

# Module 4: Helm and Kustomize – Mastering Kubernetes Configuration Management

Modern Kubernetes deployments require sophisticated configuration management tools. This guide explores two essential tools in the Kubernetes ecosystem: Helm, the package manager that simplifies application deployment, and Kustomize, the declarative configuration customization tool. Whether you're managing multi-environment deployments or standardizing application packages, these tools provide the foundation for production-grade Kubernetes operations.

# Understanding Helm: The Kubernetes Package Manager



Helm transforms Kubernetes application deployment by packaging all necessary resources into reusable charts. Think of it as the apt or yum for Kubernetes—a standardized way to define, install, and upgrade even the most complex applications.

## What is a Helm Chart?

A Helm Chart is a collection of files that describe a related set of Kubernetes resources. It's essentially a package of templates that can be configured and deployed with custom values.



### **Chart.yaml**

Metadata including name, version, and description of the application



### **values.yaml**

Default configuration values that can be overridden during installation



### **templates/**

Kubernetes manifest templates for deployments, services, and ingress

# Installing Helm: Getting Started

Setting up Helm is straightforward. The Helm binary is distributed through GitHub releases, making it easy to download and install on any Linux system. Once installed, you'll have access to the powerful helm CLI that manages your Kubernetes applications.

01

## Download Helm Binary

Navigate to the official Helm releases page and download the appropriate version for your architecture

02

## Extract the Archive

Use tar to extract the downloaded archive containing the Helm binary

03

## Install to System Path

Copy the helm binary to /usr/local/bin to make it accessible system-wide

04

## Verify Installation

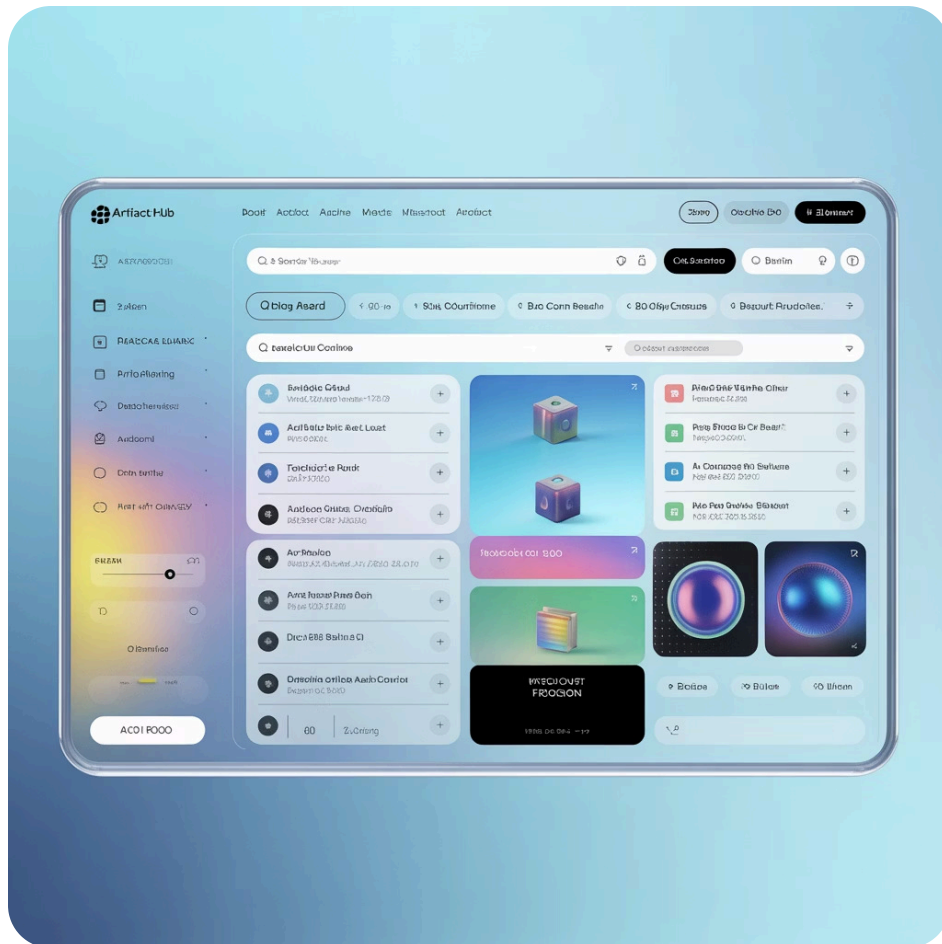
Run helm version to confirm successful installation

```
wget https://get.helm.sh/helm-v3.19.0-linux-arm64.tar.gz
tar -xvf helm-v3.19.0-linux-arm64.tar.gz
sudo cp linux-amd64/helm /usr/local/bin/helm
```

