Mithun Technologies Bangalore
devopstrainingblr@outlook.com,
devopstrainingblr@gmail.com **+91-8296242028,+91-9980923226**

Helm

Helm is a package manager for Kubernetes applications.

Helm is the package manager for Kubernetes. **It allows you to install/deploy/remove/update applications on your Kubernetes cluster in a similar** manner to yum/apt for Linux distributions.

Helm lets you fetch, deploy and manage the lifecycle of applications, both 3rd party products and your own.

Helm introduces several familiar concepts such as:

- Charts which are the Helm packages (like deb/rpm)
- Helm repositories which hold charts (similar to package repos)
- A CLI with install/upgrade/remove commands.

Why use Helm?

Kubernetes can be difficult to manage with all the objects you need to maintain (ConfigMaps/Secrets, pods, services, etc.). Helm manages all of this for you.

Helm greatly **simplifies the process of creating, managing, and deploying applications using Helm Charts.**

In addition, Helm also maintains a versioned history of every chart(Application) installation. If something goes wrong, you can simply call "helm rollback."

Similarly, if you need to upgrade a chart, you can simply call "helm upgrade."

Deploying applications to Kubernetes – the powerful and popular container-orchestration system – can be complex. Setting up a single application can involve creating multiple interdependent Kubernetes resources – such as pods, services, deployments, and replicasets – each requiring you to write a detailed YAML manifest file.

Mithun Technologies Bangalore
devopstrainingblr@outlook.com,
devopstrainingblr@gmail.com **+91-8296242028,+91-9980923226**

What is a Helm Chart?

In Helm, **a chart is basically just a collection of manifest files organized in a specific directory structure that describe a related Kubernetes resource**. There are two main components to a chart: templates and values. These templates and values go through a template rendering engine, producing a manifest that is easily digestible by Kubernetes.

Helm uses Charts to pack all the required K8S components(Manifests) for an application to deploy, run and scale.

Charts are very similar to RPM and DEB packages for Linux, making it easy for developers to distribute and consume/install applications via different repositories.

Helm concepts | Charts

Helm packages are called *charts*, and they consist of a few YAML configuration files and some templates that are rendered into Kubernetes manifest files. Here is the basic directory structure of a chart:

```
javawebapp
├── Chart.yaml # Meta data information about chart.
├── charts          # define dependent helm chart names
├── templates    # Will maintain k8's manifest files which is required for our application
│   ├── deployment.yaml
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   └── serviceaccount.yaml
│
└── values.yaml      # Will define default values which will be referred in templates.
```

These directories and files have the following functions:

- **charts/:** Manually managed chart dependencies can be placed in this directory, though it is typically better to use `requirements.yaml` to dynamically link dependencies.

- **templates/:** This directory contains template files that are combined with configuration values (from values.yaml and the command line) and rendered into Kubernetes manifests. The templates use the Go programming language's template format.
- **Chart.yaml:** A YAML file with metadata about the chart, such as chart name and version, maintainer information, a relevant website, and search keywords.
- **values.yaml:** A YAML file of default configuration values for the chart.

The helm command can install a chart from a local directory, or from a .tar.gz packaged version of this directory structure. These packaged charts can also be automatically downloaded and installed from chart repositories or *repos*.

Helm concepts | Repository

Repositories are where helm charts are held and maintained. In essence, these are a set of templates and configuration values stored as code(sometimes packed as a .tar.gz file).
An excellent example of using a Nginx ingress Chart, which used to be maintained by the company in the Helm/Charts repo.
Nginx has decided to move their product to their Helm repo, and to get the latest official Chart you can now add their repo to Helm by:

```
$ helm repo add nginx-stable https://helm.nginx.com/stable
```

```
$ helm repo update
```


