Kubernetes Manifest files using Helm Chart Guide

What is Helm?

Helm is a **package manager for Kubernetes** that makes deploying, managing, and upgrading applications easier. It helps you automate Kubernetes deployments instead of manually writing multiple YAML files.

What is a Helm Chart?

A **Helm Chart** is a **pre-configured package** containing all the Kubernetes resources (like Pods, Services, Deployments) needed to run an application. It simplifies Kubernetes application management.

Helm is beneficial for Kubernetes deployments because it simplifies, automates, and manages the deployment process effectively. Here are its key benefits:

1. Simplifies Kubernetes Deployments

- Instead of manually writing multiple Kubernetes YAML files for Deployment, Service, ConfigMap, etc., Helm packages everything into a **single chart**.

You can install an application with just one command:
helm install my-app ./my-chart

2. Manages Complex Applications

- Helm allows you to deploy and manage complex applications with multiple microservices in a structured way.

- Example: Deploying a **Flask app** along with a **PostgreSQL database** using a single Helm chart.

3. Uses Templates for Reusability

- Instead of hardcoding values in Kubernetes YAML files, Helm uses templates with placeholders that can be customized through values.yaml.

Example:

yaml

image:

repository: {{ .Values.app.image }}

- This makes it easy to modify configurations without changing the entire YAML file.

4. Version Control and Rollbacks

Helm allows **versioned releases** so you can easily roll back to a previous version if something goes wrong:
helm rollback my-app 1

5. Easy Configuration Management

Instead of modifying multiple Kubernetes YAML files, you can just update values.yaml and upgrade your deployment:
helm upgrade my-app ./my-chart

6. Dependency Management

- Helm can **manage dependencies** for complex applications (e.g., your app depends on PostgreSQL, Redis).

- Dependencies are defined in the Chart.yaml file and automatically installed.

7. Sharing and Reusability

- Helm charts can be **shared and reused** across multiple environments.

You can pull pre-built charts from **Helm repositories** like Bitnami:

helm repo add bitnami https://charts.bitnami.com/bitnami

helm install my-db bitnami/postgresql

8. Works Well with CI/CD

- Helm integrates well with **Jenkins, GitHub Actions, ArgoCD**, making automated deployments smoother.

- A CI/CD pipeline can use Helm to deploy applications in a Kubernetes cluster.

9. Security and Consistency

- Helm helps maintain **consistent deployments** across different environments (Dev, QA, Prod).

- You can enforce security policies using **Helm chart signing**.

10. Uninstallation is Easy

Removing an application is as simple as:
helm uninstall my-app

- It removes all related Kubernetes resources cleanly.

Conclusion

Helm makes Kubernetes application deployment **faster, more maintainable, and scalable**. It reduces complexity, promotes consistency, and integrates well with CI/CD pipelines.

Helm Chart Structure

A Helm Chart contains:

- **Chart.yaml** – Metadata (name, version, description).

- **values.yaml** – Default configuration settings.

- **templates/** – Kubernetes manifest files with placeholders.

- **charts/** – Optional dependencies for the chart.

Example Use Case

Instead of creating separate YAML files for:
✅ **Deployment** (How the app runs)
✅ **Service** (How the app is exposed)
✅ **ConfigMaps** (Application settings)

A Helm chart **bundles them together**, allowing deployment with a single command:

- helm install my-app ./my-chart

Project

Python with Flask and **Kubernetes with Helm**, here's how you can modify the steps to suit your new requirements.

1. Prerequisites

Ensure you have the following:

- A Kubernetes cluster (you can use Minikube, Kind, or any cloud-based Kubernetes service).
- Helm CLI installed.
- Docker image for the Python Flask application (you will create a simple "Hello World" Flask app).

