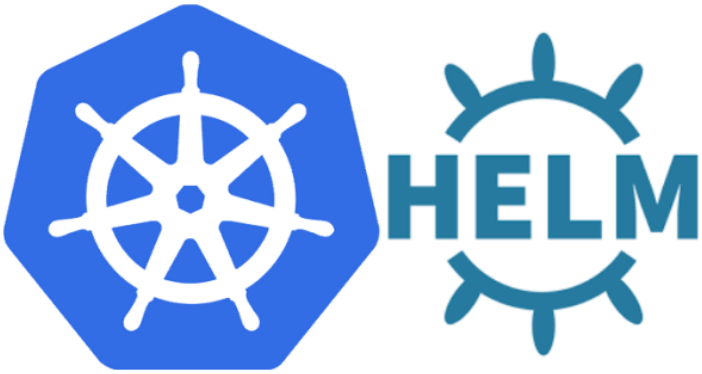


Vishal Kurane

vishalarunkurane@gmail.com | LinkedIn Profile

Contents

Introduction to Helm Chart **3**

1. What is Helm? **3**

2. Purpose of Helm in Kubernetes Ecosystem **3**

3. Key Components of Helm **3**

Helm CLI 4

Helm Charts 4

Helm Repositories 4

Helm Releases 4

4. Difference Between Helm and Plain Kubernetes Manifests **5**

5. Installation of Helm Chart **6**

Folder Structure of Helm Chart **7**

1. Chart.yaml **7**
2. values.yaml **7**
3. charts/ (Directory) **8**
4. Common Files in templates/ **8**
5. .helmignore **8**

Additional Notes on Templates **9**

Overall Significance of Helm Chart Structure **9**

Deploy an application through Helm chart **10**

1. Create a helm chart **10**
2. Update the template manifest **10**
3. Install the helm chart **10**
4. Application Deployed through Helm Chart is accessible **11**
5. Uninstall the helm chart **11**

Helm Commands **12**

1. Helm Version Commands **12**

helm version 12

2. Helm Chart Management Commands **12**

helm create 12

helm lint 12

helm package 12

helm show values 12

3. Helm Chart Repository Management Commands **13**

helm repo add 13

helm repo update 13

helm repo list 13
helm repo remove 13
4. Helm Chart Installation and Management Commands 13
helm install 13
helm upgrade 13
helm uninstall 14
helm rollback 14
helm list 14
helm status 14
5. Helm Release Management and Debugging Commands 15
Helm dry run 15
helm history 15
helm get all 15
helm get values 15
helm template 15
helm test 15
helm diff upgrade 15
6. Helm Chart Customization and Value Overrides 16
helm get manifest 16
helm show chart 16
helm show all 16
7. Helm Plugin Management Commands 16
helm plugin list 16
helm plugin install 16
helm plugin update 16
8. Helm Environment and Configuration 17
helm env 17
helm completion 17
9. Helm Chart Search and Repository Management 17
helm search repo 17
helm search hub 17

Introduction to Helm Chart

1. What is Helm?

Helm is a package manager specifically designed for Kubernetes, much like apt for Debian or yum for RHEL-based systems. It simplifies the deployment and management of applications on Kubernetes by bundling together all the Kubernetes resources required for an application into a single package called a **Helm Chart**.

Key Features of Helm:

- **Package Management**: Helm uses "charts" as the package format to describe Kubernetes resources.
- **Versioning**: Helm supports version control for application releases, allowing easy rollback to previous versions if necessary.
- **Templating**: Helm allows parameterized templates to be used in charts, making deployments flexible and customizable.
- **Release Management**: Helm tracks deployed applications as “releases” and manages them throughout their lifecycle (installation, upgrades, rollback).

2. Purpose of Helm in Kubernetes Ecosystem

Helm's purpose in the Kubernetes ecosystem is to streamline the complexity of managing Kubernetes applications by providing a higher-level tool that abstracts many repetitive and error-prone tasks involved in configuring Kubernetes resources manually.