followed by:

```
$ helm install nginxingress nginx-stable/nginx-ingress
```

Helm concepts | Release

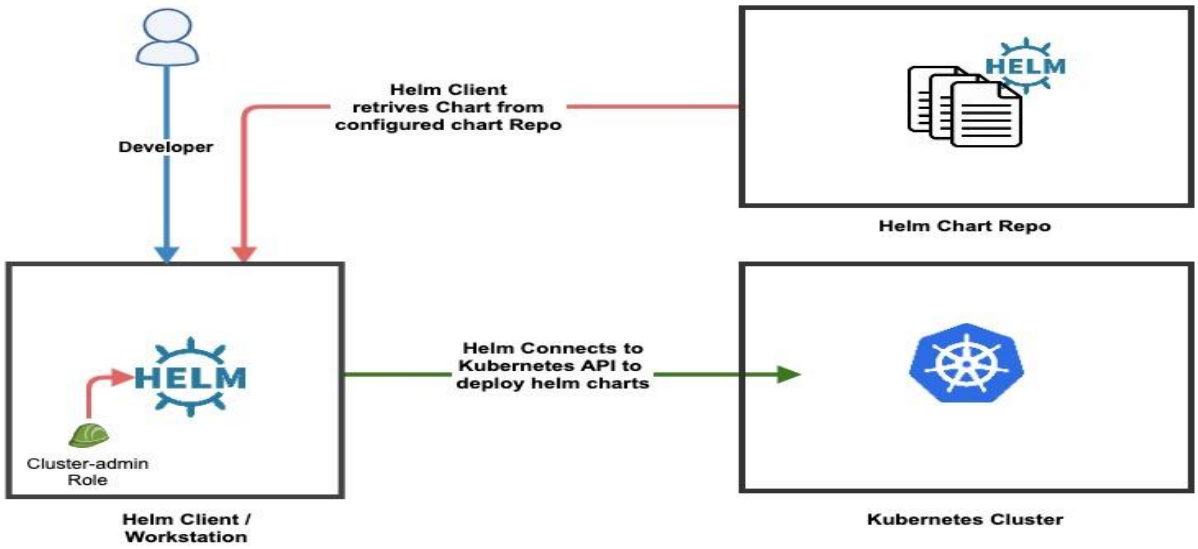
Think of a release as a mechanism to track installed applications on a K8S cluster. when an application is installed by Helm, a release is being created.

Releases can be tracked with helm ls, each would have a "revision" which is the Helm release versioning terminology; if a specific release is updated, e.g., adding more memory or update image tag to the nginx-ingress release, the revision would be incremented. Helm allows rolling back to a particular revision.

Mithun Technologies Bangalore
devopstrainingblr@outlook.com,
devopstrainingblr@gmail.com **+91-8296242028,+91-9980923226**

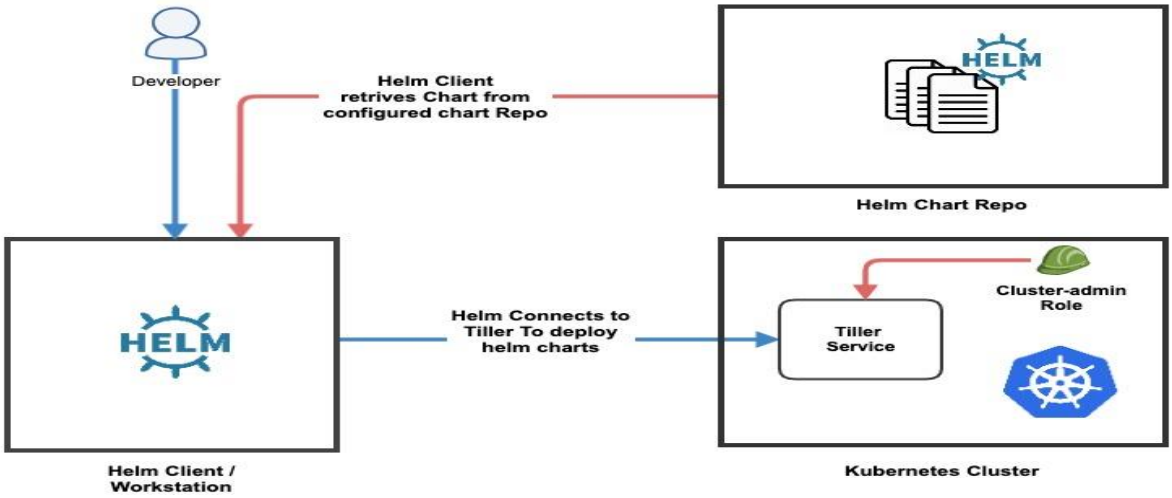
Helm 3 Architecture

In helm 3 there is no tiller component. Helm client directly interacts with the kubernetes API for the helm chart deployment.