 # Update package list and install Docker

sudo apt update

sudo apt install -y docker.io

sudo systemctl enable --now docker

 # Verify Docker installation

docker --version

 # Download and install kind

curl -Lo ./kind "https://kind.sigs.k8s.io/dl/latest/kind-linux-amd64"

chmod +x ./kind

sudo mv ./kind /usr/local/bin/kind

 # Verify kind installation

kind version

 # Create a kind cluster

kind create cluster --name my-cluster

 # Verify the cluster status

kubectl cluster-info --context kind-my-cluster

Helm Installation

sudo snap install helm --classic

helm –version

2. Project Directory Structure

Create a directory structure like this:

```
simple-flask-helm/
│
├── flask-app-chart/
│   ├── Chart.yaml
│   ├── values.yaml
│   ├── templates/
│   │   ├── deployment.yaml
│   │   ├── service.yaml
│
├── .github/workflows/
│   ├── ci-cd.yml
│
├── Jenkinsfile
├── Dockerfile
├── app.py
├── requirements.txt
└── README.md
```

3. Create a Simple Flask App

In the simple-flask-helm/ directory, create a basic Flask app. Here’s a simple app.py file for the "Hello World" app:

python

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=3000)
```

---

4. Dockerfile for Flask App

Step 1. In the same directory, create a Dockerfile for the Flask app:

dockerfile

```
FROM python:3.9-slim

WORKDIR /usr/src/app

COPY requirements.txt ./

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 3000

CMD ["python", "app.py"]
```

Step 2: Create a requirements.txt file

If you don't have a requirements.txt file, create one and add your Flask dependencies:

```
Flask
```

Step 3: Build the Docker Image

Run the following command in the same directory where your Dockerfile is located:

```
docker build -t your-dockerhub-username/flask-app:latest .
```

Replace your-dockerhub-username with your actual Docker Hub username.

Step 4: Log in to Docker Hub

Authenticate with your Docker Hub account:

docker login

Enter your **Docker Hub username** and **password** when prompted.

Step 5: Push the Docker Image to Docker Hub

After successfully logging in, push your image:

docker push your-dockerhub-username/flask-app:latest

Step 6: Run the Flask App Container (Optional Test Before Pushing)

To test the container locally, run:

docker run -p 3000:3000 your-dockerhub-username/flask-app:latest

Your Flask app should now be accessible at http://localhost:3000.

5. Helm Chart Structure

Now, let's create a Helm chart for the Flask app. Helm charts are used to define how Kubernetes resources like deployments and services should be configured.

Chart.yaml

Contains metadata about your Helm chart.

apiVersion: v2

name: flask-app

description: A simple Flask app

version: 0.1.0

---

values.yaml in Helm Chart

values.yaml is a configuration file used in Helm charts to define settings for your application, such as the **Docker image**, **replicas**, **resource limits**, and other parameters. Instead of hardcoding values in Kubernetes manifests like deployment.yaml, service.yaml, ingress.yaml etc, values.yaml allows flexibility by making configurations easily adjustable.

How values.yaml Works in Helm

1. **Overrides Default Values**: Helm uses values.yaml to customize the chart configuration.

2. **Separation of Concerns**: Instead of modifying Kubernetes YAML files directly, you can define all dynamic values in values.yaml.

3. **Easier Updates**: Changing configurations doesn't require modifying multiple files—just update values.yaml and upgrade your Helm release.

---

values.yaml

Configuration file where you define settings for the app (like Docker image, replicas, etc.).

yaml

flaskApp:

```
  name: flask-app
  image:
    repository: your-dockerhub-username/flask-app
    tag: latest
  replicaCount: 2
```

---

service:

```
  type: LoadBalancer
  port: 3000
```

---

deployment.yaml

Defines how the Flask app should be deployed on Kubernetes.

yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.flaskApp.name }}
spec:
  replicas: {{ .Values.flaskApp.replicaCount }}
  selector:
```

```
  matchLabels:
    app: {{ .Values.flaskApp.name }}
 template:
  metadata:
    labels:
      app: {{ .Values.flaskApp.name }}
  spec:
    containers:
      - name: {{ .Values.flaskApp.name }}
        image: "{{ .Values.flaskApp.image.repository }}:{{
.Values.flaskApp.image.tag }}"
        ports:
          - containerPort: 3000
```

---

service.yaml

Exposes the Flask app using a LoadBalancer service.

yaml

```
apiVersion: v1
kind: Service
metadata:
 name: {{ .Values.flaskApp.name }}
spec:
```

```
  selector:
    app: {{ .Values.flaskApp.name }}
  ports:
    - protocol: TCP
      port: {{ .Values.service.port }}
      targetPort: 3000
  type: {{ .Values.service.type }}
```

