

# Helm: Package Manager for Kubernetes

Managing Kubernetes Applications with Charts,  
Templates, and Repositories

# Why We Need Helm

---

**Helm** addresses the complexity of managing Kubernetes applications by providing templating, packaging, and deployment capabilities that make application management scalable and maintainable.

## Challenges with Raw Kubernetes YAML

- **Repetitive Configuration:** Similar YAML files across environments
- **Hard-coded Values:** Environment-specific values embedded in manifests
- **Complex Deployments:** Multiple interdependent resources
- **Version Management:** Tracking application versions and rollbacks
- **Configuration Drift:** Inconsistencies across environments
- **Dependency Management:** Managing application dependencies
- **Lifecycle Management:** Install, upgrade, rollback operations

## Problems Helm Solves

### Templating

Dynamic YAML generation with variables and logic for different environments

### Packaging

Bundle related Kubernetes resources into reusable packages called Charts

## Release Management

Track deployments, perform upgrades, and rollback to previous versions

## Dependency Management

Manage complex application dependencies and sub-charts

## Real-World Scenarios

- **Multi-Environment Deployments:** Same application across dev, staging, production
- **Microservices Architecture:** Managing dozens of interconnected services
- **Application Marketplace:** Sharing and distributing applications
- **CI/CD Pipelines:** Automated deployments with parameterized configurations
- **Disaster Recovery:** Quick restoration of complex application stacks

## Before and After Helm

Aspect	Without Helm	With Helm
Configuration	Multiple static YAML files	Single parameterized template
Deployment	<code>kubectl apply -f *.yaml</code>	<code>helm install myapp ./chart</code>
Updates	Manual YAML editing	<code>helm upgrade</code> with new values
Rollback	Manual restoration	<code>helm rollback myapp 1</code>

**Key Benefit:** Helm transforms Kubernetes from imperative resource management to declarative application management, making complex deployments simple and repeatable.

*Prepared By: Rashi Rana*

*Corporate Trainer*

# What is Helm

---

**Helm** is the package manager for Kubernetes, often called "the Kubernetes package manager." It helps you manage Kubernetes applications through Helm Charts, which are packages of pre-configured Kubernetes resources.

## Helm Overview

- **Package Manager:** Like apt, yum, or brew for Kubernetes applications
- **Templating Engine:** Generate Kubernetes YAML from templates
- **Release Manager:** Track and manage application deployments
- **Repository System:** Share and distribute applications
- **Dependency Manager:** Handle complex application dependencies

## Helm Architecture

Helm Client

communicates with

Kubernetes API Server

manages

**Charts**

**Releases**

**Repositories**

## Key Components

### Helm Client

Command-line tool that interacts with Kubernetes API server to manage charts and releases

### Charts

Packages containing Kubernetes resource templates and configuration

### Releases

Instances of charts deployed to Kubernetes cluster with specific configuration

### Repositories

Collections of charts that can be shared and distributed

## Helm Evolution

**Helm v2  
(Tiller)**



**Helm v3  
(Client-only)**



**Current  
(v3.12+)**

## Helm v3 Improvements

- **No Tiller:** Removed server-side component for better security
- **Namespace Support:** Better namespace handling and scoping

- **Library Charts:** Reusable chart components
- **JSON Schema:** Values validation support
- **Release Secrets:** Store release information as Kubernetes secrets
- **Improved Hooks:** Better lifecycle management

## Helm Workflow

- 1 **Create/Find Chart:** Develop chart or find existing one in repository
- 2 **Customize Values:** Configure chart parameters for your environment
- 3 **Install Release:** Deploy chart to Kubernetes cluster
- 4 **Manage Lifecycle:** Upgrade, rollback, or uninstall as needed

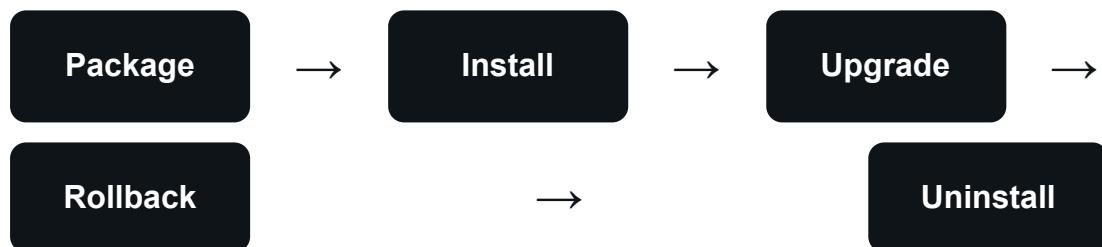
**Modern Approach:** Helm v3's client-only architecture eliminates security concerns while providing powerful templating and package management capabilities.

