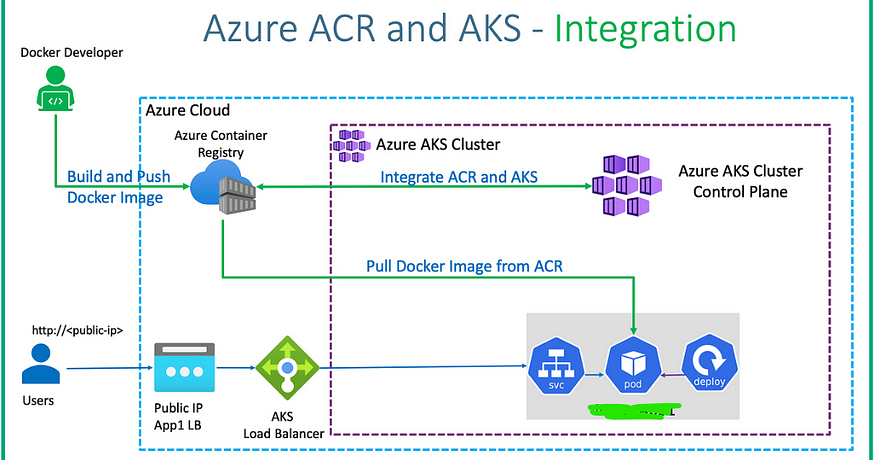
In this article, I explain the different methods to integrate ACR with AKS and given sample yaml files for a successful deployment. Also added the other requirements of service connections for the Devops pipeline.



**Introduction**

With this article I have tried to explain the integration of ACR and AKS and how to use ACR image for the AKS deployment/pods.

Apart from the AKS and ACR integration, we need few Azure Devops Service connections to enable Devops pipeline to push the docker images to registry and to do the deployment in AKS Cluster. I have covered this in the bottom of article.

This article does not contain steps for creating AKS and ACR. The purpose of this article is to demonstrate the requirements for AKS to pull the ACR image and requirements for the devops pipeline to pushimage to ACR and use that image for AKS deployment.

**Prerequisites**

1. Azure Kubernetes Cluster
2. Azure Container Registry
3. Sample Docker files to deploy App

**Types of methods to integrate ACR and AKS**

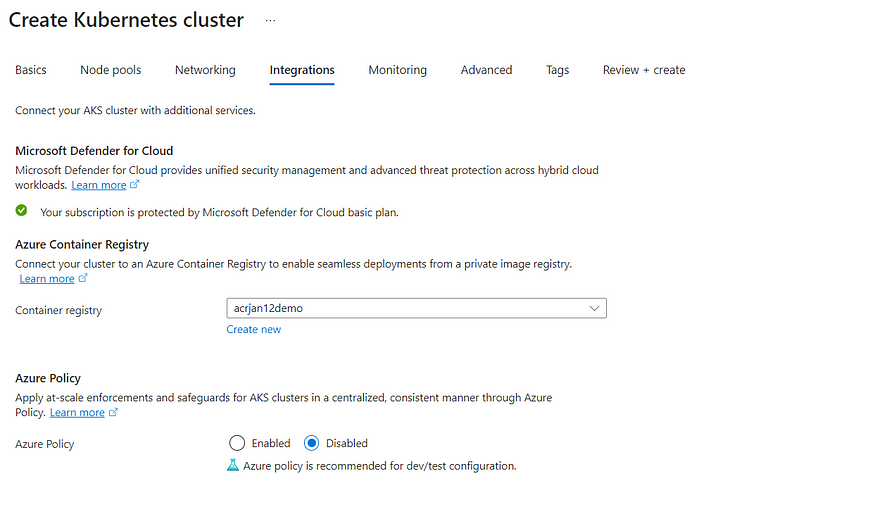
**Note:** With the first three methods mentioned below, it is worth noting that the Azure ID used to create cluster should have RBAC permission to do RBAC role assignments for the ACR (Either owner permission or user access administrator permission for the ACR)

There are different ways to integrate AKS with ACR and I have tried to articulate them as precise as possible.

First of all, we need to understand that when we say integration, what it means is, the AKS deployment or pod should be able to utilize/pull images which are stored in ACR. In other words, the AKS cluster should have permission to be able to pull images from ACR. There are different methods to achieve this.

