Bootcamp-Project 2

DEPLOY A CONTAINERIZED APPLICATION TO AKS USING AZURE DEVOPS USING HELM CHART

Mohanramrajan Erran Bothalraj

Table of Contents

Introduction

This project demonstrates a **complete CI/CD pipeline** setup to build, deploy, and monitor a containerized .net web application using **Azure DevOps, Azure Kubernetes Service (AKS), and Helm**. The pipeline automates the process from code commit to live deployment, ensuring fast, reliable, and scalable delivery. Additionally, **Azure Monitor and Prometheus** are integrated for observability and health monitoring of the deployed services.

Project Objectives

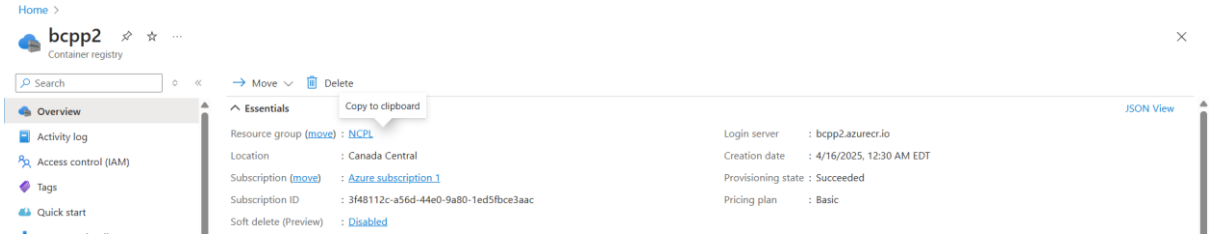
1. **Containerize the Application:**
    - Use Docker to build a container image of the application.
    - Configure the Dockerfile for production-ready deployment.
2. **Set Up a CI Pipeline (Build Pipeline):**
    - Automatically trigger on code commits.
    - Build the Docker image.
    - Tag and push the image to ACR.
3. **Set Up a CD Pipeline (Release Pipeline):**
    - Use Helm charts to define Kubernetes resources.
    - Deploy the application to AKS using the latest image from ACR.
    - Update Kubernetes workloads seamlessly.
4. **Monitoring and Observability:**
    - Enable Azure Monitor for real-time logs and metrics.
    - Deploy Prometheus (via Helm) to monitor application-specific metrics.

Expected Outcome

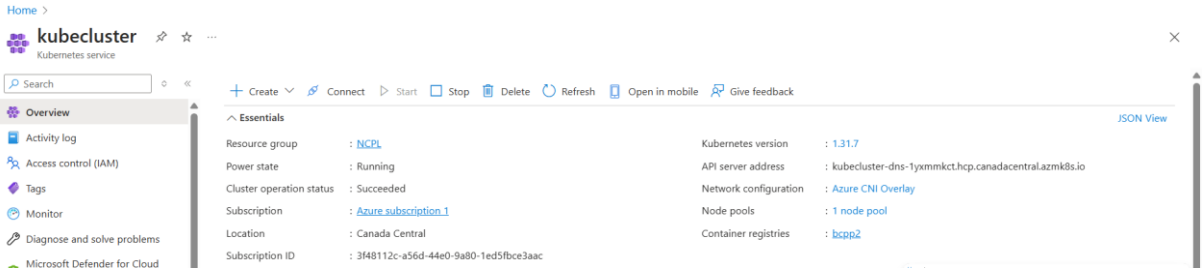
A fully automated CI/CD pipeline that builds, packages, and deploys a containerized application to **Azure Kubernetes Service (AKS) using Azure DevOps and Helm Chart**. The deployment is scalable, version-controlled, and monitored in real time using **Azure Monitor and Prometheus**—ensuring faster releases, improved reliability, and greater visibility into application health.

