variables $AKS\_RESOURCE\_GROUP, $AKS\_CLUSTER\_NAME, $ACR\_RESOURCE\_GROUP with appropriate values below and run the commands.

# Get the id of the service principal configured for AKS

CLIENT\_ID=$(az aks show --resource-group $AKS\_RESOURCE\_GROUP --name $AKS\_CLUSTER\_NAME --query "servicePrincipalProfile.clientId" --output tsv)

# Get the ACR registry resource id

ACR\_ID=$(az acr show --name $ACR\_NAME --resource-group $ACR\_RESOURCE\_GROUP --query "id" --output tsv)

# Create role assignment

az role assignment create --assignee $CLIENT\_ID --role acrpull --scope $ACR\_ID

1. Once the release is complete, launch the [Azure Cloud Shell](https://docs.microsoft.com/en-in/azure/cloud-shell/overview) and run the below commands to see the pods running in AKS:

Type **az aks get-credentials --resource-group yourResourceGroup --name yourAKSname** in the command prompt to get the access credentials for the Kubernetes cluster. Replace the variables yourResourceGroup and yourAKSname with the actual values.

kubectl get pods ( It will display all the created pods )

The deployed web application is running in the displayed pods.

1. To access the application, run the below command. If you see that **External-IP** is pending, wait for sometime until an IP is assigned.

kubectl get service <service name> --watch ( Will get an external – IP )

1. Copy the **External-IP** and paste it in the browser and press the Enter button to launch the application.

* We can conclude that here we are dockerizing that image and deploying into the AKS cluster as a Microservice.