Helm

Introduction:

Helm is an open-source CNCF project which is written in GO language and originally known as *package manager for Kubernetes*. It is focused on automating the Kubernetes applications lifecycle in a simple and consistent way.

The main objective of Helm as package manager is to manage the Kubernetes resources in automated way. We can install, update, or uninstall the packages for Kubernetes applications, and deploy them with just a few commands.

Writing and maintaining Kubernetes YAML manifest for all the required Kubernetes objects can be a time consuming and tedious task for the simplest of deployments. helm simplifies the process and create a single package that can be advertised to your cluster.

Helm key components:

Chart: It is simply Kubernetes YAML manifest combined into a single package that can be advertised to the cluster. A chart is a helm package. it contains all of the resource definitions necessary to run an application, tool or service inside of a cluster. like yum, apt etc.

Release: A release is an instance of a chart running in the cluster. One chart can often be installed multiple times into the same cluster and each time it is installed, a new release is created.

Repository: It is a location where packaged charts can be stored and shared.

Objectives:

1. Helm Installation
2. Create a Helm Chart
3. Install a Helm Chart
4. Upgrade the Helm chart
5. Rollback to earlier revision
6. Delete the Helm chart
7. Create a Helm repository
8. Deploy a WordPress application from repo
9. Deploy WordPress application using external Database
10. Helm plugins

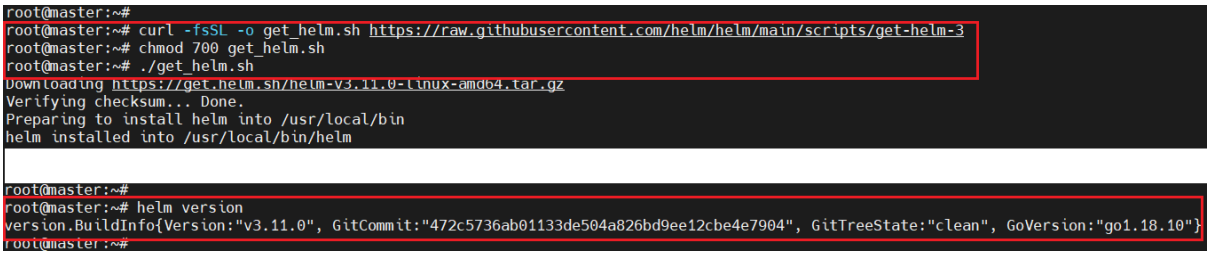
1. Helm Installation:

We are going to install Helm on our machine. Use the below script to install Helm.

```
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
```

Once it is installed, you can check the version using below command.

```
helm version
```



The above output shows that Helm has been installed in our cluster.

2. **Create a Helm Chart:**

Now let's create our first Helm chart using the below command.

```
helm create project1
```

Above command will create a **project1** directory which will include the files which can be seen in below output.

```
root@master:~/helm# helm create project1
Creating project1
root@master:~/helm#
root@master:~/helm# ls
project1
root@master:~/helm#

root@master:~/helm# tree project1/
project1/
├── Chart.yaml
├── charts
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

3 directories, 10 files
root@master:~/helm#
```

Under that project directory, we have 3 directories and 10 files.

Chart.yaml file has the information about the project.

```
root@master:~/helm/project1# cat Chart.yaml
apiVersion: v2
name: project1
description: A Helm chart for Kubernetes

# A chart can be either an 'application' or a 'library' chart.
#
# Application charts are a collection of templates that can be packaged into versioned archives
# to be deployed.
#
# Library charts provide useful utilities or functions for the chart developer. They're included as
# a dependency of application charts to inject those utilities and functions into the rendering
# pipeline. Library charts do not define any templates and therefore cannot be deployed.
type: application

# This is the chart version. This version number should be incremented each time you make changes
# to the chart and its templates, including the app version.
# Versions are expected to follow Semantic Versioning (https://semver.org/)
version: 0.1.0

# This is the version number of the application being deployed. This version number should be
# incremented each time you make changes to the application. Versions are not expected to
# follow Semantic Versioning. They should reflect the version the application is using.
# It is recommended to use it with quotes.
appVersion: "1.16.0"
```