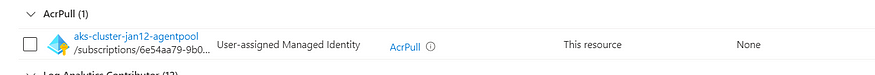
1. **Method1: Integrate during creation of AKS cluster**

During the creation of AKS cluster manually, there is an option to integrate the Azure Container Registry. Select the Azure Container Registry that is available in the subscription and create the cluster. With this option, there is no other extra configuration required for the AKS to be able to pull the ACR.



When the above option is enabled during cluster creation, it automatically creates a RBAC role “ACRPull” within the Azure Container Registry. This RBAC role provides “ACRPull” access to the managed identity of the AKS.

**Important Note:**Whenever AKS cluster is created, there are two managed identity (AKS cluster related) that gets created in Azure AD. ACR Pull permission is given to the agentpool managed identity as shown below. This is a role assignement that gets added inside the ACR.



**2. Method2: Integrate after creation of AKS Cluster**

Suppose we have not enabled the ACR integration during the creation of AKS cluster, we still have an option to enable integration using CLI. (Currently there seems to be no way to do this in portal after cluster creation)

To test, I have just detached the AKS and ACR using below command

**az aks update -n aks-cluster-jan12 -g ibo-rg — detach-acr acrjan12demo**

After detaching the ACR, could notice that the “ACRPull” RBAC permission got removed automatically

To integrate ACR with AKS again, I have used the below attach command

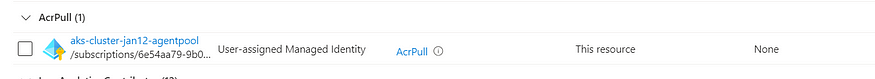
**az aks update -n aks-cluster-jan12 -g ibo-rg — attach-acr acrjan12demo**

After executing attach command, the “ACRPull” RBAC permission got added back

**3. Method3: Integration by adding the ServicePrincipal/ManagedIdentity**

As I mentioned in introduction that integration is nothing but enabling AKS pods to pull images from ACR, there is another simple way to do this after cluster creation without using the attach option.

We just manually need to add the “ACRPull” RBAC role to the Managed identity of AKS as per below screenshot. This role needs to be given inside the ACR and for the AKS managed identity (agentpool).



With this I could confirm that, after all, the integration job during cluster creation or the attach command is only doing the job of adding RBAC role to the managed identity of AKS. Lol!

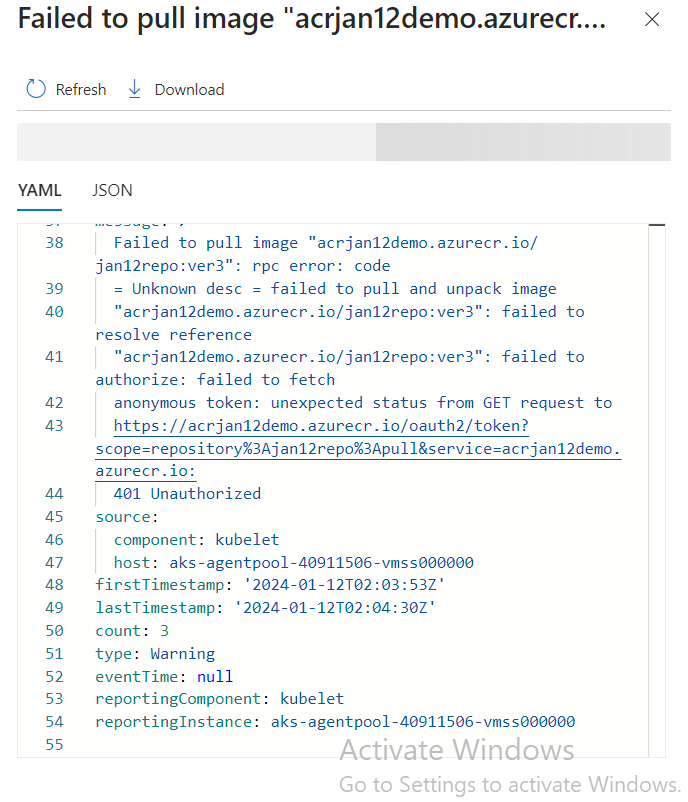
With the above methods, I was able to successfully buildandpush sample docker image to ACR and use that image for the AKS pod. Below is simple yaml that I used to achieve this via pipeline. This yaml pipeline is common for all three methods mentioned above and the deployment of AKS pod was successful using the yaml.