# Managing Kubernetes Applications with Helm

---

Helm provides a comprehensive solution for **application lifecycle management** in Kubernetes, from initial deployment through updates, scaling, and eventual decommissioning.

## Application Lifecycle with Helm



## Key Management Capabilities

### Installation

Deploy complex applications with single command, handling dependencies automatically

### Configuration

Parameterize deployments for different environments without changing templates

### Upgrades

### Rollbacks

Update applications with new versions while preserving configuration and data

Quickly revert to previous working versions when issues arise

## Release Management

Operation	Command	Description
Install	helm install myapp ./chart	Deploy new application instance
Upgrade	helm upgrade myapp ./chart	Update existing deployment
Rollback	helm rollback myapp 1	Revert to previous version
Uninstall	helm uninstall myapp	Remove application completely

## Configuration Management

- **Values Files:** Environment-specific configuration in YAML files
- **Command-line Overrides:** Quick parameter changes during deployment
- **Template Functions:** Dynamic value generation and transformation
- **Conditional Logic:** Include/exclude resources based on conditions
- **Validation:** JSON Schema validation for configuration values

## Multi-Environment Strategy

```
# Development environment
helm install myapp ./chart -f values-dev.yaml

# Staging environment
helm install myapp ./chart -f values-staging.yaml

# Production environment
helm install myapp ./chart -f values-prod.yaml

# Override specific values
helm install myapp ./chart --set image.tag=v2.0.0 --set
replicas=5
```

## Benefits for DevOps Teams

- **Standardization:** Consistent deployment patterns across teams
- **Reusability:** Share charts across projects and organizations
- **Automation:** Integrate with CI/CD pipelines seamlessly
- **Observability:** Track deployment history and changes
- **Collaboration:** Version-controlled application definitions

**Best Practice:** Always use values files for environment-specific configuration rather than modifying chart templates directly.

*Prepared By: Rashi Rana  
Corporate Trainer*

# Charts

---

**Helm Charts** are packages of pre-configured Kubernetes resources. A chart is a collection of files that describe a related set of Kubernetes resources.

## Chart Structure

```
mychart/
├── Chart.yaml # Chart metadata
├── values.yaml # Default configuration values
├── charts/ # Chart dependencies
├── templates/ # Kubernetes resource templates
│   ├── deployment.yaml
│   ├── service.yaml
│   ├── ingress.yaml
│   ├── _helpers.tpl # Template helpers
│   └── NOTES.txt # Usage notes
└── .helmignore # Files to ignore
└── README.md # Chart documentation
```

## Chart.yaml Example

```
apiVersion: v2
name: nginx-app
description: A Helm chart for Nginx application
type: application
version: 0.1.0
appVersion: "1.21.0"
keywords:
- nginx
- web
- server
```

```
home: https://nginx.org
sources:
- https://github.com/nginx/nginx
maintainers:
- name: John Doe
email: john@example.com
dependencies:
- name: postgresql
version: "11.6.12"
repository: https://charts.bitnami.com/bitnami
```

## Chart Types

### Application Charts

Complete applications with all necessary resources (deployments, services, etc.)

### Library Charts

Reusable templates and functions shared across multiple charts

### Umbrella Charts

Charts that primarily contain dependencies to other charts

### Operator Charts

Charts that deploy Kubernetes operators for complex applications

## values.yaml Example

```
# Default values for nginx-app
replicaCount: 1

image:
repository: nginx
```

```
pullPolicy: IfNotPresent
tag: "1.21"

service:
type: ClusterIP
port: 80

ingress:
enabled: false
className: ""
annotations: {}
hosts:
- host: chart-example.local
paths:
- path: /
pathType: Prefix

resources:
limits:
cpu: 500m
memory: 512Mi
requests:
cpu: 250m
memory: 256Mi

autoscaling:
enabled: false
minReplicas: 1
maxReplicas: 100
targetCPUUtilizationPercentage: 80
```

