Kubernetes Application Deployment

I. Overview

- This guide demonstrates how to deploy an application to **Azure Kubernetes Service (AKS)** using an **Azure DevOps pipeline**.
- In this demo (my repository), we'll use the provided repository as a starting point and extend it with a few additional components to align with the **assignment requirements**.

II. Implement

1. Update StorageClass for Mongodb

We need to replace current AWS provisioner to Azure provisioner

From **./mongodb.yaml** replace section StorageClass to:

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: managed-premium-retain
provisioner: disk.csi.azure.com
parameters:
  skuName: Standard_LRS
reclaimPolicy: Retain
volumeBindingMode: WaitForFirstConsumer
```

```
allowVolumeExpansion: true
```

And replace **volumeClaimTemplates** property referred to the right StorageClass:

```
  volumeClaimTemplates:
  - metadata:
      name: data-volume
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "managed-premium-retain"
      resources:
        requests:
          storage: 1Gi
```

2. Config Ingress for Grafana

From root of repository add ingress-grafana.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-grafana
  namespace: monitoring
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
      - path: /grafana(/|$)(.*)
        pathType: ImplementationSpecific
        backend:
          service:
            name: kube-prometheus-stack-grafana
            port:
              number: 80
```

Then we can access Grafana UI via <IP>/grafana

3. Update image URL for backend and frontend

From ./backend.yaml and ./frontend.yaml update image properties to

```
        image: ${REGISTRY}/backend:latest
```

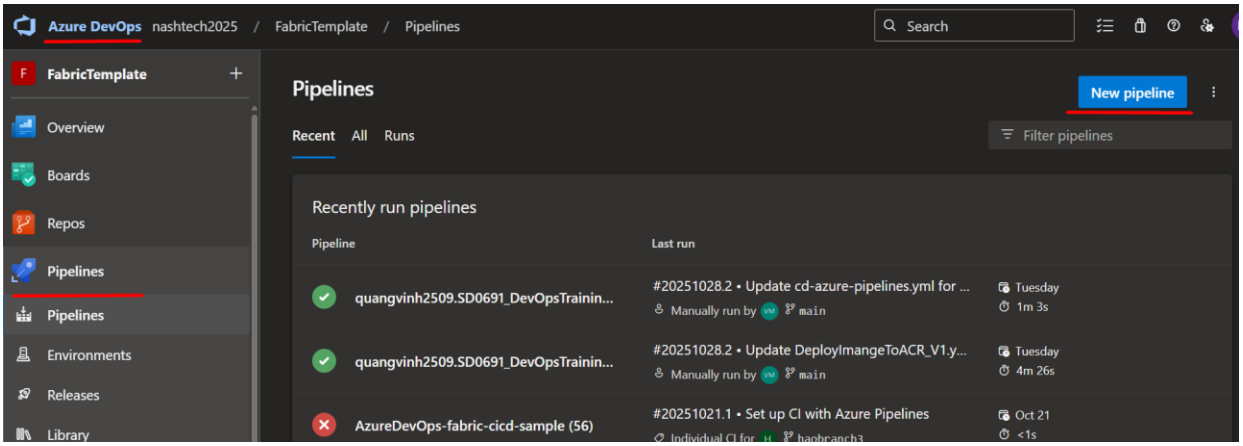
And

```
        image: ${REGISTRY}/frontend:latest
```

