INDEX

==========

1.0 : What is Helm?

Helm is a package manager for Kubernetes, often referred to as the "apt-get" or "npm" for Kubernetes.

It simplifies the process of deploying and managing applications on Kubernetes clusters by packaging Kubernetes manifest files (Resources) into charts.

These charts are reusable packages that contain all the necessary YAML files and manifests to define and deploy an application.

Package Kubernetes resources: Helm charts bundle Kubernetes resources like deployments, services, pods, and ConfigMaps into a single unit. This makes it easier to manage and share applications as a whole.

Simplify configuration: Charts can include templates for configuration values, allowing you to customize deployments for different environments without modifying the core application code.

Automate deployments: Helm commands let you install, upgrade, and uninstall charts with ease. This reduces the risk of errors and saves time compared to manual deployments.

Share and reuse charts: Helm charts can be shared publicly or privately, enabling collaboration and reuse of common application configurations.

1.1 : Here are some benefits of using Helm:

- **Increased productivity:** Helm automates repetitive tasks, saving you time and effort.
- **Improved consistency:** Charts ensure consistent deployments across different environments.
- **Reduced risk:** Helm helps avoid errors by packaging and managing configurations in a controlled way.
- **Enhanced collaboration:** Sharing charts facilitates teamwork and reuse of common application configurations.

2.0 : Three Big Concepts

- A ***Chart*** is a Helm package. It contains all of the resource definitions necessary to run an application, tool, or service inside of a Kubernetes cluster.

- A ***Repository*** is the place where charts can be collected and shared. It's like a docker image uploaded on dockerbub from where we can pull any version we want + store various revision of it as well.

- A ***Release*** is an instance of a chart running in a Kubernetes cluster. One chart can often be installed many times into the same cluster.

  Each time it is installed, a new *release* is created. Consider a MySQL chart. If you want two databases running in your cluster, you can install that chart twice. Each one will have its own *release*, which will in turn have its own *release name*.

With these concepts in mind, we can now explain Helm like this:

Helm installs *charts* into Kubernetes, creating a new *release* for each installation.

3.0 : Helm Install Resources (Manifest files) in the following order :

Helm installs resources in the following order:

- Namespace
- NetworkPolicy
- ResourceQuota
- LimitRange
- PodSecurityPolicy
- PodDisruptionBudget
- ServiceAccount
- Secret
- SecretList
- ConfigMap
- StorageClass
- PersistentVolume
- PersistentVolumeClaim
- CustomResourceDefinition
- ClusterRole
- ClusterRoleList
- ClusterRoleBinding
- ClusterRoleBindingList
- Role
- RoleList
- RoleBinding
- RoleBindingList
- Service
- DaemonSet
- Pod
- ReplicationController
- ReplicaSet
- Deployment
- HorizontalPodAutoscaler
- StatefulSet
- Job
- CronJob
- Ingress
- APIService

4.0 General Introduction :

- Helm is introduced first time in 2015.
- Helm 3 was released in 2019.
- Helm is now an official k8s project and is a part of CNCF.
- The chart can either be stored locally or fetched from remote chart repositories.
- Just like github or dockerhub, Helm Hub was a centralized repository for Helm charts launched in 2016. However, **it is no longer in use**. In 2020, it was replaced by the **CNCF Artifact Hub**. The original Helm Hub URL now redirects to the Artifact Hub.

5.0 Difference b/w Helm and Helm chart

- Helm creates a package in which all the manifest files are present and that package is called Helm chart.
- Helm chart is a collection of manifest files that becomes a package.
- Now you can easily deploy the helm chart(package) into the kubernetes cluster.

6.0 key benefits and reasons to use Helm:?

1. Simplified Kubernetes deployments:

- **Package Manager for Kubernetes:** Manage applications easily through pre-configured packages called Helm charts. No more writing complex YAML manifests for every resource.
- **Reduced Complexity:** Deploying complex microservices becomes a breeze as you bundle multiple resources into a single chart.
- **Faster Deployments:** Install, uninstall, and upgrade applications with single commands, saving time and effort.

2. Reproducible and consistent deployments:

- **Standardized configurations:** Define configuration values through chart templates, ensuring consistent deployments across environments.
- **Version control:** Track and roll back to previous versions easily, maintaining stability and predictability.
- **Repeatable process:** Streamline workflows and ensure everyone follows the same deployment process.

3. Increased Productivity and Collaboration:

- **Reuse common configuration:** Share and reuse charts across teams and projects, reducing repetitive work.
- **Community-driven ecosystem:** Access a vast repository of pre-built charts for popular applications and libraries.
- **Improved CI/CD integration:** Automate deployments within your CI/CD pipelines for enhanced agility.

4. Increased Scalability and Control:

- **Manage multiple environments:** Easily deploy and manage applications across different environments like development, staging, and production.
- **Rolling updates:** Upgrade applications smoothly with controlled rollouts to minimize downtime and risk.
- **Resource management:** Track and manage resource usage for better cost optimization.

In addition to these benefits, Helm also offers:

- **Enhanced security:** Define access controls and permissions for chart repositories.
- **Testing and rollbacks:** Integrate testing and rollback strategies into your deployments.

7.0 : How Helm helped us in the CI/CD pipeline?

- Suppose we have Dev, QA, Pre-Production and Production. For each environment you have different number of replicasets, like Dev contains 1 replicaset, QA replicaset 2 and so on.
- Now the question arises that do we need to write the number of replicasets everytime in each manifest file.
- To solve this problem, helm provides you a template which will contain the empty field in which you just need to put the values according to your replicaset.
- There will be another file in which you just need to write the numbers that will be allocated to the replicaset. In this way, you don't need to edit the YAML file everytime to change the number of replicaset.
- Instead you just need to mention the file name in the manifest file that contains the values. The key will be present in the manifest and the value will be present in another file that will be called in the manifest.
- Template engine will create the replicas that you mentioned in a file according to each of the environment.