Pre-Requisite

1. Git/Azure Repos with Dockerfile configured for the .net application: https://github.com/merranbo1989/BCP-P2.git
2. Create Azure Container Registry



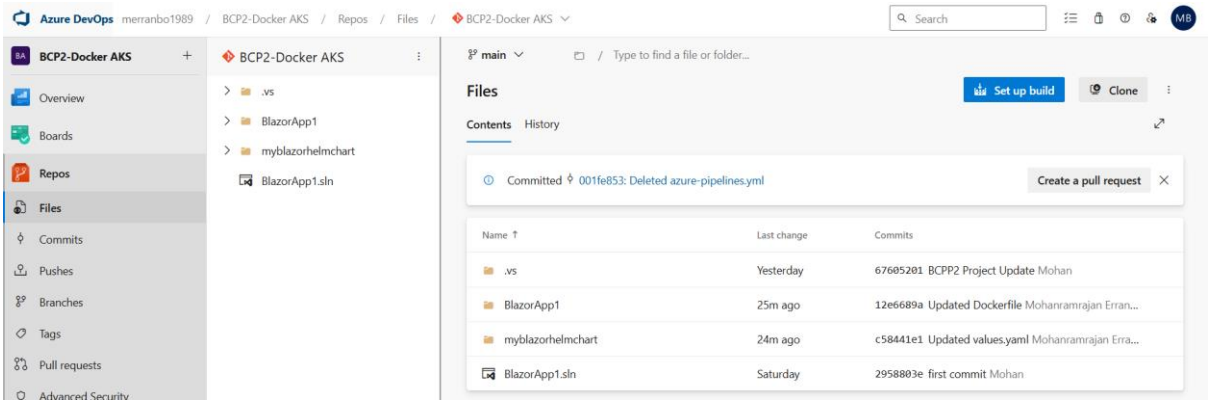
3. Create Azure Kubernetes Cluster



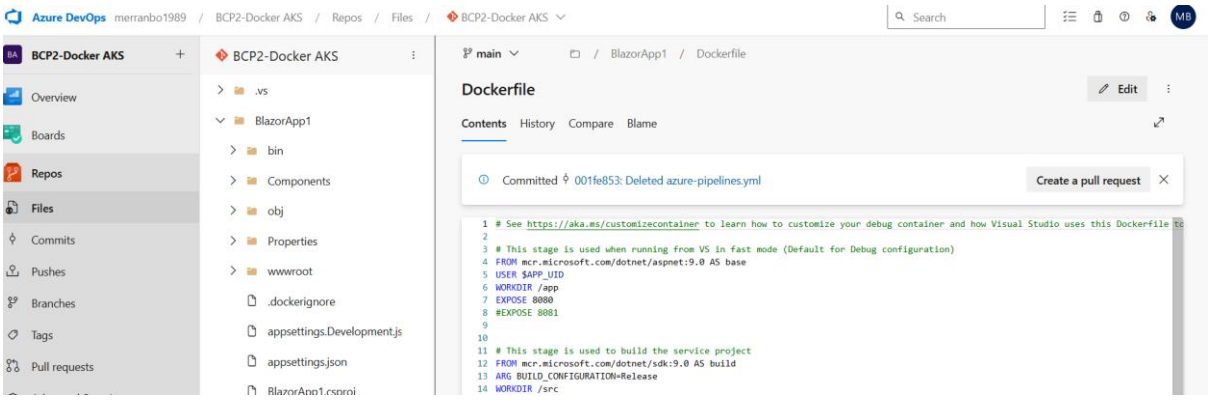
Solution Summary

Step 1 – Containerize the Application

A. Upload the .net application code into Azure Repos



B. Configure the Dockerfile for the .net application



******* Dockerfile Code Starts *******

FROM mcr.microsoft.com/dotnet/aspnet:9.0 AS base
USER $APP_UID
WORKDIR /app
EXPOSE 8080
#EXPOSE 8081

