

I.	Overview	1
II.	Implement.....	1
1.	Update StorageClass for Mongodb.....	1
2.	Config Ingress for Grafana	1
3.	Update image URL for backend and frontend	2
4.	CI/CD Azure DevOps pipeline	2
5.	Verify	9

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:

```
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
```

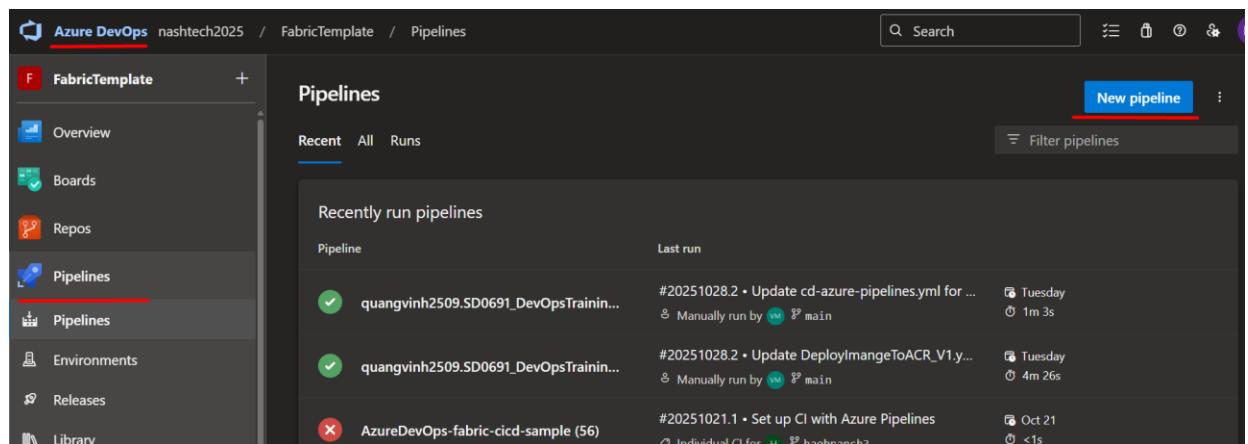
And

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

4. CI/CD Azure DevOps pipeline

Access Azure DevOps UI

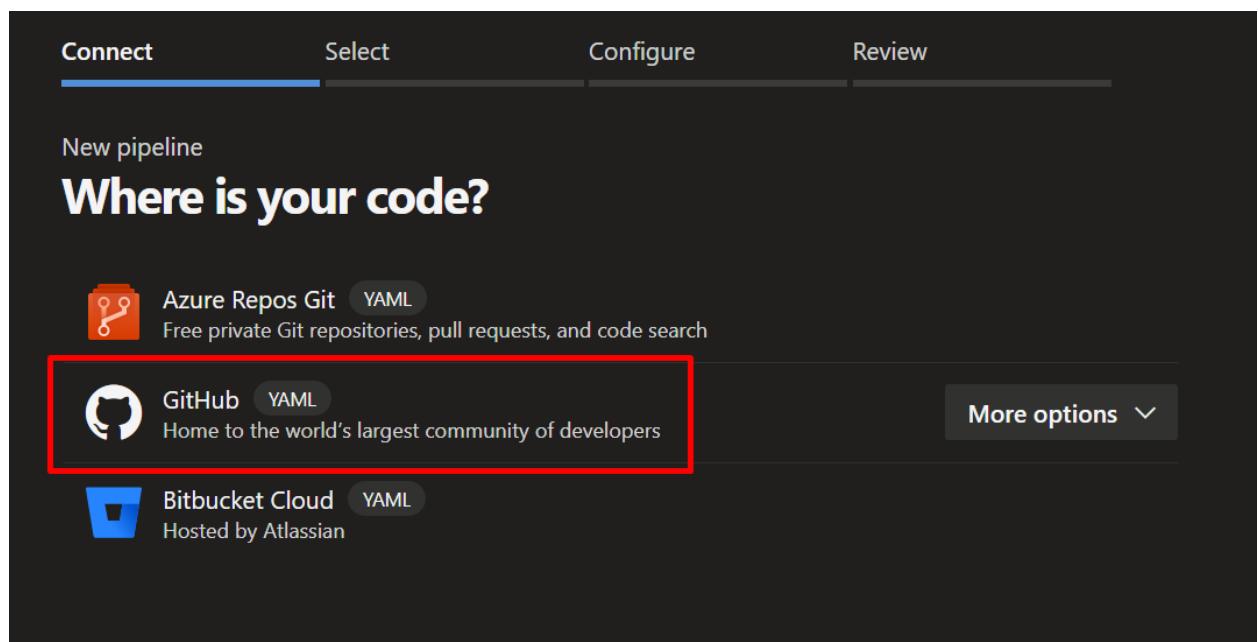
Pipelines => New pipeline



The screenshot shows the Azure DevOps Pipelines interface. On the left, there's a sidebar with links for Overview, Boards, Repos, Pipelines (which is selected), Environments, Releases, and Library. The main area is titled 'Pipelines' and shows a list of 'Recently run pipelines'. There are three entries:

Pipeline	Last run
quangvinh2509.SD0691_DevOpsTrainin...	#20251028.2 • Update cd-azure-pipelines.yml for ... Manually run by main Tuesday 1m 3s
quangvinh2509.SD0691_DevOpsTrainin...	#20251028.2 • Update Deploy\mangeToACR_V1.y... Manually run by main Tuesday 4m 26s
AzureDevOps-fabric-cicd-sample (56)	#20251021.1 • Set up CI with Azure Pipelines Individual CI for main Oct 21 <1s

Choose GitHub where store the code



The screenshot shows the 'Where is your code?' step in the Azure DevOps pipeline creation wizard. The top navigation bar has tabs: Connect, Select, Configure, and Review. The 'Connect' tab is active.

The main area displays three options:

- Azure Repos Git** (YAML): Free private Git repositories, pull requests, and code search.
- GitHub** (YAML): Home to the world's largest community of developers. This option is highlighted with a red box.
- Bitbucket Cloud** (YAML): Hosted by Atlassian.

On the right side, there's a 'More options' dropdown.

Choose repository