## Chart Commands

```
# Create new chart
helm create mychart

# Validate chart
helm lint mychart

# Package chart
helm package mychart
```

```
# Install chart
helm install myrelease mychart

# Dry run (template rendering)
helm install myrelease mychart --dry-run --debug

# Template only (no installation)
helm template myrelease mychart
```

## Chart Best Practices

- **Semantic Versioning:** Use proper version numbering
- **Resource Naming:** Use consistent naming conventions
- **Labels and Selectors:** Include standard labels
- **Documentation:** Provide clear README and NOTES.txt
- **Security:** Follow security best practices
- **Testing:** Include chart tests for validation

*Prepared By: Rashi Rana  
Corporate Trainer*

# Templates

---

**Helm Templates** use the Go template language to generate Kubernetes YAML files dynamically. Templates allow you to parameterize and customize Kubernetes resources.

## Template Syntax

### Variables

```
{{ .Values.image.repository }}  
}}
```

### Functions

```
{{ .Values.name | upper }}
```

### Conditionals

```
{%- if  
.Values.ingress.enabled %}
```

### Loops

```
{%- range .Values.hosts %}
```

## Deployment Template Example

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
name: {{ include "mychart.fullname" . }}  
labels:  
{%- include "mychart.labels" . | nindent 4 %}  
spec:
```

```

{{- if not .Values.autoscaling.enabled --}}
replicas: {{ .Values.replicaCount }}
{{- end --}}
selector:
matchLabels:
{{- include "mychart.selectorLabels" . | nindent 6 --}}
template:
metadata:
labels:
{{- include "mychart.selectorLabels" . | nindent 8 --}}
spec:
containers:
- name: {{ .Chart.Name }}
image: "{{ .Values.image.repository }}:{{ .Values.image.tag
| default .Chart.AppVersion }}"
imagePullPolicy: {{ .Values.image.pullPolicy }}
ports:
- name: http
  containerPort: 80
  protocol: TCP
resources:
{{- toYaml .Values.resources | nindent 12 --}}

```

## Built-in Objects

Object	Description	Example
<b>.Values</b>	Values from values.yaml and overrides	<code>{{ .Values.image.tag }}</code>
<b>.Chart</b>	Chart metadata from Chart.yaml	<code>{{ .Chart.Name }}</code>
<b>.Release</b>	Release information	<code>{{ .Release.Name }}</code>
<b>.Capabilities</b>	Kubernetes cluster capabilities	<code>{{ .Capabilities.KubeVersion }}</code>

Object	Description	Example
.Template	Current template information	<code>{{ .Template.Name }}</code>

## Template Functions

```

# String functions
{{ .Values.name | upper }} # Convert to uppercase
{{ .Values.name | quote }} # Add quotes
{{ .Values.name | default "nginx" }} # Default value

# Type conversion
{{ .Values.port | toString }} # Convert to string
{{ .Values.enabled | toYaml }} # Convert to YAML

# Conditionals
{{- if .Values.ingress.enabled --}}
# Include ingress resource
{{- end }}

# Loops
{{- range .Values.hosts --}}
- host: {{ . }}
{{- end }}

```

## Helper Templates (\_helpers.tpl)

```

{{/*
Expand the name of the chart.
*/}}
{{- define "mychart.name" -}}
{{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" --}}
{{- end }}