6. Deploy the Application

Package the Helm Chart

To package the Helm chart, run:

helm package ./flask-app-chart

Install the Helm Chart

Install the Helm chart into your Kubernetes cluster:

helm install flask-app ./flask-app-chart

Verify the Deployment

After the installation, check if the pods and services are running:

kubectl get pods

kubectl get services

Access the Flask App

If you're using a cloud-based Kubernetes cluster, the LoadBalancer service will expose the app with an external IP. Use kubectl get services to find the external IP and visit it in your browser. If you're using Minikube or Kind, use port forwarding:

kubectl port-forward service/flask-app 3000:3000

Then, go to http://localhost:3000 in your browser.

kubectl port-forward service/flask-app 3000:3000

 # If port 3000 is already in use, use a different port

kubectl port-forward service/flask-app 8010:3000

 # Open in browser: http://localhost:8010

Conclusion

This project helps to deploy a simple Flask application on Kubernetes using Helm. It’s a great way to understand how Helm can simplify the deployment process and make it reusable. You can scale the application by changing the replica count or customize the Docker image as needed.

CI/CD Integration

GitHub Actions Workflow

Create .github/workflows/ci-cd.yml:

```yaml
name: CI/CD Pipeline

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Code
        uses: actions/checkout@v2

      - name: Set up Python
        uses: actions/setup-python@v3
        with:
          python-version: '3.9'

      - name: Install dependencies
```

```
      run: pip install -r requirements.txt

    - name: Run Tests
      run: python -m unittest discover -s tests

    - name: Log in to Docker Hub
      run: echo "${{ secrets.DOCKERHUB_PASSWORD }}" | docker login -u "${{ secrets.DOCKERHUB_USERNAME }}" --password-stdin

    - name: Build & Push Docker Image
      run: |
        docker build -t your-dockerhub-username/flask-app:latest .
        docker push your-dockerhub-username/flask-app:latest

  deploy:
    needs: build
    runs-on: ubuntu-latest

    steps:
    - name: Install Kubernetes tools
      run: |
        curl -sL https://get.helm.sh/helm-v3.7.0-linux-amd64.tar.gz | tar xz
```

```
        sudo mv linux-amd64/helm /usr/local/bin/


    - name: Deploy with Helm
      run: |
        kubectl config set-cluster my-cluster --server=${{ secrets.KUBE_SERVER
}}
        kubectl config set-credentials my-user --token=${{ secrets.KUBE_TOKEN
}}
        kubectl config set-context my-context --cluster=my-cluster --user=my-user
        kubectl config use-context my-context
        helm upgrade --install flask-app ./flask-app-chart --set
flaskApp.image.tag=latest
```

Jenkins Pipeline (Jenkinsfile)

groovy

```
pipeline {
  agent any

  environment {
    DOCKER_CREDENTIALS_ID = 'docker-hub-credentials'
```

```
    KUBE_CONTEXT = 'kind-my-cluster'
  }

  stages {
    stage('Checkout Code') {
      steps {
        git 'https://github.com/yourusername/simple-flask-helm.git'
      }
    }

    stage('Build & Push Docker Image') {
      steps {
        script {
          docker.withRegistry('https://index.docker.io/v1/', DOCKER_CREDENTIALS_ID) {
            def appImage = docker.build("your-dockerhub-username/flask-app:latest")
            appImage.push()
          }
        }
      }
    }
```

```
        stage('Deploy to Kubernetes') {
            steps {
                sh '''
                kubectl config use-context ${KUBE_CONTEXT}
                helm upgrade --install flask-app ./flask-app-chart --set
flaskApp.image.tag=latest
                '''
            }
        }
    }
}
```

---

7. Verification

After deployment:

kubectl get pods

kubectl get services

If using Kind, forward ports:

kubectl port-forward service/flask-app 3000:3000

Visit: **http://localhost:3000**

Conclusion

This setup integrates: ✅ Flask app
✅ Docker containerization
✅ Helm for Kubernetes deployment
✅ CI/CD using GitHub Actions & Jenkins

This automates deployment whenever new code is pushed. 🚀

Advanced Helm chart.

Step-by-step guide for setting up an advanced Helm chart.

🛠️ Step 1: Create a Helm Chart

Run the following command to create a Helm chart:

helm create flask-app-chart

This creates a folder flask-app-chart/ with a default structure.