Now let’s explore the main file which is **values.yaml**.

```
# Default values for project1.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.
replicaCount: 1
image:
  repository: nginx
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion.
  tag: ""
imagePullSecrets: []
nameOverride: ""
fullnameOverride: ""
serviceAccount:
  # Specifies whether a service account should be created
  create: true
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname template
  name: ""
podAnnotations: {}
podSecurityContext: {}
  # fsGroup: 2000
securityContext: {}
```

```
  # capabilities:
  #   drop:
  #   - ALL
  # readOnlyRootFilesystem: true
  # runAsNonRoot: true
  # runAsUser: 1000
service:
  type: ClusterIP
  port: 80
ingress:
  enabled: false
  className: ""
  annotations: {}
    # kubernetes.io/ingress.class: nginx
    # kubernetes.io/tls-acme: "true"
  hosts:
    - host: chart-example.local
      paths:
        - path: /
          pathType: ImplementationSpecific
  tls: []
  #  - secretName: chart-example-tls
  #    hosts:
  #      - chart-example.local
```

```
resources: {}

  # We usually recommend not to specify default resources and to leave this as a conscious

  # choice for the user. This also increases chances charts run on environments with little

  # resources, such as Minikube. If you do want to specify resources, uncomment the following

  # lines, adjust them as necessary, and remove the curly braces after 'resources:'.

  # limits:

  #   cpu: 100m

  #   memory: 128Mi

  # requests:

  #   cpu: 100m

  #   memory: 128Mi



autoscaling:

  enabled: false

  minReplicas: 1

  maxReplicas: 100

  targetCPUUtilizationPercentage: 80

  # targetMemoryUtilizationPercentage: 80

nodeSelector: {}

tolerations: []

affinity: {}
```

The values.yaml file is the main file, where we are going to make the changes. So instead of changing the values in each file, we will change in values.yaml file. Once we are done with the changes, we can install this and all the changes will be reflected automatically.

You can see that in values.yaml, we have image, replicaCount, serviceaccount, service, securityContext etc which are required by templates.

Under templates directory, we have below files which will fetch the values from **values.yaml** file.

```
root@master:~/helm/project1#
root@master:~/helm/project1# tree templates/
templates/
├── NOTES.txt
├── _helpers.tpl
├── deployment.yaml
├── hpa.yaml
├── ingress.yaml
├── service.yaml
├── serviceaccount.yaml
└── tests
    └── test-connection.yaml

1 directory, 8 files
root@master:~/helm/project1#
```

Let’s explore deployment.yaml from above templates.

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: {{ include "project1.fullname" . }}

  labels:

    {{- include "project1.labels" . | nindent 4 }}

spec:

  {{- if not .Values.autoscaling.enabled }}

  replicas: {{ .Values.replicaCount }}

  {{- end }}

  selector:
```

```
    matchLabels:
      {{- include "project1.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      {{- with .Values.podAnnotations }}
      annotations:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      labels:
        {{- include "project1.selectorLabels" . | nindent 8 }}
    spec:
      {{- with .Values.imagePullSecrets }}
      imagePullSecrets:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      serviceAccountName: {{ include "project1.serviceAccountName" . }}
      securityContext:
        {{- toYaml .Values.podSecurityContext | nindent 8 }}
      containers:
        - name: {{ .Chart.Name }}
          securityContext:
            {{- toYaml .Values.securityContext | nindent 12 }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default
.Chart.AppVersion }}"
```

```
    imagePullPolicy: {{ .Values.image.pullPolicy }}
    ports:
      - name: http
        containerPort: {{ .Values.service.port }}
        protocol: TCP
    livenessProbe:
      httpGet:
        path: /
        port: http
    readinessProbe:
      httpGet:
        path: /
        port: http
    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 }}
```

So this deployment file will fetch the details like image, replicaCount, securityContext etc from values.yaml.

Service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: {{ include "project1.fullname" . }}
  labels:
    {{- include "project1.labels" . | nindent 4 }}
spec:
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: http
      protocol: TCP
      name: http
  selector:
    {{- include "project1.selectorLabels" . | nindent 4 }}
```

This **service.yaml** file will fetch **service type**, **service port** etc details from **values.yaml**

Similarly, other files will get the details from values.yaml.

3. **Install a Helm Chart:**

We have our chart with us, let's install this chart using the below command.

We can use lint command to check if our chart is having any errors.

```
helm lint project1
```

```
root@master:~/helm#
root@master:~/helm# helm lint project1
==> Linting project1
[INFO] Chart.yaml: icon is recommended

1 chart(s) linted, 0 chart(s) failed
root@master:~/helm#
```

Use the below command to check what all resources will be installed. It validates the YAMLs file without connecting with Kube-apiserver.

```
helm template project1
```

Use the below dry-run command to connect with kube-apisever but it will not install the resources.

```
helm install first-project --debug --dry-run project1
```

Now let's install our first release using the below command.

```
helm install first-project project1
```

```
root@master:~/helm#
root@master:~/helm# helm install first-project project1
NAME: first-project
LAST DEPLOYED: Fri Jan 20 10:22:56 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=project1,app.kubernetes.io/instance=first-project" -o jsonpath="{.items[0]
.metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT
root@master:~/helm#
```

```
root@master:~/helm# helm list
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
first-project   default         1               2023-01-20 10:22:56.75409116 +0000 UTC  deployed        project1-0.1.0  1.16.0
root@master:~/helm#
```

Our first chart is ready with the information present in values.yaml.

