**Helmcharts in Kubernetes**

Until now we only worked with two or three YAML files. Deploying these two or three YAML file of an application in the Kubernetes cluster is simple and easy way.

But deploying multiple YAML files for a single application in a Kubernetes cluster can become complex and difficult to manage.

To simplify (overcome) this, we use **Helm Charts**, which act as a package manager for Kubernetes.

Helm allows you to bundle, configure, and deploy applications with ease, reducing manual effort and improving maintainability.

**Helm:**

**Helm**is the **package manager for Kubernetes** that helps deploy and manage applications efficiently.

It simplifies the deployment, management, and versioning of Kubernetes applications by bundling them into reusable, shareable units called **Charts**.

**Helm Charts**—pre-configured templates for Kubernetes resources.

By using Helm chart we can install an application which is built/consist of multiple YAML files like (deployment.yaml, service.yaml and replica set configuration) these all are placed in a single chart.

**Helm addresses key challenges in Kubernetes deployments:**

* **Simplifies Deployment** – Instead of managing multiple YAML files manually, Helm bundles them into a single **Helm Chart**, making deployment easier.
* **Templating & Reusability** – Allows you to use variables and templates, making it reusable across different environments (dev, staging, production).
* **Version Control & Rollbacks** – Helm maintains a history of releases, allowing you to **rollback** to a previous version if needed.
* **Easier Application Management** – You can upgrade, configure, and manage applications with simple commands like **“helm install”** or **“helm upgrade”**.
* **Standardization** – Helm charts provide a standardized way to deploy applications, making it easier for teams to collaborate.
* **Supports Configuration Customization**: Helm uses “**values.yaml”** to parameterize deployments

**Key terms:**

1. **Helm Charts**: A collection of YAML files defining Kubernetes resources.
2. **Helm Repository**: A place where Helm charts are stored and shared.
3. **Helm CLI**: A command-line tool to install, upgrade, and manage applications.

Installing of Helm char in the Linux machine:

1. curl https://baltocdn.com/helm/signing.asc | sudo apt-key add –
2. sudo apt-get install apt-transport-https –yes
3. echo "deb https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
4. sudo apt-get update
5. sudo apt-get install helm
6. helm version --short
7. apt install tree: [Instead of simply listing files and directories in a flat format (like the ls command does), tree displays them in a hierarchical, tree-like structure].

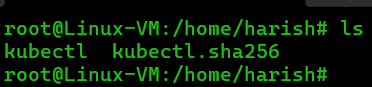


Fig: Normal listing of directories using “ls”.

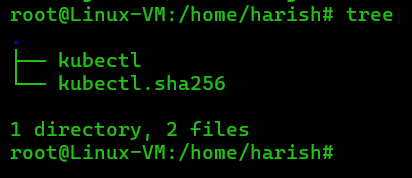
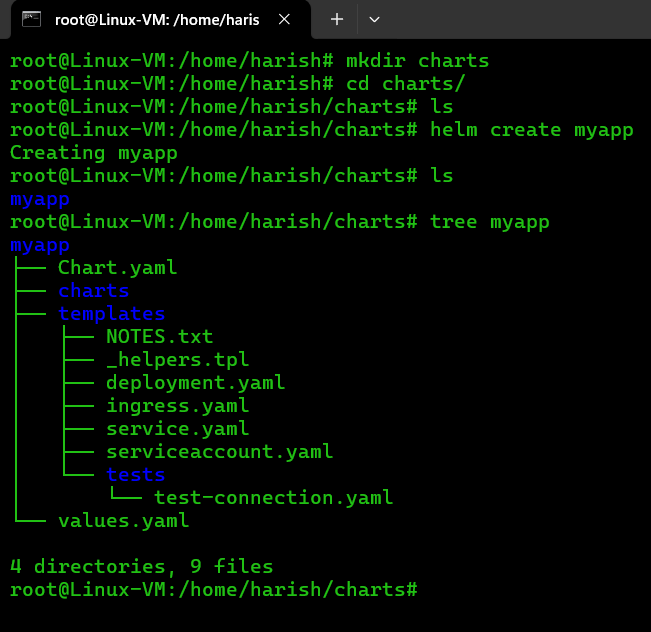


Fig: Tree like structure viewing of directories.

Let’s work with the Helm charts in practically.

**Step1:** Create a Helm chart with name “myapp” in a separate directory “charts”.

**Command to create Helm chart:** helm create myapp (it will create empty helm chart with name “myapp”).



**Chart.yaml:** This is a file, commonly used in Helm charts, that contains metadata about the chart.

**charts:** This directory typically contains subcharts, which are other Helm charts that this chart depends on (it contains dependences )

**templates:** This directory contains template files that define the Kubernetes resources that the chart will deploy.

**NOTES.txt:** This file often contains instructions or notes that are displayed to the user after the chart is deployed.

**\_helpers.tpl:** This file contains template helpers, which are reusable template snippets that can be used in other template files.

**deployment.yaml**: This file defines a Kubernetes Deployment, which manages the deployment of a set of identical pods.

**ingress.yaml:** This file defines a Kubernetes Ingress, which manages external access to the services in the cluster.

**service.yaml:** This file defines a Kubernetes Service, which provides a stable endpoint for accessing a set of pods.

**serviceaccount.yaml:** This file defines a Kubernetes ServiceAccount, which provides an identity for pods running in the cluster.

**test-connection.yaml:** This file typically contains a test that checks if the deployed application is running correctly.

**values.yaml:** This file contains default values for the variables that are used in the chart's templates.

We can also create the Helm chart manually by creating each directory and files in a hieratical format.