Helm 2 Architecture

In helm 2 there is a helm component called tiller which will be deployed in the kubernetes kube-system namespace. Tiller components is removed in helm 3 versions.



Mithun Technologies Bangalore
devopstrainingblr@outlook.com,
devopstrainingblr@gmail.com **+91-8296242028,+91-9980923226**

Installing Helm 3

Note: The workstation you are running should have the kubectl context set to the cluster you want to manage with Helm.Helm works inside Kubernetes and uses by default the kubeconfig file ("~/.kube/config").

You can use another file if you set the environment variable $KUBECONFIG. Download the latest helm 3 installation script.

```
$ curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
```

Add execute permissions to the downloaded script & run.

```
$ chmod 700 get_helm.sh

$ ./get_helm.sh
```

Validate helm installation by executing the helm command.

```
$ helm
```

Now, add the public stable helm repo for installing the stable charts.

```
$ helm repo add nginx-stable https://helm.nginx.com/stable

$ helm repo update
```

Let's install a stable nginx chart and test the setup.

```
$ helm install nginxingress nginx-stable/nginx-ingress
```

List the installed helm chart

```
$ helm ls
```

You can add more number of repos

```
$ helm repo add stable https://charts.helm.sh/stable

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

$ helm search repo stable

$ helm search repo bitnami
```

Mithun Technologies Bangalore
devopstrainingblr@outlook.com,
devopstrainingblr@gmail.com **+91-8296242028,+91-9980923226**

Basic commands

To upload a chart to Kubernetes, you can use the following command:

helm install [NAME] [CHART]

helm install [CHART] `–generate-name`

helm install [NAME] [CHART] `–dry-run --debug`

ex:

```
$ helm install nginxingress nginx-stable/nginx-ingress
```

If you want to upgrade your chart, use the following command and choose the release you want:

helm upgrade [RELEASE] [CHART]

You can add the flag -i or --install if you want to run an install before if a release by this name doesn't already exist.

Otherwise, you can do a rollback. If you do not specify the revision, it will roll back to the previous version.

helm rollback [RELEASE] [REVISION]

To retrieve a package from a package repository, and download it locally:

helm pull [CHART]

You have two commands to search:

helm search hub

This command search for charts in the Helm Hub.

helm search repo

If you want to uninstall a release, here is the command:

helm uninstall [RELEASE]

Repository commands

You have a few commands to interact with chart repositories. A repository is a place where you can find and share charts. You can add a chart repository:

helm repo add [NAME] [URL]

You can list chart repositories:

helm repo list | ls

You can remove a chart repository:

helm repo remove | rm [NAME]

And finally, you can get the latest information about charts from the respective chart repositories:

helm repo update

helm pull [NAME] --untar

Monitoring commands

helm status [RELEASE]

This command shows you the status of your release. The status consists of:

- last deployment time
- kubernetes namespace in which the release lives
- state of the release (can be: unknown, deployed, uninstalled, superseded, failed, uninstalling, pending-install, pending-upgrade or pending-rollback)
- list of resources that this release consists of, sorted by kind

- details on last test suite run, if applicable
- additional notes provided by the chart

You can list all applications that have been installed in the cluster using helm.

helm list | helm ls:

You can see with this command all the historical revisions for a given release.

helm history [RELEASE]

Create your own chart

helm create [NAME]

This command creates a chart directory with the common files and directories used in a chart.

For example, 'helm create javawebapp' will create a directory structure:

```
$ helm create javawebapp
```

```
javawebapp/
       ├── .helmignore         # Contains patterns to ignore when packaging Helm
charts.
       ├── Chart.yaml    # Information about your chart
       ├── values.yaml    # The default values for your templates
       ├── charts/      # Charts that this chart depends on
       ├── templates/    # The template files
       └── tests/             # The test files
```

Update values.yml file and templates files based on your requirement.

This command takes a path to a chart and runs a series of tests to verify that the chart is well-formed

helm lint [CHART]

```
$ helm lint javawebapp
```

This command packages a chart into a versioned chart archive file.