{{/*
Create a default fully qualified app name.
*/}}

```

```
 */ }

{{- define "mychart.fullname" -}}
{{- if .Values.fullnameOverride -}}
{{- .Values.fullnameOverride | trunc 63 | trimSuffix "-" -}}
{{- else -}}
{{- $name := default .Chart.Name .Values.nameOverride -}}
{{- if contains $name .Release.Name -}}
{{- .Release.Name | trunc 63 | trimSuffix "-" -}}
{{- else -}}
{{- printf "%s-%s" .Release.Name $name | trunc 63 | trimSuffix "-" -}}
{{- end -}}
{{- end -}}
{{- end -}}
```

## Template Best Practices

- **Use Helper Templates:** Create reusable template snippets
- **Validate Input:** Check for required values and types
- **Handle Defaults:** Provide sensible default values
- **Comment Templates:** Document complex template logic
- **Test Templates:** Use helm template to verify output

*Prepared By: Rashi Rana  
Corporate Trainer*

# Repositories

---

**Helm Repositories** are collections of Helm charts that can be shared and distributed. They serve as centralized locations for storing and accessing packaged charts.

## Repository Types

### Public Repositories

Community-maintained repositories like Bitnami, Stable, and official charts

### Private Repositories

Organization-specific repositories for internal charts and applications

### OCI Repositories

Container registry-based repositories using OCI (Open Container Initiative) format

### Local Repositories

File system-based repositories for development and testing

## Popular Public Repositories

Repository	URL	Description
<b>Bitnami</b>	<a href="https://charts.bitnami.com/bitnami">https://charts.bitnami.com/bitnami</a>	Production-ready charts for popular applications
<b>Prometheus Community</b>	<a href="https://prometheus-community.github.io/helm-charts">https://prometheus-community.github.io/helm-charts</a>	Monitoring and alerting stack charts
<b>Ingress Nginx</b>	<a href="https://kubernetes.github.io/ingress-nginx">https://kubernetes.github.io/ingress-nginx</a>	Nginx ingress controller charts
<b>Jetstack</b>	<a href="https://charts.jetstack.io">https://charts.jetstack.io</a>	cert-manager and security-focused charts

## Repository Management Commands

```
# Add repository
helm repo add bitnami https://charts.bitnami.com/bitnami

# List repositories
helm repo list

# Update repository index
helm repo update

# Search charts in repositories
helm search repo nginx

# Search Helm Hub
helm search hub wordpress

# Remove repository
helm repo remove bitnami
```

```
# Show chart information  
helm show chart bitnami/nginx  
helm show values bitnami/nginx
```

## Installing from Repositories

```
# Install from repository  
helm install my-nginx bitnami/nginx  
  
# Install specific version  
helm install my-nginx bitnami/nginx --version 13.2.23  
  
# Install with custom values  
helm install my-nginx bitnami/nginx -f my-values.yaml  
  
# Install with inline values  
helm install my-nginx bitnami/nginx \  
--set service.type=LoadBalancer \  
--set replicaCount=3
```

## Creating Private Repository

- 1 Package Charts:** Create .tgz files from your charts
- 2 Generate Index:** Create index.yaml with chart metadata
- 3 Host Files:** Serve files via HTTP server (nginx, Apache, S3, etc.)
- 4 Add Repository:** Add your repository URL to Helm

```
# Package chart  
helm package mychart  
  
# Generate repository index  
helm repo index . --url https://my-repo.example.com
```

```
# Upload to web server  
# mychart-0.1.0.tgz and index.yaml  
  
# Add private repository  
helm repo add myrepo https://my-repo.example.com
```

## OCI Registry Support

```
# Login to OCI registry  
helm registry login registry.example.com  
  
# Push chart to OCI registry  
helm push mychart-0.1.0.tgz  
oci://registry.example.com/charts  
  
# Install from OCI registry  
helm install my-release  
oci://registry.example.com/charts/mychart --version  
0.1.0
```

**Modern Approach:** OCI registries are becoming the preferred method for storing Helm charts, providing better security, versioning, and integration with existing container workflows.

*Prepared By: Rashi Rana  
Corporate Trainer*

# Helm Commands Examples

---

Master these essential **Helm commands** for effective chart and release management in your Kubernetes environment.

## Repository Management

```
# Add popular repositories
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo add ingress-nginx
https://kubernetes.github.io/ingress-nginx

# Update repository indexes
helm repo update

# List all repositories
helm repo list

# Search for charts
helm search repo nginx
helm search repo --versions nginx # Show all versions

# Get chart information
helm show chart bitnami/nginx
helm show values bitnami/nginx
helm show readme bitnami/nginx
```

## Release Management

```
# Install release
helm install my-nginx bitnami/nginx
helm install my-nginx bitnami/nginx --namespace
production --create-namespace

# Install with custom values
helm install my-nginx bitnami/nginx -f values.yaml
helm install my-nginx bitnami/nginx --set
service.type=LoadBalancer

# List releases
helm list
helm list --all-namespaces
helm list --namespace production

# Get release status
helm status my-nginx

# Get release values
helm get values my-nginx
helm get values my-nginx --all # Include default values
```

## Upgrade and Rollback

```
# Upgrade release
helm upgrade my-nginx bitnami/nginx
helm upgrade my-nginx bitnami/nginx --version 13.2.24
helm upgrade my-nginx bitnami/nginx -f new-values.yaml

