Helm Study Guide

Aleinik Cuba Lopez

2025

Table of Contents

[1. Introduction to Helm 3](#_Toc202349787)

[2. Helm Installation 4](#_Toc202349788)

[3. Most Used Helm Commands 5](#_Toc202349789)

[4. Practical Examples 7](#_Toc202349790)

[5. Creating My Own Charts 10](#_Toc202349791)

[Conclusions on the Helm Study Guide 12](#_Toc202349792)

### 1. Introduction to Helm

Helm is the **package manager for Kubernetes**. Like apt, yum, or npm for other operating systems or programming languages, Helm simplifies the deployment and management of applications on Kubernetes clusters. It allows developers and operators to package, configure, and deploy applications reproducibly.

##### Why Use Helm?

* **Simplification:** Packages multiple Kubernetes resources (Deployments, Services, Ingresses, etc.) into a single unit called a **Chart**.
* **Reusability:** Charts can be shared and reused, making it easier to deploy common applications.
* **Version Management:** Allows control over the versions of deployed applications, facilitating updates and rollbacks.
* **Dynamic Configuration:** Charts can be easily configured with different values for different environments (development, staging, production).
* **Lifecycle Management:** Facilitates the installation, updating, rolling back, and uninstallation of applications.

##### Key Concepts

* **Chart:** A Helm package that contains all the information needed to install an application or a set of applications on a Kubernetes cluster. A Chart is a directory containing configuration files and templates.
* **Release:** An instance of a Chart running on a Kubernetes cluster. When you install a Chart, Helm creates a Release with a unique name.
* **Repository:** A place where Charts are stored and shared. Repositories are essentially HTTP servers that serve Chart files.
* **Values:** A file (usually values.yaml) within a Chart that defines default configuration values. These values can be overridden during the installation or upgrade of a Chart.
* **Templates:** Files within a Chart (located in the templates/ folder) that contain Kubernetes resource definitions (.yaml) with placeholders. Helm renders these templates using the provided values to generate the final Kubernetes manifests.

### 2. Helm Installation

Helm is a command-line tool, so its installation is quite straightforward on most operating systems.

##### Prerequisites

Before installing Helm, make sure you have:

* **kubectl:** The Kubernetes command-line tool, configured to connect to your Kubernetes cluster.
* **A running Kubernetes cluster:** This can be local (like Minikube, Kind, Docker Desktop with Kubernetes) or in the cloud (GKE, AKS, EKS, etc.).

##### Installation on Linux

The easiest way to install Helm on Linux is via snap or by downloading the binary.

**Option 1: Using Snap (Recommended for Ubuntu/Debian)**

*sudo snap install helm --classic*

**Option 2: Using the Installation 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*

##### Option 3: Manual Binary Download

1. Visit the [Helm releases page on GitHub](https://github.com/helm/helm/releases).
2. Download the appropriate version for your architecture (e.g., helm-vX.Y.Z-linux-amd64.tar.gz).
3. Unzip the file:

*tar -zxvf helm-vX.Y.Z-linux-amd64.tar.gz*

1. Move the helm binary to your $PATH (e.g., /usr/local/bin/):

*sudo mv linux-amd64/helm /usr/local/bin/helm*

##### Installation on Windows

On Windows, you can install Helm using Chocolatey, Scoop, or by downloading the binary.

##### Option 1: Using Chocolatey (Recommended)

If you have Chocolatey (a package manager for Windows) installed in PowerShell:

*choco install kubernetes-helm*

##### Option 2: Using Scoop

If you have Scoop (another package manager for Windows) installed in PowerShell:

*scoop install helm*

##### Option 3: Manual Binary Download

1. Visit the [Helm releases page on GitHub](https://github.com/helm/helm/releases).
2. Download the appropriate version for your architecture (e.g., helm-vX.Y.Z-windows-amd64.zip).
3. Unzip the ZIP file.
4. Move the helm.exe executable to a location that is in your system Path (for example, C:\Program Files\Helm and add that path to the Path environment variable).

**Installation Verification**

After installation, verify that Helm is working correctly:

*helm version*

You should see the installed Helm version.