Check the helm chart installed using the below command.

```
helm list
```

So our first-project is ready with revision 1.

```
root@master:~/helm#
root@master:~/helm# kubectl get all
NAME                                          READY   STATUS    RESTARTS   AGE
pod/first-project-project1-88578c9b-rdw8c     1/1     Running   0          4m25s

NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/first-project-project1      ClusterIP   10.97.112.205   <none>        80/TCP    4m25s
service/kubernetes                  ClusterIP   10.96.0.1       <none>        443/TCP   3d23h

NAME                                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/first-project-project1      1/1     1            1           4m25s

NAME                                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/first-project-project1-88578c9b        1         1         1       4m25s
root@master:~/helm#
```

Above we can see that now we have one deployment with one pod, a replica set and a service.

4. **Upgrade the Helm chart**

Now let's make some changes in the **values.yaml** file and install the project again.

```
# Default values for project1.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

replicaCount: 2

image:
  repository: nginx
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion.
  tag: ""

imagePullSecrets: []
nameOverride: ""
fullnameOverride: ""

serviceAccount:
  # Specifies whether a service account should be created
  create: true
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname template
  name: ""
```

We have changed the replicaCount from 1 to 2. Now upgrade the first-project with this latest information. Use the below command.

```
helm upgrade first-project project1
```

It will create another revision which can be seen in below output.

```
root@master:~/helm#
root@master:~/helm# helm upgrade first-project project1
Release "first-project" has been upgraded. Happy Helming!
NAME: first-project
LAST DEPLOYED: Fri Jan 20 10:31:42 2023
NAMESPACE: default
STATUS: deployed
REVISION: 2
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=project1,app.kubernetes.io/instance=first-project" -o jsonpath="{.items[0]
.metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT
root@master:~/helm#
```

```
root@master:~/helm#
root@master:~/helm# helm list
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
first-project   default         2               2023-01-20 10:31:42.738567088 +0000 UTC deployed        project1-0.1.0  1.16.0
root@master:~/helm#
```

```
root@master:~/helm#
root@master:~/helm# kubectl get all
NAME                                        READY   STATUS    RESTARTS   AGE
pod/first-project-project1-88578c9b-hvdkf   1/1     Running   0          4m29s
pod/first-project-project1-88578c9b-rdw8c   1/1     Running   0          13m

NAME                              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/first-project-project1    ClusterIP   10.97.112.205   <none>        80/TCP    13m
service/kubernetes                ClusterIP   10.96.0.1       <none>        443/TCP   3d23h

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/first-project-project1  2/2     2            2           13m

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/first-project-project1-88578c9b   2         2         2       13m
root@master:~/helm#
```

Above we can see that now we have 2 pods as per the information provided to values.yaml.

5. Rollback to earlier revision:

We can use the below command to go back to earlier version, however it will create another revision and rollback the changes.

```
helm rollback first-project 1
```

```
root@master:~/helm#
root@master:~/helm# helm list -a
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
first-project   default         2               2023-01-20 10:31:42.738567088 +0000 UTC deployed        project1-0.1.0  1.16.0
root@master:~/helm#
root@master:~/helm#
root@master:~/helm#
root@master:~/helm# helm rollback first-project 1
Rollback was a success! Happy Helming!
root@master:~/helm#
root@master:~/helm#
root@master:~/helm# helm list -a
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
first-project   default         3               2023-01-20 10:41:25.52743425 +0000 UTC  deployed        project1-0.1.0  1.16.0
root@master:~/helm#
root@master:~/helm#
root@master:~/helm# kubectl get all
NAME                                        READY   STATUS    RESTARTS   AGE
pod/first-project-project1-88578c9b-rdw8c   1/1     Running   0          18m

NAME                              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/first-project-project1    ClusterIP   10.97.112.205   <none>        80/TCP    18m
service/kubernetes                ClusterIP   10.96.0.1       <none>        443/TCP   3d23h

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/first-project-project1  1/1     1            1           18m

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/first-project-project1-88578c9b   1         1         1       18m
root@master:~/helm#
```

So, above output shows that we are back to one pod.