# Upgrade with install fallback
helm upgrade --install my-nginx bitnami/nginx

# View release history
helm history my-nginx

# Rollback to previous version
helm rollback my-nginx
helm rollback my-nginx 2 # Rollback to specific revision

# Uninstall release
```

```
helm uninstall my-nginx
helm uninstall my-nginx --keep-history # Keep release
history
```

## Chart Development

```
# Create new chart
helm create mychart

# Validate chart
helm lint mychart

# Render templates (dry run)
helm template my-release mychart
helm template my-release mychart --debug

# Install with dry run
helm install my-release mychart --dry-run --debug

# Package chart
helm package mychart

# Test chart
helm test my-release
```

## Advanced Commands

### Dependency Management

```
helm dependency update
helm dependency build
helm dependency list
```

### Plugin Management

```
helm plugin install
<plugin-url>
helm plugin list
helm plugin update
<plugin>
```

## Environment Management

```
helm env  
export  
HELM_NAMESPACE=production  
export  
KUBECONFIG=~/.kube/config
```

## Debugging

```
helm get manifest my-  
release  
helm get hooks my-  
release  
helm get notes my-  
release
```

## Common Command Patterns

Task	Command	Use Case
Quick Install	helm install app repo/chart	Fast deployment with defaults
Production Install	helm install app repo/chart -f prod-values.yaml	Controlled production deployment
Safe Upgrade	helm upgrade app repo/chart --dry-run	Preview changes before applying
Emergency Rollback	helm rollback app	Quick revert to previous version

**Safety Tip:** Always use --dry-run flag to preview changes before applying them in production environments.

# Lab: Installing Helm

---

## Hands-on Lab: Helm Installation and Setup

Install Helm on your system and configure it for Kubernetes cluster management.

## Lab Objectives

- Install Helm CLI on your operating system
- Verify Helm installation and configuration
- Add popular Helm repositories
- Explore available charts and their configurations

## Installation Methods

### Script Installation

Quick installation using official script

### Package Manager

Install using system package managers

### Binary Download

Download and install binary manually

### Container Image

Run Helm from container image

# Step 1: Install Helm (Choose Your Method)

## Method 1: Script Installation (Recommended)

```
# Download and run installation script
curl
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

# Alternative: Download script first, then run
curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
```

## Method 2: Package Managers

```
# macOS (Homebrew)
brew install helm

# Ubuntu/Debian
curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null
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

# CentOS/RHEL/Fedora
sudo dnf install helm

# Windows (Chocolatey)
choco install kubernetes-helm
```

```
# Windows (Scoop)
scoop install helm
```

### Method 3: Binary Download

```
# Download latest release
wget https://get.helm.sh/helm-v3.12.0-linux-amd64.tar.gz

# Extract and install
tar -zxvf helm-v3.12.0-linux-amd64.tar.gz
sudo mv linux-amd64/helm /usr/local/bin/helm

# Make executable
chmod +x /usr/local/bin/helm
```

## Step 2: Verify Installation

```
# Check Helm version
helm version

# Check Helm environment
helm env

# Verify Kubernetes connection
kubectl cluster-info
helm list
```

## Step 3: Configure Helm

```
# Check Helm configuration
helm env

# Set default namespace (optional)
export HELM_NAMESPACE=default

# Enable shell completion (bash)
```

```
echo 'source <(helm completion bash)' >> ~/.bashrc
source ~/.bashrc

# Enable shell completion (zsh)
echo 'source <(helm completion zsh)' >> ~/.zshrc
source ~/.zshrc
```

*Prepared By: Rashi Rana  
Corporate Trainer*

# Lab: Repository Management

---

## Step 4: Add Popular Repositories

```
# Add Bitnami repository (popular applications)
helm repo add bitnami https://charts.bitnami.com/bitnami

# Add Prometheus Community (monitoring stack)
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts

# Add Ingress Nginx (ingress controller)
helm repo add ingress-nginx
https://kubernetes.github.io/ingress-nginx

# Add Jetstack (cert-manager)
helm repo add jetstack https://charts.jetstack.io

# Update repository indexes
helm repo update
```

## Step 5: Explore Repositories

```
# List all repositories
helm repo list

# Search for charts
helm search repo nginx
helm search repo database
helm search repo monitoring

# Search with versions
helm search repo nginx --versions

# Search Helm Hub (public registry)
```

```
helm search hub wordpress
helm search hub postgresql
```

## Step 6: Examine Chart Details

