**wrExpectation:** You are tasked to build, containerize, and deploy a simple "Hello World" web application on Azure Free Tier while ensuring efficient resource utilization, scalability, and automation using tools like Docker, Azure Kubernetes Service (AKS), and Azure DevOps, all within a real-world DevOps workflow?

**What you need to do:** You’ll first create Azure resources like a Resource Group, VNet, Subnet, and a VM for development. Next, develop a static "Hello World" web app using HTML/CSS and containerize it using Docker. Push the container to Azure Container Registry (ACR). Then, set up Azure Kubernetes Service (AKS) to host the app. Finally, automate the process using Azure DevOps Classic Pipelines to build, push, and deploy the app. Ensure all resources stay within free-tier limits.

**Objective:** To familiarize the associate with cloud infrastructure, containerization, and automation. By deploying a "Hello World" web app using HTML and CSS on Azure Free Tier, participants will learn to set up resources like VMs, AKS, and ACR, containerize applications using Docker, and automate deployment with Azure DevOps pipelines. The goal is to provide hands-on experience in DevOps practices, CI/CD workflows, and resource optimization within free-tier limits.

**Required Setup and Installations**

**1. Azure Resource Group**

* **Name:** DevEnvironment-RG
* **Region:** East US
* **Purpose:** Group all resources related to this project.
* **Bash Command:** az group create --name DevEnvironment-RG --location "East US"

**2. Virtual Network (VNet)**

* **Name:** Dev-VNet
* **Address Space:** 10.0.0.0/16
* **Purpose:** Enable communication between resources in the Azure environment.
* **Bash Command:** az network vnet create \

--name Dev-VNet \

--resource-group DevEnvironment-RG \

--address-prefix 10.0.0.0/16

**3. Subnet**

* **Name:** Dev-Subnet
* **Address Range:** 10.0.1.0/24
* **Purpose:** Assign IP addresses for resources inside the VNet.
* **Bash Command:** az network vnet subnet create \

--vnet-name Dev-VNet \

--resource-group DevEnvironment-RG \

--name Dev-Subnet \

--address-prefix 10.0.1.0/24

**4. Storage Account**

* **Name:** devstorage<uniqueid> (e.g., devstorage12345)
* **Location:** East US
* **SKU:** Standard\_LRS
* **Purpose:** Store logs and application-related data.
* **Bash Command:** az storage account create \

--name devstorage12345 \

--resource-group DevEnvironment-RG \

--location "East US" \

--sku Standard\_LRS \

--kind StorageV2 \

--tags Project=DevEnvironment Owner=azure-dev-user

**5. Virtual Machine**

* **Name:** Dev-VM
* **Operating System:** Ubuntu 20.04 LTS
* **Size:** Standard\_B1s (1 vCPU, 1 GB RAM, Free Tier eligible)
* **Admin Username:** azure-dev-user
* **Tags:** Project=DevEnvironment, Owner=azure-dev-user
* **Purpose:** Development environment to host tools and test applications.
* **Bash Command:**  az vm create \

--resource-group DevEnvironment-RG \

--name Dev-VM \

--image UbuntuLTS \

--size Standard\_B1s \

--admin-username azure-dev-user \

--generate-ssh-keys \

--vnet-name Dev-VNet \

--subnet Dev-Subnet \

--tags Project=DevEnvironment Owner=azure-dev-user

**6. Network Security Group (NSG)**

* **Name:** Dev-NSG
* **Rules:** Allow HTTP traffic on port 80 and SSH traffic on port 22.
* **Bash Command to Create NSG:** az network nsg create --resource-group DevEnvironment-RG --name Dev-NSG

Command to Add Rules: az network nsg rule create \

--resource-group DevEnvironment-RG \

--nsg-name Dev-NSG \

--name Allow-HTTP \

--priority 100 \

--direction Inbound \

--access Allow \

--protocol Tcp \

--destination-port-ranges 80

az network nsg rule create \

--resource-group DevEnvironment-RG \

--nsg-name Dev-NSG \

--name Allow-SSH \

--priority 110 \

--direction Inbound \

--access Allow \

--protocol Tcp \

--destination-port-ranges 22

**7. Docker Installation on VM**

* **Version:** Latest version available via apt
* **Purpose:** Containerize the application for deployment.
* **Bash Commands:** sudo apt update

sudo apt install -y docker.io

sudo systemctl start docker

sudo systemctl enable docker

sudo usermod -aG docker azure-dev-user

**8. Git Installation on VM**

* **Version:** Latest version available via apt
* **Purpose:** Version control for application files.
* **Commands:** sudo apt install -y git