1. mkdir myapp
2. cd myapp/
3. touch Chart.yaml

apiVersion: v2

name: myapp

version: 0.1.0

appVersion: 1.0

description: A Helm chart for Kubernetes

1. mkdir templates
2. mkdir charts
3. Vi Chart.yaml

**Fig:** Chart.yaml file. (vi Chart.yaml)

1. cd template/
2. touch deployment.yaml
3. Vi deployment.yaml

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: nginx-replicaset

labels:

app: nginx

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

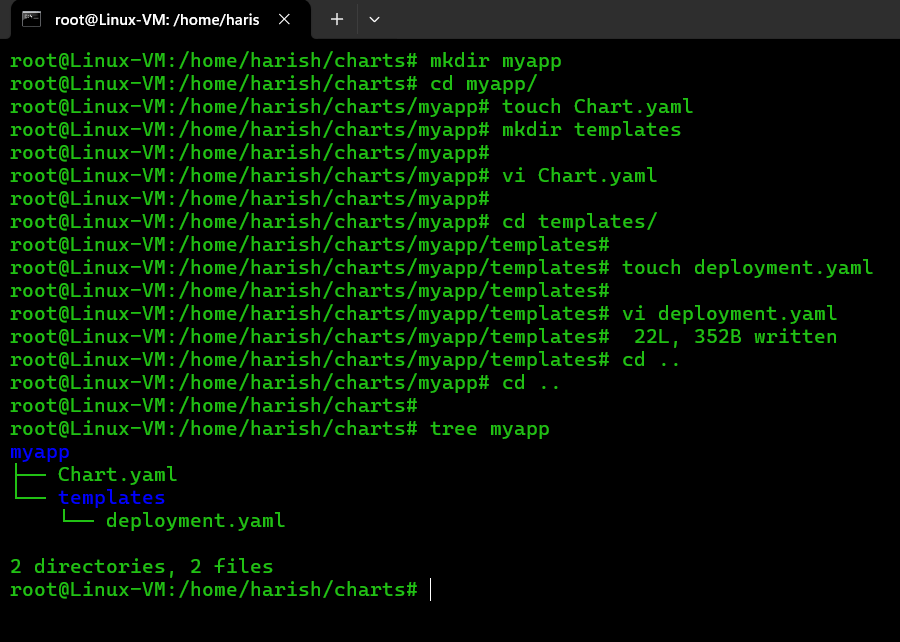
- name: nginx-container

image: nginx:latest

ports:

- containerPort: 80

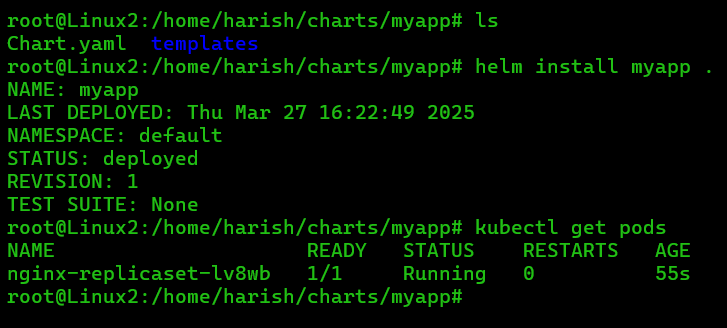
**Fig:** deployment.yaml file (vi deployment.yaml).



**Step2:** Now install the myapp helm chart using below command.

**Command:** helm install myapp .

**Note:** Execute this command by standing in the path of “myapp” directory only as shown in below figure.



Hence helm is installed successfully as show in above figure.

**Note:** If we have multiple YAML files in the helm chart by using **“heml install <chart-name>”** we can execute all YAML files at a time with single command.

**#upgrading of Helm chart:**

**Step3:** Upgrading of changes of Helm chart using the command **“helm upgrad <chart-name>”.**

Let’s make any changes in any YAML file and then upgrade the changes of Helm Chart.

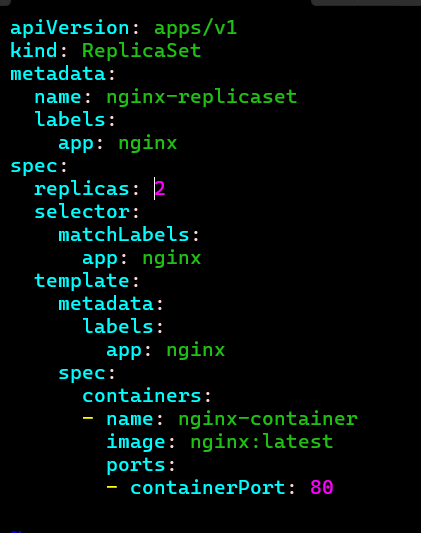


Fig:changed deployment.yaml file.

In above figure we made a change in the replica set value from 1 to 2 of a deploymet.yaml file. Then upgrade the Helm chart by using the below command.

