**🔍 What is a Helm Chart?**

A **Helm Chart** is a collection of **YAML templates** that describe a related set of Kubernetes resources (like Deployments, Services, ConfigMaps, etc.).

Each chart includes:

* Chart.yaml: Metadata about the chart (name, version, etc.)
* values.yaml: Default configuration values for the templates
* templates/: The actual Kubernetes manifest templates
* Optional charts/: Dependencies (sub-charts)

**✅ Why Do We Use Helm Charts?**

Here are the main reasons Helm Charts are used:

**1. Simplifies Kubernetes Deployment**

Instead of writing many Kubernetes YAML files manually, you can package them into a reusable chart.

**2. Templating Support**

You can create dynamic YAML files using Go templating with variables ({{ .Values.<key> }}), making charts reusable and configurable.

**3. Version Control**

Charts can be versioned and stored in repositories, making it easy to roll back to previous application versions.

**4. Easier Configuration**

Users can customize deployments by overriding values in values.yaml without changing the core template files.

**5. Dependency Management**

Helm allows you to define and manage application dependencies (other charts) cleanly.

**6. Consistent Deployments**

Using charts ensures that your applications are deployed consistently across environments (dev, staging, prod).

**🔧 Example Use Case**

Let’s say you want to deploy a PostgreSQL database on Kubernetes.

Without Helm:

* Write multiple YAML files for Deployment, Service, PVC, etc.
* Manually maintain and update them

With Helm:

* Use an existing chart (bitnami/postgresql)
* Configure just the values you need (helm install my-db bitnami/postgresql --set ...)

**Summary**

| **Feature** | **Helm Chart Benefit** |
| --- | --- |
| Reusability | One chart, many environments |
| Customizability | Override values per deployment |
| Complexity Management | Handles multiple Kubernetes objects |
| Deployment Speed | Single command to deploy whole app stack |

Let me know if you want a visual breakdown of the chart structure or a simple demo YAML example.

**You said:**

i need complete workflow from dockerfile to aks deployment

**ChatGPT said:**

Great! Here's a **complete workflow** from writing a Dockerfile to deploying your application on **Azure Kubernetes Service (AKS)** using **Helm**. This will give you the full CI/CD-style deployment pipeline — from development to production.

**🚀 Workflow: From Dockerfile to AKS Deployment Using Helm**

**📦 1. Create a Dockerfile**

This defines how your application is containerized.

# Dockerfile

FROM node:18

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["npm", "start"]

**🧪 2. Build and Test Docker Image Locally**

docker build -t myapp:latest .

docker run -p 3000:3000 myapp

Make sure everything runs fine locally.

**📤 3. Push Image to Azure Container Registry (ACR)**

**🔧 Login to ACR**

az login

az acr login --name <your-acr-name>

**🏷️ Tag and Push**

docker tag myapp:latest <your-acr-name>.azurecr.io/myapp:1.0.0

docker push <your-acr-name>.azurecr.io/myapp:1.0.0

**📁 4. Create Helm Chart**

Create the Helm chart structure:

helm create myapp-chart

This will generate:

myapp-chart/

├── Chart.yaml

├── values.yaml

├── templates/

│ ├── deployment.yaml

│ ├── service.yaml

│ └── ...

**✏️ Edit values.yaml to include your image**

image:

repository: <your-acr-name>.azurecr.io/myapp

tag: "1.0.0"

pullPolicy: IfNotPresent

service:

type: LoadBalancer

port: 80

containerPort: 3000

**✏️ Update deployment.yaml template to use values**

In templates/deployment.yaml, modify:

containers:

- name: {{ .Chart.Name }}

image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"

imagePullPolicy: {{ .Values.image.pullPolicy }}

ports:

- containerPort: {{ .Values.containerPort }}

**☸️ 5. Create Azure Kubernetes Cluster (AKS)**

az aks create \

--resource-group myResourceGroup \

--name myAKSCluster \

--node-count 2 \

--enable-addons monitoring \

--generate-ssh-keys

**🔐 6. Connect AKS to ACR**

az aks update \

--name myAKSCluster \

--resource-group myResourceGroup \

--attach-acr <your-acr-name>

**🔗 7. Get AKS Credentials**

az aks get-credentials --resource-group myResourceGroup --name myAKSCluster

Now your local kubectl is configured to interact with AKS.

**📦 8. Deploy to AKS with Helm**

From inside your myapp-chart/ directory:

helm install myapp-release .

You can override values like image tag with:

helm install myapp-release . \

--set image.tag=1.0.0

**✅ 9. Verify Deployment**

kubectl get all

kubectl get svc

Find the external IP from the LoadBalancer service to access your app.