# Exploring Helm Repositories and Charts

## Finding Charts

Helm Hub (now Artifact Hub) serves as the central registry for discovering Helm charts. You can browse thousands of pre-built charts for popular applications, from databases to monitoring tools.



## Managing Repositories

Helm repositories are collections of charts hosted on web servers. Adding repositories gives you access to curated chart collections maintained by the community or vendors.



### Search Hub

Use `helm search hub` to find charts across all public repositories



### Add Repository

Add specific repositories with `helm repo add` for direct access



### Update Charts

Run `helm repo update` to sync with latest chart versions

```
helm search hub ingress
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
```

# Deploying NGINX Ingress Controller

The NGINX Ingress Controller is a critical component for managing external access to services in a Kubernetes cluster. Helm makes deploying this complex application remarkably simple, whether you're running in the cloud or on local VMs.

1

## Cloud Deployment Strategy

For cloud environments, modify the default Deployment to use a DaemonSet. This ensures an ingress controller runs on every node, providing high availability and distributed load handling.


- Fetch the chart with `helm fetch`
- Edit `values.yaml` to change kind from Deployment to DaemonSet
- Install using `helm install myingress .`

2

## Local VM Configuration

Local deployments require direct host network binding. Configure the ingress to use `hostNetwork` and `hostPort`, eliminating the need for NodePort or LoadBalancer services.

- Create a custom values file with `hostNetwork` settings
- Set `controller.kind` to DaemonSet
- Disable external services with `service.enabled: false`

 **Pro Tip:** Use `helm show values` to inspect all available configuration options before installation. This helps you understand what can be customized without diving into template source code.



# Working with Helm Installations



## Inspect Charts

Before installing, examine chart metadata, default values, and documentation using `helm show` commands. This provides visibility into what resources will be created.



## Install Applications

Deploy charts with custom configurations using values files or inline `--set` flags. Helm handles the complexity of creating all Kubernetes resources in the correct order.



## Create Ingress Rules

Once the controller is running, define ingress resources to route external traffic to your services based on hostnames and paths.

### Creating Ingress Resources

```
kubectl create ingress simple \
  --class=nginx \
  --
  rule="web.example.net/=web:80
"
```

### Testing the Configuration

```
curl -H "Host: web.example.net" \
  http://master
```

# Introduction to Kustomize

While Helm uses templating, Kustomize takes a different approach: template-free customization of Kubernetes YAML files. Built directly into kubectl, Kustomize allows you to maintain a base configuration and create environment-specific overlays without modifying the original files. This approach reduces complexity and makes configurations easier to understand and maintain.