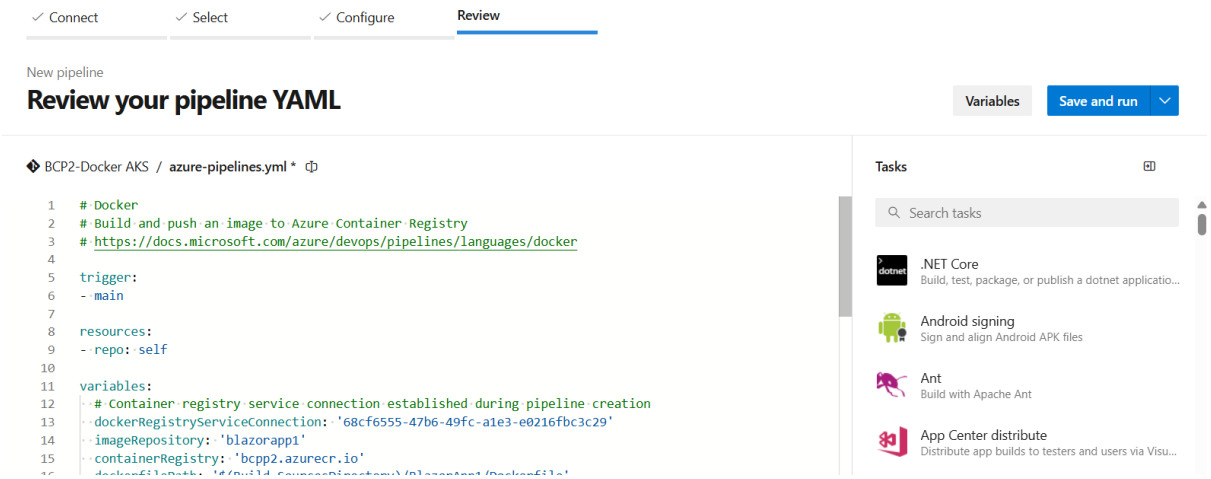
 # This stage is used to build the service project

```
FROM mcr.microsoft.com/dotnet/sdk:9.0  AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["BlazorApp1.csproj",  "."]
RUN dotnet restore "./BlazorApp1.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "./BlazorApp1.csproj"  -c $BUILD_CONFIGURATION -o /app/build

# This stage is used to publish the service project to be copied to the final stage
FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "./BlazorApp1.csproj"  -c $BUILD_CONFIGURATION -
o /app/publish /p:UseAppHost=false

# This stage is used in production or when running from VS in regular mode (Default when not using t
he Debug configuration)
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "BlazorApp1.dll"]
```

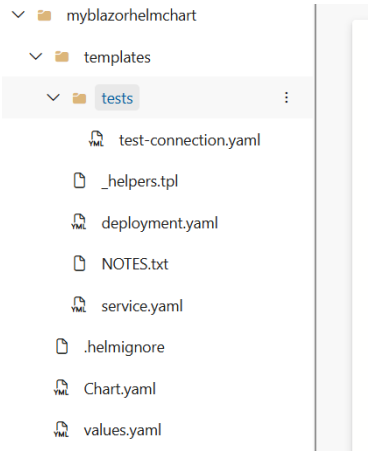
******* Code ends here *******

Step 2 – Create the CI Pipeline

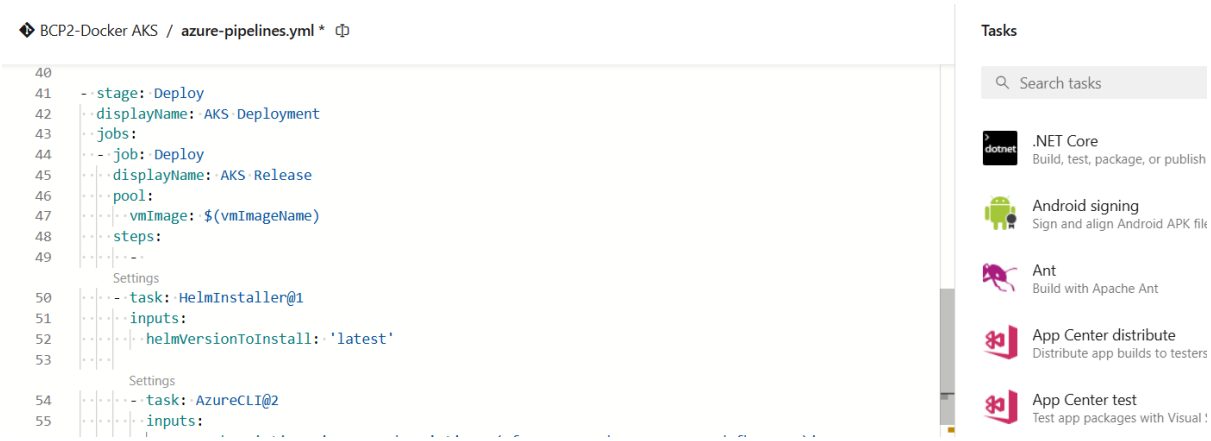
A. Setup the CI/CD build to build & push the Docker Image to ACR