### 3. Most Used Helm Commands

Here are the Helm commands you'll use most frequently in your day-to-day work:

| Command | Description |
| --- | --- |
| *helm help* | Shows general help or specific help for a command. |
| *helm version* | Shows the installed Helm version. |
| *helm repo add [NAME] [URL]* | Adds a new Chart repository. |
| *helm repo list* | Lists configured Chart repositories. |
| *helm repo update* | Updates Chart information from all repositories. |
| *helm search repo [KEYWORD]* | Searches for Charts in configured repositories. |
| *helm search hub [KEYWORD]* | Searches for Charts on Artifact Hub (a centralized registry for CNCF Charts and other artifacts). |
| *helm install [NAME] [CHART]* | Installs a Chart on the cluster, creating a new Release. |
| *helm upgrade [RELEASE\_NAME] [CHART]* | Upgrades an existing Release with a new Chart version or new values. |
| *helm rollback [RELEASE\_NAME] [REVISION]* | Rolls back a Release to a previous revision. |
| *helm uninstall [RELEASE\_NAME]* | Uninstalls a Release, deleting all associated Kubernetes resources. |
| *helm list* | Lists all Releases in the cluster. |
| *helm status [RELEASE\_NAME]* | Shows the status of a specific Release. |
| *helm history [RELEASE\_NAME]* | Shows the revision history of a Release. |
| *helm show values [CHART]* | Shows the default values of a Chart. |
| *helm create [NAME]* | Creates a new Chart with a basic structure to start developing your own Chart. |
| *helm lint [CHART\_PATH]* | Analyzes a local Chart for errors and best practices. |
| *helm package [CHART\_PATH]* | Packages a Chart into a .tgz file for distribution. |

### 4. Practical Examples

Let's look at some common examples of how to use Helm.

##### Example 1: Adding a Repository and Searching for a Chart

Suppose we want to install Nginx, a very popular web server. First, we need to add a repository that contains the Nginx Chart. The official Chart repository is a good starting point.

1. **Add the bitnami repository:** Bitnami has many popular and well-maintained Charts.

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

Expected output: "bitnami" has been added to your repositories

1. **Update repositories:** To ensure we have the latest Chart information.

*helm repo update*

Expected output: Successfully got an update from the "bitnami" chart repository

1. **Search for the Nginx Chart:**

*helm search repo nginx*

You will see several results, look for the Bitnami one which will likely be named bitnami/nginx.

##### Example 2: Installing an Application (Nginx)

Now that we've found the Nginx Chart, let's install it.

*helm install my-nginx bitnami/nginx*

* my-nginx: This is the **Release name**. It's a unique name you give to this Nginx instance.
* bitnami/nginx: This is the **Chart** we are installing.

Expected output (similar to):

NAME: my-nginx

LAST DEPLOYED: Thu Jun 26 11:00:00 2025

NAMESPACE: default

STATUS: deployed

REVISION: 1

NOTES:

... (instructions to access Nginx, IP, etc.)

##### Example 3: Listing Installed Releases

To see all applications (releases) you have installed in your cluster:

*helm list*

Expected output:

NAME NAMESPACE REVISION UPDATED STATUS CHART APP VERSION

my-nginx default 1 2025-06-26 11:00:00.1234567 -0400 EDT deployed nginx-15.0.0 1.25.1

##### Example 4: Viewing the Status of a Release

To get more details about a specific Release:

*helm status my-nginx*

This will show a summary of the Release, including the Kubernetes resources it has created.

##### Example 5: Updating an Application with New Values

Suppose you want to change the number of Nginx replicas. You can do this by passing a new value.

1. **Get the current Chart values:**

*helm show values bitnami/nginx > nginx-values.yaml*

This will save the default values to an nginx-values.yaml file.

1. **Modify nginx-values.yaml:** Open the file and find the replicaCount section. Change it to 2 (or your desired number).

YAML

# ... other configurations

replicaCount: 2

# ... other configurations

1. **Upgrade the Release:**

*helm upgrade my-nginx bitnami/nginx -f nginx-values.yaml*

my-nginx: The name of your Release.

bitnami/nginx: The Chart.

-f nginx-values.yaml: Tells Helm to use the values from this file.

Expected output (similar to):

Release "my-nginx" has been upgraded. Happy Helming!

NAME: my-nginx

LAST DEPLOYED: Thu Jun 26 11:15:00 2025

NAMESPACE: default

STATUS: deployed

REVISION: 2

NOTES:

...

We can verify the change with kubectl get deployments.