Normal Deployment file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
```

==========

==========

Helm deployment file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: nginx
```

8.0 : Helm 3 vs Helm 2

Feature	Helm 2	Helm 3
Architecture	Client-server (Tiller)	Client-based
Security	Less secure (Tiller permissions)	More secure (RBAC)
Storage	ConfigMaps/Secrets under Tiller namespace	Secrets in user namespace
Upgrades and rollbacks	Can overwrite manual changes, tricky rollbacks	Preserves manual changes, smarter handling
Features	Less feature-rich	More features (chart dependencies, testing/rollback)
Setup	Requires Tiller setup	No Tiller, simpler setup
Recommendation	Not recommended for new deployments	Recommended for modern deployments

9.0 General Diagram of Helm Working :

```
 ---------    Install                                        --------------------
|         |  --------------->  ----------------             |  --------------  |
|  User   |                    |  Helm-client  | ----------> |  |  API-SERVER | |
|         |  <---------------  ----------------             |  --------------  |
 ---------     Feedback          ↓      ↑                   |        ↓         |
              (Install           ↓      ↑ Fetch             |   ------------   |  K8s
               Status)   Request/↓      ↑                   |   |  Node 1  |   |  Cluster
                         Check   ↓      ↑                   |   ------------   |
                             -----------------              |   ------------   |
                             | Helm registry |              |   | Node 2 ,3 |  |
@vishalk17                   -----------------              |   ------------   |
                                                            --------------------
```

1. Chart Installation Request: The user initiates a chart installation request using Helm commands.

2. Helm Client Interaction:

- ➢ Interacts with the Helm artifact/registry to check chart availability.
- ➢ Fetches the chart locally if it's available.

3. Release Creation : The Helm client creates a new Helm release within the cluster.

4. Helm Client to API Server Communication (k8s Cluster):

- ➢ The Helm client submits Kubernetes manifests (generated from the chart) to the Kubernetes API server.
- ➢ The Kubernetes API server validates and stores the manifests.
- ➢ The Kubernetes scheduler assigns pods to available nodes based on the manifests.
- ➢ The Kubelet on each node creates and runs the specified pods.

5. Chart Installation/ FeedBack (Status) to the user from Helm Client :

- ➢ The Chart is successfully installed in the Kubernetes cluster, with its components deployed as specified in the manifests.

10.0 : General Directory Structure of Helm Chart Source :

```
vishalk17-deployment/  <-- Main chart directory
├── charts/      <------------ Optional, for nested charts
├── Chart.yaml   <------------ Chart metadata
├── .helmignore  <------------ Similar to a .gitignore file,
├── templates/   <------------- Template files ( Contains Manifest files)
│    ├
│    ├── _helpers.tpl <-- Reusable template functions
│    ├── NOTES.txt   <-- Post-installation instructions
│    ├── hpa.yaml
│    ├── ingress.yaml
│    ├── deployment.yaml
│    ├── serviceaccount.yaml
│    ├── service.yaml
│    └── tests/             <-- Tests for the chart (optional)
│        └── test.yaml
└── values.yaml        <-- Default configuration values
```

vishalk17-deployment/ (Main chart directory):

- This is the root directory of your Helm chart.

charts/ (Optional, for nested charts):

- This directory is used to store dependencies or sub-charts. If your main chart relies on other charts, Helm will download and unpack them in this directory.

Chart.yaml:

- This file contains metadata about the chart, such as the chart name, version, description, and maintainers. It provides essential information about the chart.

==========

.helmignore:

- Similar to a .gitignore file, this file specifies patterns of files that should be ignored when packaging the chart.

templates/ (Template files - Contains Manifest files):

- This directory contains template files for Kubernetes resources that will be deployed. Each file typically corresponds to a Kubernetes manifest (YAML) file.

tests/ (Tests for the chart - Optional):

- This directory contains test files that Helm runs after the chart is deployed. These tests help ensure that the chart functions as expected. In this case, there is a `test.yaml` file inside the `tests` directory.

values.yaml:

- This file contains default configuration values for the chart. Users can override these values when installing the chart, either by providing a custom values file or by specifying individual values on the command line.

11.0 : POC 1 : How to Install / Uninstall already available Helm Charts.

11.1 : Various Helm Commands :

1. Managing Repositories:

- helm repo list: Displays a list of configured chart repositories.
- helm repo add <repo-name> <URL>: Adds a new chart repository to Helm's list.
- helm repo remove <repo-name>: Removes a chart repository from Helm's list.

2. Searching for Charts:

- helm search repo <chart-name>: Searches for charts within configured repositories.

3. Inspecting Chart Information:

- helm show <values | chart | readme | all> <chart-name>: Displays specific information about a chart before installation:
  - values: Shows default configuration values.
  - chart: Shows chart metadata.
  - readme: Shows the chart's README file (if available).
  - all: Shows all available information.
  - crds : show crds

4. Installing Charts / List Installed helm Charts Releases :

- helm install <release-name> <chart-name>: Installs a chart (latest), creating a new release in the Kubernetes cluster.
- helm install <release-name> <chart-name> --version <chart-version>: Installs a chart, with specific version
- helm install <release-name> <chart-name> --wait --timeout 10s: Installs a chart and waits up to 10 seconds for installation to complete before displaying results.
- helm list -A : list of helm releases installed in all namespaces

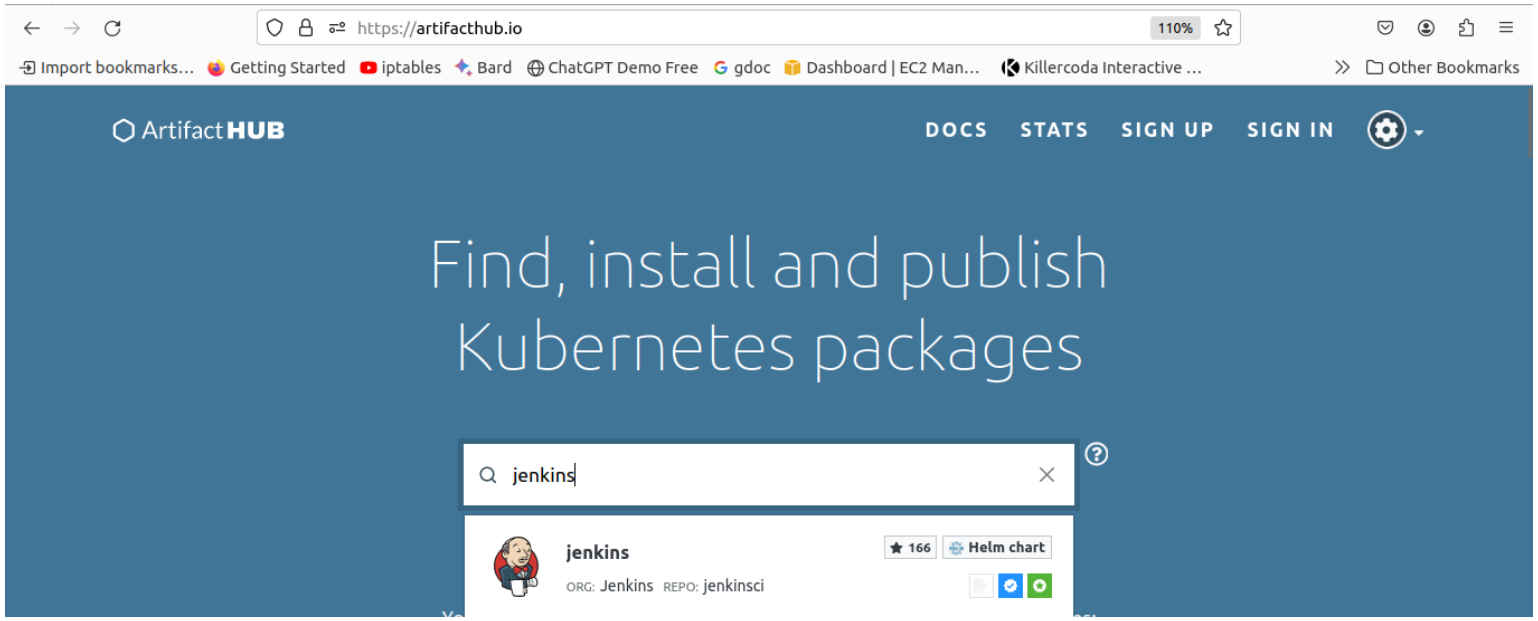
5. Uninstall Helm Chart Release:

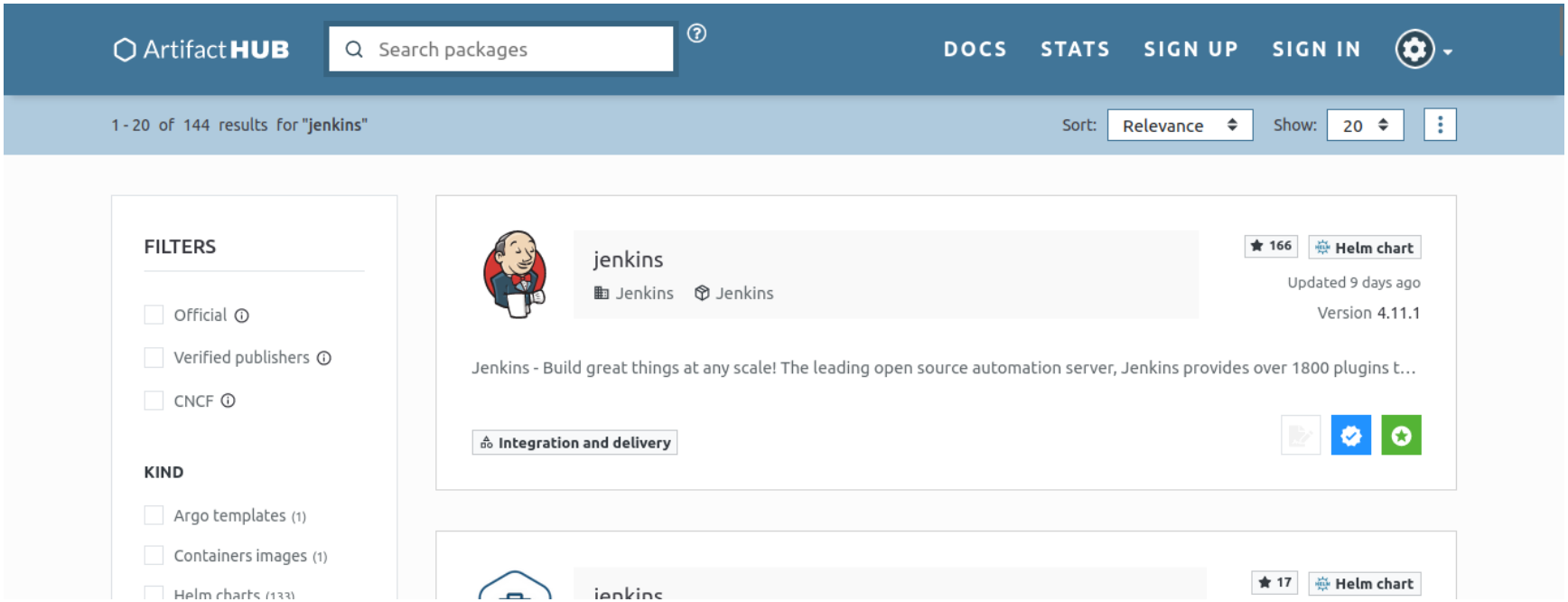
- helm uninstall <release-name>: uninstalls Helm chart release.

Key Points:

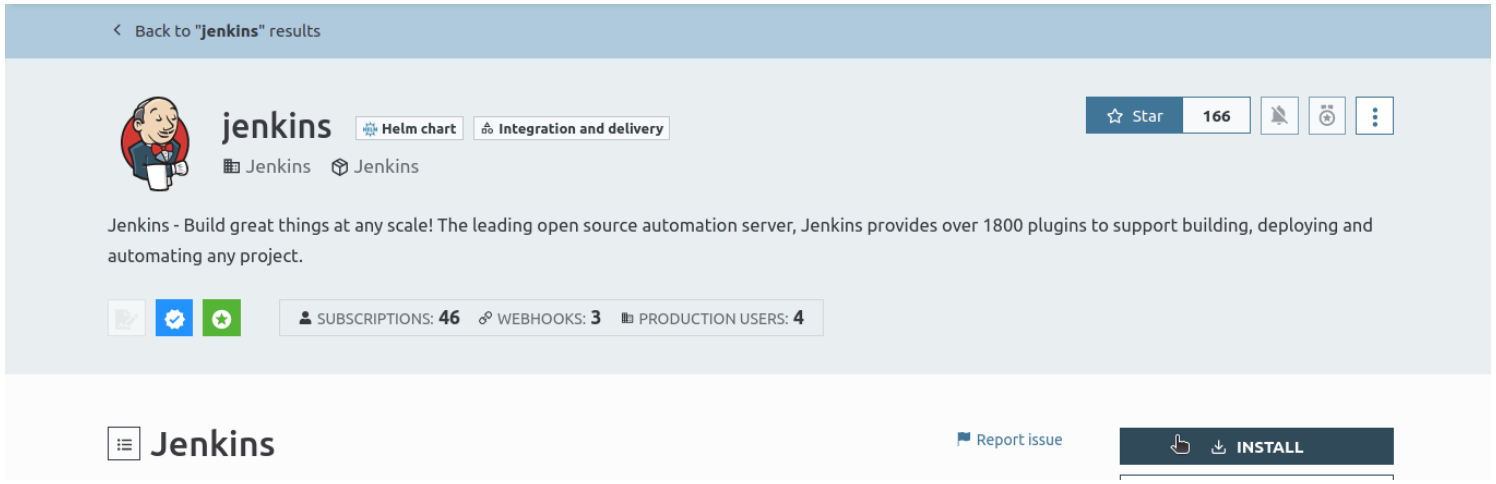
- Chart repositories are collections of pre-packaged charts that can be easily downloaded and installed using Helm.
- The `helm repo` commands are used to manage these repositories.
- Searching and inspecting charts allows you to find the right charts for your needs and understand their configuration options.
- The `helm install` command is used to deploy charts into your Kubernetes cluster, creating releases.
- The `--wait` and `--timeout` flags provide control over the installation process and feedback.