```
# Show chart information
helm show chart bitnami/nginx

# Show default values
helm show values bitnami/nginx

# Show chart README
helm show readme bitnami/nginx

# Show all chart information
helm show all bitnami/nginx
```

## Expected Output Examples

```
# helm repo list output
NAME URL
bitnami https://charts.bitnami.com/bitnami
prometheus-community https://prometheus-
community.github.io/helm-charts
ingress-nginx https://kubernetes.github.io/ingress-nginx
jetstack https://charts.jetstack.io

# helm search repo nginx output
NAME CHART VERSION APP VERSION DESCRIPTION
bitnami/nginx 13.2.23 1.25.2 NGINX Open Source is a web
server that can be a...
ingress-nginx/ingress-nginx 4.7.1 1.8.1 Ingress controller
for Kubernetes using NGINX a...
```

## Troubleshooting Common Issues

## Permission Issues

Use sudo for system-wide installation or install to user directory

## Network Issues

Check firewall settings and proxy configuration

## Kubernetes Access

Verify kubectl configuration and cluster connectivity

## Version Compatibility

Ensure Helm version is compatible with Kubernetes version

**Verification:** If you can run `helm version` and `helm repo list` successfully, your Helm installation is complete and ready for use!

*Prepared By: Rashi Rana  
Corporate Trainer*

# Lab: Install NGINX using Helm Chart

---

## Hands-on Lab: Deploy NGINX Application

Install and manage NGINX web server using Helm charts with different configurations and service types.

## Lab Objectives

- Install NGINX using Bitnami Helm chart
- Customize deployment with values files
- Perform upgrades and rollbacks
- Manage release lifecycle

## Step 1: Basic NGINX Installation

```
# Install NGINX with default values
helm install my-nginx bitnami/nginx

# Check installation status
helm status my-nginx

# List all releases
helm list

# Check created Kubernetes resources
kubectl get all -l app.kubernetes.io/instance=my-nginx
```

## Step 2: Create Custom Values File

```
# nginx-values.yaml
replicaCount: 2

image:
tag: "1.25.2"

service:
type: NodePort
nodePorts:
http: 30080

resources:
limits:
cpu: 500m
memory: 512Mi
requests:
cpu: 250m
memory: 256Mi

ingress:
enabled: true
hostname: nginx.local
path: /

metrics:
enabled: true

autoscaling:
enabled: true
minReplicas: 2
maxReplicas: 5
targetCPU: 70
```

## Step 3: Install with Custom Values

```
# Create the values file
cat > nginx-values.yaml << 'EOF'
replicaCount: 2
service:
```

```
type: NodePort
nodePorts:
http: 30080
resources:
limits:
cpu: 500m
memory: 512Mi
requests:
cpu: 250m
memory: 256Mi
EOF

# Install with custom values
helm install my-nginx-custom bitnami/nginx -f nginx-
values.yaml

# Verify installation
helm get values my-nginx-custom
kubectl get pods -l app.kubernetes.io/instance=my-nginx-
custom
```

## Step 4: Test the Deployment

```
# Get service information
kubectl get services -l app.kubernetes.io/instance=my-
nginx-custom

# Test NodePort service (if using NodePort)
kubectl get nodes -o wide
curl http://<node-ip>:30080

# Alternative: Port forward for testing
kubectl port-forward service/my-nginx-custom 8080:80

# Test from another terminal
curl http://localhost:8080
```

## Step 5: Upgrade the Release

```
# Update values file for upgrade
cat > nginx-values-v2.yaml << 'EOF'
replicaCount: 3
service:
  type: LoadBalancer
resources:
limits:
  cpu: 1000m
  memory: 1Gi
requests:
  cpu: 500m
  memory: 512Mi
EOF