##### Example 6: Rolling Back an Application to a Previous Version

If the upgrade went wrong, you can roll back to a previous revision. First, check the history:

*helm history my-nginx*

Expected output (similar to):

REVISION UPDATED STATUS CHART APP VERSION DESCRIPTION

1 Thu Jun 26 11:00:00 2025 deployed nginx-15.0.0 1.25.1 Install complete

2 Thu Jun 26 11:15:00 2025 deployed nginx-15.0.0 1.25.1 Upgrade complete

To roll back to revision 1:

*helm rollback my-nginx 1*

Expected output:

Rollback was a success! Happy Helming!

##### Example 7: Uninstalling an Application

To completely remove an application deployed with Helm:

*helm uninstall my-nginx*

Expected output:

release "my-nginx" uninstalled

This will delete all Kubernetes resources associated with the my-nginx Release.

### 5. Creating My Own Charts

Helm allows you to package your own applications.

##### Basic Chart Structure

You can create a skeleton Chart with the helm create command:

*helm create my-app*

This will create a my-app directory with the following structure:

my-app/

Chart.yaml # Information about the Chart (name, version, description)

values.yaml # Default values for the Chart

charts/ # Chart dependencies (sub-charts)

templates/ # Directory where your Kubernetes manifests (.yaml) will go

NOTES.txt # Notes displayed after installation

\_helpers.tpl # File to define reusable templates and functions

deployment.yaml # Template for a Kubernetes Deployment

service.yaml # Template for a Kubernetes Service

ingress.yaml # Template for a Kubernetes Ingress (if enabled)

serviceaccount.yaml # Template for a ServiceAccount (if enabled)

tests/ # Tests for the Chart (e.g., test-connection.yaml)

##### Modifying Templates and Values

Inside templates/, you can modify the .yaml files to define your Kubernetes resources. Values defined in values.yaml can be injected into the templates using Go template syntax.

For example, in templates/deployment.yaml, you'll see something like:

YAML

apiVersion: apps/v1

kind: Deployment

metadata:

name: {{ include "my-app.fullname" . }}

labels:

{{- include "my-app.labels" . | nindent 4 }}

spec:

replicas: {{ .Values.replicaCount }}

# ...

Here, {{ .Values.replicaCount }} is a placeholder that Helm will replace with the replicaCount value defined in values.yaml (or a value provided during installation/upgrade).

**Linting and Packaging**

Before sharing or deploying your Chart, it's a good idea to validate and package it:

* **Linting (Validation):**

*helm lint my-app/*

* **Packaging:**

*helm package my-app/*

This will create a my-app-X.Y.Z.tgz file that you can distribute.

### Conclusions on the Helm Study Guide

This study guide is designed to be your starting point in the world of **Helm**, the **de facto package manager for Kubernetes**. Throughout this document, and with the repository structure we've laid out, you've explored the **fundamental concepts** that make Helm an indispensable tool in the Kubernetes ecosystem.

By understanding what a **Chart** is, how a **Release** works, the utility of **Repositories**, and the flexibility offered by **Values** and **Templates**, you've laid the groundwork for efficient application management. The **installation** section has provided you with clear steps to get Helm up and running in your **Linux** and **Windows** environments, ensuring you can start working without delay.

We've covered the **most used commands** in Helm, which are your daily bread and butter when interacting with your deployments. From adding repositories and searching for Charts, to installing, upgrading, rolling back, and uninstalling applications, these commands give you complete control over the lifecycle of your Kubernetes deployments. The **practical examples** accompanying the guide are designed for you to see Helm in action and replicate these scenarios, solidifying your learning with direct experience.

Finally, the section on **creating your own Charts** opens the door to true customization and reusability. Being able to package your own applications allows you to standardize deployments, apply version control to your infrastructure, and share solutions reproducibly—a fundamental pillar in DevOps practices.

In summary, this guide not only introduces you to Helm but also equips you with the knowledge and tools to start leveraging its power. Mastering Helm will allow you to:

* **Simplify the complexity** of Kubernetes manifests.
* **Accelerate your deployments** and improve repeatability.
* **Manage application lifecycles** robustly.
* **Collaborate more effectively** on Kubernetes projects.

We hope this guide serves as a valuable resource and encourages you to further explore Helm's capabilities on your journey toward excellence in container orchestration.