Drawbacks of k8s:

1. Static files
2. Consistency – If a dev directly changes the k8s cluster without making changes to yaml files, it will make changes inconsistent
3. K8s doesn’t maintain revision history for us

Helm: Package Manager

Charts: packages

Charts are hosted on chart repository

Charts are required to create k8s resources.

Advantages of Helm

1. Simplify the k8s deployment by extracting out the complexities.
2. Helm maintain revision history
3. Dynamic configuration – We can pass values to deployment.yaml, service.yaml, configmap.yaml and secret.yaml using Values.yaml
4. Consistency- We use helm to do the upgrade on k8s cluster
5. Lifecycle hooks – Actions to perform before any helm commands
6. Security

Helm and K8s use same config files

To overwrite configuration files set KUBECONFIG variable

Helm repo list – Show list of configured repositories

Helm repo add {name} {repo\_url}: To add chart repository

Helm search repo {chart\_name}

Helm repo remove {repo\_name}

Helm status {chart\_name}

**To install same chartname again use another k8s namespace**

**Helm install {chart\_name} –namespace {namespace\_name} –create\_namespace**

Helm list –namespace : list packages install on k8s cluster

Helm uninstall {package\_name} –namespace {namespace\_name} : Uninstal

Helm repo update- update all charts – It will upgrade the revision number

--reuse-values: It will reuse config values while upgrading the chart.

Helm upgrade –install – Do upgrade if it does not exist perform install

Release records: When ever we install or upgrade charts on helm it will store the version history in kubectl get secrets command.

--dry-run = Load chart and its dependencies , substitute values.yaml, validate the schemas and generate k8s files but do not submit those files to k8s to execute and create the objects.

Helm template {chart\_name} –values={path\_to\_values\_yaml\_file}

This will generate k8s file for storing in Code repository and later can be used to generate the objects on the k8s cluster

Helm get notes {chart\_name}: This will display release notes

Helm get values {chart\_name}: This will display values.yaml used while install or upgrading the chart

Helm history {chart\_name}: History of install , upgrade and failures of a particular chart

Helm rollback {chart\_name} {version\_number}

Helm get manifest {chart\_name}: it will get the manifest of the release which was sent to k8s

--keep-history – Keeps all the history related to a particular chart

Helm install {chart\_name} {repo\_path} –wait --timeout 5m10s : Wait and timeout till all the resource are created

Flag used during Upgrade command

--atomic – Will rollback to previous successful release.

--force – During upgrade it will delete the old deployment and create the new deployment

--cleanup-on-failure - this will remove any dangling resource during helm upgrade

Chart Commands:

Helm create {chart\_name} – chart.yaml, values.yaml, charts, templates folder

Chart.yaml – it will have metadata about the chart created

Charts (Folder) – If current chart is using any other chart dependency it will come into this folder.

Templates(folder) – deployment.yaml, service.yaml etc these files will be used to render the k8s manifest

Values.yaml – Default values going in the chart

\_helpers.tpl – IT will have template functions which can be used in deployment.yaml, service.yaml files etc.

Values.yaml – IT will have default values which are used across all the files under templates directory.

Helm package {chart\_folder\_path} -u – Generate tgz file with version with updated dependencies.

.helmignore – It will remove unnecessary files to be removed from helm package

Helm lint – check for syntax or indent issues

**Go Templating Syntax**

* It used to remove any leading or trailing whitespaces

.Values - It is subobject referring to values.yaml file

| - to combine multiple function calls

{{ .Values.my.custom.data | default “testfefault” | upper | quote }}

Nindent – newline with 4 spaces

toYaml – Convert current object to yaml

{{- if .Values.my.flag }}

{{“Output of if” | nident 2}}

{{- else}}

{{“Output of else” | nident 2}}

{{- end}}

To bind list of values to the template

{{- with .Values.my.values}}

{{- toYaml . | nident 2 }}

{{- end}}

**Variable**

{{ $myFlag := “test” }}

Loops

{{- range .Values.my.values}}

- {{. | upper | quote }}

{{- end}}

Dict

{{- range $key,$value .Values.my.values}}

- {{$key}} : {{$value}} | upper | quote }}

{{- end}}

Dependencies are defined in chart.yaml inside dependencies block

Helm dependency update {release\_name}

Tags: are used to send multiple flags

Import-values: fetch child chart values in parent chart

[In summary, **ClusterIP** is used for pod-to-pod communication within the same Kubernetes cluster, while **NodePort** is used for communication between applications within the cluster and external clients outside the cluster 2](https://kodekloud.com/blog/clusterip-nodeport-loadbalancer/)

Values.schema.json- It defines the format , type and values used in values.yaml file

Above file is used in helm lint command

Helm env HELM\_DATA\_HOME – The folder in which helm expects to store a starter template.

Helm create vs apply difference

Helm has inbuilt support for security and provenance using PGP: Pretty Good Privacy

Helm install –verify using GNUPG files