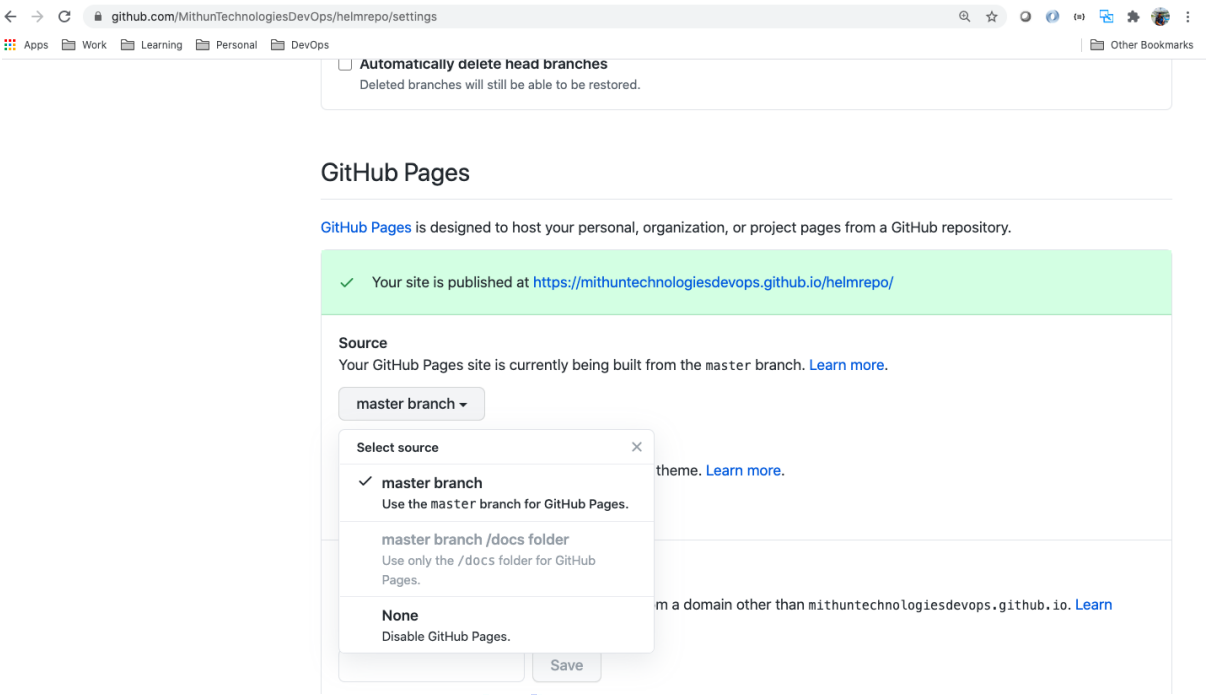
Mithun Technologies Bangalore
devopstrainingblr@outlook.com,
devopstrainingblr@gmail.com **+91-8296242028,+91-9980923226**

helm package [CHART]

```
$ helm package javawebapp
```

Configure the "helm-chart" repository as a Github pages site

Now it's time to publish the contents of our git repository **as Github page**s. Go back to your browser, in the **"settings"** section of your git repository, scroll down to **Github Pages** section and configure it as follow:



Generate index.yaml file and add chart tar into git hub.

```
$ git clone https://github.com/MithunTechnologiesDevOps/helmrepo.git

$ mv javawebapp-0.1.0.tgz helmrepo

$ helm repo index helmrepo --url
https://mithuntechnologiesdevops.github.io/helmrepo/

Note: Above URL(Repo) is GitHub Page Link Which We can get from Git Hub Pages
of our repo.


$ cd helmrepo

$ git add .
$ git commit -a -m "Added Chart Files"
$ git push origin
```

OK, we did it! Now we can add this repo to another Helm installation:

Repo URL is Git Hub Page URL.

```
$ helm repo add helmrepo https://mithuntechnologiesdevops.github.io/helmrepo/
```

```
$ helm repo list
```