📝 Step 2: Update values.yaml

Modify values.yaml to include **custom settings for advanced features** like **HPA, PVC, ConfigMaps, Secrets, RBAC, Network Policies, etc.**

✅ values.yaml Example

yaml

```
# Image settings
image:
  repository: your-dockerhub-username/flask-app
  tag: latest
  pullPolicy: IfNotPresent

# Replica count
replicaCount: 2

# Service settings
service:
  type: ClusterIP
  port: 3000
```

```
# Ingress settings
ingress:
  enabled: true
  host: flask-app.example.com
```

 # Horizontal Pod Autoscaler (HPA)

hpa:

  enabled: true

  minReplicas: 2

  maxReplicas: 5

  cpuThreshold: 75

 # Persistent Volume Claim (PVC)

pvc:

  enabled: true

  storage: 1Gi

 # Database Secret

database:

```
  password: "mysecurepassword"
```

 # Pod Disruption Budget

```
pdb:
  enabled: true
  minAvailable: 1
```

 # RBAC Configuration

```
rbac:
  enabled: true
```

 # Network Policy Configuration

```
networkPolicy:
  enabled: true
```

Step 3: Create Advanced YAML Resources in templates/ Folder

📂1️⃣deployment.yaml – Defines the Flask App Deployment

yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Release.Name }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
        - name: flask-app
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          ports:
            - containerPort: 3000
```

🔹 **Benefits**

✔️ **Ensures application availability**: Runs multiple replicas for high availability.
✔️ **Auto-healing**: Kubernetes restarts failed containers automatically.
✔️ **Easy updates**: Helm upgrades the deployment when values.yaml changes.

📂2️⃣service.yaml – Exposes the Flask App

yaml

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Release.Name }}
spec:
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: 3000
  selector:
    app: flask-app
```

🔹 **Benefits**

✔️ **Enables communication between pods**: Internal and external services can reach your app.
✔️ **Abstracts pod IPs**: A stable service name is used instead of changing pod IPs.
✔️ **Supports different types**: ClusterIP (internal), NodePort (external access), or LoadBalancer.

📂3️⃣ingress.yaml – Manages External Traffic

yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: {{ .Release.Name }}-ingress
```

```
spec:
  rules:
    - host: {{ .Values.ingress.host }}
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: {{ .Release.Name }}
                port:
                  number: {{ .Values.service.port }}
```

🔹 **Benefits**

✔️ **Allows domain-based routing**: Maps requests from flask-app.example.com to your app.
✔️ **Reduces cost**: Uses a single LoadBalancer for multiple services.
✔️ **Supports HTTPS**: TLS certificates can be added for security.

📂4️⃣ **hpa.yaml – Auto-Scales the App**

yaml

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: {{ .Release.Name }}-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: {{ .Release.Name }}
```

```
  minReplicas: {{ .Values.hpa.minReplicas }}
  maxReplicas: {{ .Values.hpa.maxReplicas }}
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: {{ .Values.hpa.cpuThreshold }}
```

🔹 **Benefits**

✔️ **Automatically scales pods** when CPU usage increases.
✔️ **Optimizes cost**: Uses fewer pods when traffic is low.
✔️ **Ensures performance**: Prevents app crashes during high load.

📂5️⃣ secret.yaml – Stores Sensitive Data Securely

yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: {{ .Release.Name }}-secret
type: Opaque
data:
  DATABASE_PASSWORD: {{ .Values.database.password | b64enc }}
```

🔹 **Benefits**

✔️ **Keeps sensitive data safe**: Stores passwords, API keys, etc., securely.
✔️ **Prevents hardcoding**: Avoids exposing secrets in deployment files.
✔️ **Encapsulates encryption**: Kubernetes Secrets are encrypted in the cluster.

📂6️⃣ pvc.yaml – Persistent Storage for Flask App

yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: {{ .Release.Name }}-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: {{ .Values.pvc.storage }}
```

🔹 **Benefits**

✔️ **Prevents data loss**: Data is saved even if the pod restarts.
✔️ **Supports stateful apps**: Good for databases, logs, and file storage.
✔️ **Improves performance**: Can use SSD-based Persistent Volumes.

