COMPLETE HELM TUTORIAL FROM BASICS TO ADVANCED TEMPLATING

www.linkedin.com/in/saraswathilakshman

saraswathilakshman.medium.com

Complete Helm Tutorial: From Basics to Advanced Templating

Table of Contents

What is Helm?

Helm is often called the "package manager for Kubernetes." It's a tool that helps you manage Kubernetes applications by providing a way to define, install, and upgrade complex Kubernetes applications using packages called "charts."

Key Benefits:

- **Simplifies deployments**: Deploy complex applications with a single command
- **Version management**: Track and manage different versions of your applications
- **Configuration management**: Use templates to customize deployments for different environments
- **Dependency management**: Handle application dependencies automatically
- **Rollback capabilities**: Easily rollback to previous versions

---

Helm as a Package Manager

Just like apt for Ubuntu, yum for Red Hat, or npm for Node.js, Helm manages packages for Kubernetes. These packages are called "charts" and contain all the necessary Kubernetes resources to run an application.

Package Manager Features:

- **Search**: Find charts in repositories
- **Install**: Deploy applications to your cluster
- **Upgrade**: Update applications to newer versions
- **Uninstall**: Remove applications cleanly
- **Repository management**: Add, update, and manage chart repositories

---

Helm Charts

A Helm chart is a package containing all the necessary resource definitions to run an application, tool, or service inside a Kubernetes cluster.

Chart Structure

```
my-chart/
├── Chart.yaml          # Chart metadata
├── values.yaml         # Default configuration values
├── charts/             # Chart dependencies
├── templates/          # Kubernetes manifest templates
│   ├── deployment.yaml
│   ├── service.yaml
│   ├── ingress.yaml
│   └── _helpers.tpl    # Template helpers
└── .helmignore         # Files to ignore when packaging
```

Example Chart.yaml

```yaml
apiVersion: v2
name: my-web-app
description: A simple web application chart
type: application
version: 0.1.0
appVersion: "1.0.0"
maintainers:
  - name: Your Name
    email: your.email@example.com
keywords:
  - web
  - nginx
home: https://github.com/your-repo/my-web-app
sources:
  - https://github.com/your-repo/my-web-app
```

Example values.yaml

```yaml
# Default values for my-web-app
replicaCount: 1

image:
  repository: nginx
  pullPolicy: IfNotPresent
  tag: "1.21"

nameOverride: ""
fullnameOverride: ""
```

```
service:
  type: ClusterIP
  port: 80

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

resources:
  limits:
    cpu: 100m
    memory: 128Mi
  requests:
    cpu: 100m
    memory: 128Mi

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

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

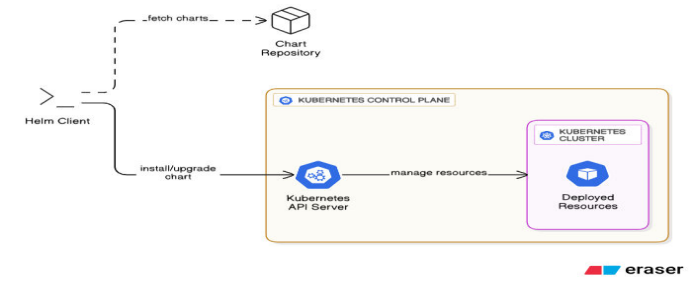
Example Deployment Template

```
# templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "my-web-app.fullname" . }}
  labels:
    {{- include "my-web-app.labels" . | nindent 4 }}
spec:
  {{- if not .Values.autoscaling.enabled }}
  replicas: {{ .Values.replicaCount }}
  {{- end }}
  selector:
    matchLabels:
      {{- include "my-web-app.selectorLabels" . | nindent 6 }}
  template:
    metadata:
```

```
      labels:
        {{- include "my-web-app.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 }}
```

Helm's Architecture and Key Components

Architecture Overview



Key Components:

1. **Helm Client**: The command-line tool that interacts with Kubernetes
2. **Charts**: Packaged applications containing Kubernetes manifests
3. **Repositories**: Collections of charts (like Docker Hub for containers)
4. **Releases**: Instances of charts deployed to Kubernetes
5. **Values**: Configuration parameters for customizing charts

---

Setting up Kubernetes Cluster (Minikube)

Prerequisites

- Docker installed on your machine
- At least 2GB of RAM available
- VT-x/AMD-v virtualization support

Installing Minikube

Linux