The screenshot shows the 'Select a repository' step in a pipeline creation wizard. The top navigation bar includes 'Connect', 'Select' (which is highlighted in blue), 'Configure', and 'Review'. Below the bar, it says 'New pipeline' and 'Select a repository'. A search bar with 'Filter by keywords' and a dropdown for 'My repositories' are visible. Three repositories are listed: 'levanminh83868386-art/application-deployment-demo' (3h ago), 'levanminh83868386-art/aks-demo' (4h ago), and 'levanminh83868386-art/aks_deployment' (Tuesday). A note at the bottom explains the repository selection logic.

① Showing the most recently used repositories where you are a collaborator.
If you can't find a repository, make sure you provide access.
You may also select a specific connection.

Define CI/CD process code:

```
trigger: none

pool:
  vmImage: 'ubuntu-latest'

variables:
  azureSubscription: 'minhAzureServiceConnection'      # your Azure DevOps
  service connection
  aksResourceGroup: 'myTFRResourceGroup'      # 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

/ DevOpsProjHao / Pipelines / minh-cicd-pipeline / 20251030.4

Search

← Jobs in run #202...
minh-cicd-pipeline

Jobs

CI 3m 8s

- ✓ Initialize job 1s
- ✓ Checkout leva... 2s
- ✓ Build and ... 3m 3s
- ✓ Post-job: Che... <1s
- ✓ Finalize Job <1s

CD 1m 5s

- ✓ Initialize job 1s
- ✓ Checkout leva... 3s
- ✓ Connect to A... 25s
- ✓ Create names... 2s
- ✓ Create storage 5s

CD

1 Pool: Azure Pipelines
2 Image: ubuntu-latest
3 Agent: Hosted Agent
4 Started: Today at 11:09 AM
5 Duration: 1m 5s
6
7 ► Job preparation parameters

1 Pool: Azure Pipelines
2 Image: ubuntu-latest
3 Agent: Hosted Agent
4 Started: Today at 11:09 AM
5 Duration: 1m 5s
6
7 ► Job preparation parameters

5. Verify

Get all resource from AKS

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