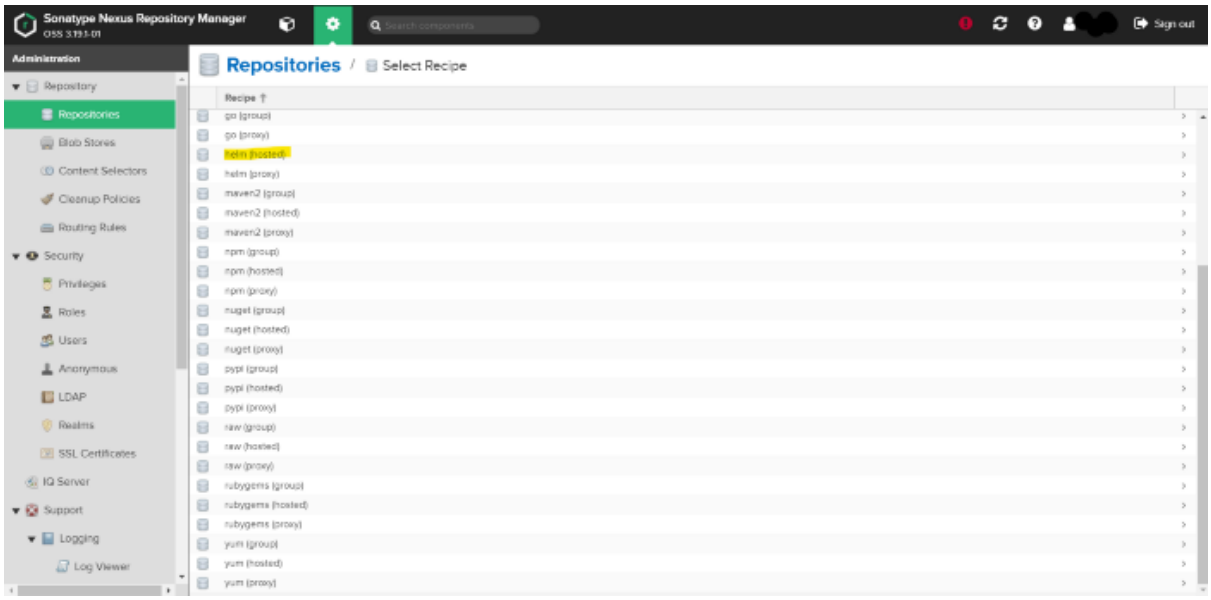
```
NAME            URL
helmrepo    https://mithuntechnologiesdevops.github.io/helmrepo/
```

Now check it by creating a new deploy from the repo:

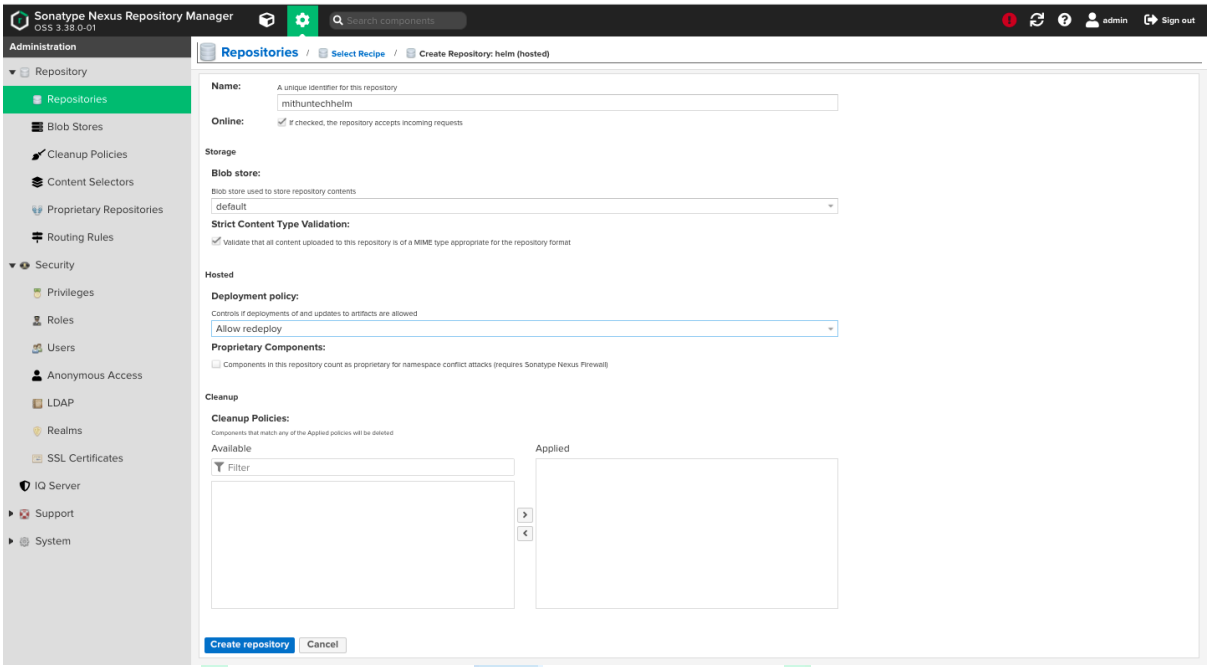
```
$ helm install helmrepo/javawebapp --name=javawebapp
```

