**HELM Theory Doc**

**Helm Detailed Documentation with Commands**

Helm is a Kubernetes package manager that simplifies the deployment, upgrade, and management of applications in Kubernetes clusters. Below is a comprehensive guide to using Helm, including installation, creating and managing charts, and common commands.

**1. Installing Helm**

**Install Helm on Linux:**

curl -fsSL https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

**Install Helm on macOS:**

brew install helm

**Install Helm on Windows:**

1. Download Helm from the [official releases page](https://github.com/helm/helm/releases).
2. Extract and add it to your PATH.

**2. Helm Basics**

**Helm Components:**

* **Chart**: A package of pre-configured Kubernetes resources.
* **Release**: A deployed instance of a chart in a Kubernetes cluster.
* **Repository**: A collection of charts.

**Helm Workflow:**

1. Create or download a Helm chart.
2. Install the chart in a Kubernetes cluster to create a release.
3. Manage releases with Helm commands.

**3. Creating a Helm Chart**

**Command to Create a New Chart:**

helm create <chart-name>

**Directory Structure:**

<chart-name>/

├── Chart.yaml # Metadata about the chart

├── values.yaml # Default values for the chart

├── templates/ # Kubernetes resource templates

│ ├── deployment.yaml

│ ├── service.yaml

│ ├── ingress.yaml

│ ├── \_helpers.tpl # Template helpers

**1. Chart.yaml**

**Description**: This file contains metadata about the Helm chart, such as the chart name, version, and description.

**Example**:

apiVersion: v2

name: bankapp

description: A Helm chart for deploying the BankApp application

type: application

version: 1.0.0

appVersion: "1.0.0"

* **apiVersion**: Defines the Helm chart API version (v2 for Helm 3).
* **name**: The name of the chart.
* **description**: A brief description of the chart.
* **type**: Type of chart (application or library).
* **version**: Version of the chart itself.
* **appVersion**: Version of the application being deployed.

**2. values.yaml**

**Description**: This file defines default values for the chart's templates. These values can be overridden during installation.

**Example**:

replicaCount: 2

image:

repository: adijaiswal/bankapp

tag: latest

pullPolicy: IfNotPresent

service:

type: LoadBalancer

port: 80

resources:

limits:

cpu: 500m

memory: 512Mi

requests:

cpu: 250m

memory: 256Mi

* **replicaCount**: Default number of replicas for a deployment.
* **image**: Configuration for the container image.
* **service**: Configuration for the Kubernetes service.
* **resources**: Resource limits and requests for the application.

**3. templates/ Directory**

This directory contains Kubernetes manifest templates. These templates are rendered using the values from values.yaml.

**a. deployment.yaml**

**Description**: Defines a Kubernetes Deployment for the application.

**Example**:

apiVersion: apps/v1

kind: Deployment

metadata:

name: {{ .Release.Name }}-bankapp

labels:

app: bankapp

spec:

replicas: {{ .Values.replicaCount }}

selector:

matchLabels:

app: bankapp

template:

metadata:

labels:

app: bankapp

spec:

containers:

- name: bankapp

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

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

resources:

limits:

cpu: {{ .Values.resources.limits.cpu }}

memory: {{ .Values.resources.limits.memory }}

requests:

cpu: {{ .Values.resources.requests.cpu }}

memory: {{ .Values.resources.requests.memory }}

ports:

- containerPort: 8080

**Key Points**:

* Templates use the {{ .Values.<key> }} syntax to access values from values.yaml.
* The .Release.Name placeholder ensures each release has a unique name.

**b. service.yaml**

**Description**: Defines a Kubernetes Service to expose the application.

**Example**:

apiVersion: v1

kind: Service

metadata:

name: {{ .Release.Name }}-bankapp

spec:

type: {{ .Values.service.type }}

ports:

- port: {{ .Values.service.port }}

targetPort: 8080

selector:

app: bankapp

**Key Points**:

* The type and port values are parameterized using values.yaml.
* The selector matches the app label from the deployment.