11.2 : Open https://artifacthub.io/ and search for any chart :





- Click on Jenkins.



- Click on Install , You will get installation instructions.

- helm repo list: Displays a list of configured chart repositories.
- helm repo add <repo-name> <URL>: Adds a new chart repository to Helm's list.
- helm install <release-name> <chart-name>: Installs a chart (latest), creating a new release in the Kubernetes cluster.
- helm install <release-name> <chart-name> --version <chart-version>: Installs a chart, with specific version

In our case,

helm repo add jenkins https://charts.jenkins.io/ … add jenkins repo to the helm list.

```
vishal@vishalk17:~/vishal$ helm repo list
NAME                    URL
datadog                 https://helm.datadoghq.com
grafana                 https://grafana.github.io/helm-charts
neuvector               https://neuvector.github.io/neuvector-helm/
neuvectorcharts         https://neuvector.github.io/neuvector-helm/
prometheus-community    https://prometheus-community.github.io/helm-charts
loki                    https://grafana.github.io/helm-charts
kube-prom-stack         https://prometheus-community.github.io/helm-charts
loki-tempo              https://grafana.github.io/helm-charts
dnationcloud            https://dnationcloud.github.io/helm-hub/
vishal@vishalk17:~/vishal$
vishal@vishalk17:~/vishal$ helm repo add  jenkins https://charts.jenkins.io/
"jenkins" has been added to your repositories
vishal@vishalk17:~/vishal$
vishal@vishalk17:~/vishal$ helm list
NAME    NAMESPACE       REVISION        UPDATED STATUS  CHART   APP VERSION
vishal@vishalk17:~/vishal$ helm repo list
NAME                    URL
datadog                 https://helm.datadoghq.com
grafana                 https://grafana.github.io/helm-charts
neuvector               https://neuvector.github.io/neuvector-helm/
neuvectorcharts         https://neuvector.github.io/neuvector-helm/
prometheus-community    https://prometheus-community.github.io/helm-charts
loki                    https://grafana.github.io/helm-charts
kube-prom-stack         https://prometheus-community.github.io/helm-charts
loki-tempo              https://grafana.github.io/helm-charts
dnationcloud            https://dnationcloud.github.io/helm-hub/
jenkins                 https://charts.jenkins.io/
vishal@vishalk17:~/vishal$
```

- helm search repo <chart-name>: Searches for charts within configured repositories.
- helm search repo <chart-name> -l : Searches for charts within configured repositories with list of chart versions.

```
vishal@vishalk17:~/vishal$ helm search repo jenkins
NAME            CHART VERSION   APP VERSION     DESCRIPTION
jenkins/jenkins 4.11.1          2.426.2         Jenkins - Build great things at any scale! The ...
vishal@vishalk17:~/vishal$
```

- Install jenkins helm chart

  - helm install <any-release-name> <repo/chart-name>: Installs a chart (latest), creating a new release in the Kubernetes cluster.

helm install **vishal-jenkins** **jenkins/jenkins**

```
vishal@vishalk17:~/vishal$ kubectl get ns
NAME              STATUS        AGE
kube-system       Active        13d
kube-public       Active        13d
kube-node-lease   Active        13d
default           Active        13d
dev               Terminating   9d
test              Terminating   9d
observability     Terminating   5d4h
vishal@vishalk17:~/vishal$ helm install vishal-jenkins jenkins/jenkins  -n default
NAME: vishal-jenkins
LAST DEPLOYED: Sat Dec 30 00:36:27 2023
NAMESPACE: default
STATUS: deployed
```

- Installed helm chart in default namespace in k8s

```
vishal@vishalk17:~/vishal$ kubectl get pods -n default
NAME                                                      READY   STATUS        RESTARTS   AGE
vishal-jenkins-0                                          0/2     Pending       0          5m15s
prometheus-vishalk17-kube-prometheus-prometheus-0         0/2     Terminating   16         10d
alertmanager-vishalk17-kube-prometheus-alertmanager-0     0/2     Terminating   16         10d
alertmanager-kube-prometheus-alertmanager-0               0/2     Terminating   16         10d
prometheus-kube-prometheus-prometheus-0                   0/3     Terminating   24         10d
```

- Pods scheduled successfully,

11.3 : Uninstall helm chart release that we were previously installed .

- helm uninstall <release-name>: uninstalls Helm chart release.
- helm list -A : list of helm releases installed in all namespaces

helm uninstall vishal-jenkins -n default
kubectl get pods -n default

```
vishal@vishalk17:~/vishal$ helm list -A
NAME            NAMESPACE       REVISION        UPDATED                                   STATUS          CHART           APP VERSION
vishal-jenkins  default         1               2023-12-30 00:36:27.651589052 +0530 IST deployed        jenkins-4.11.1  2.426.2
vishal@vishalk17:~/vishal$
vishal@vishalk17:~/vishal$ helm uninstall vishal-jenkins -n default
release "vishal-jenkins" uninstalled
vishal@vishalk17:~/vishal$ kubectl get pods -n default
NAME                                                      READY   STATUS        RESTARTS   AGE
prometheus-vishalk17-kube-prometheus-prometheus-0         0/2     Terminating   16         10d
alertmanager-vishalk17-kube-prometheus-alertmanager-0     0/2     Terminating   16         10d
alertmanager-kube-prometheus-alertmanager-0               0/2     Terminating   16         10d
prometheus-kube-prometheus-prometheus-0                   0/3     Terminating   24         10d
```

- As you can seee, I uninstalled the Helm chart for Jenkins, resulting in the removal of the Jenkins deployment and pod.

12.0: POC 2 - Override Custom Values over Default Values when installing Helm Chart

1. Understanding Values Files:

- **Default values.yaml:** Charts include a `values.yaml` file that defines default values for various configuration options.
- **Custom values files:** You can create your own YAML files to specify custom values.