**Command:** helm upgrade <chart-name> .

**EX:** helm upgrade myapp .

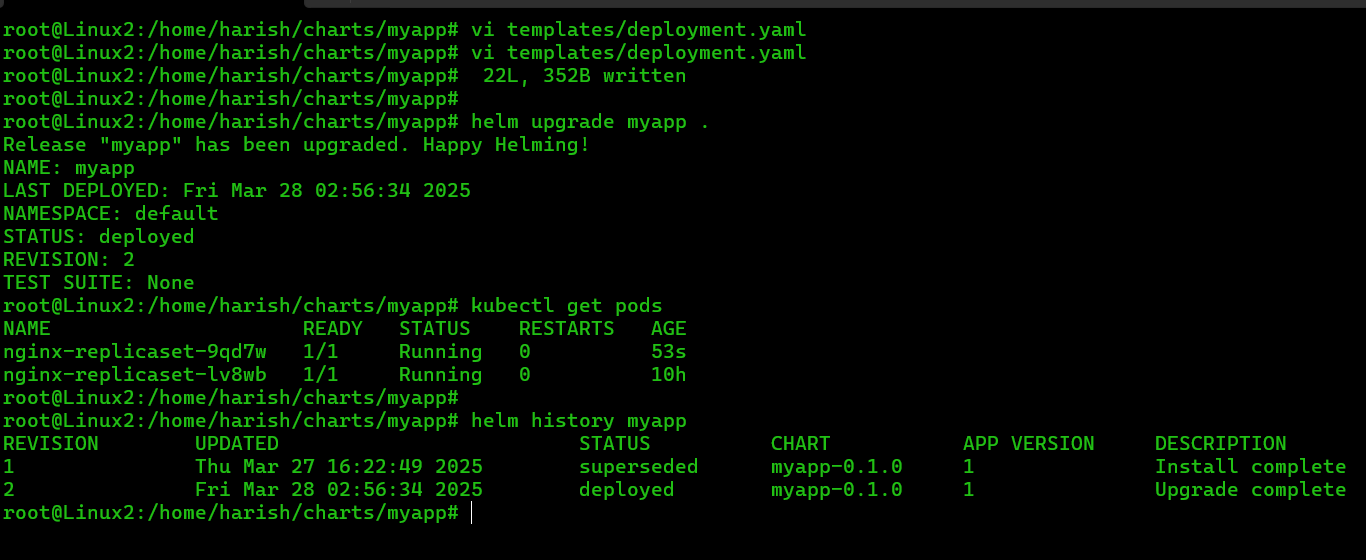
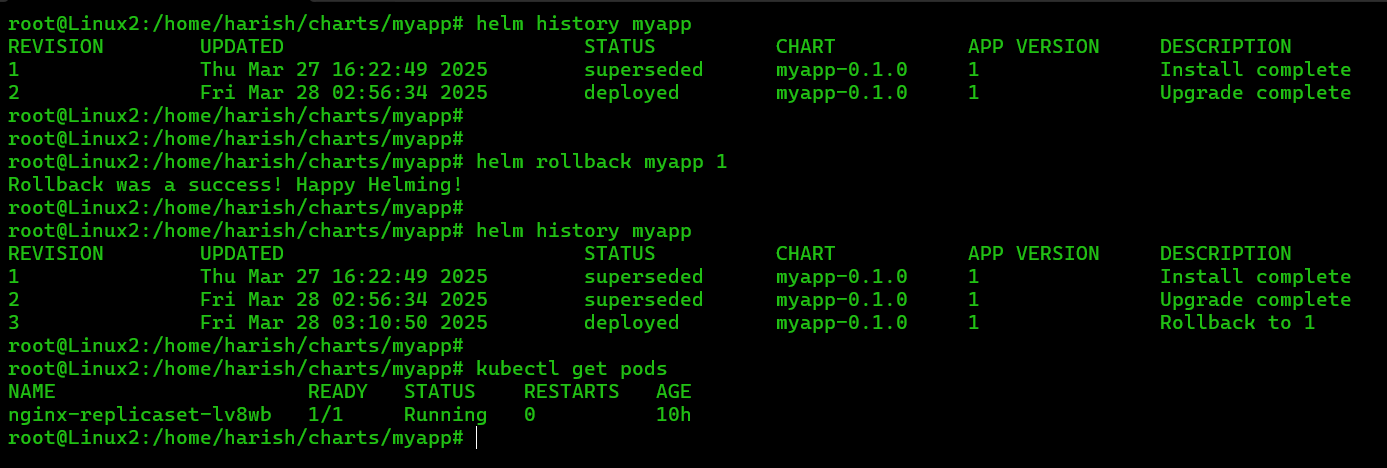


Fig: Helm chart is upgraded successfully.

**Helm Revision:**

* Helm, the package manager for Kubernetes, maintains a history of releases, where each upgrade, rollback, or modification creates a new **revision**.
* In Helm, a "revision" represents a specific version of a release.
* Each time you install, upgrade, or rollback a Helm chart, a new revision is created.
* Essentially, a revision is a snapshot of the chart's state at a particular point in time.

**#rollback of helm chart: (command:** heml rollback myapp <revision-number>**)**

****

From above figure we can say that each rollback of helm chart creates a new revision as shown in above figure (revision: 3).

By using rollback command we go to previous version if we needed

In revision 2 we have two pods, when we roll back to the revision 1, then number of pods become 1, so we roll backed from revision 2 to revision 1. That means version-1 to the version-2.

**Step4:** Now add a “service.yaml” file to the helm chart “myapp”.

**Note:** This service.yaml (All YAML files) files are placed to the template folder only.

apiVersion: v1

kind: Service

metadata:

name: my-loadbalancer-service

spec:

type: LoadBalancer

ports:

- port: 80

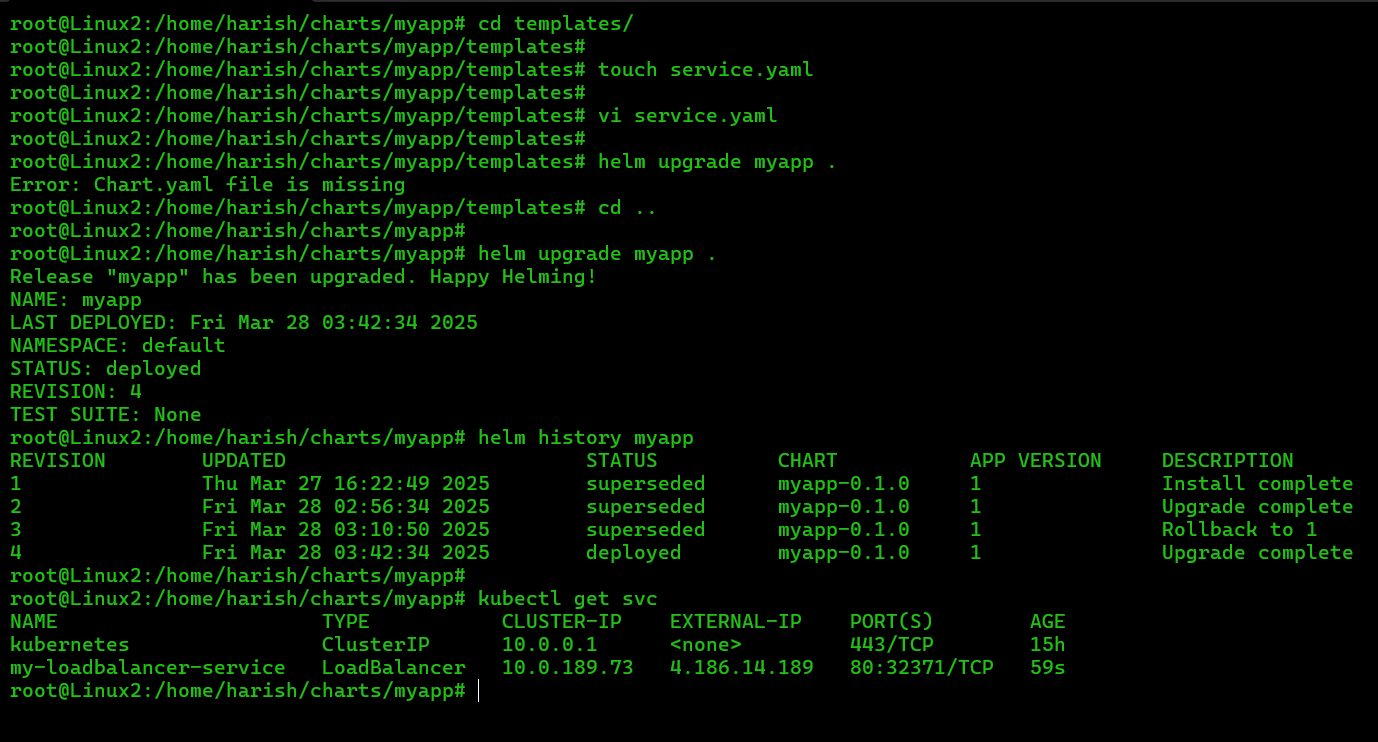
targetPort: 80

protocol: TCP

selector:

app: nginx

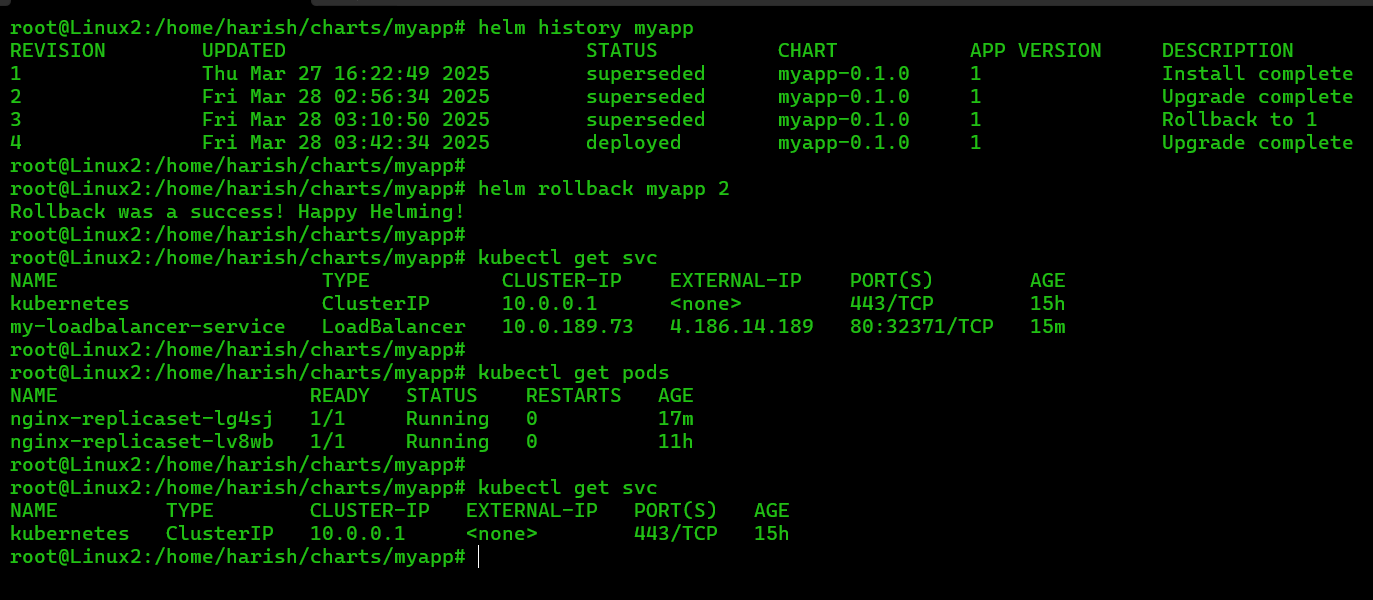
Fig: Service.yaml file.



When we add the service to the helm chart it will also add one more revision (revision 4) as shown in the above figure.