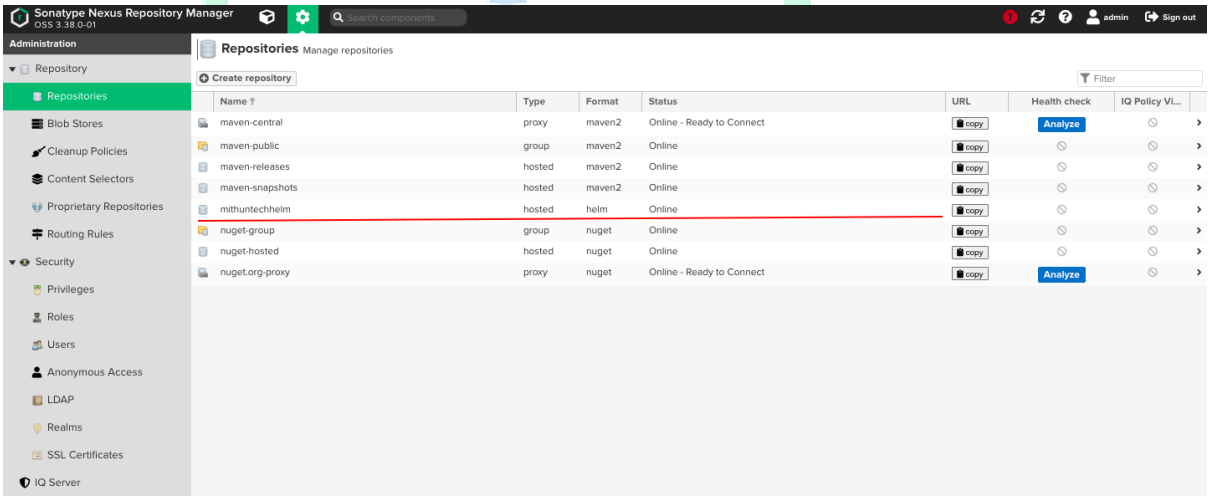
Nexus Helm Repo

Create a helm hosted repo in nexus



Mithun Technologies Bangalore
devopstrainingblr@outlook.com,
devopstrainingblr@gmail.com **+91-8296242028,+91-9980923226**





Create a package out of helm chart folder

helm package <chartFolderPath>

```
$ helm package javawebapp
```

```
Upload helm chart tgz file to nexus helm repo using below command replace with
your nexus credentials and url and repo name and helm chart tgz file name
```

```
$ curl -u admin:admin123 http://13.233.136.253:8081/repository/mithuntechhelm/
--upload-file javawebapp-0.1.0.tgz
```

Add nexus helm repo to helm client

```
helm repo add <repoName>
http://<username:password>@<NexusIP>:<Port>/repositrory/<repoName>
```

EX:

```
$ helm repo add nexus
http://admin:admin123@13.233.136.253:8081/repository/mithuntechhelm/
```

Install application using helm chart from nexus helm repo

```
$ helm install nexus/javawebapp --name=javawebapp
```

No more maintaining random groups of YAML files (or very long ones) describing pods, replica sets, services, RBAC settings, etc. With helm, there is a structure and a convention for a software package that defines a layer of YAML **templates** and another layer that changes the templates called **values.** Values are injected into templates, thus allowing a separation of configuration, and defines where changes are allowed. This whole package is called a **Helm Chart**.

Essentially you create structured application packages that contain everything they need to run on a Kubernetes cluster; including **dependencies** the application requires