6. **Delete the Helm chart:**

Use the below command to delete the installed Helm chart.

```
helm delete first-project
```

```
root@master:~/helm#
root@master:~/helm# helm delete first-project
release "first-project" uninstalled
root@master:~/helm#
root@master:~/helm#
root@master:~/helm#
root@master:~/helm# helm list
NAME    NAMESPACE       REVISION        UPDATED STATUS  CHART   APP VERSION
root@master:~/helm#
root@master:~/helm# kubectl get pods
No resources found in default namespace.
root@master:~/helm#
root@master:~/helm#
```

Now we do not have any pod, deployment and service in our cluster, as the helm chart has been deleted.

7. **Create a Helm repository:**

As of now, we do not have any repository added in our cluster. You can check that using below command.

```
helm repo list
```

Search for the repository from the hub, use the below command.

```
helm search hub
```

It will show us all the repositories in our hub.

```
helm search hub wordpress --max-col-width=0
```

Above command will search and list the repo which has the WordPress chart.

Now let's add the Bitnami repository in our cluster and name it **bitnami-repo.**

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

```
root@master:~/helm#
root@master:~/helm# helm repo add bitnami-repo https://charts.bitnami.com/bitnami
"bitnami-repo" has been added to your repositories
root@master:~/helm#
root@master:~/helm# helm repo list
NAME            URL
bitnami-repo    https://charts.bitnami.com/bitnami
root@master:~/helm#
```

Our repo has been added.

8. **Deploy a WordPress application from repo**

Let's search for the WordPress chart from the repo

```
helm search repo wordpress
```

Above command will show us the version of Wordpress chart available in our repo.

```
root@master:~/helm# helm search repo wordpress
NAME                          CHART VERSION   APP VERSION     DESCRIPTION
bitnami-repo/wordpress        15.2.31         6.1.1           WordPress is the world's most popular blogging ...
bitnami-repo/wordpress-intel  2.1.31          6.1.1           DEPRECATED WordPress for Intel is the most popu...
root@master:~/helm#
```

Use the below command to get the instructions to install this package.

```
helm show readme bitnami-repo/wordpress --version 15.2.31
```

Now let's check the values which will be used in this chart.

```
helm show values bitnami-repo/wordpress --version 15.2.31
```

This application is using the dynamic volumes so we would be creating this application on PAAS service like GKE or EKS.

Use the below command to install Kubernetes.

```
helm install my-release \

  --set wordpressUsername=admin \

  --set wordpressPassword=password \

  --set mariadb.auth.rootPassword=secretpassword \

    bitnami-repo/wordpress
```

Above we are providing the values in the command line itself.

```
helm install my-release \
  --set wordpressUsername=admin \
  --set wordpressPassword=password \
  --set mariadb.auth.rootPassword=secretpassword \
    bitnami-repo/wordpress
```

```
W0120 12:36:01.532019    1055 warnings.go:70] Autopilot increased resource requests for Deployment default/my-release-wordpress to meet requirements. See http://g.co/gke/autopilot-resources
W0120 12:36:01.938692    1055 warnings.go:70] Autopilot set default resource requests for StatefulSet default/my-release-mariadb, as resource requests were not specified. See http://g.co/gke/a
utopilot-defaults
NAME: my-release
LAST DEPLOYED: Fri Jan 20 12:35:44 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: wordpress
CHART VERSION: 15.2.31
APP VERSION: 6.1.1

** Please be patient while the chart is being deployed **

Your WordPress site can be accessed through the following DNS name from within your cluster:

    my-release-wordpress.default.svc.cluster.local (port 80)

To access your WordPress site from outside the cluster follow the steps below:

1. Get the WordPress URL by running these commands:

  NOTE: It may take a few minutes for the LoadBalancer IP to be available.
        Watch the status with: 'kubectl get svc --namespace default -w my-release-wordpress'

   export SERVICE_IP=$(kubectl get svc --namespace default my-release-wordpress --include "{{ range (index .status.loadBalancer.ingress 0) }}{{ . }}{{ end }}")
   echo "WordPress URL: http://$SERVICE_IP/"
   echo "WordPress Admin URL: http://$SERVICE_IP/admin"

2. Open a browser and access WordPress using the obtained URL.

3. Login with the following credentials below to see your blog:

  echo Username: admin
  echo Password: $(kubectl get secret --namespace default my-release-wordpress -o jsonpath="{.data.wordpress-password}" | base64 -d)
```