B. Configure the Helm Chart to deploy into AKS.



C. Setup the CD pipeline to deploy the Docker image from ACR to AKS



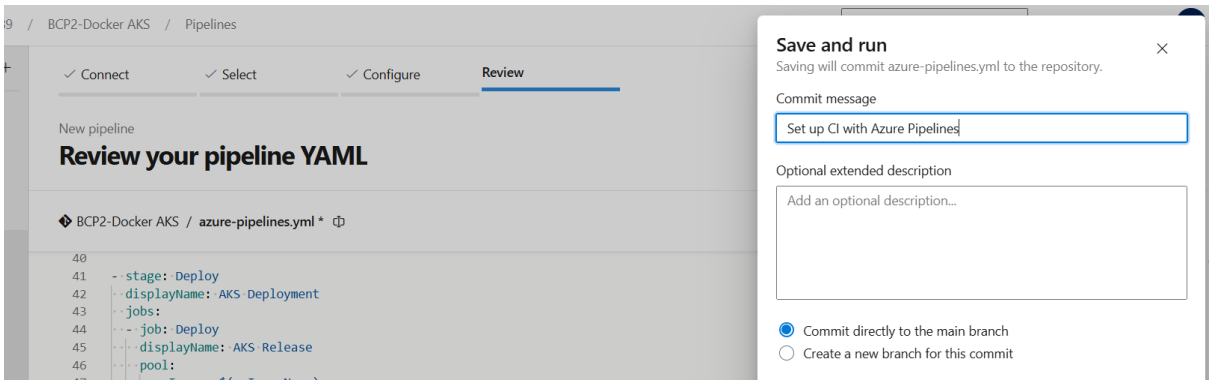
******* Code for azure-pipeline.yml *******

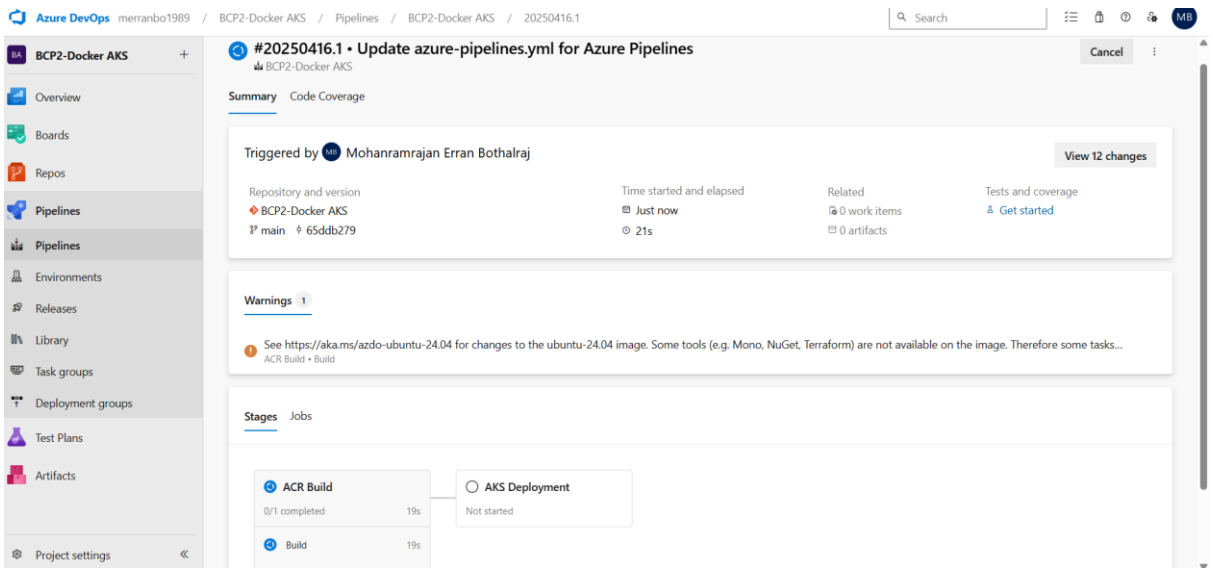
```
trigger:
- master

resources:
- repo: self

variables:
  # Container registry service connection established during pipeline creation
  dockerRegistryServiceConnection: '1da2fadb-236f-4fe3-9b9d-1257a81b20d0'
  azureSubscription: '3f48112c-a56d-44e0-9a80-1ed5fbce3aac'
  imageRepository: 'blazorapp1'
  containerRegistry: 'bcpp2.azurecr.io'
  dockerfilePath: '$(Build.SourcesDirectory)/BlazorApp1/Dockerfile'
  tag: 'latest'

  # Agent VM image name
  vmImageName: 'ubuntu-latest'

stages:
- stage: Build
  displayName: ACR Docker Build
  jobs:
```