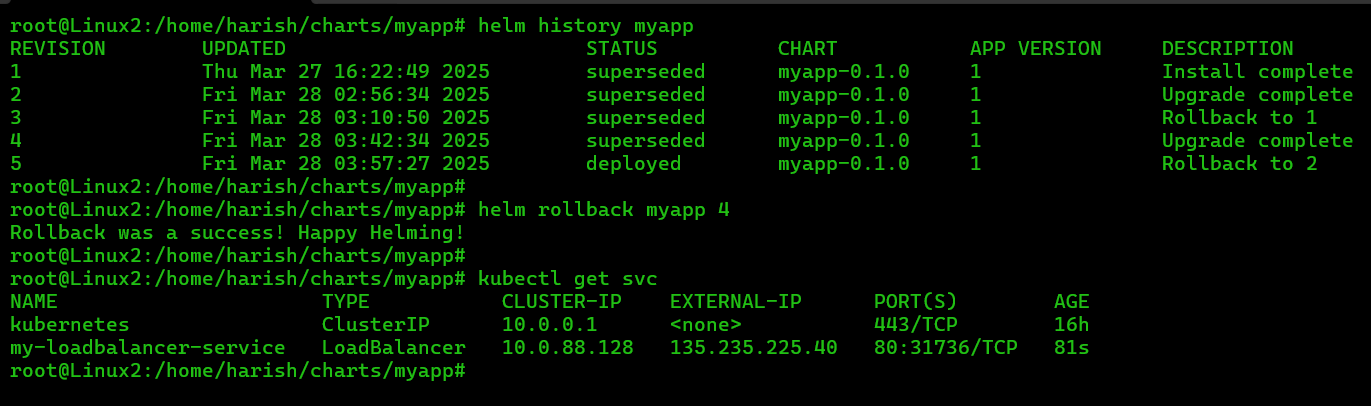
**Step5:** Now again do the role back to the previous revision (revision 2).

When we do rollback to the revision 2 then the service.yaml file will be deleted and two pods gets remains as shown in below figure.



The service.yaml file is attached to the revision 4 of the helm chart.

If we roll back to the revision 4 then we will get the service.yaml again as shown in below figure.



**Step6:** Uninstall the helm chart “myapp”.

**Command:** helm uninstall <helm chart-name> .

**Ex:** helm uninstall myapp .

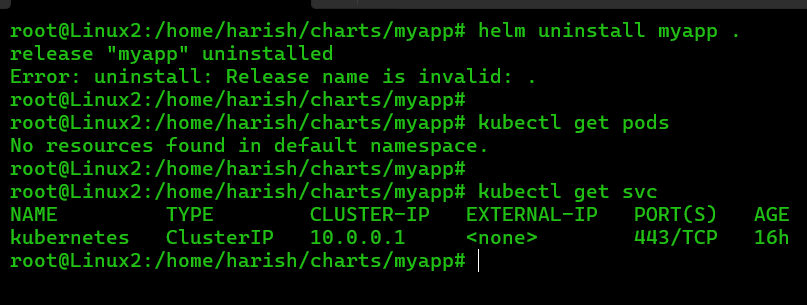
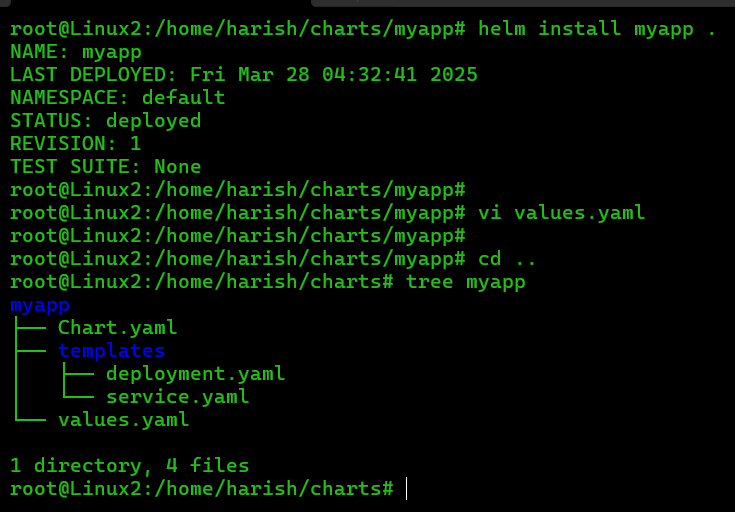


Fig: Helm chart is uninstalled successfully.

**Working with “values.yaml” file in the Helm chart:**

**Step1:** Create a values.yaml file in the myapp directory by installing “myapp” again.



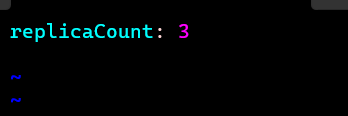


Fig: values.yaml

**Step2:** Edit the deployment.yaml file by placing the variable at replica set.

vi deployment.yaml

replicas: {{ .Values.replicaCount}}

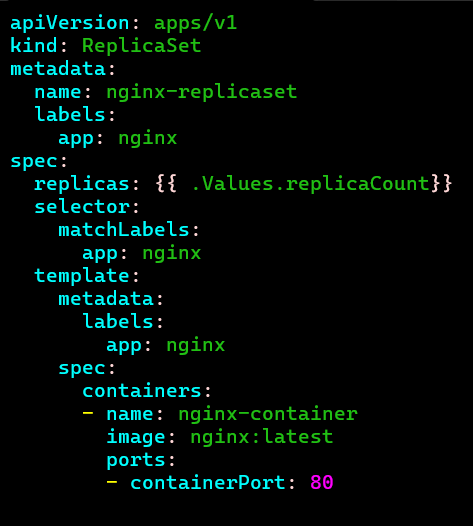


Fig: Editing of deployment.yaml file.