2. Precedence Order:

When installing a chart, Helm applies values in a specific order of precedence:

1. **--set flag (highest priority):** Overrides values directly on the command line.
2. **User-supplied values files (-f flag):** Values from custom files take precedence over the default `values.yaml`.
3. **Default values.yaml:** Values from the chart's built-in file are used as a baseline.
4. **Chart dependencies:** Values from dependent charts are merged, with lower-level charts taking precedence.

3. Overriding Methods:

A. Using --set flag:

- Set individual values during installation:
  (`helm install <release-name> <chart> --set <parent>.<child-field>=value`)

```
helm install mychart ./mychart --set service.replicaCount=3 --set image.tag=v1.2.0
```

B. Using a custom values file:

- Create a YAML file (e.g., `myvalues.yaml`) with your custom values:

```
service:
  replicaCount: 2
image:
  tag: latest
```

Pass this file during installation:

```
helm install mychart ./mychart -f myvalues.yaml
```

C. Combining methods:

- Use `--set` flag to override specific values + values from a custom file:
  `helm install <release-name> <chart> --set <parent>.<child-field>=values`

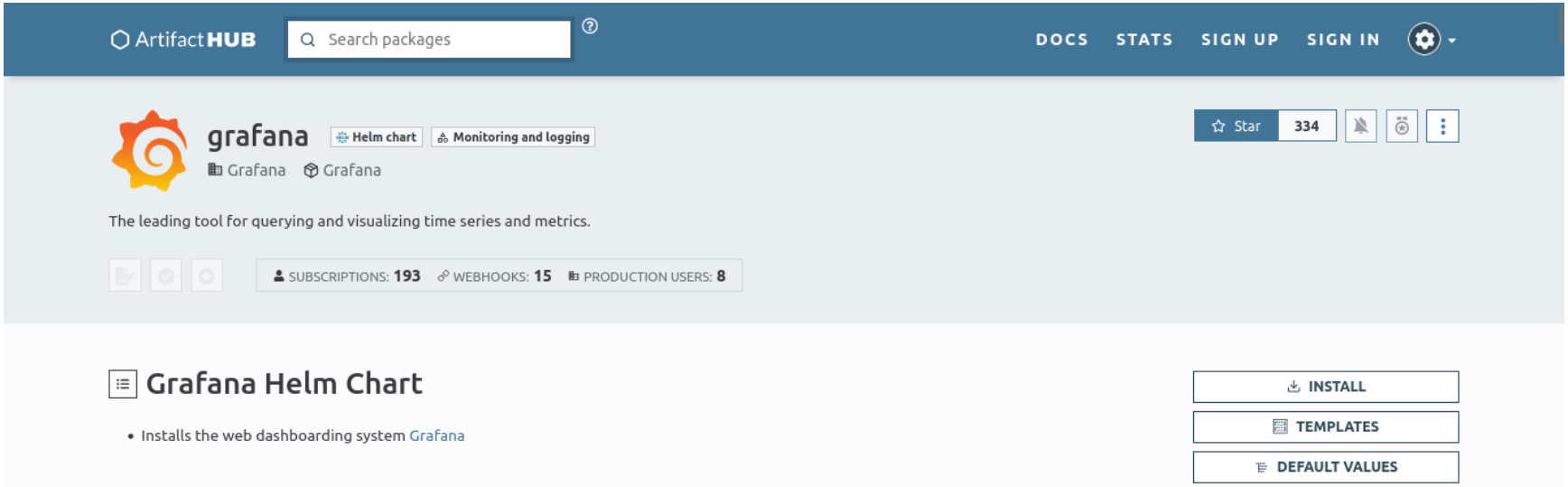
```
helm install mychart ./mychart -f myvalues.yaml --set image.tag=v1.1.5
```

4. Key Points:

- Use `helm show values` to inspect available values and their defaults for a chart using cli.
- Use `helm install release-name <helm-chart> --dry-run` to preview the rendered manifests before actual installation.
- Ensure correct YAML syntax in your custom values files.
- Understand the precedence order to avoid unexpected overrides.

Step 1 : Search for grafana on helm artifact

https://artifacthub.io/packages/helm/grafana/grafana



- Click on Default Values.
  (alternatively : Use `helm show values <chart-name>` to inspect available values and their defaults for a chart using cli.)

Default values

Compare to version ...

Search path

```
407      capabilities:
408        add:
409          - CHOWN
410
411  # Administrator credentials when not using an existing secret (see below)
412  adminUser: admin
413  # adminPassword: strongpassword
```

We are going to override adminuser of grafana , Here it is 'admin'
Lets, Change it to vishalk17-user using --set flag

```
54  grafana.ini:
55    paths:
56      data: /var/lib/grafana/
57      logs: /var/log/grafana
58      plugins: /var/lib/grafana/plugins
59      provisioning: /etc/grafana/provisioning
60    analytics:
61      check_for_updates: true
```

- Change logs path to /var/vishalk17/log Using custom-values.yaml

Create custom-values.yaml

```
grafana.ini:
  paths:
    logs: /var/vishalk17/log
```

```
vishal@vishalk17:~/helm$ ls
custom-values.yaml
vishal@vishalk17:~/helm$ cat custom-values.yaml
grafana.ini:
  paths:
    logs: /var/vishalk17/log
vishal@vishalk17:~/helm$
vishal@vishalk17:~/helm$
```

Now check helm chart before install by adding --dry-run ,

helm install <Release-name> --set adminUser=vishalk17 -f custom-values.yaml <helm-chart>

helm install my-grafana --set adminUser=vishalk17 -f custom-values.yaml grafana/grafana --dry-run

```
vishal@vishalk17:~/helm$    helm install my-grafana --set adminUser=vishalk17 -f custom-values.yaml grafana/grafana  --dry-run
NAME: my-grafana
LAST DEPLOYED: Sun Dec 31 13:57:44 2023
NAMESPACE: default
STATUS: pending-install
REVISION: 1
HOOKS:
---
# Source: grafana/templates/tests/test-serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
```

```
              secretKeyRef:
                name: my-grafana
                key: admin-password
            - name: GF_PATHS_DATA
              value: /var/lib/grafana/
            - name: GF_PATHS_LOGS
              value: /var/vishalk17/log
            - name: GF_PATHS_PLUGINS
              value: /var/lib/grafana/plugins
            - name: GF_PATHS_PROVISIONING
```