trigger:  
- none  
  
pool:  
 vmImage: ubuntu-latest  
  
variables:   
 acr\_repo: jan12repo  
 tags: |  
 ver3  
  
stages:  
 - stage: Build\_Docker\_Image\_Stage  
 displayName: Build\_Docker\_Image\_Stage  
 jobs:  
 - job: Build\_docker\_Image\_Job  
 displayName: Build\_Docker\_Image\_Job  
 steps:  
   
 - task: Docker@2  
 displayName: buildandpush  
 inputs:  
 containerRegistry: 'acr-jan12-manualsvc'  
 repository: $(acr\_repo)  
 command: 'buildAndPush'  
 Dockerfile: '$(Build.sourcesdirectory)/app/Dockerfile'  
 tags: $(tags)  
  
   
  
 - stage: kube\_deploy\_stage  
  
 jobs:  
   
 - job: kube\_deploy\_job  
 steps:   
 - task: KubernetesManifest@1  
 displayName: deploy\_app\_job  
 inputs:  
 action: 'deploy'  
 connectionType: 'azureResourceManager'  
 azureSubscriptionConnection: 'aks-jan12-manualsvc'  
 azureResourceGroup: 'ibo-rg'  
 kubernetesCluster: 'aks-cluster-jan12'  
 manifests: '$(build.sourcesdirectory)/aks-files/\*\*'

kubernetes deployment yaml:

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: app2-devops-deployment  
   
spec:  
 replicas: 4  
 selector:  
 matchLabels:  
 app: app2-devops  
 template:  
 metadata:  
 labels:  
 app: app2-devops  
 spec:  
 containers:  
 - name: app2-devops  
 image: acrjan12demo.azurecr.io/repo\_ver3:ver3  
  
 ports:  
 - containerPort: 80  
---  
apiVersion: v1  
kind: Service  
metadata:  
 name: app2-nginx-loadbalancer-service  
   
spec:  
 type: LoadBalancer  
 selector:  
 app: app2-devops  
 ports:  
 - port: 80  
 targetPort: 80

I would just like to mention that when the “ACRPull” RBAC role is removed for the AKS managed identity from the ACR, below is the error when we run the pipeline and when the pod tries to pull the ACR image, so it is very clear that this RBAC role is crucial for the AKS to pull image from ACR



**4. Method4: Integrate in the yaml using createsecret**

Without using above three methods and without adding any extra RBAC to the ACR, we would be able to achieve this after cluster creation using the yaml given below. This yaml has got another kubernetes task (createsecret) which integrates the AKS and ACR. As below, we have added an additional task in the yaml to createsecret and then call that secret in the deploy task (imagePullSecrets: ‘abctest’)

trigger:  
- none  
  
pool:  
 vmImage: ubuntu-latest  
  
variables:   
 acr\_repo: repo\_ver3  
 tags: |  
 ver3  
   
stages:  
 - stage: Build\_Docker\_Image\_Stage  
 displayName: Build\_Docker\_Image\_Stage  
 jobs:  
 - job: Build\_docker\_Image\_Job  
 displayName: Build\_Docker\_Image\_Job  
 steps:  
   
 - task: Docker@2  
 displayName: buildandpush  
 inputs:  
 containerRegistry: 'acr-jan12-manualsvc'   
 repository: $(acr\_repo)  
 command: 'buildAndPush'  
 Dockerfile: '$(Build.sourcesdirectory)/app/Dockerfile'  
 tags: $(tags)  
  
   
   
 - stage: kube\_deploy\_stage  
 jobs:   
 - job: kube\_deploy\_job  
  
 steps:  
 - task: KubernetesManifest@1  
 displayName: secret\_creation\_job  
   
 inputs:  
 action: 'createSecret'  
 connectionType: 'azureResourceManager'  
 azureSubscriptionConnection: 'aks-jan12-manualsvc'  
 azureResourceGroup: 'ibo-rg'  
 kubernetesCluster: 'aks-cluster-jan12'  
 secretType: 'dockerRegistry'  
 secretName: 'abctest'  
 dockerRegistryEndpoint: 'acr-jan12-manualsvc'  
  
 - task: KubernetesManifest@1  
 displayName: deploy\_app\_job  
 inputs:  
 action: 'deploy'  
 connectionType: 'azureResourceManager'  
 azureSubscriptionConnection: 'aks-jan12-manualsvc'  
 azureResourceGroup: 'ibo-rg'  
 kubernetesCluster: 'aks-cluster-jan12'  
 manifests: '$(build.sourcesdirectory)/aks-files/\*\*'  
 imagePullSecrets: 'abctest'