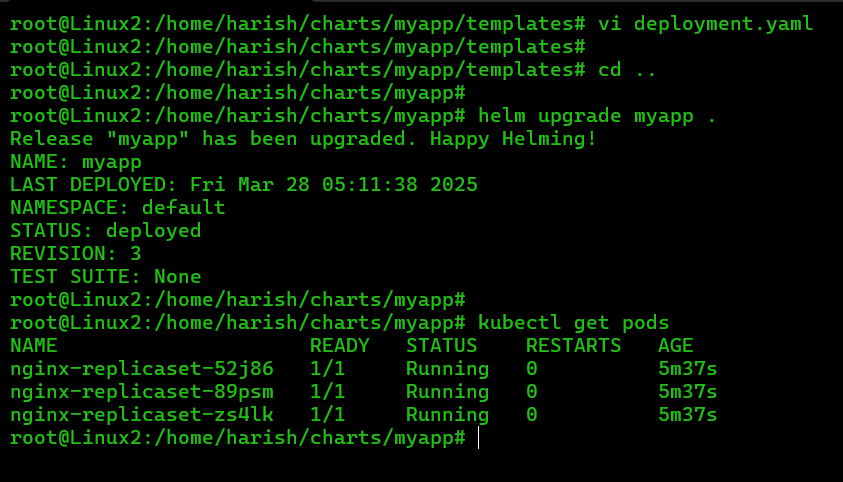


Fig: Three pods are created successfully created using values.yaml file.

We can also set the replica set using command line as well by using below command.

**Command:** helm upgrade <chart-name> . --set replicaCount=4

**Ex:** helm upgrade myapp . – set replicaCount=4

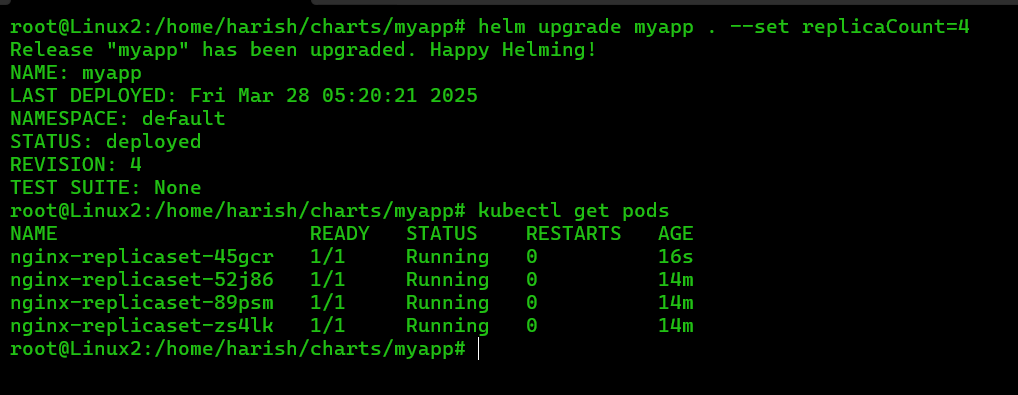


Fig: Updated the deployment.yaml file replica set using CLI

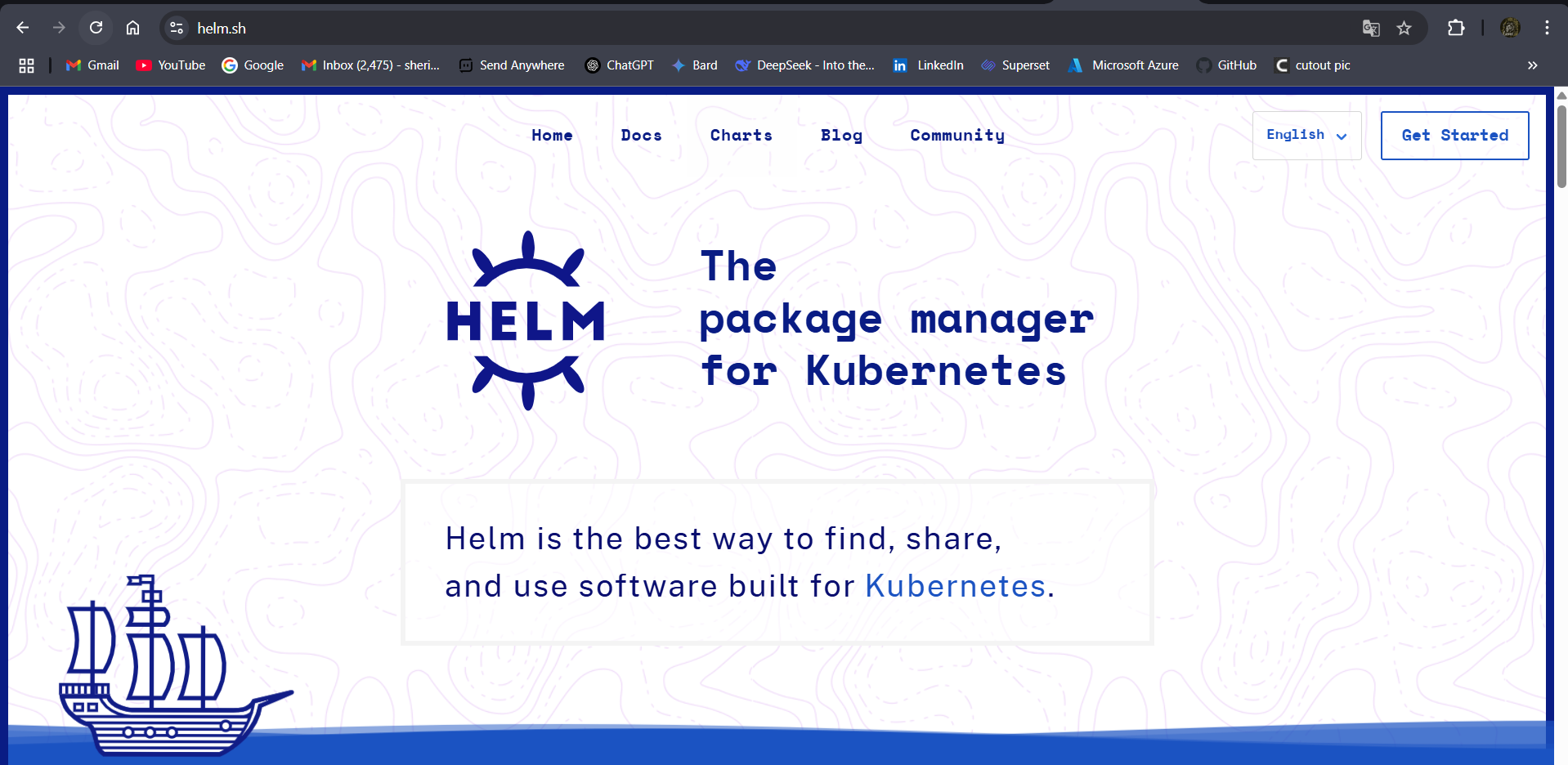
The number of pods is increased to 4.