- All looks good , Now we can install it directly,

helm install my-grafana --set adminUser=vishalk17 -f custom-values.yaml grafana/grafana

```
vishal@vishalk17:~/helm$ helm install my-grafana --set adminUser=vishalk17 -f custom-values.yaml grafana/grafana
NAME: my-grafana
LAST DEPLOYED: Sun Dec 31 14:01:08 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get your 'vishalk17' user password by running:

   kubectl get secret --namespace default my-grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo
```

- Status : Deployed

```
vishal@vishalk17:~/helm$ helm list
NAME            NAMESPACE       REVISION        UPDATED                                   STATUS          CHART           APP VERSION
my-grafana      default         1               2023-12-31 14:01:08.208982034 +0530 IST deployed        grafana-7.0.19  10.2.2
vishal@vishalk17:~/helm$
vishal@vishalk17:~/helm$ kubectl get all
NAME                              READY   STATUS    RESTARTS   AGE
pod/my-grafana-5657c75f85-cz2dj   1/1     Running   0          79s

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.152.183.1    <none>        443/TCP   15h
service/my-grafana   ClusterIP   10.152.183.81   <none>        80/TCP    80s

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/my-grafana   1/1     1            1           80s

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/my-grafana-5657c75f85   1         1         1       80s
vishal@vishalk17:~/helm$
```

- As You Can see , It has been installed Successfully.

Lets check whether changes successfully deployed or not

`helm list` : List of installed helm releases

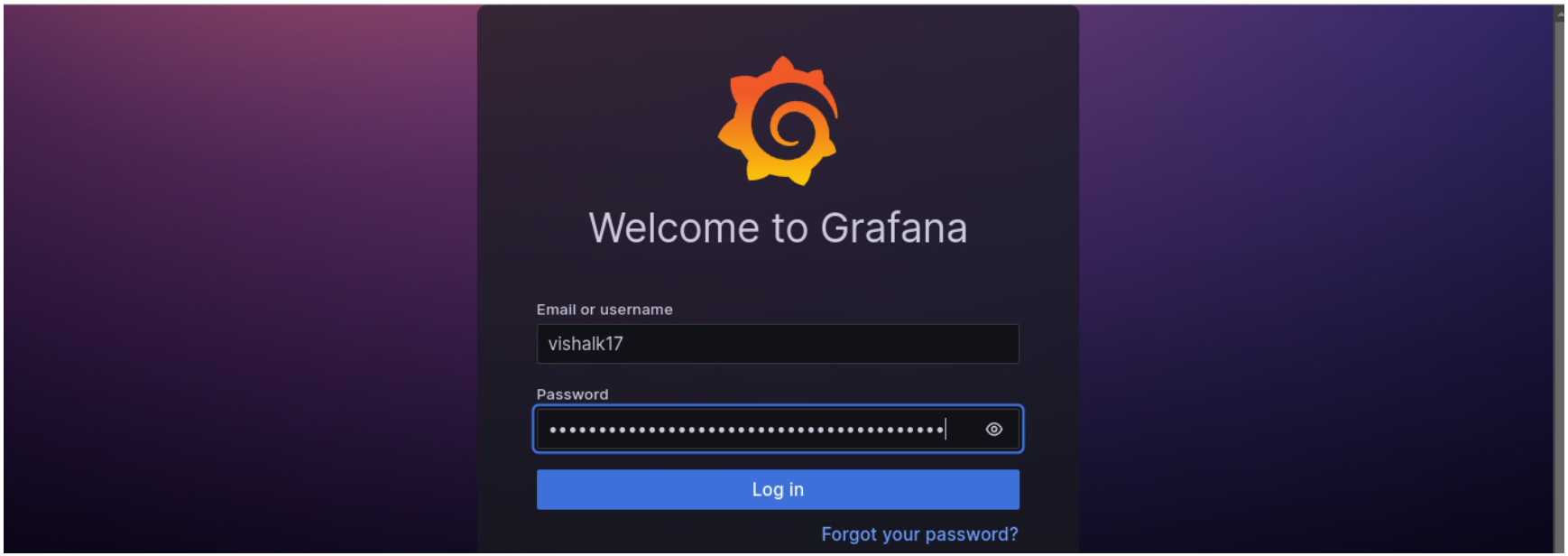
`helm get values <release-name>` : show changed values after deployed

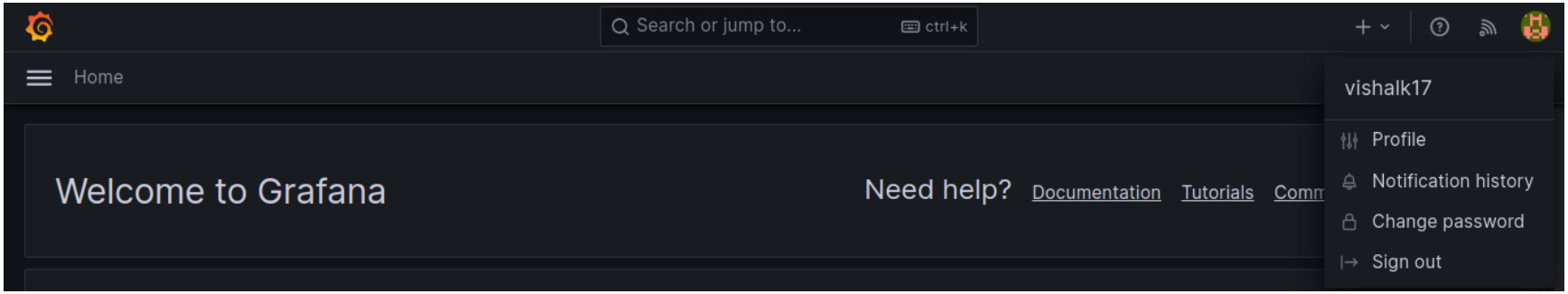
my-grafana released deployed in default namespace

```
vishal@vishalk17:~/helm$ helm list -A
NAME            NAMESPACE       REVISION        UPDATED                                   STATUS          CHART           APP VERSION
my-grafana      default         1               2023-12-31 14:01:08.208982034 +0530 IST deployed        grafana-7.0.19  10.2.2
vishal@vishalk17:~/helm$
vishal@vishalk17:~/helm$ helm get values my-grafana -n default
USER-SUPPLIED VALUES:
adminUser: vishalk17
grafana.ini:
  paths:
    logs: /var/vishalk17/log
vishal@vishalk17:~/helm$
```

-

Access grafana Dashboard : (We already changed default admin user to vishalk17 , Lets Check)





13.0 How to , Helm Upgrade / Rollback

Helm Upgrade:

Helm upgrade is a command used to update a release to a new version of a chart or to modify the configuration settings of a running release. This is particularly useful when you want to apply changes to your Kubernetes application without manually deleting and redeploying resources.

