**What is a Helm Chart?**

A **Helm chart** is a package that contains all the Kubernetes resources needed to deploy an application. It simplifies the process of deploying and managing applications on Kubernetes by providing pre-configured templates for resources like Pods, Services, and more.

A typical Helm chart contains the following components:

- **Chart.yaml**: Contains metadata about the chart, such as its name, version, and description.
- **values.yaml**: A configuration file where default values for the chart's parameters are set. These values can be customized during installation.
- **templates/**: A folder with Kubernetes manifest templates (written in Go templating syntax) that define the resources to be created.
- **charts/**: A folder for any dependent charts.
- **LICENSE and README.md**: Optional files that provide licensing and documentation.

---

**1. Install Helm**

Helm is a package manager for Kubernetes that simplifies application deployment and management.

**For Linux/macOS:**

```
curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee
/usr/share/keyrings/helm.gpg > /dev/null
sudo apt-get install apt-transport-https --yes
echo "deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/
all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm
```

**For Windows:** Download the installer from Helm Install Docs.

**Verify Installation:**

helm version

**2. Create a New Helm Chart**

**To create a new Helm chart, run:**

helm create my-chart

**This generates a folder structure with default files:**

- **Chart.yaml**: Chart metadata.
- **values.yaml**: Default configuration values.
- **charts/**: Folder for dependencies.
- **templates/**: Kubernetes resource templates.

---

**3. Customize the Chart's Metadata**

Edit the **Chart.yaml** file to include metadata for your chart:

yaml

```
apiVersion: v2
name: my-chart
description: A Helm chart for Kubernetes resources
version: 0.1.0
appVersion: "1.0"
```

---

**4. Modify the values.yaml File**

The **values.yaml** file contains default values that can be overridden during installation.

**Example values.yaml**:

yaml

```yaml
replicaCount: 3
image:
  repository: nginx
  tag: stable
  pullPolicy: IfNotPresent
service:
  type: ClusterIP
  port: 80
```

**Key Sections Explained:**

1. **replicaCount**: Specifies the number of application replicas (pods) for redundancy and load balancing.

2. **image**: Defines the Docker image to use, its version (tag), and pull policy.

   **repository**: This is the name of the Docker image repository. In this case, nginx is the official image for the Nginx web server, which is a popular web server and reverse proxy.

   **tag**: This specifies the version of the Docker image. stable is a tag that points to the stable version of the Nginx image. You can also specify other tags like latest, or specific version numbers like 1.19.0.

   **pullPolicy**: This defines when Kubernetes should download (or pull) the Docker image. The option IfNotPresent means Kubernetes will only pull the image if it is not already present on the node. Other options include:

a. Always: Always pull the image, even if it's already present.
b. Never: Never pull the image; Kubernetes will only use the local image

3. **Service**: Configures how the application is exposed to the network, e.g., ClusterIP for internal access.

   **type**: This defines how the service is exposed. There are several types of services:

   a. ClusterIP: This is the default type. It exposes the service only within the Kubernetes cluster (internal access).
   b. NodePort: This exposes the service on a static port on each node's IP, allowing external access to the service.
   c. LoadBalancer: This creates an external load balancer (if supported by the cloud provider) and exposes the service to the internet.

---

**5. Create Kubernetes Manifest Templates**

In the **templates/** folder, create Kubernetes resource definitions using Helm's templating syntax.

**Example deployment.yaml**:

yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Release.Name }}-deployment
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: {{ .Release.Name }}
```

```
  template:
    metadata:
      labels:
        app: {{ .Release.Name }}
    spec:
      containers:
        - name: {{ .Release.Name }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          ports:
            - containerPort: 80
```

**Example service.yaml**:

```
yaml
apiVersion: v1
kind: Service
metadata:
  name: {{ .Release.Name }}-service
spec:
  selector:
    app: {{ .Release.Name }}
  ports:
    - protocol: TCP
      port: {{ .Values.service.port }}
      targetPort: 80
  type: {{ .Values.service.type }}
```

---

**6. Install the Helm Chart**

**To install your Helm chart:**

```
helm install my-release ./my-chart
```

---

**7. View the Generated Manifests**

**To preview the manifests without deploying:**

```
helm template my-release ./my-chart > output.yaml
```

---

**8. Customize Installation with --set**

Override values during installation without editing values.yaml:

```
helm install my-release ./my-chart --set replicaCount=5 --set image.tag=1.0.0
```

---

**9. Package the Chart**

To share your Helm chart, package it into a .tgz file:

```
helm package ./my-chart
```

---

**Summary**

- **Helm Chart**: A set of files describing Kubernetes resources.
- **values.yaml**: Defines default configurations.
- **templates/**: Contains resource templates with placeholders.
- **helm install**: Deploys the chart.
- **helm template**: Generates manifests without deploying.

---

**Best Practices**

- Use **values.yaml** to manage configurations effectively.

- Organize charts logically (e.g., separate frontend and backend).
- Store Helm charts in a Git repository for version control.

---

**Exercise: Create a Redis Helm Chart**

Modify values.yaml to use:
yaml
```
replicaCount: 3
image:
  repository: redis
  tag: "6.2"
```

**Install the chart:**
```
helm install my-redis ./my-redis
kubectl get pods
```

---

**How Helm Charts Work**

- **Templates**: Use placeholders like {{ .Values.key }} to inject configurations.
- **Values**: values.yaml provides defaults, which can be overridden during installation.

---

# 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).

**2. Project Directory Structure**

Create a directory structure like this:

rust

```
simple-flask-helm/
│
├── flask-app-chart/
│   ├── Chart.yaml
│   ├── values.yaml
│   ├── templates/
│   │   ├── deployment.yaml
│   │   ├── service.yaml
│
└── Dockerfile (for Flask app)
```

**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**

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"]

**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.

yaml

apiVersion: v2

name: flask-app

description: A simple Flask app

version: 0.1.0

**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.

**Summary of Fields to Update:**

- **values.yaml**:
    - flaskApp.image.repository and flaskApp.image.tag (use your Docker Hub username and image tag).
    - flaskApp.replicaCount (number of replicas/pods for the app).
    - service.type and service.port (service type and port to expose).
- **deployment.yaml**:
    - flaskApp.image.repository and flaskApp.image.tag (use your Docker image details).
- **service.yaml**:

- service.type and service.port (service type and port to expose).

**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.

---

**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.