```
# Download and install minikube
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube

# Start minikube
minikube start --driver=docker
```

Windows

```
# Using Chocolatey
choco install minikube

# Or download from GitHub releases
# https://github.com/kubernetes/minikube/releases

# Start minikube
minikube start --driver=docker
```

Verify Minikube Installation

```
# Check cluster status
minikube status

# Get cluster info
kubectl cluster-info

# Check nodes
kubectl get nodes
```

Installing Helm

Prerequisites

- Kubernetes cluster running (minikube, EKS, GKE, etc.)
- kubectl configured to communicate with your cluster

Installation Methods

Linux

```
# Method 1: Using the install script
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: Using package manager (Ubuntu/Debian)
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

# Method 3: Using Snap
sudo snap install helm --classic
```

Windows

```
# Method 1: Using Chocolatey
choco install kubernetes-helm

# Method 2: Using Scoop
scoop install helm

# Method 3: Download binary
# Download from: https://github.com/helm/helm/releases
# Extract and add to PATH
```

Verifying Kubernetes Connection

```
# Check Helm version
helm version

# Check if Helm can connect to Kubernetes
helm list

# Add the official Helm stable repository
helm repo add stable https://charts.helm.sh/stable
helm repo update

# Search for available charts
helm search repo stable
```

Helm Chart Fundamentals

Creating Your First Chart

```
# Create a new chart
helm create my-first-chart

# Examine the chart structure
ls -la my-first-chart/
```

Understanding Chart Dependencies

```
# Chart.yaml - Adding dependencies
dependencies:
  - name: mysql
    version: 9.4.1
    repository: https://charts.bitnami.com/bitnami
  - name: redis
    version: 17.3.7
    repository: https://charts.bitnami.com/bitnami
```

```
# Download dependencies
helm dependency update my-first-chart/
```

Basic Helm Commands

```
# Install a chart
helm install my-release my-first-chart/

# List releases
helm list

# Get release status
helm status my-release

# Upgrade a release
helm upgrade my-release my-first-chart/

# Rollback to previous version
helm rollback my-release 1

# Uninstall a release
helm uninstall my-release

# Dry run (see what would be deployed)
helm install my-release my-first-chart/ --dry-run --debug
```

Working with Helm Templates

Understanding Go Templating Language

Helm uses Go's template language with additional functions. Templates are processed to generate Kubernetes manifests.

Basic Syntax

```
# Variable substitution
name: {{ .Values.name }}

# With default value
name: {{ .Values.name | default "default-name" }}

# Pipeline operations
name: {{ .Values.name | upper | quote }}
```

Using If Statements for Conditional Logic

```
# Simple if statement
{{- if .Values.ingress.enabled }}
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: {{ include "my-app.fullname" . }}
spec:
  # ... ingress spec
{{- end }}

# If-else statement
{{- if eq .Values.service.type "LoadBalancer" }}
  type: LoadBalancer
{{- else }}
  type: ClusterIP
{{- end }}

# Complex conditions
{{- if and .Values.persistence.enabled (not .Values.persistence.existingClaim) }}
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: {{ include "my-app.fullname" . }}
spec:
  # ... PVC spec
{{- end }}
```

Looping Through Lists with Range

```
# Simple range over a list
{{- range .Values.environments }}
- name: {{ . }}
  value: "active"
{{- end }}

# Range with index
{{- range $index, $env := .Values.environments }}
- name: ENV_{{ $index }}
  value: {{ $env }}
{{- end }}

# Range over map
{{- range $key, $value := .Values.configMap }}
{{ $key }}: {{ $value | quote }}
{{- end }}

# Example values.yaml for above
environments:
  - development
  - staging
  - production

configMap:
  DATABASE_URL: "postgresql://localhost:5432/mydb"
  CACHE_TTL: "3600"
  DEBUG_MODE: "false"
```

Defining and Using Named Templates

```
# templates/_helpers.tpl
{{/*
Expand the name of the chart.
*/}}
{{- define "my-app.name" -}}
{{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" }}
{{- end }}

