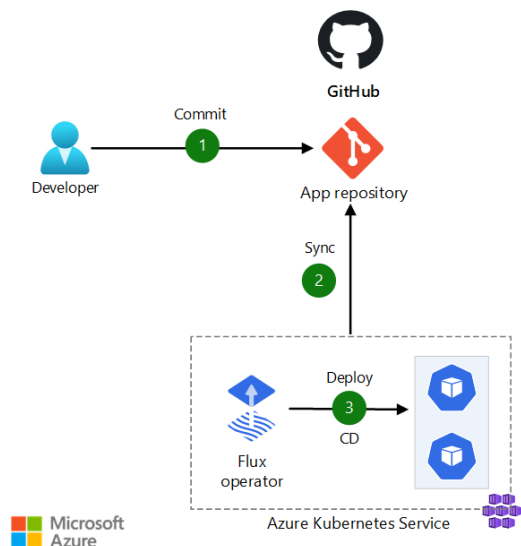


# GitHub & DevOps – using AKS

## PROBLEM STATEMENT :

In today's rapidly changing business landscape, traditional application deployment falls short in meeting the growing demands for scalability and cross-platform compatibility. To address these challenges, a DevOps solution combining GitHub Actions and Azure Kubernetes Services (AKS) is crucial. This solution automates container management, ensuring efficient, secure, and consistent application deployment across diverse environments.