# Perform upgrade
helm upgrade my-nginx-custom bitnami/nginx -f nginx-
values-v2.yaml

# Check upgrade status
helm status my-nginx-custom
helm history my-nginx-custom
```

## Step 6: Rollback and Cleanup

```
# View release history
helm history my-nginx-custom

# Rollback to previous version
helm rollback my-nginx-custom 1

# Verify rollback
kubectl get pods -l app.kubernetes.io/instance=my-nginx-
custom
helm get values my-nginx-custom

# Uninstall releases
helm uninstall my-nginx
helm uninstall my-nginx-custom

# Verify cleanup
```

```
helm list  
kubectl get all -l app.kubernetes.io/name=nginx
```

## Expected Outputs

```
# helm status output  
NAME: my-nginx-custom  
LAST DEPLOYED: Mon Oct 23 10:30:00 2023  
NAMESPACE: default  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None  
NOTES:  
NGINX can be accessed through the following DNS name from  
within your cluster:  
my-nginx-custom.default.svc.cluster.local (port 80)  
  
# kubectl get pods output  
NAME READY STATUS RESTARTS AGE  
my-nginx-custom-7d4b7b8c9d-abc12 1/1 Running 0 2m  
my-nginx-custom-7d4b7b8c9d-def34 1/1 Running 0 2m
```

**Lab Complete:** You've successfully installed, configured, upgraded, and managed NGINX using Helm charts!

*Prepared By: Rashi Rana  
Corporate Trainer*

# Lab: Write a Basic Custom Helm Chart

---

## Hands-on Lab: Create Custom Helm Chart

Build a custom Helm chart for a simple web application with deployment, service, and configurable values.

## Lab Objectives

- Create a new Helm chart from scratch
- Customize templates for deployment and service
- Configure values.yaml for parameterization
- Test and deploy the custom chart

## Step 1: Create New Chart

```
# Create new chart
helm create webapp

# Explore chart structure
tree webapp
# or
find webapp -type f

# Chart structure:
# webapp/
#   └── Chart.yaml
```

```
# └── values.yaml
# └── templates/
#   ├── deployment.yaml
#   ├── service.yaml
#   ├── ingress.yaml
#   ├── _helpers.tpl
#   └── NOTES.txt
# └── charts/
```

## Step 2: Customize Chart.yaml

```
# webapp/Chart.yaml
apiVersion: v2
name: webapp
description: A simple web application Helm chart
type: application
version: 0.1.0
appVersion: "1.0.0"
keywords:
- web
- application
- demo
home: https://github.com/myorg/webapp
maintainers:
- name: Your Name
email: your.email@example.com
```

## Step 3: Configure values.yaml

```
# webapp/values.yaml
replicaCount: 2

image:
repository: nginx
pullPolicy: IfNotPresent
tag: "1.25.2"

nameOverride: ""
fullnameOverride: ""
```

```
service:
  type: ClusterIP
  port: 80
  targetPort: 80

  ingress:
    enabled: false
    className: ""
    annotations: {}
    hosts:
      - host: webapp.local
    paths:
      - path: /
    pathType: Prefix
    tls: []

  resources:
    limits:
      cpu: 500m
      memory: 512Mi
    requests:
      cpu: 250m
      memory: 256Mi

  autoscaling:
    enabled: false
    minReplicas: 1
    maxReplicas: 100
    targetCPUUtilizationPercentage: 80

  nodeSelector: {}
  tolerations: []
  affinity: {}
```

## Step 4: Customize Deployment Template

```
# webapp/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "webapp.fullname" . }}
```

```

labels:
{{- include "webapp.labels" . | nindent 4 --}}
spec:
{{- if not .Values.autoscaling.enabled --}}
replicas: {{ .Values.replicaCount }}
{{- end --}}
selector:
matchLabels:
{{- include "webapp.selectorLabels" . | nindent 6 --}}
template:
metadata:
labels:
{{- include "webapp.selectorLabels" . | nindent 8 --}}
spec:
containers:
- name: {{ .Chart.Name }}
image: "{{ .Values.image.repository }}:{{ .Values.image.tag
| default .Chart.AppVersion }}"
imagePullPolicy: {{ .Values.image.pullPolicy }}
ports:
- name: http
containerPort: {{ .Values.service.targetPort }}
protocol: TCP
livenessProbe:
httpGet:
path: /
port: http
initialDelaySeconds: 30
periodSeconds: 10
readinessProbe:
httpGet:
path: /
port: http
initialDelaySeconds: 5
periodSeconds: 5
resources:
{{- toYaml .Values.resources | nindent 12 --}}

```

## Step 5: Test and Deploy Chart