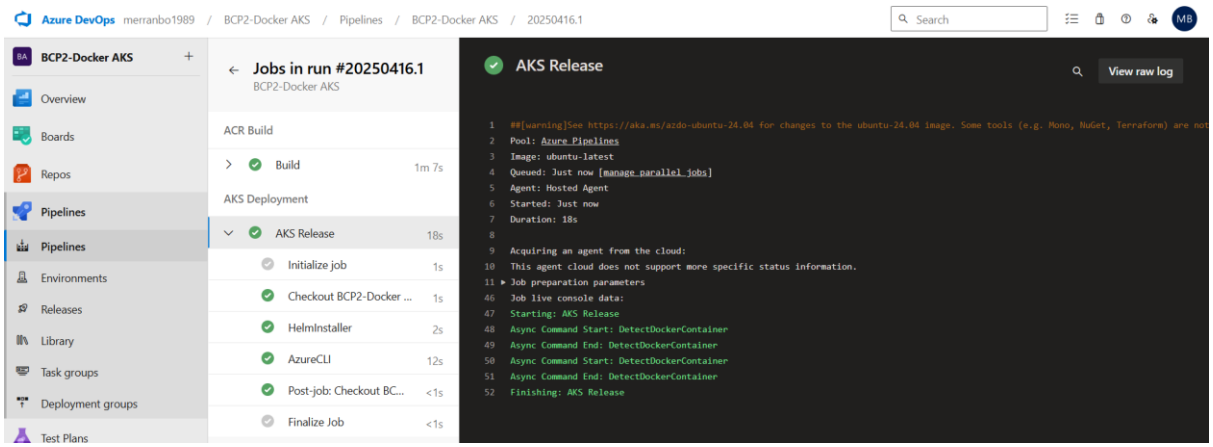
```
  - job: Build
    displayName: Build Docker Image
    pool:
      vmImage: $(vmImageName)
    steps:

    - task: Docker@2
      inputs:
        containerRegistry: 'bcpp2'
        repository: 'blazorapp1'
        command: 'buildAndPush'
        Dockerfile: '**/Dockerfile'
        tags: $(tag)
- stage: Deploy
  displayName: AKS Deployment
  jobs:
  - job: Deploy
    displayName: Deploy to AKS
    pool:
      vmImage: $(vmImageName)
    steps:
      - task: HelmInstaller@1
        inputs:
          helmVersionToInstall: 'latest'

      - task: AzureCLI@2
        inputs:
          azureSubscription: 'Azure subscription 1(3f48112c-a56d-44e0-9a80-1ed5fbce3aac)'
          scriptType: 'bash'
          scriptLocation: 'inlineScript'
          inlineScript: |

            echo "Getting AKS credentials..."
            az aks get-credentials --resource-group NCPL --name kubecluster --overwrite-existing
            echo "Deploying with Helm..."
            helm upgrade --install myblazorapp1 ./myblazorhelmchart
```