Key Benefits of Helm in Kubernetes:

- **Simplified Deployment**: Helm packages multiple Kubernetes resources (pods, services, config maps, etc.) into a single unit (a Helm chart) and deploys them with a single command.
- **Reusability and Sharing**: Helm charts can be shared and reused across different projects. This allows teams to use pre-built charts for popular software (e.g., databases, web servers) or build their own reusable, parameterized templates.
- **Lifecycle Management**: Helm helps with the lifecycle management of applications, including deployment, upgrade, and rollback, while keeping track of previous deployments as versions.
- **Parameterization**: Helm allows dynamic configuration of applications through the values.yaml file, making it easy to deploy the same application with different settings across different environments (development, staging, production).
- **Multi-Environment Support**: Helm makes it easy to deploy the same application to multiple Kubernetes environments (e.g., dev, QA, prod) with different configurations by leveraging Helm's templating and values system.

3. Key Components of Helm

Helm is made up of several core components, which work together to make Kubernetes deployments easier and more manageable.

Helm CLI

- The command-line interface (CLI) is the main tool for interacting with Helm. With the CLI, you can manage your Helm repositories, install charts, upgrade applications, roll back to previous versions, and more.
- Common commands:
    - helm install: Install a new chart.
    - helm upgrade: Upgrade an existing release.
    - helm rollback: Roll back to a previous version.
    - helm repo add: Add a Helm chart repository.
    - helm template: Render templates locally without deploying them.

Helm Charts

- **Helm Charts** are the core packaging format in Helm. They define the structure of your Kubernetes application and its dependencies. A chart is a collection of YAML templates that describe Kubernetes resources such as deployments, services, and config maps.
- Chart components include:
    - Chart.yaml: Defines metadata about the chart, including its name and version.
    - values.yaml: Contains default configurations that can be overridden at deployment time.
    - templates/: Directory containing template files that define Kubernetes resources.
    - requirements.yaml: Lists any dependencies that the chart has on other charts.
- Charts make applications easily portable and configurable.

Helm Repositories

- Helm repositories are locations where charts are stored and distributed. They are similar to software repositories in traditional package managers (e.g., npm or PyPI).
- Public repositories, such as the **Artifact Hub** (a Helm community hub for sharing and discovering charts), make it easy to discover charts for common software like MySQL, Nginx, etc.
- You can also create private repositories for internal charts.
- Commands like helm repo add or helm repo update help manage repositories.

Helm Releases

- A **release** is a specific deployment of a Helm chart to your Kubernetes cluster. Every time you use helm install to deploy a chart, a new release is created.
- Helm manages releases using a versioning system, which enables you to upgrade and roll back applications.
- Each release is named (e.g., myapp-v1) and tracked within Kubernetes. Helm also stores the release history, making it easy to return to a previous version if necessary (using helm rollback).

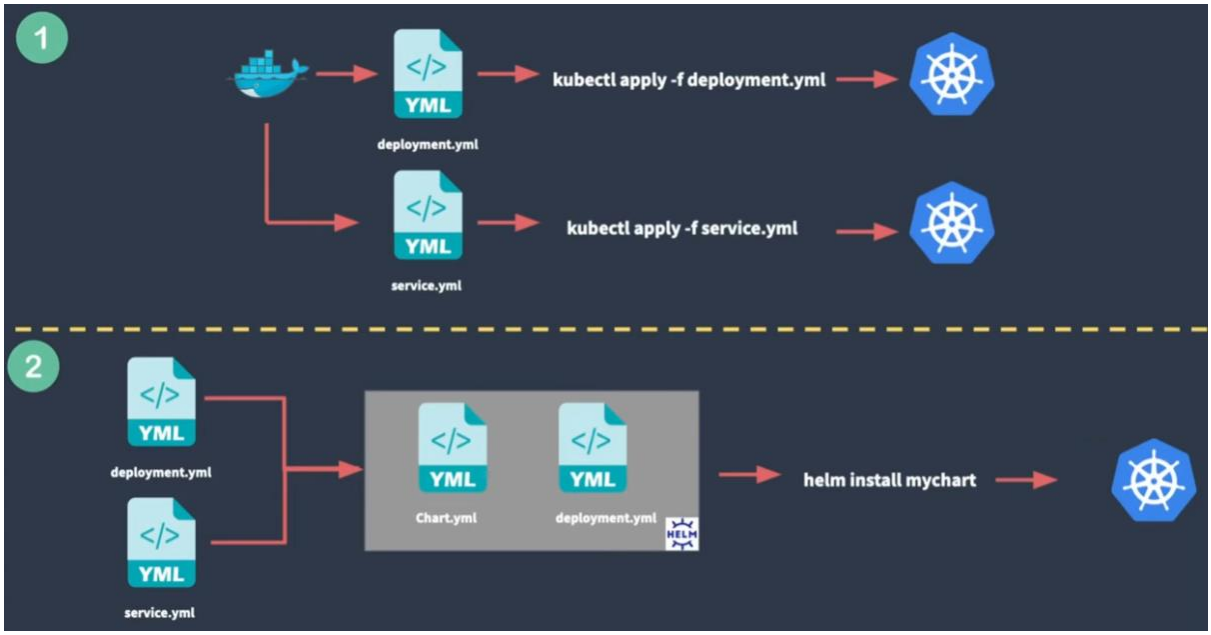
4. Difference Between Helm and Plain Kubernetes Manifests