**9. HTML and CSS Web Application**

* **Folder Name:** hello-world-app
* **File Name:** index.html
* **Content of html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Hello World</title>

<style>

body {

font-family: Arial, sans-serif;

text-align: center;

margin-top: 20%;

background-color: #f4f4f4;

}

h1 {

color: #333;

}

</style>

</head>

<body>

<h1>Hello, World!</h1>

<p>Welcome to your first deployed web application!</p>

</body>

</html>

**10. Dockerfile**

* **Purpose:** Containerize the static web application.
* **Content of Dockerfile:** FROM nginx:latest

COPY . /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

**11. Docker Commands**

* **Build the Image:**

docker build -t hello-world-app .

**Run the Image Locally:**

docker run -p 80:80 hello-world-app

**12. Azure Kubernetes Service (AKS)**

* **Cluster Name:** Dev-AKS
* **Node Count:** 1
* **Node Size:** Standard B2s (Free Tier eligible)
* **Commands to Create Cluster:**

az aks create \

--resource-group DevEnvironment-RG \

--name Dev-AKS \

--node-count 1 \

--enable-addons monitoring \

--generate-ssh-keys

**13. Azure Container Registry (ACR)**

* **Name:** DevEnvironmentACR
* **Commands to Create and Push Docker Image:**

az acr create \

--resource-group DevEnvironment-RG \

--name DevEnvironmentACR \

--sku Basic

az acr login --name DevEnvironmentACR

docker tag hello-world-app DevEnvironmentACR.azurecr.io/hello-world-app:latest

docker push DevEnvironmentACR.azurecr.io/hello-world-app:latest

14. Kubernetes Deployment Manifest

File Name: deployment.yaml

Content of yaml:

apiVersion: apps/v1

kind: Deployment

metadata:

name: hello-world-app

spec:

replicas: 1

selector:

matchLabels:

app: hello-world-app

template:

metadata:

labels:

app: hello-world-app

spec:

containers:

- name: hello-world-app

image: DevEnvironmentACR.azurecr.io/hello-world-app:latest

ports:

- containerPort: 80

Command to Deploy:

kubectl apply -f deployment.yaml

**Azure DevOps Classic Pipeline**

* **Tasks:**
  + Clone Repository
  + Build Docker Image
  + Push to ACR
  + Deploy to AKS

**CI/CD pipeline setup** for Azure DevOps Classic Pipelines

**15. Azure DevOps Classic Pipeline**

**CI/CD Pipeline Overview**

* **CI (Build Pipeline):**
  + Build the Docker image from the application code.
  + Push the image to Azure Container Registry (ACR).
* **CD (Release Pipeline):**
  + Pull the Docker image from ACR.
  + Deploy the application to Azure Kubernetes Service (AKS).

**Step-by-Step CI/CD Tasks**

**Continuous Integration (CI): Build Pipeline**

1. **Create a New Build Pipeline in Azure DevOps**
   * Navigate to **Pipelines** > **Builds** > **New Pipeline** > **Use Classic Editor**.
   * Choose the **repository** where the application code is stored.
   * Select **"Start with an empty job"**.
2. **Add CI Tasks**
   * **Task 1: Clone the Repository**
     + Task Type: **Get Sources**
     + Ensure the repository containing the application code and Dockerfile is selected.
   * **Task 2: Build the Docker Image**
     + Task Type: **Command Line**
     + Display Name: "Build Docker Image"
     + Script: docker build -t hello-world-app .
   * **Task 3: Tag the Docker Image**
     + Task Type: **Command Line**
     + Display Name: "Tag Docker Image"
     + Script: docker tag hello-world-app DevEnvironmentACR.azurecr.io/hello-world-app:$(Build.BuildId)
   * **Task 4: Push Docker Image to ACR**
     + Task Type: **Command Line**
     + Display Name: "Push Docker Image"
     + Script: az acr login --name DevEnvironmentACR

docker push DevEnvironmentACR.azurecr.io/hello-world-app:$(Build.BuildId)

1. **Publish the Build Artifact**
   * Publish metadata about the built image for later use in the release pipeline.
2. **Save and Queue the Build**
   * Name the pipeline CI-HelloWorld-App.
   * Run the pipeline to ensure it successfully builds and pushes the Docker image to ACR.