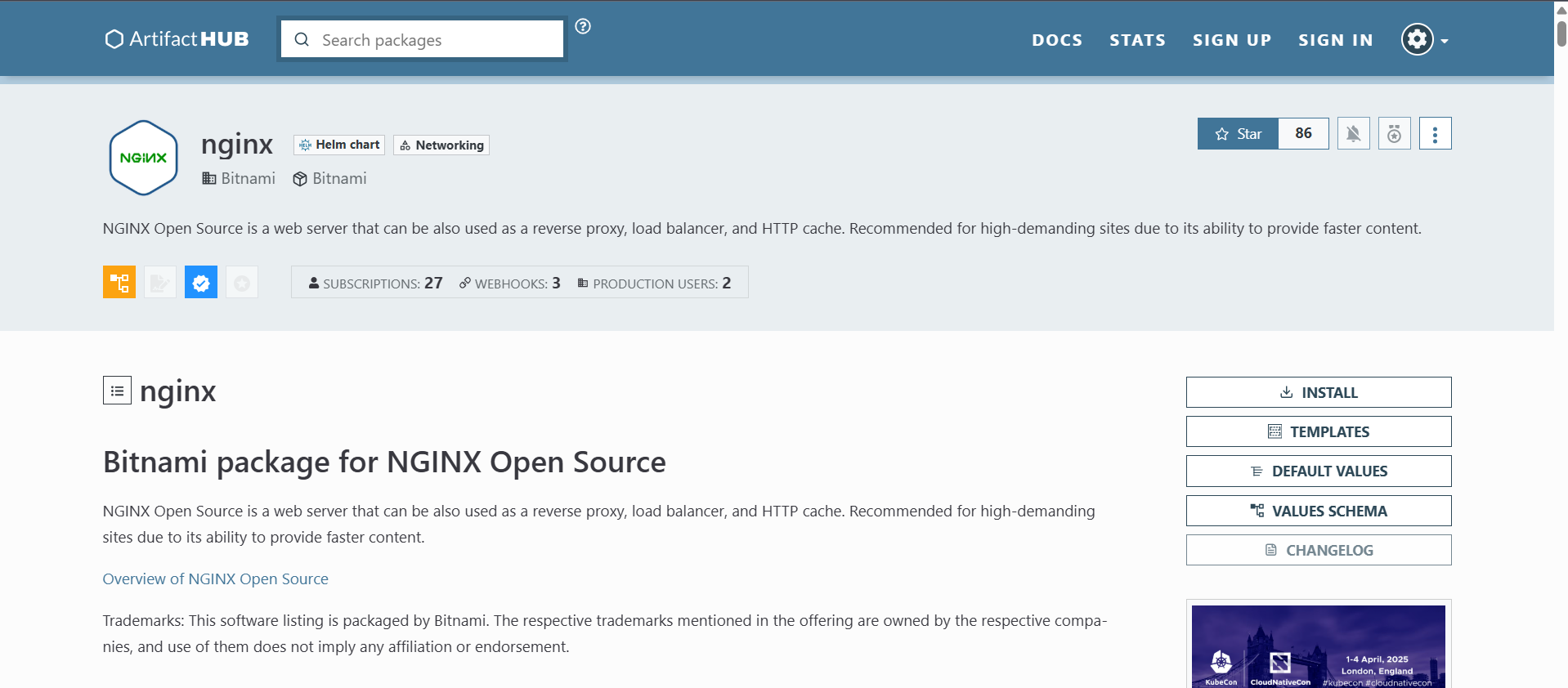
**Working with the Dependences (chart) in Helm chart:**

Now will talk about dependency management..