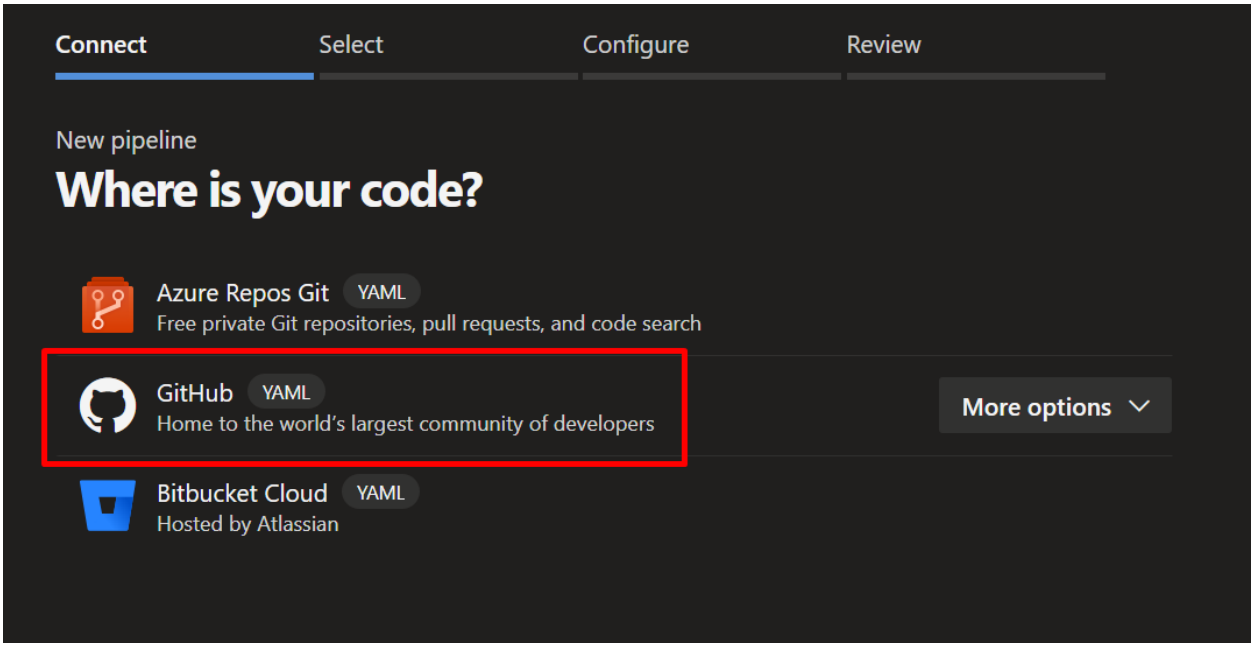
4. CI/CD Azure DevOps pipeline

Access Azure DevOps UI

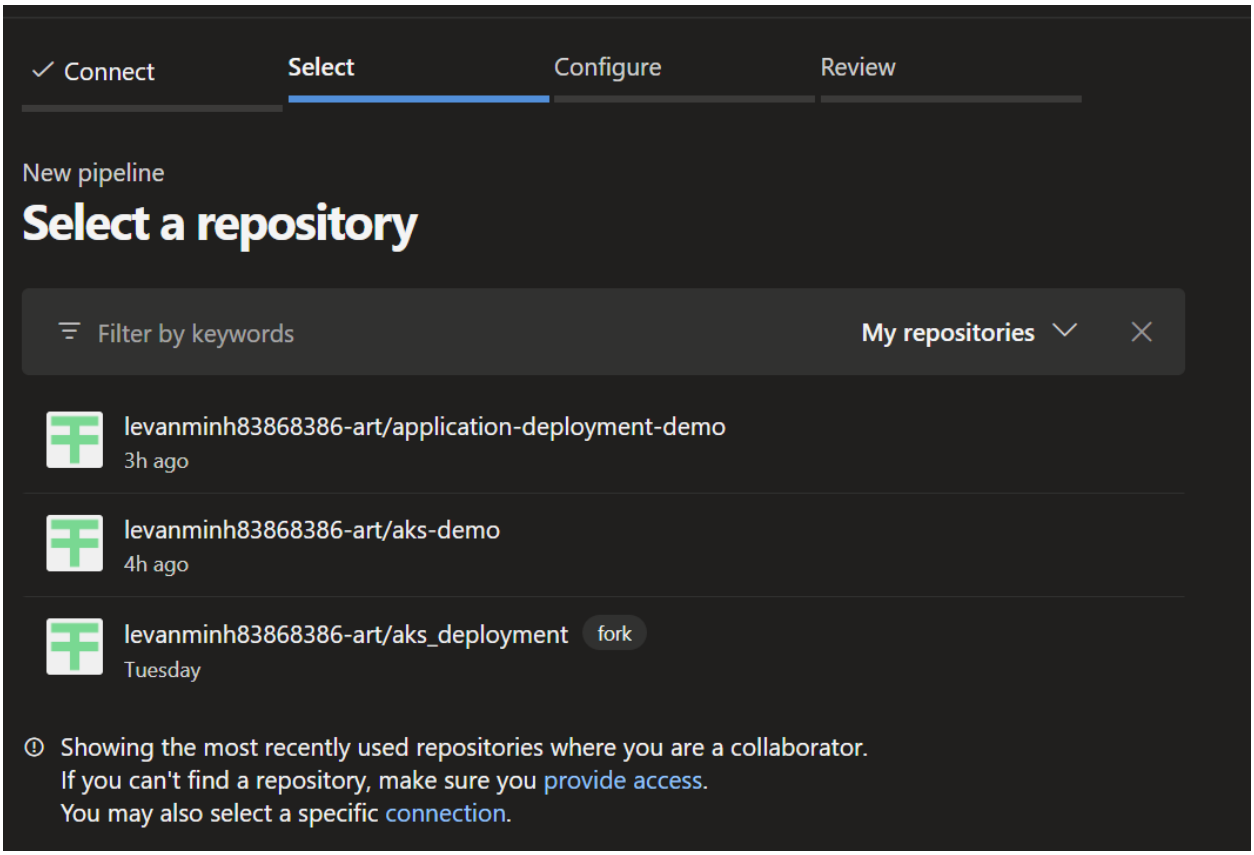
Pipelines => New pipeline



Choose GitHub where store the code



Choose repository



Define CI/CD process code:

```
trigger: none

pool:
  vmImage: 'ubuntu-latest'

variables:
  azureSubscription: 'minhAzureServiceConnection'      # your Azure DevOps
service connection
  aksResourceGroup: 'myTFResourceGroup'    # resource group of AKS
  aksCluster: 'minh-test-aks'             # AKS cluster name
  namespace: 'dev'
  containerRegistry: 'minhtestacr.azurecr.io'
  containerRegistryName: 'minhtestacr'
  deploymentContextPath: '.'
  tag: 'latest'


jobs:
- job: CI
```

```
  steps:
  - task: AzureCLI@2
    displayName: 'Build and Push Image to ACR'
    inputs:
      azureSubscription: '$(azureSubscription)'
      scriptType: 'bash'
      scriptLocation: 'inlineScript'
      inlineScript: |
        echo  "🚀 Starting backend image build and push..."
        az acr build --registry  $(containerRegistryName) --image
backend:latest $(deploymentContextPath)/src/backend/
        echo "☑ Backend image build and push completed."
        echo  "🚀 Starting frontend image build and push..."
        az acr build --registry  $(containerRegistryName) --image
frontend:latest $(deploymentContextPath)/src/frontend/
        echo "☑ Frontend image build and push completed."
```

This **CI job** automates building and pushing Docker images for both the backend and frontend applications to **Azure Container Registry (ACR)** using the `AzureCLI@2` task.

It runs inline Bash scripts to build images directly from the specified source paths and tags them as `latest`. This ensures that the most recent versions of both services are available for deployment.

```
- job: CD
  dependsOn: CI
  steps:
  # Connect to AKS
  - task: AzureCLI@2
    displayName: 'Connect to AKS'
    inputs:
      azureSubscription: '$(azureSubscription)'
      scriptType: 'bash'
      scriptLocation: 'inlineScript'
      inlineScript: |
        az aks get-credentials --resource-group $(aksResourceGroup) --name
$(aksCluster) --overwrite-existing
        kubectl config current-context

  # Create namespace if not exists
  - script: |
      kubectl get namespace $(namespace) || kubectl create namespace
$(namespace)
      kubectl config set-context --current --namespace=$(namespace)
```