## SOLUTION ARCHITECTURE :

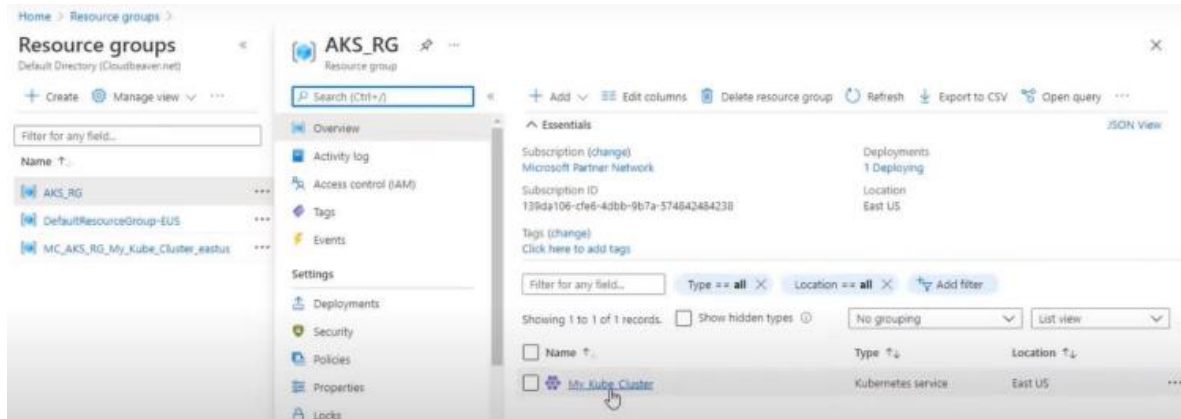


## Prerequisites:

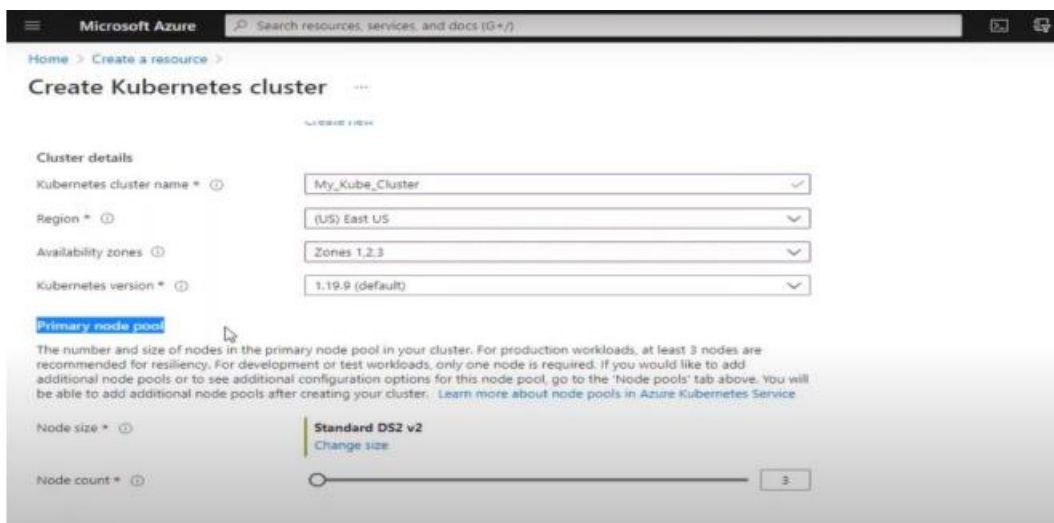
- Access to an Azure subscription
- Access to a GitHub account
- Basic knowledge of executing commands by using the Azure CLI
- Basic knowledge of Kubernetes and its concepts
- Basic knowledge of AKS and its concepts
- Basic knowledge of Git and GitHub

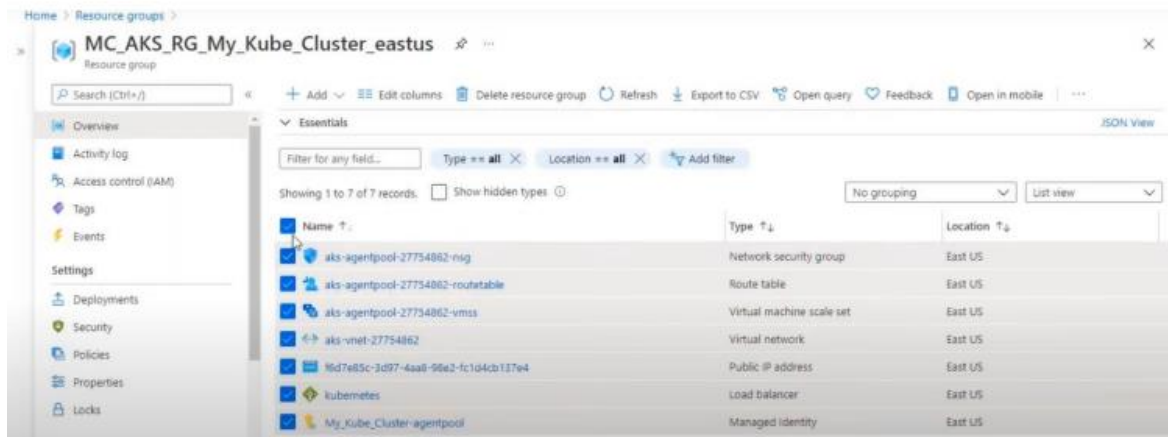
## Step 1: Creating a Kubernetes cluster in azure portal.

Before creating a Kubernetes cluster you need to create a resource group in which your cluster will reside



When setting up your cluster, the critical step is selecting the right node size and determining the number of nodes required. The node count represents the VMs within your cluster, with a maximum limit of 1000 nodes. Remember that you can adjust the node count post-deployment, but the instance type remains fixed. Keep in mind that if you opt for Linux as the OS, hosting Windows applications won't be possible, and vice versa.





You can also use CLI :

### Add windows server 2019 node pool:

Ex:

```
az aks nodepool add \
  */used to add a node pool named npwin
--resource-group myResourceGroup \
--cluster-name myAKSCluster \
--os-type Windows \
--name npwin \
--node-count 1
```

### Add a windows server node pool 2022:

For newer versions we need to specify os sku type.

Ex:

```
az aks nodepool add \
--resource-group myResourceGroup \
--cluster-name myAKSCluster \
--os-type Windows \
--os-sku Windows2022 \
--name npwin \
--node-count 1
```

### connect to the cluster:

- To connect to the cluster we use the kubectl it can installed locally using the command `az aks install-cli`. `az aks get-credentials --resource-group`

myResourceGroup --name myAKSCluster is the command used to configure kubectl.