******* Code ends here *******

D. Trigger the Pipeline

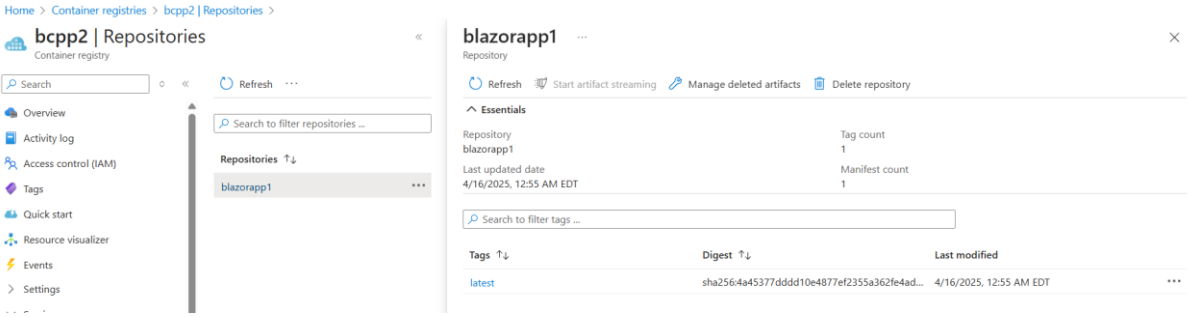




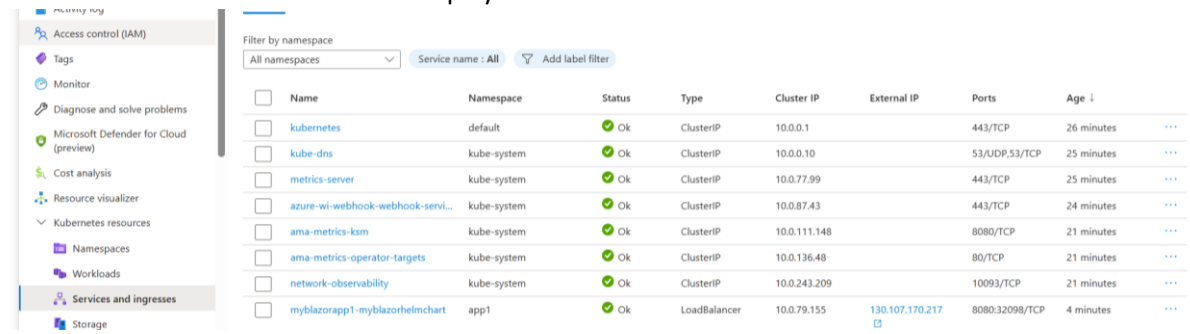


Step 3 – Validation

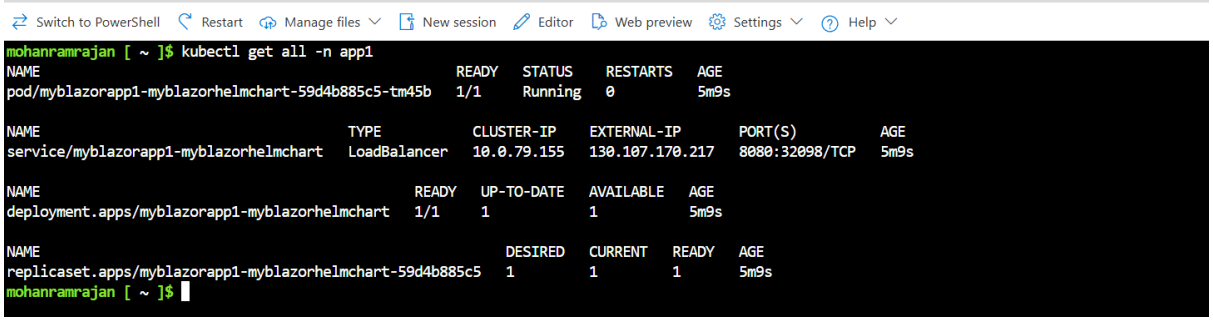
A. Check the ACR in Azure to see if the image is available