**🔄 10. Upgrade (Optional)**

If you push a new image version:

docker tag myapp:latest <your-acr-name>.azurecr.io/myapp:1.0.1

docker push <your-acr-name>.azurecr.io/myapp:1.0.1

helm upgrade myapp-release . --set image.tag=1.0.1

**🧹 11. Cleanup (Optional)**

helm uninstall myapp-release

az aks delete --name myAKSCluster --resource-group myResourceGroup --yes

**✅ Summary**

| **Step** | **Description** |
| --- | --- |
| 1 | Create Dockerfile |
| 2 | Build & Test locally |
| 3 | Push to ACR |
| 4 | Create Helm chart |
| 5 | Deploy AKS cluster |
| 6 | Link ACR to AKS |
| 7 | Install app using Helm |
| 8 | Access via LoadBalancer |
| 9 | Upgrade using Helm |

Perfect! Let's create a **complete GitHub repo structure** with a **GitHub Actions CI/CD pipeline** that:

1. Builds your Docker image from a Dockerfile
2. Pushes the image to **Azure Container Registry (ACR)**
3. Deploys (or upgrades) your app to **Azure Kubernetes Service (AKS)** using **Helm**

**📁 Project Structure**

Here’s the suggested folder structure:

myapp/

├── .github/

│ └── workflows/

│ └── ci-cd.yaml <-- GitHub Actions pipeline

├── Dockerfile <-- Builds app image

├── helm/

│ └── myapp-chart/ <-- Helm chart for deployment

│ ├── Chart.yaml

│ ├── values.yaml

│ └── templates/

│ ├── deployment.yaml

│ └── service.yaml

├── src/ <-- Your actual app code (Node.js, Python, etc.)

├── package.json (if Node)

└── README.md

**⚙️ GitHub Actions CI/CD Pipeline (ci-cd.yaml)**

Here's a basic pipeline file: .github/workflows/ci-cd.yaml

name: Build and Deploy to AKS

on:

push:

branches:

- main

env:

ACR\_NAME: youracrname

ACR\_LOGIN\_SERVER: youracrname.azurecr.io

AKS\_CLUSTER\_NAME: your-aks-name

AKS\_RESOURCE\_GROUP: your-resource-group

IMAGE\_NAME: myapp

HELM\_RELEASE\_NAME: myapp-release

jobs:

build-and-deploy:

runs-on: ubuntu-latest

steps:

- name: Checkout Code

uses: actions/checkout@v3

- name: Azure Login

uses: azure/login@v1

with:

creds: ${{ secrets.AZURE\_CREDENTIALS }}

- name: Docker Login to ACR

run: |

az acr login --name $ACR\_NAME

- name: Build and Push Docker Image

run: |

IMAGE\_TAG=${{ github.sha }}

docker build -t $ACR\_LOGIN\_SERVER/$IMAGE\_NAME:$IMAGE\_TAG .

docker push $ACR\_LOGIN\_SERVER/$IMAGE\_NAME:$IMAGE\_TAG

echo "IMAGE\_TAG=$IMAGE\_TAG" >> $GITHUB\_ENV

- name: Set AKS context

run: |

az aks get-credentials --resource-group $AKS\_RESOURCE\_GROUP --name $AKS\_CLUSTER\_NAME

- name: Deploy via Helm

run: |

helm upgrade --install $HELM\_RELEASE\_NAME ./helm/myapp-chart \

--set image.repository=$ACR\_LOGIN\_SERVER/$IMAGE\_NAME \

--set image.tag=${{ env.IMAGE\_TAG }} \

--set image.pullPolicy=Always

**🔐 Required GitHub Secrets**

Go to your repo → **Settings** → **Secrets and variables** → **Actions → New repository secret**

You need to add:

| **Secret Name** | **Description** |
| --- | --- |
| AZURE\_CREDENTIALS | JSON from az ad sp create-for-rbac for GitHub to authenticate to Azure |

Generate like this:

az ad sp create-for-rbac \

--name github-deploy \

--role contributor \

--scopes /subscriptions/<your-subscription-id> \

--sdk-auth

Copy the JSON output and add to GitHub as AZURE\_CREDENTIALS.

**📝 values.yaml Example (Helm)**

image:

repository: youracrname.azurecr.io/myapp

tag: "latest"

pullPolicy: IfNotPresent

service:

type: LoadBalancer

port: 80

containerPort: 3000

**🚀 How It Works**

1. You push code to the main branch.
2. GitHub Actions:
   * Logs into Azure
   * Builds your Docker image
   * Pushes it to ACR
   * Deploys or updates the app in AKS using Helm

**✅ Result**

With this setup, your deployment is fully automated. Just **git push**, and your app is live in AKS 🎉