📂7️⃣ pdb.yaml – Ensures App Availability During Node Failures

yaml

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: {{ .Release.Name }}-pdb
spec:
  minAvailable: {{ .Values.pdb.minAvailable }}
  selector:
```

```
  matchLabels:
    app: flask-app
```

🔹 **Benefits**

✔️ **Prevents downtime**: Ensures at least one pod is always running.
✔️ **Helps during maintenance**: Ensures safe node upgrades.

📂8️⃣ networkpolicy.yaml – Restricts Unauthorized Access

yaml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: {{ .Release.Name }}-network-policy
spec:
  podSelector:
    matchLabels:
      app: flask-app
  policyTypes:
    - Ingress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              role: backend
      ports:
        - protocol: TCP
          port: 3000
```

🔹 **Benefits**

✔️ **Prevents unauthorized access**: Blocks unwanted external traffic.
✔️ **Improves security**: Allows only specific services to talk to the app.
✔️ **Reduces attack surface**: Blocks unnecessary connections.

📂9️⃣ role.yaml & rolebinding.yaml – Implements RBAC Security

📌 role.yaml – Defines Permissions

yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: flask-app-role
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "watch", "list"]
```

📌 rolebinding.yaml – Assigns Role to Service Account

yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: flask-app-rolebinding
subjects:
  - kind: ServiceAccount
    name: default
roleRef:
  kind: Role
  name: flask-app-role
  apiGroup: rbac.authorization.k8s.io
```

🔹 **Benefits**

✔️ **Restricts access**: Only allows certain users or services to interact with the app.
✔️ **Enhances security**: Limits damage from unauthorized actions.

🎯 **Summary of YAML File Benefits**

YAML File	Purpose	Key Benefits
deployment.yaml	Deploys Flask App	Ensures high availability & auto-recovery
service.yaml	Exposes the app	Enables communication between pods & services
ingress.yaml	Manages external access	Supports domain-based routing & TLS
hpa.yaml	Auto-scales pods	Prevents crashes due to high load
secret.yaml	Stores sensitive data	Keeps passwords & API keys safe
pvc.yaml	Persistent storage	Prevents data loss after pod restarts
pdb.yaml	Pod disruption control	Ensures uptime during node failures
networkpolicy.yaml	Restricts network access	Prevents unauthorized communication
role.yaml & rolebinding.yaml	Implements RBAC	Controls access to Kubernetes resources

---

🎯 **Summary of Steps**

✅ **Step 1: Create Helm chart using helm create flask-app-chart.**

✅ **Step 2: Update values.yaml with required configurations.**

✅ **Step 3: Create advanced YAML templates inside templates/.**

✅ **Step 4: Install the Helm chart using helm install flask-app ./flask-app-chart -f values.yaml.**

✅ **Step 5: Upgrade using helm upgrade or**

✅ **Step 6:Uninstall using helm uninstall.**

Helm Commands

helm install <release-name> <chart>: Install an app (called a "chart") with a custom name.

helm upgrade <release-name> <chart>: Update an app to a new version.

helm uninstall <release-name>: Remove an app from your Kubernetes cluster.

helm repo add <repo-name> <repo-url>: Add a new source to get Helm charts.

helm repo update: Refresh the list of charts from all added sources.

helm create <chart-name>: Create a new chart template for your app.

helm package <chart-directory>: Package your chart into a .tgz file for sharing.

helm lint <chart-directory>: Check your chart for errors or issues.

helm show values <chart>: View the default settings for a chart.

helm pull <chart>: Download a chart without installing it.

helm list: List all the apps you've installed with Helm.

helm status <release-name>: Get the status of a specific app.

helm get all <release-name>: Get all details about a specific app.

helm history <release-name>: See the version history of an app.

helm rollback <release-name> <revision>: Go back to a previous version of an app.

helm test <release-name>: Run tests to check if an app is working correctly.

helm get values <release-name>: View the settings used for a specific app.

helm upgrade --set <key=value>: Update an app with custom settings.

helm upgrade --values <file>: Upgrade an app using settings from a file.

helm template <chart>: Preview what Kubernetes resources a chart will create.

helm get manifest <release-name>: Get the raw configuration for an app.

helm get hooks <release-name>: View any special actions tied to an app.

helm repo list: List all the chart repositories you've added.

helm repo remove <repo-name>: Remove a chart repository.

helm search <repo> <chart-name>: Search for a chart in a specific repository.

helm search hub <chart-name>: Search for a chart in the Helm Hub.

helm plugin list: View all Helm plugins installed.

helm plugin install <plugin-url>: Install a Helm plugin from a URL.

helm completion: Enable autocompletion for Helm commands in the terminal.

helm version: Check which version of Helm you are using.