The above steps are required for the AKS to pull image from ACR and complete a successful integration

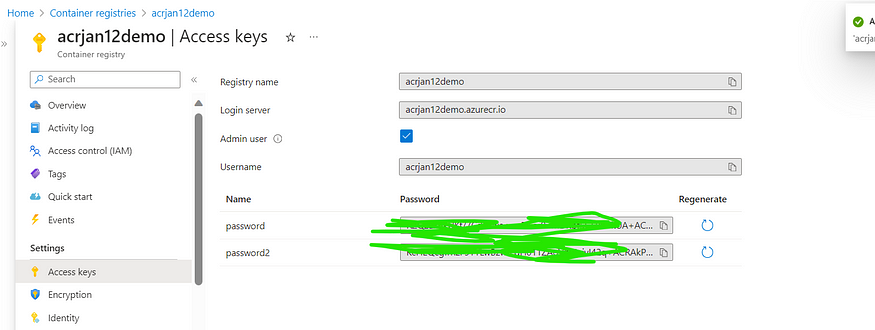
**Other Requirements for successful pipeline**

In addition to the above integration, we need to create two Azure Devops service connections for the Azure Containter Registry and Azure Kubernetes cluster. This is required for the Devops pipeline to have permissions to push image to ACR and to do deployments for the AKS.

**Devops Service Connection for ACR:**

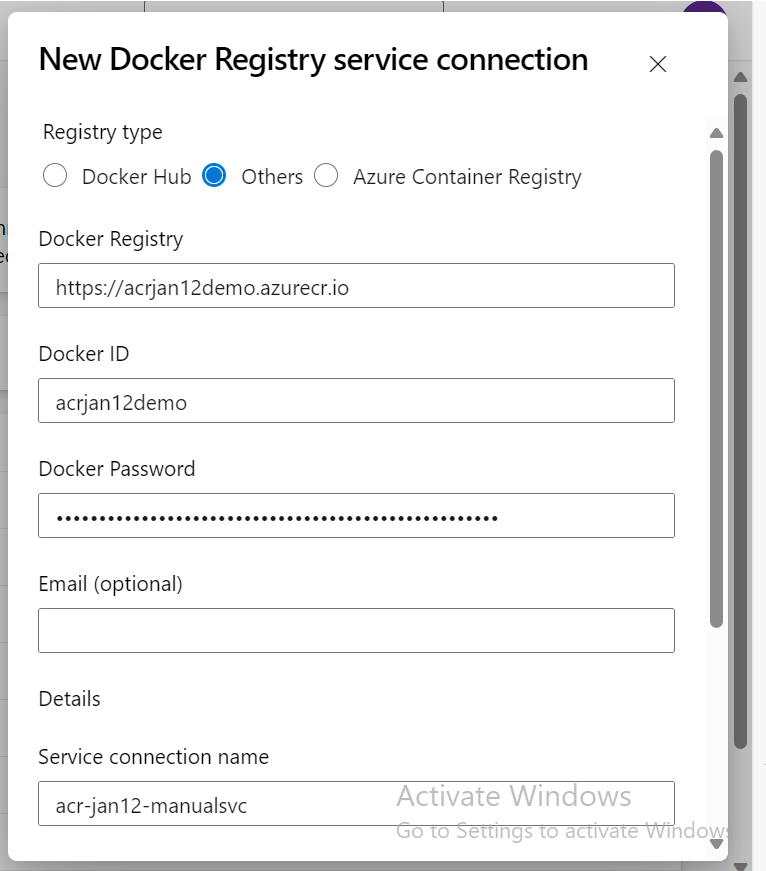
Steps:

1. Enable Access keys in the ACR



2. We need to create a Service Connection in Azure Devops, fill the details that we got from the previous step and save.

(No additional RBACS Required)



**Devops Service Connection for AKS :**

Steps:

1. Create an App registration in Azure AD and create secret for the APP. I have created an app named “aks-jan12-manualsvc”
2. This App should have the below RBAC role in the AKS, otherwise pipeline wont be able to do the deployment and will fail. After we create the APP in Azure AD, we need to add this RBAC role in the AKS cluster for the APP.