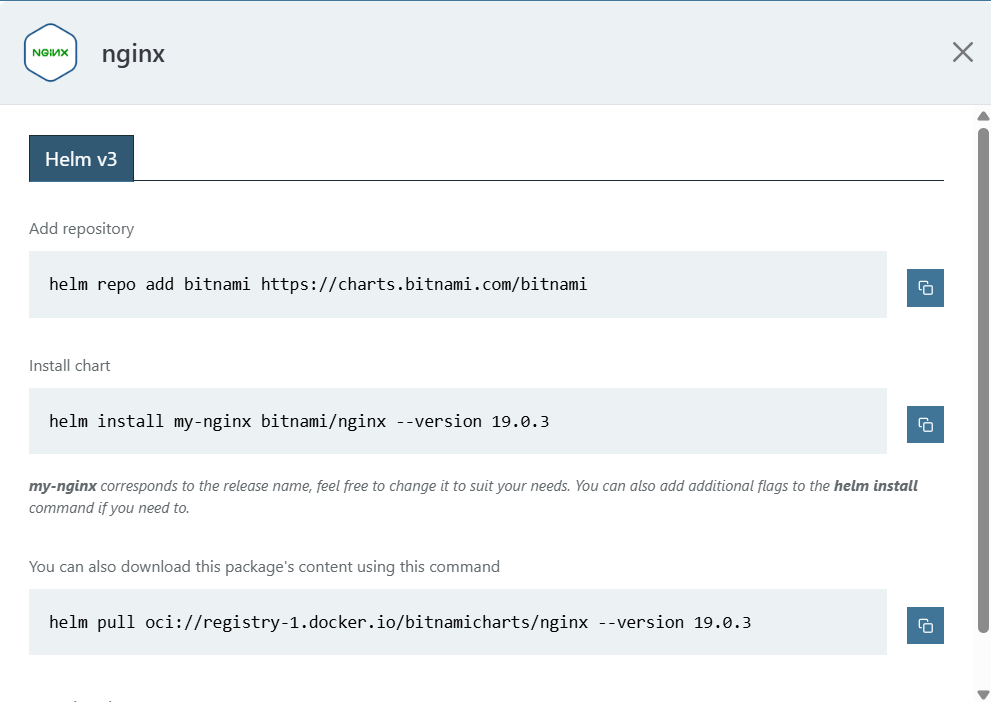
For example my chart is depending on another chart before deploying my chart, I need to deploy another chart.

You don’t need to install, but you can specify the specific chart from the ‘**helm.sh”** web browser.

This web browser is just like a Docker Hub but here we can find lot of charts like Nginx, MySQL and so on. 



Let’s download the dependences of Nginx by creating the chart folder/directory and editing the Chart.yaml file.



dependencies:

- name: nginx

version: "19.0.3"

repository: " "https://charts.bitnami.com/bitnami "

Fig: dependancy’s.

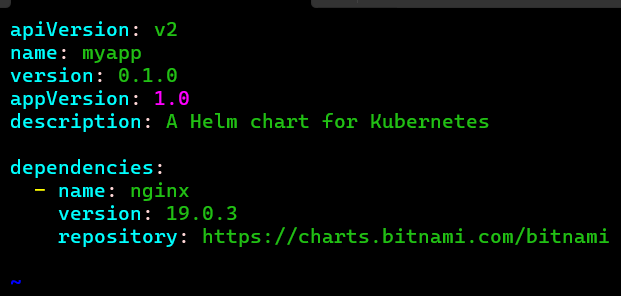
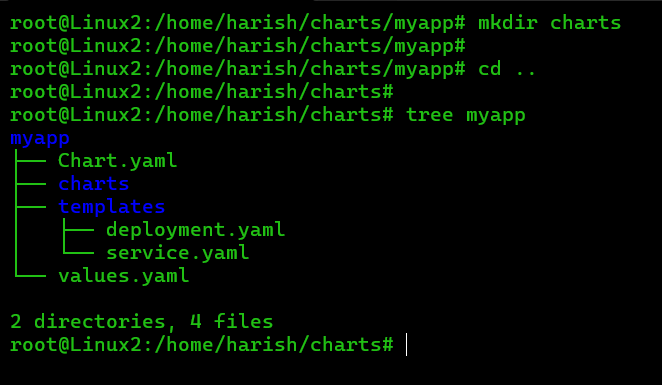


Fig: chart.yaml file

**Step2:** Create the chart directory in the myapp folder.

In chart directory all the .exe of Nginx are downloads in it.



Now you need to build dependencies using the below commands

**Command:** helm dep build

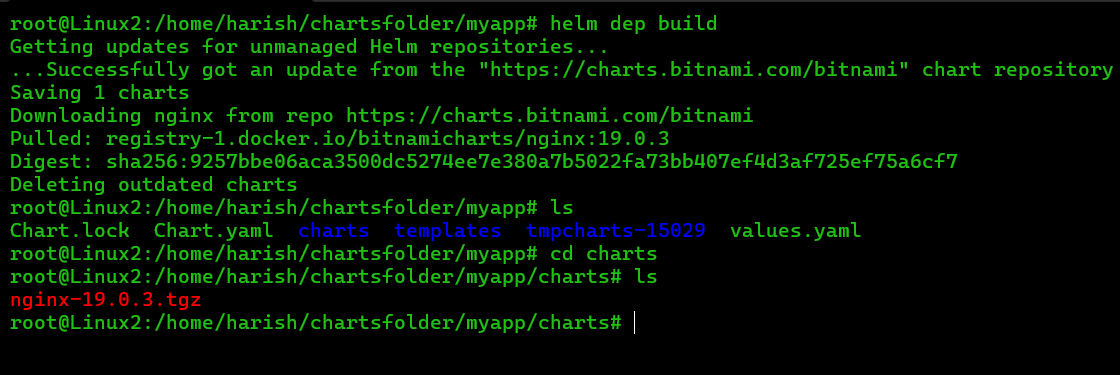


Fig: Nginx zip file is downloaded successfully.

Now you can see, in the charts folder, you wil have zip file