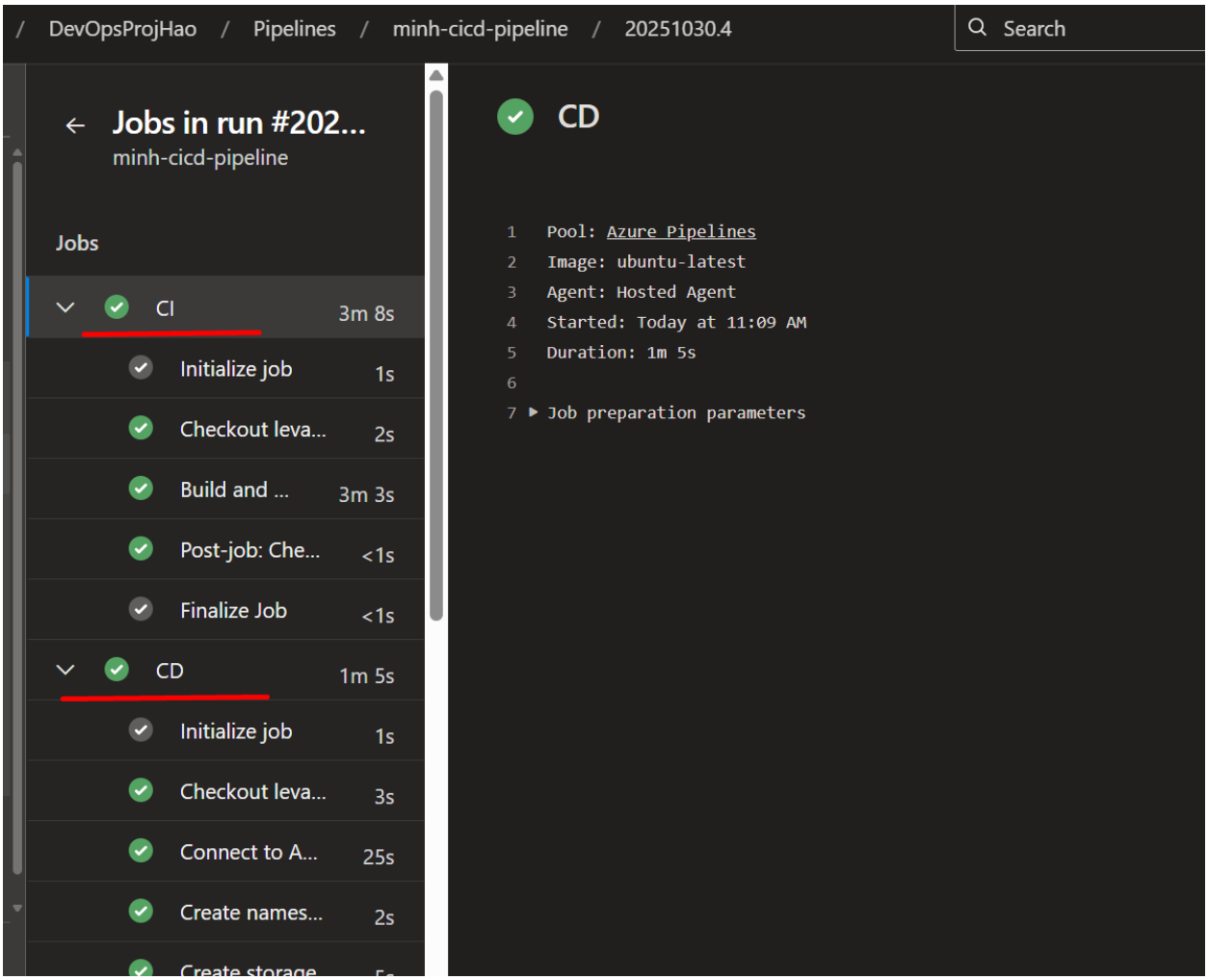
```
  displayName: 'Create namespace if not exists'

# Database
- script: |
    echo "Create storageclass, pvc and database"
    kubectl apply -f $(deploymentContextPath)/mongodb.yaml
  displayName: 'Create storageclass, pvc and database'

- script: |
    echo -e "\nverify storage class"
    kubectl get sc

    echo -e "\nverify pvc"
    kubectl get pvc

    echo -e "\nverify StatefulSet"
    kubectl get sts

    echo -e "\nverify database pods"
    kubectl get pod

    echo -e "\nverify database service"
    kubectl get service

    echo -e "\nverify configmap"
    kubectl get configmap

    echo -e "\ncheck configmap details"
    kubectl describe configmap mongo-config

    echo -e "\nverify secret"
    kubectl get secret

    echo -e "\ncheck secret details"
    kubectl get secret mongo-secrets -o yaml

  displayName: 'Verify storageclass, pvc and database'

# Backend
- script: |
    echo "Using registry: $(containerRegistry)"
    echo "Namespace: $(namespace)"
```

```
      echo -e "\nDeploying backend..."
      export REGISTRY=$(containerRegistry)
      envsubst < $(deploymentContextPath)/backend.yaml | kubectl apply -f
-
    displayName: 'Deploy backend image to AKS'

  - script: |
      echo "verify backend pods"
      kubectl get pods

      echo -e "\nverify backend service"
      kubectl get svc
    displayName: 'Verify backend deployment'

  # Frontend
  - script: |
      echo "Using registry: $(containerRegistry)"
      echo "Namespace: $(namespace)"

      echo -e "\nDeploying frontend..."
      export REGISTRY=$(containerRegistry)
      envsubst < $(deploymentContextPath)/frontend.yaml | kubectl apply -f
-
    displayName: 'Deploy frontend image to AKS'

  - script: |
      echo "verify frontend pods"
      kubectl get pods

      echo -e "\nverify frontend service"
      kubectl get svc
    displayName: 'Verify frontend deployment'

  # Ingress
  - script: |
      echo "install NGINX ingress controller"
      kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-
v1.7.1/deploy/static/provider/cloud/deploy.yaml
    displayName: 'Application Ingress'
```

```
  - script: |
      echo "verify ingress controller installation"
      kubectl get pods --namespace=ingress-nginx | grep nginx

      echo -e "\ninstall application ingress"
      kubectl apply -f $(deploymentContextPath)/ingress.yml

      echo -e "\nverify application ingress"
      kubectl get ingress -o wide

    displayName: 'Verify ingress'



  # Grafana ingress
  - script: |
      envsubst < $(deploymentContextPath)/ingress-grafana.yaml | kubectl
apply -f -
    displayName: 'Config Ingress for Grafana'
```