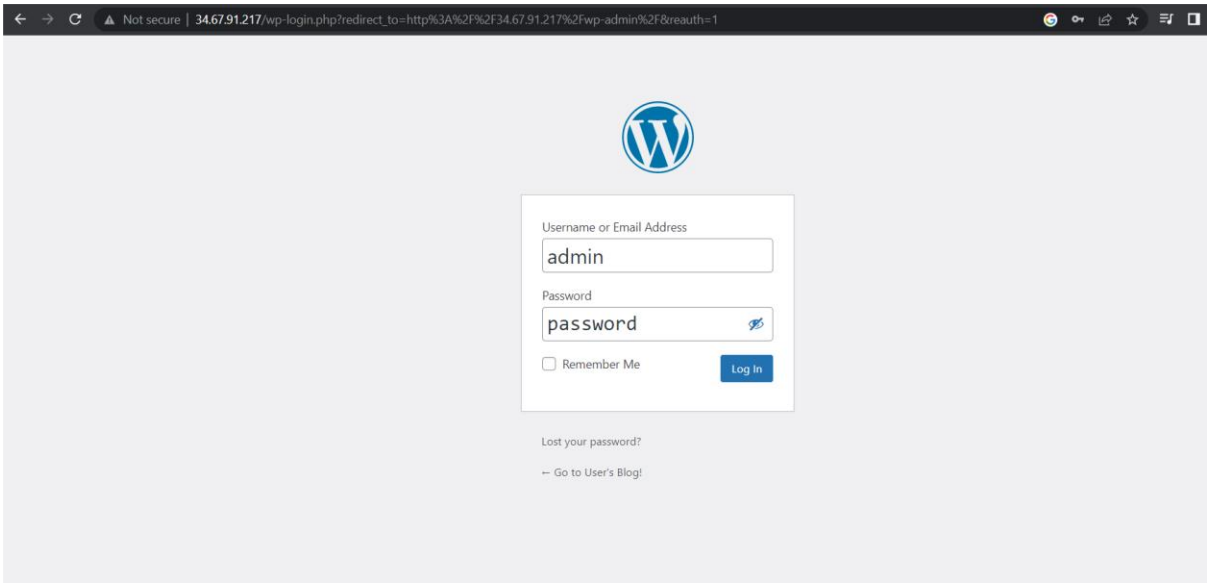
Once installed, it will show us the above output.

```
$ kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/my-release-mariadb-0                1/1     Running   0          16m
pod/my-release-wordpress-7c4555f8-dv865   1/1     Running   0          16m

NAME                           TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)                      AGE
service/kubernetes             ClusterIP      10.8.128.1     <none>         443/TCP                      26m
service/my-release-mariadb     ClusterIP      10.8.128.106   <none>         3306/TCP                     16m
service/my-release-wordpress   LoadBalancer   10.8.130.25    34.67.91.217   80:31068/TCP,443:30938/TCP   16m

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/my-release-wordpress   1/1     1            1           16m

NAME                                             DESIRED   CURRENT   READY   AGE
replicaset.apps/my-release-wordpress-7c4555f8   1         1         1       16m

NAME                                 READY   AGE
statefulset.apps/my-release-mariadb   1/1     16m
```

Now our application is ready to be accessed. We can access the application using the service which is exposing our frontend.

Above the service has received the IP address 34.67.91.217, let's open this in our browser using below.

https://34.67.91.217/admin



9. **Deploy WordPress application using an external Database:**

We are going to use an external database and we will provide the values from a file while installing our WordPress application.

Step 1:

Let's install the MySQL database:

Use the below command first to check the chart in our repo.

```
helm search repo mysql
```

```
root@master:~/helm#
root@master:~/helm# helm search repo mysql
NAME                            CHART VERSION   APP VERSION     DESCRIPTION
bitnami-repo/mysql              9.4.6           8.0.31          MySQL is a fast, reliable, scalable, and easy t...
bitnami-repo/phpmyadmin         10.4.0          5.2.0           phpMyAdmin is a free software tool written in P...
bitnami-repo/mariadb            11.4.4          10.6.11         MariaDB is an open source, community-developed ...
bitnami-repo/mariadb-galera     7.4.11          10.6.11         MariaDB Galera is a multi-primary database clus...
root@master:~/helm#
```

We would get above chart list, now we are going to install mysql and will set some of the values.

```
helm install my-release --set auth.rootPassword=secretpassword,auth.database=app_database bitnami-repo/mysql
```

Use the above command to install MySQL. We have provided below values during the installation which will be used by our WordPress application.

auth.rootPassword=secretpassword

auth.database=app_database

```
                                        ; kubectl get all
NAME                        READY   STATUS    RESTARTS   AGE
pod/my-release-mysql-0      1/1     Running   0          2m25s

NAME                                TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
service/kubernetes                  ClusterIP   10.8.128.1     <none>        443/TCP    148m
service/my-release-mysql            ClusterIP   10.8.130.152   <none>        3306/TCP   2m26s
service/my-release-mysql-headless   ClusterIP   None           <none>        3306/TCP   2m26s

NAME                                READY   AGE
statefulset.apps/my-release-mysql   1/1     2m26s
```

Our database is ready now.