B. Check the AKS to see if the deployment is available

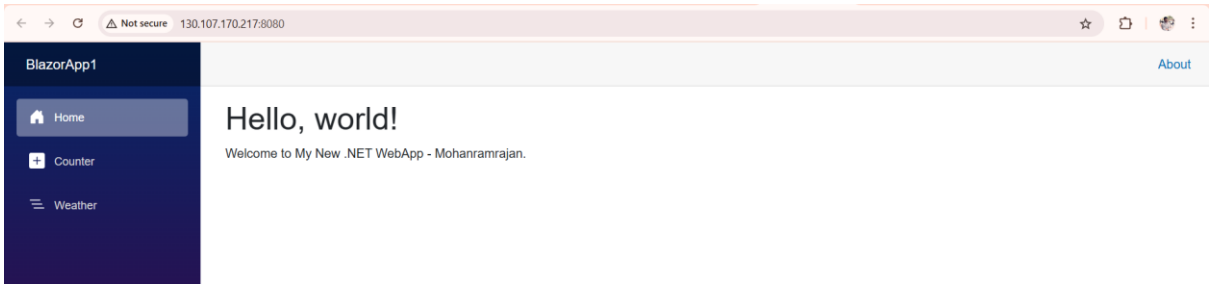


C. Launch the Azure CLI and verify if the nodes, pods, and services are running and available in the namespace



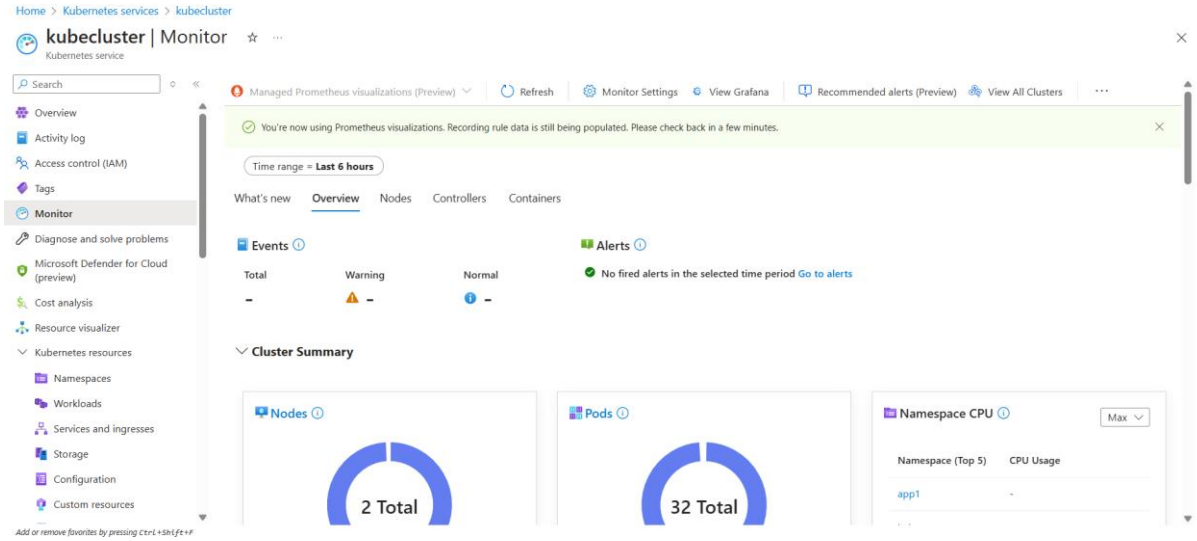
D. Use the Load Balancer IP address and verify that the application is working as expected.