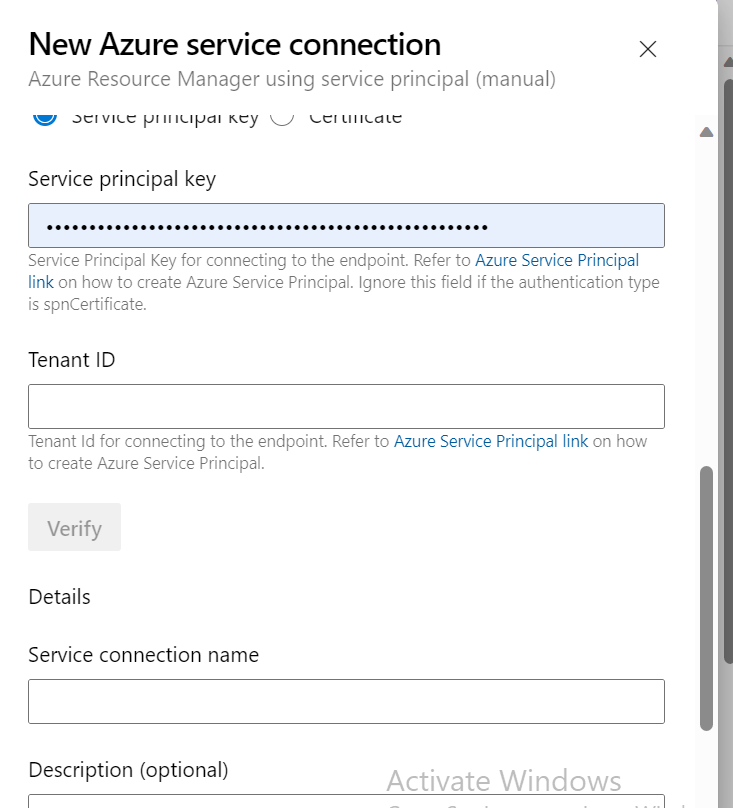


2. Get the client id and the secret of the app registration

3. Create new Azure Devops service connection

Azure Devops portal →New service connection →Azure Resource Manager → Service principal (Manual)

Enter all details obtained from Azure AD and verify



After creating the Devops service connections, the yaml will look like below which has got the service connections

(just to avoid confusions — This yaml is same as the yaml I have given above in the AKS -ACR integration, Just put this here to show the role of service connection)

trigger:  
- none  
  
pool:  
 vmImage: ubuntu-latest  
  
variables:   
 acr\_repo: repo\_ver3  
 tags: |  
 ver3  
   
stages:  
 - stage: Build\_Docker\_Image\_Stage  
 displayName: Build\_Docker\_Image\_Stage  
 jobs:  
 - job: Build\_docker\_Image\_Job  
 displayName: Build\_Docker\_Image\_Job  
 steps:  
   
 - task: Docker@2  
 displayName: buildandpush  
 inputs:  
 containerRegistry: 'acr-jan12-manualsvc' #Devops Service Connection for ACR   
 repository: $(acr\_repo)  
 command: 'buildAndPush'  
 Dockerfile: '$(Build.sourcesdirectory)/app/Dockerfile'  
 tags: $(tags)  
  
   
   
 - stage: kube\_deploy\_stage  
 jobs:   
 - job: kube\_deploy\_job  
  
 steps:  
 - task: KubernetesManifest@1  
 displayName: secret\_creation\_job  
   
 inputs:  
 action: 'createSecret'  
 connectionType: 'azureResourceManager'  
 azureSubscriptionConnection: 'aks-jan12-manualsvc' #Devops Service Connection for AKS :  
 azureResourceGroup: 'ibo-rg'  
 kubernetesCluster: 'aks-cluster-jan12'  
 secretType: 'dockerRegistry'  
 secretName: 'abctest'  
 dockerRegistryEndpoint: 'acr-jan12-manualsvc' #Devops Service Connection for ACR  
  
 - task: KubernetesManifest@1  
 displayName: deploy\_app\_job  
 inputs:  
 action: 'deploy'  
 connectionType: 'azureResourceManager'  
 azureSubscriptionConnection: 'aks-jan12-manualsvc'#Devops Service Connection for AKS :  
 azureResourceGroup: 'ibo-rg'  
 kubernetesCluster: 'aks-cluster-jan12'  
 manifests: '$(build.sourcesdirectory)/aks-files/\*\*'  
 imagePullSecrets: 'abctest'

With this we come to an end. I have tried to clear confusions as much as possible. Hope this helps someone.

4