```

# Validate chart syntax
helm lint webapp

```

```
# Render templates (dry run)
helm template my-webapp webapp

# Install with dry run
helm install my-webapp webapp --dry-run --debug

# Install the chart
helm install my-webapp webapp

# Check deployment
helm status my-webapp
kubectl get all -l app.kubernetes.io/instance=my-webapp
```

## Step 6: Test with Custom Values

```
# Create custom values file
cat > custom-values.yaml << 'EOF'
replicaCount: 3
service:
  type: NodePort
  port: 8080
  image:
    tag: "1.24.0"
  resources:
    requests:
      cpu: 100m
      memory: 128Mi
EOF

# Install with custom values
helm install my-webapp-custom webapp -f custom-
values.yaml

# Verify custom configuration
helm get values my-webapp-custom
kubectl describe deployment my-webapp-custom
```

## Step 7: Package and Share

```
# Package the chart
helm package webapp

# This creates: webapp-0.1.0.tgz

# Install from package
helm install my-webapp-pkg ./webapp-0.1.0.tgz

# Cleanup
helm uninstall my-webapp
helm uninstall my-webapp-custom
helm uninstall my-webapp-pkg
```

**Chart Created!** You've successfully created, customized, and deployed your own Helm chart. This chart can now be shared, versioned, and reused across different environments.

*Prepared By: Rashi Rana  
Corporate Trainer*

# Summary and Next Steps

---

## Key Concepts Covered

- Why Helm is essential for Kubernetes application management
- Helm architecture and core components (Charts, Releases, Repositories)
- Chart structure and templating with Go templates
- Repository management and chart distribution
- Essential Helm commands for lifecycle management
- Hands-on installation and configuration of Helm
- Deploying NGINX using existing Helm charts
- Creating custom Helm charts from scratch

## Practical Skills Gained

### Helm Installation

Multiple installation methods and environment setup

### Repository Management

Adding, updating, and searching chart repositories

## Application Deployment

Installing and managing applications with Helm

## Chart Development

Creating custom charts with templates and values

## Lab Accomplishments

- **Helm Setup:** Installed Helm and configured popular repositories
- **NGINX Deployment:** Deployed and managed NGINX with custom configurations
- **Release Management:** Performed upgrades, rollbacks, and lifecycle operations
- **Custom Chart:** Created a complete Helm chart with templates and values
- **Best Practices:** Applied Helm best practices for production use

## Advanced Helm Topics

- 1 **Advanced Templating:** Complex template functions, conditionals, and loops
- 2 **Chart Dependencies:** Managing sub-charts and dependency relationships
- 3 **Hooks and Tests:** Pre/post-install hooks and chart testing
- 4 **Security:** Chart signing, verification, and security best practices
- 5 **CI/CD Integration:** Automated chart testing and deployment pipelines
- 6 **Helm Plugins:** Extending Helm functionality with custom plugins

7

**GitOps with Helm:** ArgoCD and Flux integration patterns

8

**Multi-Environment:** Advanced strategies for environment management

## Recommended Tools and Ecosystem

- **Chart Development:** Helm lint, helm-docs, chart-testing
- **Security:** Helm secrets, SOPS, sealed-secrets
- **GitOps:** ArgoCD, Flux, Helmfile
- **Repository Hosting:** ChartMuseum, Harbor, Artifactory
- **Monitoring:** Helm dashboard, Prometheus operator charts

## Best Practices Learned

- **Values Organization:** Use structured values files for different environments
- **Template Reusability:** Create helper templates for common patterns
- **Version Management:** Follow semantic versioning for charts
- **Documentation:** Maintain clear README and NOTES.txt files
- **Testing:** Always use --dry-run before production deployments
- **Security:** Never hardcode secrets in values files

## Next Learning Path

- **Kubernetes Operators:** Advanced application management patterns
- **Service Mesh:** Istio, Linkerd with Helm integration
- **Monitoring Stack:** Prometheus, Grafana, AlertManager deployment
- **Security Hardening:** Pod Security Standards, Network Policies

- **Multi-Cluster Management:** Cluster API, fleet management

**Congratulations!** You've mastered Helm fundamentals and can now efficiently manage Kubernetes applications using charts, templates, and repositories. Continue practicing with complex applications and explore advanced Helm features for production environments.

*Prepared By: Rashi Rana  
Corporate Trainer*