Plain Kubernetes manifests consist of YAML files that define Kubernetes resources such as Pods, Services, and ConfigMaps. Managing applications using plain manifests can be repetitive, complex, and error-prone, especially for larger and more sophisticated deployments.

Key Differences:

Feature	Helm	Plain Kubernetes Manifests
Templating	Helm provides templating support, allowing parameterized YAML configurations via values.yaml.	Kubernetes manifests are static and require manual editing for different environments or configurations.
Application Packaging	Helm bundles multiple Kubernetes resources into a single chart, making it easy to deploy an entire application with one command.	You must manually deploy each YAML file and manage them separately.
Versioning	Helm tracks releases, allowing easy upgrades and rollbacks with full history tracking.	Kubernetes does not natively track versioning across deployments. You need to manage this manually.
Dependency Management	Helm supports chart dependencies, so if your application depends on other charts, Helm can install and manage those automatically.	Kubernetes manifests do not have built-in dependency management. You must handle this manually.
Customization	You can easily customize Helm charts for different environments using values.yaml or --set flags at deployment time.	Customizing Kubernetes manifests for different environments requires manual editing or separate YAML files.
Lifecycle Management	Helm tracks the state of your deployments and offers tools to manage upgrades, rollbacks, and uninstallation easily.	Kubernetes requires custom scripting or manual intervention to handle updates and rollbacks.
Reusability	Helm charts are reusable templates that can be shared across projects and teams, fostering consistency and efficiency.	Kubernetes manifests are typically tailored to individual applications and aren’t easily reusable.

In summary, Helm extends Kubernetes manifests by adding structure, reusability, and management features like templating, versioning, and dependency management, making it a more efficient tool for managing complex Kubernetes applications.



5. Installation of Helm Chart

- ➔ 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
- ➔ helm version

```
vishal@MasterNode:~$ helm  version
version.BuildInfo{Version:"v3.16.1", GitCommit:"5a5449dc42be07001fd5771d56429132984ab3ab", GitTreeState:"clean",
 GoVersion:"go1.22.7"}
vishal@MasterNode:~$
```

Folder Structure of Helm Chart

➔ Helm create demo
➔ Tree demo

```
vishal@MasterNode:~$ tree demo/
demo/
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

3 directories, 10 files
vishal@MasterNode:~$
```

The folder structure of a Helm chart is crucial for managing Kubernetes applications effectively. A Helm chart is essentially a collection of files that define a related set of Kubernetes resources. The files and directories within a chart describe the deployment, configuration, and structure of the Kubernetes application.

1. **Chart.yaml**
   - **Purpose**: This is the main file of a Helm chart. It contains metadata about the chart such as its name, version, description, and other dependencies.
   - **Key fields**:
     - apiVersion: The version of the Chart API used (e.g., v2 for Helm 3).
     - name: Name of the chart (must be unique).
     - version: Version of the chart.
     - description: Short description of the chart.
     - appVersion: The version of the application this chart deploys (e.g., MySQL version).
     - keywords: List of keywords to describe the chart.
     - dependencies: Defines other charts this chart depends on.

Significance: This file provides essential metadata about the Helm chart, allowing Helm to properly identify, version, and manage the application.

2. **values.yaml**
   - **Purpose**: Contains default configuration values for the chart, which can be overridden by users at install or upgrade time.
   - **Usage**: Users provide custom values to override these defaults during chart installation by using the --values or -f flag.

Significance: It's a key file where customizable options like replicas, image versions, resource limits, and other configurations are defined. It allows users to reuse the chart with different configurations by simply changing values.