Step 2: Install WordPress application

Let's create a file named **wordpress.values.yaml.**

```
vi wordpress.values.yaml
```

And paste the below information.

```
wordpressUsername: techlanders

wordpressPassword: techlanders123

wordpressEmail: contactus@techlanders.com

wordpressFirstName: techlanders

wordpressLastName: training

wordpressBlogName: techlanders.com

service:

  type: LoadBalancer



externalDatabase:

   host: my-release-mysql.default.svc.cluster.local

   user: root

   password: secretpassword

   database: app_database
```

```
mariadb:

 enabled: false
```

Above we have provided credentials for WordPress admin and external database. This helm chart is using mariadb so we are disabling it as we are going for external database.

Let's install the helm chart now.

```
helm install wordpress bitnami-repo/wordpress –values wordpress.values.yaml
```

```
NAME                                READY   STATUS    RESTARTS   AGE
pod/my-release-mysql-0              1/1     Running   0          15m
pod/wordpress-6b48fd45cb-6k5xc      1/1     Running   0          101s

NAME                                TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)                      AGE
service/kubernetes                  ClusterIP      10.8.128.1     <none>         443/TCP                      161m
service/my-release-mysql            ClusterIP      10.8.130.152   <none>         3306/TCP                     15m
service/my-release-mysql-headless   ClusterIP      None           <none>         3306/TCP                     15m
service/wordpress                   LoadBalancer   10.8.130.255   34.67.91.217   80:32441/TCP,443:30126/TCP   102s

NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/wordpress   1/1     1            1           102s

NAME                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/wordpress-6b48fd45cb   1         1         1       103s

NAME                                READY   AGE
statefulset.apps/my-release-mysql   1/1     15m
```

Now our application is ready and we will access it using the service.

In our case, we are having the IP address **34.67.91.217/admin** to access the WordPress application.

10. Helm Plugins:

We can use Helm plugins to understand more about Helm release. In this exercise, we are going to use diff plugin which is used to show the difference between two versions.

To get the list of plugins installed, we can use below command.

```
helm plugin list
```

Let's install diff plugin, use below command.

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

```
root@master:~#
root@master:~# helm plugin list
NAME    VERSION DESCRIPTION
root@master:~#
```

```
root@master:~# helm plugin install https://github.com/databus23/helm-diff
Downloading https://github.com/databus23/helm-diff/releases/latest/download/helm-diff-linux-amd64.tgz
Preparing to install into /root/.local/share/helm/plugins/helm-diff
helm-diff installed into /root/.local/share/helm/plugins/helm-diff/helm-diff

The Helm Diff Plugin

* Shows a diff explaining what a helm upgrade would change:
    This fetches the currently deployed version of a release
  and compares it to a local chart plus values. This can be
  used visualize what changes a helm upgrade will perform.

* Shows a diff explaining what had changed between two revisions:
    This fetches previously deployed versions of a release
  and compares them. This can be used visualize what changes
  were made during revision change.

* Shows a diff explaining what a helm rollback would change:
    This fetches the currently deployed version of a release
  and compares it to the previously deployed version of the release, that you
  want to rollback. This can be used visualize what changes a
  helm rollback will perform.

Usage:
  diff [flags]
  diff [command]

Available Commands:
  completion  Generate the autocompletion script for the specified shell
  release     Shows diff between release's manifests
  revision    Shows diff between revision's manifests
  rollback    Show a diff explaining what a helm rollback could perform
  upgrade     Show a diff explaining what a helm upgrade would change.
  version     Show version of the helm diff plugin
```

```
root@master:~#
root@master:~# helm plugin list
NAME    VERSION DESCRIPTION
diff    3.6.0   Preview helm upgrade changes as a diff
root@master:~#
```

Now our plugin has been installed.

Let’s explore this by creating our first project using helm chart.