Helm Rollback :

Helm rollback is a command used in Helm, the Kubernetes package manager, to revert a release to a previous version. This is useful when you deploy a new version of your application using Helm, and you encounter issues or want to revert to a stable state. The rollback process allows you to switch back to a previous release revision.;

Helm Commands :

`helm list [flags]` : List releases

`helm status RELEASE_NAME` : View status

`helm get values <release-name>` : show changed values after deployed

`helm history <RELEASE_NAME>` : View release history

`helm upgrade <RELEASE_NAME> <CHART>` : Upgrade release

`helm rollback RELEASE_NAME [REVISION]` : Rollback release

- A ***Release*** is an instance of a chart running in a Kubernetes cluster. One chart can often be installed many times into the same cluster.

  Each time it is installed, a new *release* is created. Consider a MySQL chart. If you want two databases running in your cluster, you can install that chart twice. Each one will have its own *release*, which will in turn have its own *release name*.

13.1 : Poc Helm Upgrade:

- We Have previously installed grafana Chart ,

```
vishal@vishalk17:~/helm$ helm list
NAME            NAMESPACE       REVISION        UPDATED                                   STATUS          CHART           APP VERSION
my-grafana      default         1               2023-12-31 14:01:08.208982034 +0530 IST deployed        grafana-7.0.19  10.2.2
vishal@vishalk17:~/helm$
vishal@vishalk17:~/helm$
```

  - Check replica values in default values.yaml

Default values Compare to version ...

Search path

```
41  ## Service account annotations. Can be templated.
42  #  annotations:
43  #    eks.amazonaws.com/role-arn: arn:aws:iam::123456789000:role/iam-role-name-here
44    autoMount: true
45
46  replicas: 1
```

- Current status

```
vishal@vishalk17:~/helm$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
my-grafana-5657c75f85-cz2dj   1/1     Running   0          49m
vishal@vishalk17:~/helm$
```

Now Upgrade it, Change replica value to 4

helm install my-grafana --set adminUser=vishalk17 -f custom-values.yaml grafana/grafana --set replicas=4

```
vishal@vishalk17:~/helm$ helm upgrade my-grafana --set adminUser=vishalk17 -f custom-values.yaml grafana/grafana --set replicas=4
Release "my-grafana" has been upgraded. Happy Helming!
NAME: my-grafana
LAST DEPLOYED: Sun Dec 31 14:52:20 2023
NAMESPACE: default
STATUS: deployed
REVISION: 2
NOTES:
1. Get your 'vishalk17' user password by running:
```

- Status deployed , revision 2 (New) , Release upgraded

```
vishal@vishalk17:~/helm$ helm get values my-grafana
USER-SUPPLIED VALUES:
adminUser: vishalk17
grafana.ini:
  paths:
    logs: /var/vishalk17/log
replicas: 4
vishal@vishalk17:~/helm$
```

```
vishal@vishalk17:~/helm$ helm list
NAME            NAMESPACE       REVISION        UPDATED                                   STATUS          CHART           APP VERSION
my-grafana      default         2               2023-12-31 14:52:20.254264009 +0530 IST deployed        grafana-7.0.19  10.2.2
vishal@vishalk17:~/helm$
vishal@vishalk17:~/helm$ helm history my-grafana
REVISION        UPDATED                         STATUS          CHART           APP VERSION     DESCRIPTION
1               Sun Dec 31 14:01:08 2023        superseded      grafana-7.0.19  10.2.2          Install complete
2               Sun Dec 31 14:52:20 2023        deployed        grafana-7.0.19  10.2.2          Upgrade complete
vishal@vishalk17:~/helm$
vishal@vishalk17:~/helm$ kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
my-grafana-764f6b84dd-qschl     1/1     Running   0          88s
my-grafana-764f6b84dd-8ptb2     1/1     Running   0          87s
my-grafana-764f6b84dd-78vkw     1/1     Running   0          67s
my-grafana-764f6b84dd-52wxn     1/1     Running   0          67s
vishal@vishalk17:~/helm$
```

13.2 : Poc Helm Rollback :

`helm rollback RELEASE_NAME [REVISION]` : Rollback release

```
vishal@vishalk17:~/helm$ helm list
NAME            NAMESPACE       REVISION        UPDATED                                   STATUS          CHART           APP VERSION
my-grafana      default         2               2023-12-31 14:52:20.254264009 +0530 IST deployed        grafana-7.0.19  10.2.2
vishal@vishalk17:~/helm$
vishal@vishalk17:~/helm$ helm history my-grafana
REVISION        UPDATED                         STATUS          CHART           APP VERSION     DESCRIPTION
1               Sun Dec 31 14:01:08 2023        superseded      grafana-7.0.19  10.2.2          Install complete
2               Sun Dec 31 14:52:20 2023        deployed        grafana-7.0.19  10.2.2          Upgrade complete
vishal@vishalk17:~/helm$
vishal@vishalk17:~/helm$ helm rollback my-grafana 1
Rollback was a success! Happy Helming!
vishal@vishalk17:~/helm$
vishal@vishalk17:~/helm$ kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
my-grafana-5657c75f85-mmkd8     1/1     Running   0          30s
vishal@vishalk17:~/helm$
vishal@vishalk17:~/helm$ helm history my-grafana
REVISION        UPDATED                         STATUS          CHART           APP VERSION     DESCRIPTION
1               Sun Dec 31 14:01:08 2023        superseded      grafana-7.0.19  10.2.2          Install complete
2               Sun Dec 31 14:52:20 2023        superseded      grafana-7.0.19  10.2.2          Upgrade complete
3               Sun Dec 31 14:58:36 2023        deployed        grafana-7.0.19  10.2.2          Rollback to 1
vishal@vishalk17:~/helm$
```

- Status rollback to revision 1, Changes reverted to Revision 1.

14.0 Setting Up a Private Helm Chart Repository (Artifactory) on Sonatype Nexus for Storing Helm Chart Packages

Hosting Helm charts, or any package for that matter, in a repository serves several purposes and provides several advantages:

1. **Centralized Distribution:**
   - A hosted repository provides a central location for storing and distributing Helm charts. This makes it easy for teams and developers to access the charts they need in a standardized and organized manner.
2. **Version Control:**
   - Hosting Helm charts in a repository allows for versioning. Teams can easily reference and use specific versions of charts, ensuring consistency in deployments and reproducibility.
3. **Access Control:**
   - Repositories often come with access control mechanisms. You can control who has read and write access to the Helm charts, ensuring that only authorized individuals or systems can modify or deploy specific versions of the charts.