{{/*
Create a default fully qualified app name.
*/}}
{{- define "my-app.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 }}

{{/*
Common labels
*/}}
{{- define "my-app.labels" -}}
helm.sh/chart: {{ include "my-app.chart" . }}
{{ include "my-app.selectorLabels" . }}
{{- if .Chart.AppVersion }}
app.kubernetes.io/version: {{ .Chart.AppVersion | quote }}
{{- end }}
app.kubernetes.io/managed-by: {{ .Release.Service }}
{{- end }}

{{/*
Selector labels
*/}}
{{- define "my-app.selectorLabels" -}}
app.kubernetes.io/name: {{ include "my-app.name" . }}
app.kubernetes.io/instance: {{ .Release.Name }}
{{- end }}

# Using named templates in other files
metadata:
  name: {{ include "my-app.fullname" . }}
  labels:
    {{- include "my-app.labels" . | nindent 4 }}
```

Working with Built-in Helm Objects

```
# Chart object
name: {{ .Chart.Name }}
version: {{ .Chart.Version }}
app-version: {{ .Chart.AppVersion }}

# Release object
release-name: {{ .Release.Name }}
release-namespace: {{ .Release.Namespace }}
release-service: {{ .Release.Service }}
release-revision: {{ .Release.Revision }}

# Values object
replica-count: {{ .Values.replicaCount }}
image-tag: {{ .Values.image.tag }}

# Template object
template-name: {{ .Template.Name }}
template-basepath: {{ .Template.BasePath }}
```

```
# Files object (reading files from chart)
config-data: |
{{ .Files.Get "config/app.conf" | indent 2 }}

# Capabilities object
api-versions: {{ .Capabilities.APIVersions }}
kube-version: {{ .Capabilities.KubeVersion }}
```

Debugging Helm Templates

```
# Render templates locally without installing
helm template my-release my-chart/

# Render with specific values
helm template my-release my-chart/ --values custom-values.yaml

# Debug with dry-run
helm install my-release my-chart/ --dry-run --debug

# Render specific template
helm template my-release my-chart/ --show-only templates/deployment.yaml

# Validate chart
helm lint my-chart/

# Check for issues
helm template my-release my-chart/ --validate
```

Advanced Template Functions

```
# String functions
{{ "hello world" | title }}                # Hello World
{{ "HELLO" | lower }}                      # hello
{{ "hello" | upper }}                      # HELLO
{{ "  hello  " | trim }}                   # hello
{{ "hello-world" | replace "-" "_" }}      # hello_world

# List functions
{{ list "a" "b" "c" | join "," }}          # a,b,c
{{ .Values.tags | toJson }}                # JSON format
{{ .Values.config | toYaml | indent 2 }}   # YAML format

# Logical functions
{{ and .Values.enabled .Values.ready }}
{{ or .Values.dev .Values.debug }}
{{ not .Values.production }}

# Comparison functions
{{ eq .Values.env "production" }}
```

```
{{ ne .Values.replicas 1 }}
{{ lt .Values.cpu "100m" }}
{{ gt .Values.memory "256Mi" }}

# Date functions
{{ now | date "2006-01-02" }}
{{ now | unixEpoch }}

# Encoding functions
{{ "hello" | b64enc }}                    # base64 encode
{{ "aGVsbG8=" | b64dec }}                 # base64 decode
```

Real-World Helm Application

Let's create a complete web application with database, caching, and monitoring.

Project Structure

```
webapp-chart/
├── Chart.yaml
├── values.yaml
├── values-prod.yaml
├── values-dev.yaml
└── templates/
    ├── _helpers.tpl
    ├── configmap.yaml
    ├── secret.yaml
    ├── deployment.yaml
    ├── service.yaml
    ├── ingress.yaml
    ├── hpa.yaml
    └── servicemonitor.yaml
```

Chart.yaml

```yaml
apiVersion: v2
name: webapp
description: A production-ready web application
type: application
version: 1.0.0
appVersion: "2.1.0"
dependencies:
  - name: postgresql
    version: 12.1.2
    repository: https://charts.bitnami.com/bitnami
  - name: redis
    version: 17.4.0
    repository: https://charts.bitnami.com/bitnami
```

values.yaml (Production Configuration)