URL: http://130.107.170.217:8080

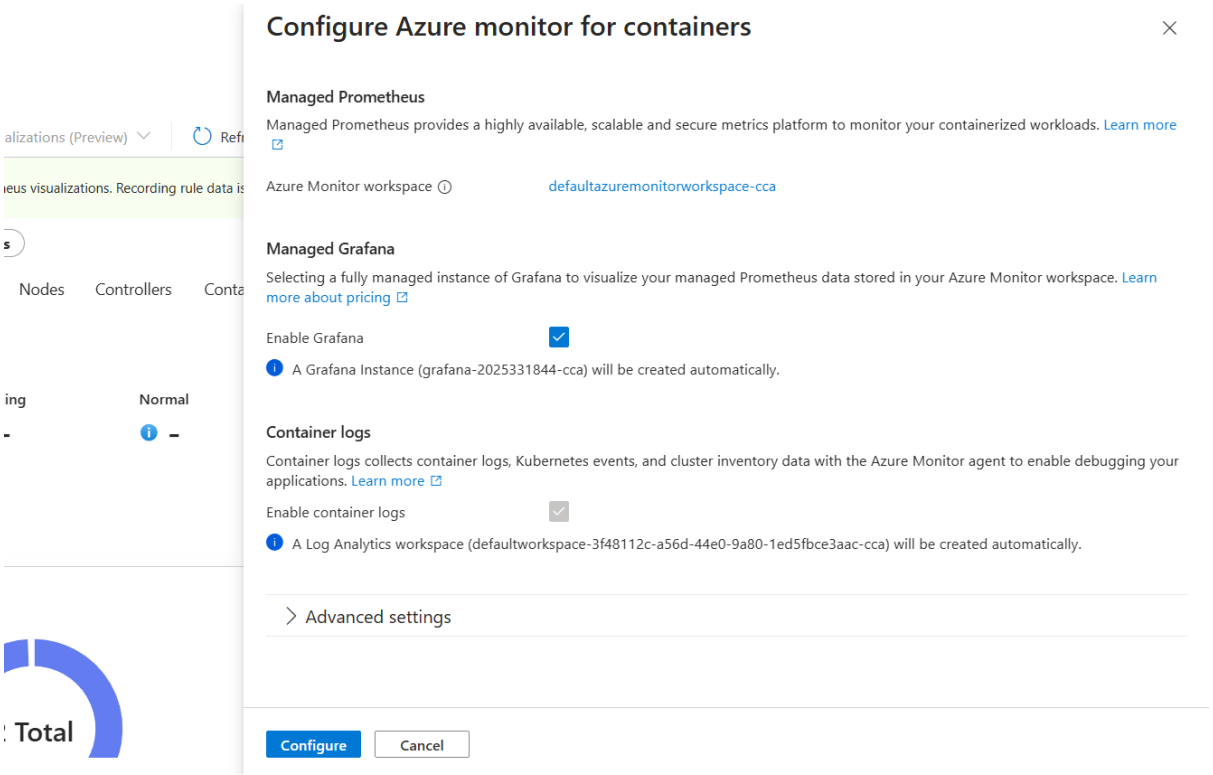


Step 4 – Azure Monitor with Prometheus and Grafana

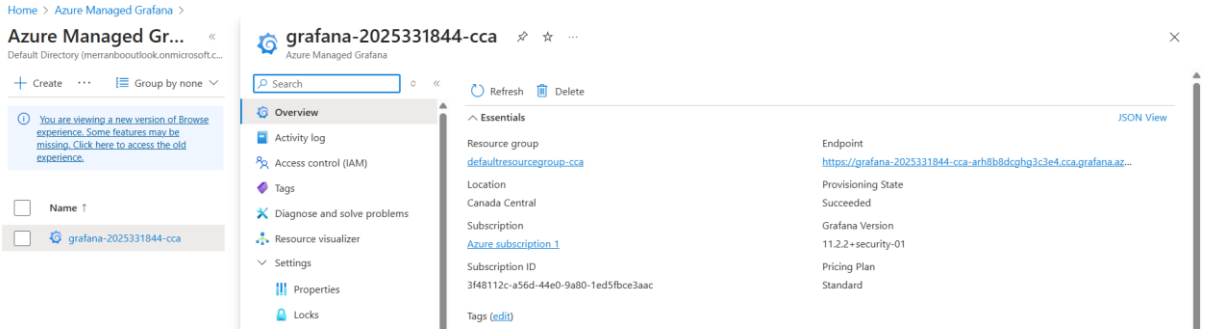
A. Prometheus Dashboard



B. Enabling Grafana



C. Configure Grafana Dashboard



D. View the Grafana Dashboard