Use the below command.

```
helm install first-project project
```

Below output shows us that we have a deployment created with 2 pods which are exposed using a ClusterIP service.

```
root@master:~/helm#
root@master:~/helm#
root@master:~/helm# helm install first-project project1
NAME: first-project
LAST DEPLOYED: Mon Jan 23 05:22:56 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=project1,app.kubernetes.io/instance=first-project" -o jsonpath="{.items[0]
.metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT
root@master:~/helm#
```

```
root@master:~/helm#
root@master:~/helm# kubectl get all
NAME                                        READY   STATUS    RESTARTS   AGE
pod/first-project-project1-88578c9b-tpbrm   1/1     Running   0          6s
pod/first-project-project1-88578c9b-wwcm9   1/1     Running   0          6s

NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/first-project-project1      ClusterIP   10.104.221.76   <none>        80/TCP     6s
service/kubernetes                  ClusterIP   10.96.0.1       <none>        443/TCP    6d18h

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/first-project-project1   2/2     2            2           6s

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/first-project-project1-88578c9b   2         2         2       6s
```

Let’s make changes in values.yaml file.

We have changed the number of replicas and the service from ClusterIP to NodePort, as shown in below picture.

```yaml
replicaCount: 4

image:
  repository: nginx
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion.
  tag: ""

imagePullSecrets: []
nameOverride: ""
fullnameOverride: ""

serviceAccount:
  # Specifies whether a service account should be created
  create: true
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname template
  name: ""

podAnnotations: {}

podSecurityContext: {}
  # fsGroup: 2000

securityContext: {}
  # capabilities:
  #   drop:
  #   - ALL
  # readOnlyRootFilesystem: true
  # runAsNonRoot: true
  # runAsUser: 1000

service:
  type: NodePort
  port: 80
```

Let’s upgrade our project using below command.

```
Helm upgrade first-project project1
```

Below you can see that our upgrade has been completed.

```
root@master:~/helm#
root@master:~/helm# helm upgrade first-project project1
Release "first-project" has been upgraded. Happy Helming!
NAME: first-project
LAST DEPLOYED: Mon Jan 23 05:26:47 2023
NAMESPACE: default
STATUS: deployed
REVISION: 2
NOTES:
1. Get the application URL by running these commands:
  export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services first-project-project1)
  export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
  echo http://$NODE_IP:$NODE_PORT
root@master:~/helm#
```

Now we have a deployment with 4 pods and service has been changed to NodePort.

Our revision has also changed from 1 to 2.

```
root@master:~/helm#
root@master:~/helm# helm list
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
first-project   default         2               2023-01-23 05:26:47.627645478 +0000 UTC deployed        project1-0.1.0  1.16.0
root@master:~/helm#
root@master:~/helm#
```

```
root@master:~/helm#
root@master:~/helm# kubectl get all
NAME                                            READY   STATUS    RESTARTS   AGE
pod/first-project-project1-88578c9b-rf5d8       1/1     Running   0          15s
pod/first-project-project1-88578c9b-tf8tq       1/1     Running   0          14s
pod/first-project-project1-88578c9b-tpbrm       1/1     Running   0          4m5s
pod/first-project-project1-88578c9b-wwcm9       1/1     Running   0          4m5s

NAME                              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/first-project-project1    NodePort    10.104.221.76   <none>        80:30358/TCP   4m5s
service/kubernetes                ClusterIP   10.96.0.1       <none>        443/TCP        6d18h

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/first-project-project1     4/4     4            4           4m5s

NAME                                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/first-project-project1-88578c9b       4         4         4       4m5s
root@master:~/helm#
root@master:~/helm#
```

Now we can use the below diff command to see the difference between the revisions.

```
root@master:~/helm#
root@master:~/helm# helm diff revision first-project 1 2
default, first-project-project1, Deployment (apps) has changed:
  # Source: project1/templates/deployment.yaml
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: first-project-project1
    labels:
      helm.sh/chart: project1-0.1.0
      app.kubernetes.io/name: project1
      app.kubernetes.io/instance: first-project
      app.kubernetes.io/version: "1.16.0"
      app.kubernetes.io/managed-by: Helm
  spec:
-   replicas: 2
+   replicas: 4
    selector:
      matchLabels:
        app.kubernetes.io/name: project1
        app.kubernetes.io/instance: first-project
    template:
      metadata:
        labels:
          app.kubernetes.io/name: project1
          app.kubernetes.io/instance: first-project
```

```
  spec:
-   type: ClusterIP
+   type: NodePort
    ports:
      - port: 80
        targetPort: http
        protocol: TCP
        name: http
```

Above output will show us the difference.