3. **charts/ (Directory)**
    - **Purpose**: Stores any dependencies that the current chart may have, such as other charts.
    - **Usage**: Helm allows you to bundle other Helm charts (dependencies) into your chart. These dependencies are added here, either manually or by specifying them in Chart.yaml under the dependencies field.
    - **Significance**: Dependencies are managed here, making it easier to include third-party charts (e.g., databases or other microservices) that your application relies on.

templates/ (Directory)

- **Purpose**: Contains all the Kubernetes resource templates that define the application. These templates are written in YAML but are processed by Helm to substitute variables with actual values.
- **Significance**: The files in this folder define the actual Kubernetes resources, like deployments, services, config maps, and ingresses. Helm replaces placeholders with values from values.yaml during deployment.

4. **Common Files in templates/**
    - **deployment.yaml**: Defines the Kubernetes Deployment resource, which handles how the application is deployed, including replica settings, container images, environment variables, and resource constraints.
    - **service.yaml**: Defines the Kubernetes Service resource, which exposes the application to other services or the outside world.
    - **ingress.yaml**: Defines the Ingress resource for routing external traffic to the service.
    - **configmap.yaml**: Defines a ConfigMap resource for storing non-sensitive configuration data that can be consumed by containers.
    - **_helpers.tpl**: Contains helper templates that can be used in other templates to avoid repetition. Often contains functions for generating names or combining strings.
        - **Significance**: Keeps the templates DRY (Don't Repeat Yourself) by providing reusable code snippets.
    - **NOTES.txt**: Contains helpful instructions that are printed to the user after chart installation, often including how to access the application.
        - **Significance**: Provides useful post-installation information for end users.

5. **.helmignore**
    - **Purpose**: Similar to .gitignore, this file specifies files or directories that should be ignored when packaging the Helm chart.
    - **Usage**: Helps to exclude unnecessary files (e.g., .DS_Store, temporary files) from being included in the Helm chart package.

- **Significance**: Helps reduce the size of the Helm package and prevents sensitive or irrelevant files from being uploaded.

Additional Notes on Templates

- **Template Processing**: Helm templates support Go templating. Variables, loops, conditions, and functions can be used to dynamically build the YAML configurations. This is how the values from values.yaml are injected into the templates during runtime.
- **Values Insertion**: Helm injects values into the templates using the {{ .Values }} syntax. For example, in deployment.yaml, the image might be specified as:

  spec:

  containers:

  - name: mycontainer

  image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"

This line dynamically inserts the image name and tag from values.yaml.

Overall Significance of Helm Chart Structure

- **Modular and Reusable**: By organizing Kubernetes resources into templates and providing customizable values in values.yaml, Helm charts allow users to deploy applications flexibly across different environments.
- **Version Control**: The Chart.yaml file maintains versioning for the chart, allowing for easy upgrades and rollbacks.
- **Dependency Management**: The charts/ directory handles dependencies, making it possible to bundle other charts (such as databases or third-party services) that your application might rely on.
- **Post-installation Guidance**: The NOTES.txt file gives users important information on how to use the deployed resources, making the chart more user-friendly.

Each component of the Helm chart folder structure serves a purpose, contributing to the chart's reusability, configurability, and ease of deployment in Kubernetes.

Deploy an application through Helm chart

1. Create a helm chart

```
vishal@MasterNode:~$ helm create flaskapp-helmchart
Creating flaskapp-helmchart
vishal@MasterNode:~$ ls -latr | grep flaskapp-helm
drwxr-xr-x  4 vishal vishal     4096 Sep 18 12:16 flaskapp-helmchart
vishal@MasterNode:~$ tree  flaskapp-helmchart/
flaskapp-helmchart/
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

3 directories, 10 files
vishal@MasterNode:~$
```

2. Update the template manifest

```
root@MasterNode:/home/vishal# tree flaskapp-helmchart/
flaskapp-helmchart/
├── charts
├── Chart.yaml
└── templates
    ├── deployment.yaml
    ├── dockercreds.yaml
    ├── ingress.yaml
    └── service.yaml

2 directories, 5 files
```

3. Install the helm chart

```
root@MasterNode:/home/vishal# helm install flaskapp flaskapp-helmchart --namespace helm-flaskapp
NAME: flaskapp
LAST DEPLOYED: Wed Sep 18 13:33:51 2024
NAMESPACE: helm-flaskapp
STATUS: deployed
REVISION: 1
TEST SUITE: None
root@MasterNode:/home/vishal# helm list
NAME    NAMESPACE       REVISION        UPDATED STATUS  CHART   APP VERSION
root@MasterNode:/home/vishal# helm list --namespace helm-flaskapp
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                   APP VERSION
flaskapp        helm-flaskapp   1               2024-09-18 13:33:51.529289315 +0530 IST deployed        flaskapp-helmchart-0.1.0        1.16.0
root@MasterNode:/home/vishal#
```