```yaml
global:
  environment: production

replicaCount: 3

image:
  repository: mycompany/webapp
  pullPolicy: IfNotPresent
  tag: "2.1.0"

service:
  type: ClusterIP
  port: 8080

ingress:
  enabled: true
  className: nginx
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-prod
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
  hosts:
    - host: webapp.company.com
      paths:
        - path: /
          pathType: Prefix
  tls:
    - secretName: webapp-tls
      hosts:
        - webapp.company.com

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

autoscaling:
  enabled: true
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 70

config:
  database:
    host: "{{ .Release.Name }}-postgresql"
    port: 5432
```

```
    name: webapp
  cache:
    host: "{{ .Release.Name }}-redis-master"
    port: 6379
  features:
    enableMetrics: true
    enableTracing: true

secrets:
  database:
    username: webapp
    password: "" # Set via --set or external secret

monitoring:
  enabled: true
  serviceMonitor:
    enabled: true

# PostgreSQL configuration
postgresql:
  enabled: true
  auth:
    postgresPassword: "secure-password"
    username: webapp
    password: "webapp-password"
    database: webapp

# Redis configuration
redis:
  enabled: true
  auth:
    enabled: true
    password: "redis-password"
```

Deployment Template with Advanced Features

```
# 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:
      annotations:
        checksum/config: {{ include (print $.Template.BasePath "/configmap.yaml")
. | sha256sum }}
        checksum/secret: {{ include (print $.Template.BasePath "/secret.yaml") . |
sha256sum }}
      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: 8080
              protocol: TCP
          env:
            - name: ENVIRONMENT
              value: {{ .Values.global.environment }}
            - name: DATABASE_HOST
              valueFrom:
                configMapKeyRef:
                  name: {{ include "webapp.fullname" . }}-config
                  key: database-host
            - name: DATABASE_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: {{ include "webapp.fullname" . }}-secret
                  key: database-password
            {{- range $key, $value := .Values.config }}
            {{- if kindIs "string" $value }}
            - name: {{ $key | upper | replace "." "_" }}
              value: {{ tpl $value $ | quote }}
            {{- end }}
            {{- end }}
          livenessProbe:
            httpGet:
              path: /health
              port: http
            initialDelaySeconds: 30
            periodSeconds: 10
          readinessProbe:
            httpGet:
              path: /ready
              port: http
            initialDelaySeconds: 5
            periodSeconds: 5
          resources:
            {{- toYaml .Values.resources | nindent 12 }}
      {{- with .Values.nodeSelector }}
      nodeSelector:
        {{- toYaml . | nindent 8 }}
```

```
      {{- end }}
      {{- with .Values.affinity }}
      affinity:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      {{- with .Values.tolerations }}
      tolerations:
        {{- toYaml . | nindent 8 }}
      {{- end }}
```

Deployment Commands

```
# Deploy to development
helm install webapp-dev webapp-chart/ -f values-dev.yaml

# Deploy to production
helm install webapp-prod webapp-chart/ -f values-prod.yaml

# Upgrade with new image tag
helm upgrade webapp-prod webapp-chart/ --set image.tag=2.1.1

# Rollback if needed
helm rollback webapp-prod 1

# Check deployment status
helm status webapp-prod
kubectl get pods -l app.kubernetes.io/name=webapp
```

Best Practices and Tips

Chart Development

1. **Use semantic versioning** for chart versions
2. **Document your values** in values.yaml with comments
3. **Use consistent naming** conventions
4. **Validate charts** with `helm lint`
5. **Test templates** with `helm template`

Security

1. **Never hardcode secrets** in charts
2. **Use proper RBAC** when needed
3. **Scan images** for vulnerabilities
4. **Limit resource usage** with resource quotas

Production Readiness

1. **Always set resource limits**

2. **Configure health checks**
3. **Use multiple replicas**
4. **Set up monitoring and alerting**
5. **Plan for disaster recovery**

This tutorial covers the essential aspects of Helm from basic concepts to advanced templating. Practice with these examples and gradually build more complex applications as you become comfortable with Helm's capabilities.