**c. ingress.yaml**

**Description**: Defines a Kubernetes Ingress resource to expose the application externally.

**Example**:

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: {{ .Release.Name }}-bankapp

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

rules:

- host: {{ .Values.ingress.host }}

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: {{ .Release.Name }}-bankapp

port:

number: {{ .Values.service.port }}

**Key Points**:

* The host value can be set in values.yaml for custom domain names.
* NGINX-specific annotations are used to rewrite paths.

**d. \_helpers.tpl**

**Description**: This file contains reusable template functions. These functions reduce redundancy and simplify template management.

**Example**:

{{- define "bankapp.labels" -}}

app: bankapp

chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "\_" }}

release: {{ .Release.Name }}

{{- end }}

**Usage**: This helper can be used in other templates:

yaml

Copy code

metadata:

labels:

{{- include "bankapp.labels" . | nindent 4 }}

**4. Chart Values and Overrides**

During installation or upgrade, values from values.yaml can be overridden using:

**Override Individual Values:**

helm install bankapp ./bankapp-chart --set replicaCount=3,image.tag=v1.1.0

**Override Using a Values File:**

helm install bankapp ./bankapp-chart -f custom-values.yaml

**Benefits of This Structure**

1. **Modularity**: Separate files for each resource.
2. **Reusability**: Use helpers (\_helpers.tpl) to define common patterns.
3. **Parameterization**: Easily customize deployments with values.yaml.
4. **Version Control**: Manage the chart as a single unit with a version.

**4. Helm Commands**

**1. Repositories**

**Add a Repository**:

helm repo add <repo-name> <repo-url>

**List Repositories**:

helm repo list

**Update Repository Cache**:

helm repo update

**Search for Charts**:

helm search repo <chart-name>

**2. Chart Installation**

**Install a Chart**:

helm install <release-name> <chart-name>

**Install a Chart with Custom Values**:

helm install <release-name> <chart-name> -f custom-values.yaml

**Verify Installation**:

helm list

**3. Managing Releases**

**Upgrade a Release**:

helm upgrade <release-name> <chart-name>

**Rollback a Release**:

helm rollback <release-name> <revision>

**Uninstall a Release**:

helm uninstall <release-name>

**View Release History**:

helm history <release-name>

**4. Chart Management**

**Package a Chart**:

helm package <chart-directory>

**Lint a Chart**:

helm lint <chart-directory>

**Render Templates Locally**:

helm template <chart-name>

**Push a Chart to a Repository**:

helm push <chart.tgz> <repo-name>

**5. Advanced Helm Usage**

**1. Using Values**

**Override Values During Install/Upgrade**:

helm install <release-name> <chart-name> --set key1=value1,key2=value2

**Use a Values File**:

helm install <release-name> <chart-name> -f custom-values.yaml

**2. Debugging**

**Dry-Run to Preview Deployment**:

helm install <release-name> <chart-name> --dry-run --debug

**Inspect Rendered Templates**:

helm template <chart-name>

**3. Managing Dependencies**

**Add Dependencies**: Add dependencies to Chart.yaml under dependencies:

dependencies:

- name: my-dependency

version: "1.2.3"

repository: "https://example.com/charts"

**Update Dependencies**:

helm dependency update

**6. Best Practices**

1. **Keep Charts Modular**:
   * Separate application and infrastructure charts.
   * Use dependencies for shared components.
2. **Version Control**:
   * Version charts and releases to track changes over time.
3. **Use Values Effectively**:
   * Define sane defaults in values.yaml.
   * Allow overrides for customization.
4. **Testing**:
   * Use helm lint to validate charts.
   * Perform dry-runs before deploying to production.
5. **Leverage Templates**:
   * Use \_helpers.tpl for reusable template logic.
   * Parameterize resources to increase reusability.

**Examples**

**Example 1: Install a Chart from a Repository**

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

helm install my-mysql bitnami/mysql

**Example 2: Upgrade with Custom Values**

helm upgrade my-mysql bitnami/mysql --set auth.rootPassword=mysecretpassword

**Example 3: Uninstall a Release**

helm uninstall my-mysql

**Example 4: Create a Custom Chart**

helm create mychart

cd mychart

helm install myrelease ./mychart