Kubernetes Helm – Complete Guide

What is Helm?

Helm is a **package manager for Kubernetes**—similar to how apt works for Ubuntu or yum for CentOS, but specifically designed for managing Kubernetes applications.

When deploying an application on Kubernetes, you typically need to write multiple YAML files (for deployments, services, secrets, config maps, etc.). This process can be time-consuming, repetitive, and prone to human error.

Helm simplifies this by:

Packaging everything into a single unit called a **Helm Chart**
Using variables and templates, so you can avoid repeating the same YAML across environments
Installing, upgrading, and uninstalling applications with simple Helm commands

In short, Helm streamlines and standardizes the way you define, install, and manage Kubernetes applications.

Without Helm (Manual Kubernetes Deployment)

When deploying Kubernetes apps **manually**, you deal with multiple raw YAML files:

- deployment.yaml – Application deployment
- service.yaml – Exposes the app
- configmap.yaml – Environment configs
- secret.yaml – Credentials and tokens
- ingress.yaml – Ingress routing (optional)

Problem:

For every environment (dev, staging, prod), you:

- Copy-paste the same files
- Manually edit image tags, replica counts, resource limits
- Risk introducing human errors
- Lose consistency between environments

Downsides:

- **Time-consuming**
- **Hard to maintain**
- **Error-prone**
- **Poor scalability for teams**

With Helm (Templated Deployment)

With Helm, you use a **Chart** – a reusable, parameterized package for Kubernetes apps.

Helm Chart Structure:

```
myapp-chart/
├── Chart.yaml       # Metadata (name, version)
├── values.yaml      # Default configs
└── templates/       # Templated YAMLs (Deployment, Service, etc.)
```

Benefits:

- Templates like:

  ```
  image: {{ .Values.image.repository }}:{{ .Values.image.tag }}
  replicas: {{ .Values.replicaCount }}
  ```

- No need to rewrite YAMLs for each environment

How it helps:

- Use **the same chart** for all environments
- Just change values.yaml (or override with -f)
- Deploy/upgrade/remove apps with simple CLI commands:

```
helm install myapp ./myapp-chart
helm upgrade myapp ./myapp-chart -f prod-values.yaml
helm uninstall myapp
```

Helm Components

1. Chart

A **Chart** is a Helm package. It's like a zip file containing everything Kubernetes needs to deploy your app.

What's inside:

- **templates/**: Templated Kubernetes YAML files
- **values.yaml**: Default configuration values
- **Chart.yaml**: Metadata (name, version, description)

Example:
An nginx chart may contain:

- A Deployment
- A Service
- Default values like:

```
image:
  repository: nginx
  tag: "1.25"
```

2. values.yaml

A configuration file that holds default values used by the templates.

Why it's useful:
Change just this file to deploy in different environments (dev, staging, prod) — no need to modify the actual templates.

Example:

```
replicaCount: 2
appName: myapp
image:
  repository: nginx
  tag: "1.21"
```

3. Templates

Go-templated Kubernetes YAML files (*.yaml) in the templates/ folder that generate actual manifests at runtime.

Why it's useful:
Dynamic, reusable deployments. Templates get values from values.yaml.

Example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.appName }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    spec:
      containers:
        - name: {{ .Values.appName }}
          image: {{ .Values.image.repository }}:{{ .Values.image.tag }}
```

4. Chart.yaml

Metadata file that describes your chart.

Why it's useful:
Provides identity, versioning, and ownership details.

Example:

```
apiVersion: v2
name: myapp-chart
description: A Helm chart for deploying MyApp
version: 1.0.0        # Chart version
appVersion: "1.21"    # App version
maintainers:
  - name: Alice DevOps
    email: alice@example.com
```

5. Release

A **release** is a deployed instance of a Helm chart in your Kubernetes cluster.

Why it's useful:
You can have multiple releases of the same chart (e.g., dev and prod) with different values.

Example:

```
helm install my-nginx bitnami/nginx
# "my-nginx" is the release name
```

6. Repository

A **Helm repository** stores and distributes charts — like npm (Node.js) or PyPI (Python).

Why it's useful:
You can use public repositories (like Bitnami) or host your own private chart repo.

Example:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm search repo bitnami/nginx
```

Helm Commands

Basic Helm Setup & Help

```
# Check Helm version
helm version

# Get general Helm help
helm help

# Help for a specific command
helm help install
```

Repository Management

```
# Add a Helm repository
helm repo add <repo-name> <repo-url>
helm repo add bitnami https://charts.bitnami.com/bitnami

# Update local repository cache
helm repo update

# List all added repositories
helm repo list

# Search charts in repositories
helm search repo <keyword>
helm search repo nginx
```