To check the status of each node execute the following command:

```
kubectl get nodes -o wide
```

Step 2 :

In Azure CLI clone the github repository

```
Bash
git clone https://github.com/{your-username}/mslearn-aks-deployment-pipeline-github-actions
The command creates a new directory called mslearn-aks-deployment-pipeline-github-actions.
```

```
Bash
cd mslearn-aks-deployment-pipeline-github-actions
```

```
Bash
bash init.sh
```

Step 3:

Build the Application Workflow

forked from [MicrosoftDocs/mslearn-aks-deployment-pipeline-github-actions](#)

<> Code Pull requests **🔄 Actions** Projects Wiki Settings

## Get started with GitHub Actions

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow template to get started.

Skip this and [set up a workflow yourself](#) →

Step 4:

**Create a YAML file for pushing and deploying**

name: Build and push the latest build to staging

on:

push:

branches: [ main ]

jobs:

build\_push\_image:

runs-on: ubuntu-20.04

steps:

- uses: actions/checkout@v2

- name: Set up Buildx

uses: docker/setup-buildx-action@v1

- name: Docker Login

uses: docker/login-action@v1

with:

registry: \${{ secrets.ACR\_NAME }}

username: \${{ secrets.ACR\_LOGIN }}

password: \${{ secrets.ACR\_PASSWORD }}

- name: Build and push staging images

uses: docker/build-push-action@v2

with:

context: .

push: true

tags: \${{secrets.ACR\_NAME}}/contoso-website:latest

Commit the Changes:

**Start commit** ▾

## Commit new file

Create build steps for staging

Add an optional extended description...

Choose which email address to associate with this commit

Commit directly to the `main` branch.

Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

**Commit new file**

Step 5:

Connect the AKS CLI

— az aks get-credentials --resource-group (name) --name (cluster name)

```
$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
aks-agentpool-27754862-vmss000000	Ready	agent	3m31s	v1.19.9
aks-agentpool-27754862-vmss000001	Ready	agent	3m47s	v1.19.9

Step 6 :

apiVersion: apps/v1

kind: Deployment

metadata:

name: sam

labels:

```
app: sam
spec:
replicas: 1
template:
metadata:
name: sam
labels:
app: sam
spec:
nodeSelector:
"kubernetes.io/os": windows
containers:
- name: sam
image: E:/dotnet/framework/sam:asp netapp
resources:
limits:
cpu: 1
memory: 800M
ports:
- containerPort: 80
selector:
matchLabels:
app: sam
---
apiVersion: v1
kind: Service
metadata:
name: sam
spec:
type: LoadBalancer
ports:
- protocol: TCP
```

port: 80

selector:

app: sample

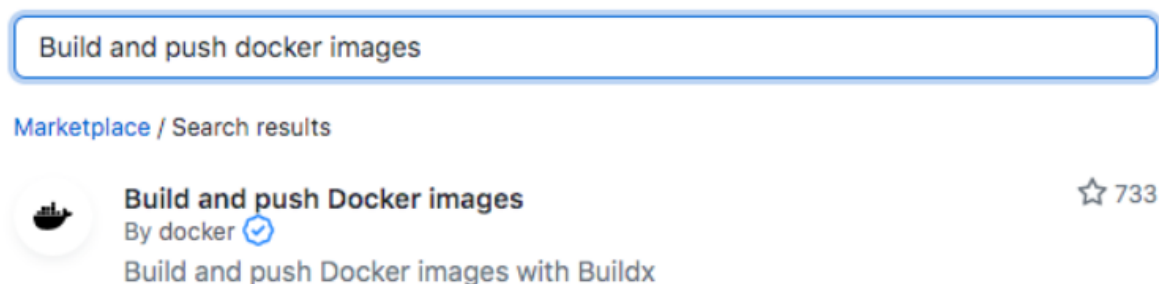
## Deploy the application using kubectl apply:

```
kubectl apply -f sam.yaml
```

**Your application is deployed.**

Docker Configuration:

search for **Docker Login**. Select the first result published by **Docker**.



---

### Create a personal access token (PAT)

1. Go to the fork of the sample repository in the GitHub website. On the top right hand corner, select your profile photo, then select **Settings**.
2. Select **Developer settings** at the bottom of the left menu.
3. Select **Personal access tokens**.
4. Select **Generate new token**.
5. Provide a name for your PAT, such as *myPersonalAccessToken*
6. Select the checkbox next to **public\_repo**.



- GitHub Apps
- OAuth Apps
- Personal access tokens

## New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

### Note

myPersonalAccessToken

What's this token for?

### Expiration \*

7 days The token will expire on Wed, Apr 13 2022

### Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

- |   |                                      |
|---|--------------------------------------|
| <input type="checkbox"/> repo                   | Full control of private repositories |
| <input type="checkbox"/> repo:status            | Access commit status                 |
| <input type="checkbox"/> repo:deployment        | Access deployment status             |
| <input checked="" type="checkbox"/> public_repo | Access public repositories           |
| <input type="checkbox"/> repo:invite            | Access repository invitations        |
| <input type="checkbox"/> security_events        | Read and write security events       |

## Install Helm

In this exercise, you use Helm version v3.3.1. Azure has a built action that downloads and installs Helm.

- Below the runs-on key, add a new steps key. Then, search for **Helm tool installer**. Select the first result published by **Azure**.

deploy:

runs-on: ubuntu-20.04

needs: build\_push\_image

steps:

- uses: actions/checkout@v2

- name: Helm tool installer

uses: Azure/setup-helm@v1

with:

# Version of helm

version: # default is latest

## Helm tool installer

By Azure  v3.3  77

Install a specific version of helm binary. Acceptable values are latest or any semantic version string like 1.15.0

[View full Marketplace listing](#)

### Installation

Copy and paste the following snippet into your `.yaml` file.

Version: v3.3 ▾



1. To commit the changes, select the **Start commit** button. Enter a description for the commit, and then select **Commit new file**.
2. In Cloud Shell, run `git pull` to fetch the latest changes. Then, run the following command to tag and push the changes:

BashCopy

```
git tag -a v2.0.1 -m 'Creating first production deployment' && git push --tags
```

3. When prompted, provide your GitHub username, and the PAT created previously as the password.
4. Open the **Actions** tab and see the running process.

### CHALLENGES FACED:

1. **Initial Configuration Complexity:** Setting up AKS to meet project requirements initially posed a challenge. Detailed documentation from Microsoft and reference materials from various GitHub repositories were crucial resources.
2. **Multi-Cloud Deployment Complexity:** Deploying applications across multiple cloud infrastructures increases the complexity of managing Kubernetes clusters. Coordinating networking and storage across different environments can be challenging.

3. **Storage Management:** Handling storage can be a significant challenge, especially without relying on a cloud service. Kubernetes requires efficient storage solutions, and configuring this can be complex.
4. **Security Concerns:** Kubernetes is an open-source tool, which means security may not be as robust as desired. Ensuring the security of containerized applications and the cluster itself can be a challenge.
5. **Tooling and Interface Transition:** Transitioning from one environment to another, such as moving from CLI to PowerShell, can be challenging. Different environments offer varying features and require adaptation, adding complexity to DevOps processes

## BUSINESS BENEFITS

Kubernetes offers robust load balancing capabilities for efficient traffic distribution.

It facilitates declarative configuration and automation, reducing the risk of human errors in managing complex deployments.

Implementing CI/CD with GitHub Actions and AKS brought several significant business benefits. It streamlined and automated the development pipeline, reducing manual image builds and deployments, which saved valuable time. This increased efficiency translated into cost savings, as the company no longer needed to allocate resources for time-consuming manual tasks. Furthermore, the solution improved the overall agility of the development process, enabling quicker feature releases and enhancing the company's competitiveness in the market.