**Continuous Deployment (CD): Release Pipeline**

1. **Create a New Release Pipeline**
   * Navigate to **Pipelines** > **Releases** > **New Pipeline** > **Empty Job**.
2. **Add an Artifact**
   * Select the build pipeline (CI-HelloWorld-App) as the artifact source.
   * Enable **Continuous Deployment Trigger** to automatically deploy after a successful build.
3. **Add CD Tasks**
   * **Task 1: Pull the Docker Image from ACR**
     + Task Type: **Kubernetes Service Connection**
     + Display Name: "Pull Docker Image"
     + Configuration:
       - Set the image reference: DevEnvironmentACR.azurecr.io/hello-world-app:$(Build.BuildId)
   * **Task 2: Deploy to AKS**
     + Task Type: **Kubernetes Deployment**
     + Display Name: "Deploy to AKS"
     + Configuration:
       - Kubernetes Cluster: Select the connected AKS cluster.
       - Namespace: Default (or create a namespace if necessary).
       - Deployment YAML: Use the deployment.yaml file to deploy the application.
4. **Save and Create the Release**
   * Name the pipeline CD-HelloWorld-App.
   * Run the release pipeline to deploy the application.

**Pipeline Visualization**

Build Pipeline (CI):

Trigger: Push to the main branch or manual trigger.

Tasks:

Clone Repository

Build Docker Image

Tag Docker Image

Push Docker Image to ACR

Output: Docker image available in ACR.

Release Pipeline (CD):

Trigger: Successful build of the CI pipeline.

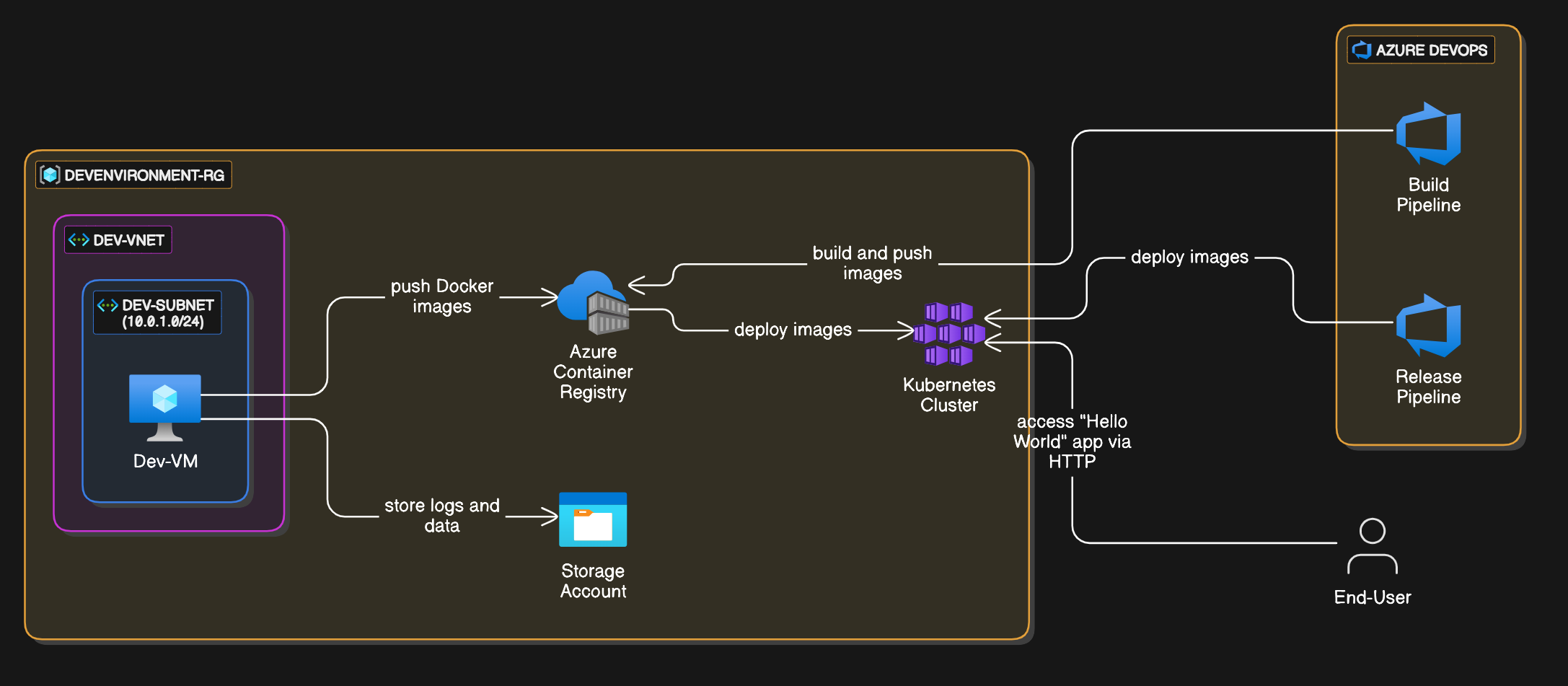
Tasks:

Pull Docker Image from ACR

Deploy the application to AKS

Output: "Hello World" application deployed on AKS.

**Architecture reference**



**Additional Integrations | Health Check Script for Kubernetes Cluster (PowerShell)**

**Script Objective:**

The script will:

1. Check the health of Kubernetes nodes and pods.
2. Report if all components are healthy or identify any issues.
3. Exit with an appropriate status code to integrate seamlessly with the CI/CD pipeline.

**Health Check Script (PowerShell):**

**Create File:** health-check.ps1

**Function to check the health of Kubernetes nodes**

function Check-Nodes {

Write-Host "Checking Kubernetes Nodes..."

$nodes = kubectl get nodes --no-headers | ForEach-Object {

$fields = $\_ -split '\s+'

[PSCustomObject]@{

NodeName = $fields[0]

Status = $fields[1]

}

}

$allHealthy = $true

foreach ($node in $nodes) {

if ($node.Status -ne "Ready") {

Write-Error "Node $($node.NodeName) is not healthy. Status: $($node.Status)"

$allHealthy = $false

} else {

Write-Host "Node $($node.NodeName) is healthy."

}

}

if (-not $allHealthy) {

exit 1

}

}

# Function to check the health of Kubernetes pods

function Check-Pods {

Write-Host "Checking Kubernetes Pods..."

$pods = kubectl get pods --all-namespaces --no-headers | ForEach-Object {

$fields = $\_ -split '\s+'

[PSCustomObject]@{

PodName = $fields[1]

Status = $fields[2]

}

}

$allHealthy = $true

foreach ($pod in $pods) {

if ($pod.Status -ne "Running" -and $pod.Status -ne "Completed") {

Write-Error "Pod $($pod.PodName) is not healthy. Status: $($pod.Status)"

$allHealthy = $false

} else {

Write-Host "Pod $($pod.PodName) is healthy."

}

}

if (-not $allHealthy) {

exit 1

}

}

# Run the checks

Check-Nodes

Check-Pods

Write-Host "All Kubernetes components are healthy."

exit 0

**Steps to Integrate into the Pipeline**

**1. Add the Script to the Repository**

* Save the health-check.ps1 script in the repository under a folder, e.g., scripts/.

**2. Add a Task in the Azure DevOps Pipeline**

* **Pipeline Integration:**
  + Add a **PowerShell** task in the build or release pipeline to execute the script.
  + Ensure the Kubernetes cluster context is set using kubectl.
* **Task Configuration:**
  + **Task Type:** PowerShell
  + **Display Name:** "Kubernetes Cluster Health Check"
  + **Script Path:** scripts/health-check.ps1
* **Inline Script Example:**

powershell

Copy code

pwsh scripts/health-check.ps1

**3. Behaviour in the Pipeline**

* **Success Scenario:** If all nodes and pods are healthy, the script exits with 0, allowing the pipeline to proceed.
* **Failure Scenario:** If any node or pod is unhealthy, the script exits with 1, failing the pipeline and alerting the team to investigate.

**Pipeline Workflow with Health Check**

1. **Build Pipeline (CI):**
   * Clone Repository.
   * Build Docker Image.
   * Push Docker Image to ACR.
2. **Release Pipeline (CD):**
   * Pull Docker Image from ACR.
   * **Health Check Task:** Run health-check.ps1 to ensure Kubernetes cluster health before deployment.
   * Deploy Docker Image to AKS.

End Game