CD job above helps deliver latest application version to AKS via kubectl apply command

Save and run the pipeline



5. Verify

Get all resource from AKS

Switch to Bash Restart Manage files New session Editor Web preview

```
PS /home/minh> kubectl get all -n dev
NAME                                READY   STATUS    RESTARTS   AGE
pod/backend-bf9bddc8c-ss5rr         1/1     Running   0          3h21m
pod/frontend-784dbb9896-5qnd2       1/1     Running   0          3h21m
pod/mongo-0                         1/1     Running   0          3h21m

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
service/backend      ClusterIP   10.0.8.142      <none>        3000/TCP    3h21m
service/frontend     ClusterIP   10.0.226.91     <none>        3000/TCP    3h21m
service/mongo        ClusterIP   10.0.158.123    <none>        27017/TCP   3h21m

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/backend      1/1     1            1           3h21m
deployment.apps/frontend     1/1     1            1           3h21m

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/backend-bf9bddc8c       1         1         1       3h21m
replicaset.apps/frontend-784dbb9896     1         1         1       3h21m

NAME                       READY   AGE
statefulset.apps/mongo     1/1     3h21m
PS /home/minh>
```

Get ingress

```
PS /home/minh> kubectl get ingress -n dev
NAME               CLASS   HOSTS   ADDRESS           PORTS   AGE
ingress-frontend   nginx   *       132.164.150.166   80      3h23m
PS /home/minh>
```

Use this public ip to access frontend application

Check Grafana ingress

```
PS /home/minh> kubectl get ingress -n monitoring
NAME                CLASS   HOSTS   ADDRESS           PORTS   AGE
ingress-grafana     nginx   *       132.164.150.166   80      3h26m
PS /home/minh>
```

Now we can also access Grafana ui via 132.164.150.166/grafana