We can also check the difference between two releases. Let's create another release from the same chart with few changes like replicaCount=2.

helm install second-project project1

```
root@master:~/helm#
root@master:~/helm# helm install second-project project1
NAME: second-project
LAST DEPLOYED: Mon Jan 23 06:01:00 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services second-project-project1)
  export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
  echo http://$NODE_IP:$NODE_PORT
root@master:~/helm#
```

```
root@master:~/helm#
root@master:~/helm# kubectl get all
NAME                                             READY   STATUS    RESTARTS   AGE
pod/first-project-project1-88578c9b-rf5d8        1/1     Running   0          36m
pod/first-project-project1-88578c9b-tf8tq        1/1     Running   0          36m
pod/first-project-project1-88578c9b-tpbrm        1/1     Running   0          40m
pod/first-project-project1-88578c9b-wwcm9        1/1     Running   0          40m
pod/second-project-project1-99bd8b666-dv4z4      1/1     Running   0          5s
pod/second-project-project1-99bd8b666-wzsv2      1/1     Running   0          5s

NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/first-project-project1      NodePort    10.104.221.76   <none>        80:30358/TCP   40m
service/kubernetes                  ClusterIP   10.96.0.1       <none>        443/TCP        6d19h
service/second-project-project1     NodePort    10.96.102.82    <none>        80:31238/TCP   5s

NAME                                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/first-project-project1      4/4     4            4           40m
deployment.apps/second-project-project1     2/2     2            2           5s

NAME                                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/first-project-project1-88578c9b        4         4         4       40m
replicaset.apps/second-project-project1-99bd8b666      2         2         2       5s
root@master:~/helm#
```

```
root@master:~/helm#
root@master:~/helm# helm list
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
first-project   default         2               2023-01-23 05:26:47.627645478 +0000 UTC deployed        project1-0.1.0  1.16.0
second-project  default         1               2023-01-23 06:03:13.81981629 +0000 UTC  deployed        project1-0.1.0  1.16.0
root@master:~/helm#
root@master:~/helm#
```

Above output shows us that we have two releases installed with name **first-project** and **second-project**.

Now let's find out the difference between these releases using below command.

helm diff release first-project second-project

```
root@master:~/helm#
root@master:~/helm# helm diff release first-project second-project
project1/templates/deployment.yaml has changed:
  # Source: project1/templates/deployment.yaml
  apiVersion: apps/v1
  kind: Deployment
  metadata:
-   name: first-project-project1
+   name: second-project-project1
    labels:
      helm.sh/chart: project1-0.1.0
      app.kubernetes.io/name: project1
-     app.kubernetes.io/instance: first-project
+     app.kubernetes.io/instance: second-project
      app.kubernetes.io/version: "1.16.0"
      app.kubernetes.io/managed-by: Helm
  spec:
-   replicas: 4
+   replicas: 2
    selector:
      matchLabels:
        app.kubernetes.io/name: project1
-       app.kubernetes.io/instance: first-project
+       app.kubernetes.io/instance: second-project
    template:
      metadata:
        labels:
          app.kubernetes.io/name: project1
-         app.kubernetes.io/instance: first-project
+         app.kubernetes.io/instance: second-project
      spec:
-       serviceAccountName: first-project-project1
+       serviceAccountName: second-project-project1
project1/templates/service.yaml has changed:
  # Source: project1/templates/service.yaml
  apiVersion: v1
  kind: Service
  metadata:
-   name: first-project-project1
+   name: second-project-project1
    labels:
      helm.sh/chart: project1-0.1.0
      app.kubernetes.io/name: project1
-     app.kubernetes.io/instance: first-project
+     app.kubernetes.io/instance: second-project
      app.kubernetes.io/version: "1.16.0"
      app.kubernetes.io/managed-by: Helm
  spec:
    type: NodePort
    ports:
      - port: 80
        targetPort: http
        protocol: TCP
        name: http
    selector:
      app.kubernetes.io/name: project1
-     app.kubernetes.io/instance: first-project
+     app.kubernetes.io/instance: second-project
project1/templates/serviceaccount.yaml has changed:
  # Source: project1/templates/serviceaccount.yaml
  apiVersion: v1
  kind: ServiceAccount
  metadata:
-   name: first-project-project1
+   name: second-project-project1
```

There is another command, we can see the difference what a rollback can do using the below command.

```
helm diff rollback first-project 1
```

We can uninstall the plugin using below command.

```
helm plugin uninstall diff
```

Use the below command to package our release.

```
helm package project1
```

Use below command to pull a chart from our hub.

```
helm pull bitnami-repo/wordpress
```

Use below command to pull a chart from our hub in unzipped format.

```
helm pull bitnami-repo/wordpress --untar
```

See the output below, we have the packages ready which can be pushed or shared with other team.

```
root@master:~/helm#
root@master:~/helm# ls
project1  project1-0.1.0.tgz  wordpress  wordpress-15.2.31.tgz
root@master:~/helm#
```