Chart Development / Local Work

```
# Create a new chart scaffold
helm create <chart-name>
helm create mychart

# Lint the chart for errors
helm lint <chart-directory>
helm lint mychart

# Render templates locally (without installing)
helm template <release-name> <chart-directory> -f values.yaml
helm template myrelease ./mychart -f values.yaml

# Package the chart
helm package <chart-directory>
helm package mychart

# Show chart metadata and default values
helm show chart <chart-file>
helm show chart mychart-1.0.0.tgz

helm show values <chart-file>
helm show values mychart-1.0.0.tgz
```

Installing & Managing Releases

```
# Install a release
helm install <release-name> <chart> -f values.yaml
helm install myrelease ./mychart -f values.yaml

# Upgrade a release
helm upgrade <release-name> <chart> -f values.yaml
helm upgrade myrelease ./mychart -f values.yaml

# Upgrade or install if not already present
helm upgrade --install <release-name> <chart> -f values.yaml
helm upgrade --install myrelease ./mychart -f values.yaml

# Uninstall (delete) a release
helm uninstall <release-name>
helm uninstall myrelease

# List all installed releases
helm list

# Get the status of a specific release
helm status <release-name>
helm status myrelease

# Dry-run install or upgrade
helm install --dry-run --debug <release-name> <chart>
helm install --dry-run --debug myrelease ./mychart
```

Rollback & History

```
# View release history
helm history <release-name>
helm history myrelease

# Roll back to a specific revision
helm rollback <release-name> <revision>
helm rollback myrelease 1
```

Roll back to the previous version
helm rollback <release-name>
helm rollback myrelease

Managing Dependencies (Subcharts)

Update dependencies listed in Chart.yaml
helm dependency update <chart-directory>
helm dependency update mychart

Build dependencies (create charts/ folder)
helm dependency build <chart-directory>
helm dependency build mychart

Sample app and deploy it with Helm.

Structure:

```
my-k8s-app/
├── helm-chart/
│   ├── Chart.yaml
│   ├── values.yaml
│   └── templates/
│       ├── deployment.yaml
│       └── service.yaml
├── app/
│   └── index.html
```

Step-by-Step Instructions

1. SSH into your EC2 and Setup

ssh -i your-key.pem ec2-user@<your-public-ip>

Install prerequisites (if not already):

sudo yum update -y
sudo yum install -y git curl docker

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
chmod +x kubectl && sudo mv kubectl /usr/local/bin/
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 |
bash
```

2. Ensure Kubernetes (Kops) is Ready

Make sure your Kops cluster is up:

```
kubectl get nodes
```

3. Create a Project Directory

```
mkdir my-k8s-app && cd my-k8s-app
mkdir app && echo "<h1>Hello from Helm!</h1>" > app/index.html
```

4. Create a Helm Chart

```
helm create helm-chart
```

This creates a starter chart with templates.

5. Clean the Default Files

Inside helm-chart/templates/, delete unnecessary files:

```
cd helm-chart/templates/
rm -f tests/* hpa.yaml ingress.yaml serviceaccount.yaml
```

6. Edit Chart Files

Chart.yaml

```
apiVersion: v2
name: my-k8s-app
version: 0.1.0
description: A simple HTML app deployed with Helm
```

values.yaml

```
yaml
CopyEdit
replicaCount: 2
image:
  repository: nginx
  tag: latest
  pullPolicy: IfNotPresent

  service:
    type: LoadBalancer
    port: 80

  resources: {}
```

templates/deployment.yaml

```
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: {{ .Release.Name }}-nginx
  spec:
    replicas: {{ .Values.replicaCount }}
    selector:
      matchLabels:
        app: {{ .Release.Name }}
    template:
      metadata:
        labels:
          app: {{ .Release.Name }}
      spec:
        containers:
          - name: nginx
            image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
            ports:
              - containerPort: 80
```

templates/service.yaml

```
  apiVersion: v1
  kind: Service
  metadata:
    name: {{ .Release.Name }}-service
```

```
spec:
  type: {{ .Values.service.type }}
  selector:
    app: {{ .Release.Name }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: 80
```

7. Deploy the Helm Chart

Go back to project root:

```
cd ~/my-k8s-app
helm install myapp ./helm-chart
```

8. Verify the Deployment

```
kubectl get all
kubectl get svc
```

Look for an external IP if using LoadBalancer (might take a minute).

9. Clean Up

```
helm uninstall myapp
```

Result

You've:

- Created a Helm chart
- Deployed it to your cluster
- Exposed a sample Nginx web app