## Declarative Approach

Pure YAML with strategic patches  
—no templating language  
required

## Base + Overlays

Define common resources once,  
customize per environment

## Native kubectl

Integrated into kubectl with the `-k` flag for seamless workflow

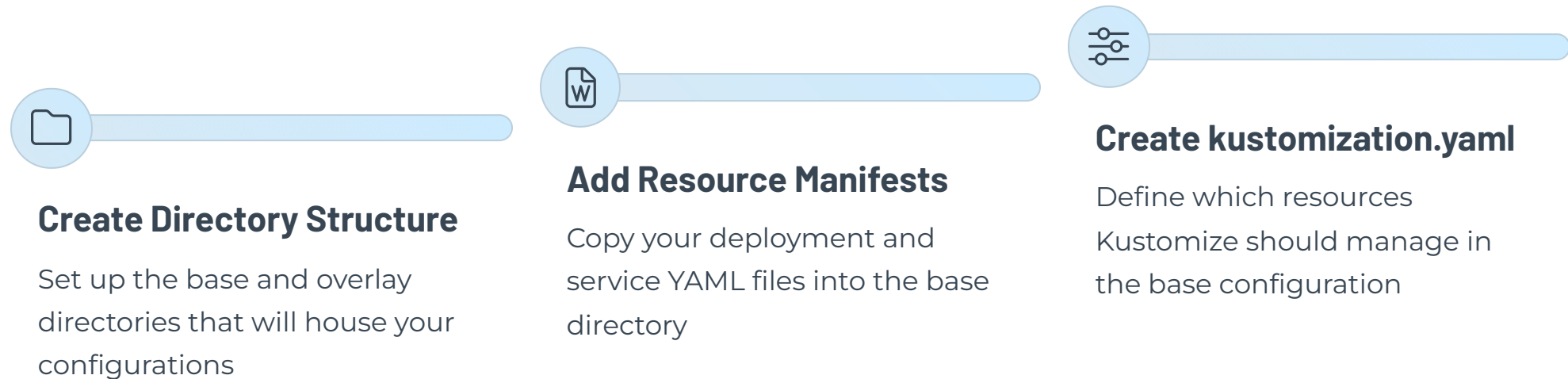
## Directory Structure

Kustomize uses a hierarchical directory structure. The base directory contains common resources, while overlay directories contain environment-specific customizations.

```
my-app/  
├── base/  
│   ├── deployment.yaml  
│   ├── service.yaml  
│   └── kustomization.yaml  
├── overlays/  
│   ├── dev/  
│   │   └── kustomization.yaml  
│   └── prod/  
│       └── kustomization.yaml
```

# Building the Kustomize Base

The base directory is the foundation of your Kustomize configuration. It contains the core Kubernetes manifests that are common across all environments. By centralizing these resources, you ensure consistency while retaining the flexibility to customize for specific deployment targets.



```
mkdir -p web/{base,overlays/{dev,prod}}
cp deploy-web.yaml web/base/
kubectl expose -f deploy-web.yaml --dry-run=client -o yaml > web/base/service.yaml

cat << EOF > web/base/kustomization.yaml
resources:
- deployment.yaml
- service.yaml
EOF
```

- ❏ The base configuration should contain only resources that are truly common across all environments. Environment-specific settings belong in overlays.



# Customizing with Overlays

Overlays are where Kustomize truly shines. Each overlay references the base configuration and applies targeted patches for specific environments. This approach keeps your configurations DRY (Don't Repeat Yourself) while providing complete flexibility for environment-specific requirements.

## Development Overlay

The dev overlay adds prefixes for easy identification and reduces replica counts for resource efficiency. Labels help identify development resources at a glance.

- Adds "dev-" prefix to resource names
- Patches replica count to 2
- Applies environment: dev labels

## Production Overlay

Production configurations use different container images and typically higher resource allocations. Patches allow surgical modifications without duplicating entire manifests.

- References production-specific patches
- Modifies container images
- Maintains production-grade settings

## Apply Development Config

```
kubectl create ns dummy
kubectl -n dummy apply \
  -k web/overlays/dev
kubectl -n dummy get all
```

## Apply Production Config

```
kubectl -n dummy apply \
  -k web/overlays/prod

# Verify the changes
kubectl -n dummy get deployment
```

# Best Practices and Key Takeaways



## Choose the Right Tool

Use Helm for packaging and distributing applications. It excels when you need to share charts with others or deploy complex applications with many configurable options.



## Leverage Kustomize for Configuration

Use Kustomize for managing environment-specific configurations. Its patch-based approach is ideal for teams that prefer pure YAML and minimal abstraction.

### Version Control

Always version your Helm charts and Kustomize configurations in Git. This enables rollbacks and provides an audit trail of changes.

### Test Before Production

Deploy to dev and staging environments first. Validate configurations work as expected before promoting to production.

### Clean Up Resources

Use `kubectl delete -k` or `helm uninstall` to remove resources cleanly when no longer needed.

Both Helm and Kustomize are powerful tools in the Kubernetes ecosystem, each with distinct strengths. Helm's templating and package management capabilities make it ideal for distributing applications, while Kustomize's declarative approach provides elegant configuration management. Many teams use both: Helm for third-party applications and Kustomize for internal services. Master these tools to build reliable, maintainable Kubernetes deployments.

#### # Cleanup commands

```
kubectl delete -k web/overlays/prod
kubectl delete -k web/overlays/dev
kubectl delete ns dummy
```