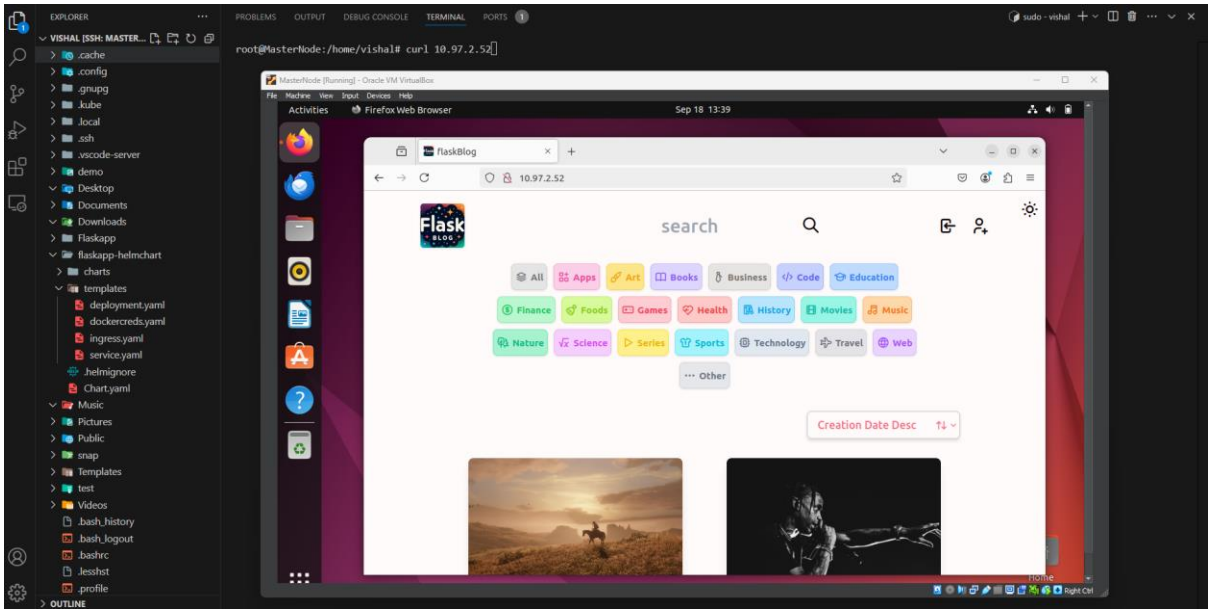
```
root@MasterNode:/home/vishal# kubectl get all -n helm-flaskapp
NAME                                     READY   STATUS    RESTARTS   AGE
pod/flaskapp-deployment-7dc8579d94-hbn4b   1/1     Running   0          107s

NAME                        TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
service/flaskapp-service    ClusterIP   10.97.2.52    <none>        80/TCP    107s

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/flaskapp-deployment   1/1     1            1           107s

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/flaskapp-deployment-7dc8579d94   1         1         1       107s
root@MasterNode:/home/vishal#
```

4. **Application Deployed through Helm Chart is accessible**



5. **Uninstall the helm chart**

```
root@MasterNode:/home/vishal# helm uninstall flaskapp --namespace helm-flaskapp
release "flaskapp" uninstalled
root@MasterNode:/home/vishal#
```

Helm Commands

1. Helm Version Commands

helm version

- Displays the Helm version information.
- Command:

➔ *helm version*

Example output: version.BuildInfo{Version:"v3.10.2", GitCommit:"...", GoVersion:"go1.18"}

2. Helm Chart Management Commands

helm create

- Creates a new Helm chart with a default directory structure.
- Command:

➔ *helm create <chart-name>*

This will generate a new directory structure for a Helm chart named <chart-name>.

helm lint

- Runs a linter to check for syntax issues and best practices in Helm charts.
- Command:

➔ *helm lint <chart-name>*

The command checks for issues with the chart templates before installation.

helm package

- Packages a Helm chart directory into a .tgz archive file for distribution.
- Command:

➔ *helm package <chart-directory>*

The .tgz package can be uploaded to a Helm chart repository.

helm show values

- Displays the default values for a Helm chart.
- **Command**:

➔ *helm show values <chart>*

3. Helm Chart Repository Management Commands

helm repo add

- Adds a new Helm chart repository.
- Command:

➔ *helm repo add <repo-name> <repo-url>*

Example:

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

helm repo update

- Fetches the latest list of charts from all configured repositories.
- Command:

➔ *helm repo update*

helm repo list

- Lists all configured Helm repositories.
- Command:

➔ *helm repo list*

helm repo remove

- Removes a configured repository.
- Command:

➔ *helm repo remove <repo-name>*

4. Helm Chart Installation and Management Commands

helm install

- Installs a Helm chart to a Kubernetes cluster.
- Command:

➔ *helm install <release-name> <chart> --namespace <namespace>*

Example:

helm install my-release bitnami/nginx --namespace my-namespace

helm upgrade

- Upgrades an existing Helm release with a new chart or updated values.
- Command:

➔ *helm upgrade <release-name> <chart> --namespace <namespace>*

Example:

helm upgrade my-release bitnami/nginx --set replicaCount=3

helm uninstall

- Uninstalls a Helm release and deletes associated resources.
- Command:

➔ *helm uninstall <release-name> --namespace <namespace>*

Example:

helm uninstall my-release --namespace my-namespace

helm rollback

- Rolls back a Helm release to a previous version.
- Command:

➔ *helm rollback <release-name> <revision-number>*

Example:

helm rollback my-release 1

helm list

- Lists all Helm releases in a namespace (or all namespaces if --all-namespaces is specified).
- Command:

➔ *helm list --namespace <namespace>*

Example:

helm list --all-namespaces

helm status

- Displays the status of a specific Helm release.
- Command:

➔ *helm status <release-name> --namespace <namespace>*

Example:

helm status my-release

5. Helm Release Management and Debugging Commands

Helm dry run

- It will validate and verify your chart by connecting to kubernetes api server and after successful validation it will render the manifest in the form of YAMLs(kubernetes resources)
- Command:

➔ *helm install <release-name> --debug --dry-run <chart>*

helm history

- Shows the revision history of a Helm release.
- Command:

➔ *helm history <release-name> --namespace <namespace>*

Example:

helm history my-release

helm get all

- Retrieves all information about a Helm release, including deployed resources and chart values.
- Command:

➔ *helm get all <release-name>*

helm get values

- Shows the values currently used in a Helm release.
- Command:

➔ *helm get values <release-name>*

helm template

- Renders templates locally without deploying them to Kubernetes. Useful for debugging and previewing.
- Command:

➔ *helm template <release-name> <chart>*

Example:

helm template my-release bitnami/nginx

helm test

- Runs tests (if available) for a Helm release.
- Command:

➔ *helm test <release-name> --namespace <namespace>*

helm diff upgrade

- Compares the currently deployed version of a release with a new chart or values, showing the differences.
- Command:

➔ *helm diff upgrade <release-name> <chart>*

(This requires the helm-diff plugin to be installed).

6. Helm Chart Customization and Value Overrides

helm get manifest

- Shows the Kubernetes manifest that was generated by Helm from a specific release.
- Command:

➔ *helm get manifest <release-name>*

helm show chart

- Displays the metadata of a Helm chart (name, version, appVersion, etc.).
- Command:

➔ *helm show chart <chart>*

helm show all

- Displays detailed information about a Helm chart, including chart metadata, values, and templates.
- Command:

➔ *helm show all <chart>*

7. Helm Plugin Management Commands

helm plugin list

- Lists all installed Helm plugins.
- Command:

➔ *helm plugin list*

helm plugin install

- Installs a Helm plugin.
- Command:

➔ *helm plugin install <url>*

Example:

helm plugin install https://github.com/databus23/helm-diff

helm plugin update

- Updates an installed Helm plugin.
- Command:

➔ *helm plugin update <plugin-name>*

8. Helm Environment and Configuration

helm env

- Displays environment variables that Helm is using.
- Command:

➔ *helm env*

helm completion

- Generates shell autocompletions for Helm.
- Command:

➔ *helm completion > /etc/_completion.d/helm*

➔ *source /etc/_completion.d/helm*

9. Helm Chart Search and Repository Management

helm search repo

- Searches for charts in a Helm repository.
- Command:

➔ *helm search repo <search-term>*

Example:

helm search repo nginx

helm search hub

- Searches for Helm charts in the Helm Hub.
- Command:

➔ *helm search hub <search-term>*