4. Security:

- By hosting Helm charts in a private repository, you have greater control over the security of your charts. You can enforce secure communication (HTTPS), implement authentication mechanisms, and restrict access to sensitive or proprietary charts.

In summary, hosting Helm charts in a repository offers a centralized, controlled, and efficient way to manage, distribute, and deploy charts in various environments. It provides a set of tools and features that enhance collaboration, security, and reliability in the context of Kubernetes application deployment.

==========

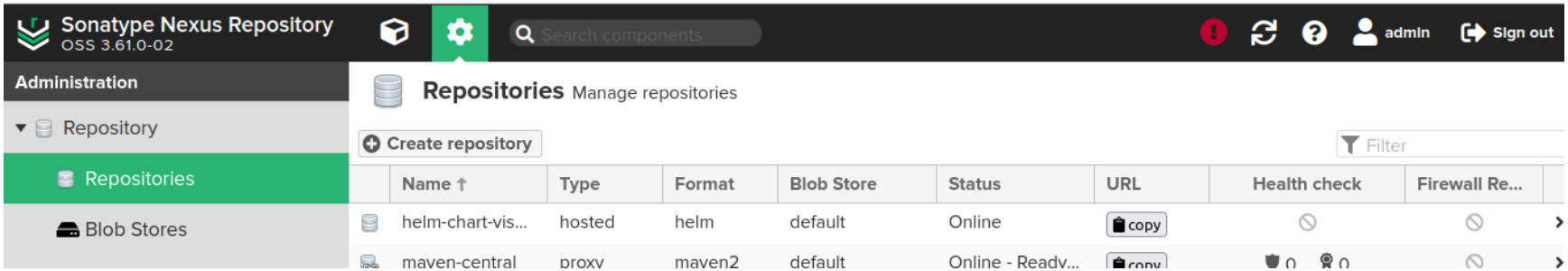
Access Nexus Repository Manager:

Ensure that your Nexus Repository Manager is running and accessible. You can usually access it through a web browser using the URL: `http://<your-nexus-host>:<port>`.

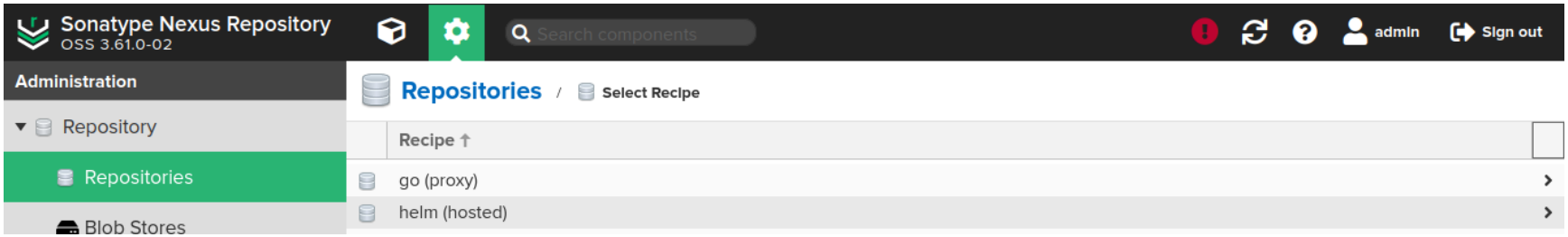
Log in with your Nexus credentials.

==========

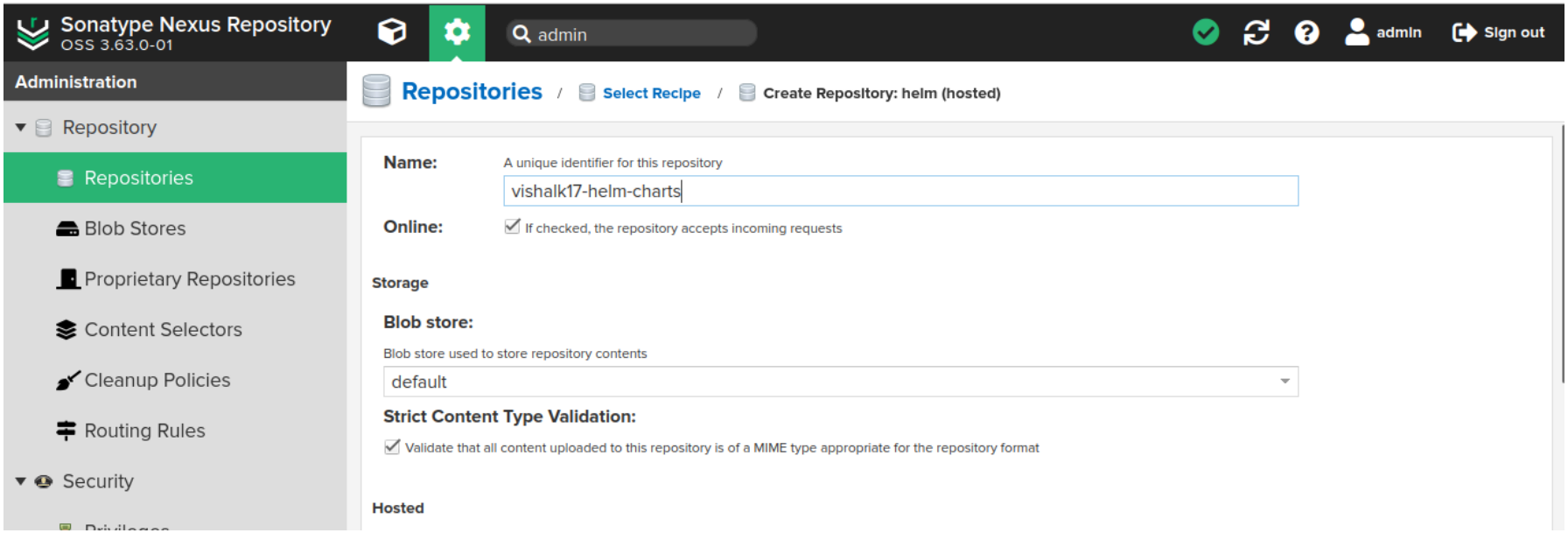
Step 1 : Sonatype nexus repo dashboard >> setting > repositories >> Create Repository



Step 2 : Click on helm hosted



==========



Step 3 : Click on create repository

- As you can see, its created

15.0 : Create & deploy our 1st Helm Chart for wordpress application.

Task: Create a Helm chart for the WordPress application and standardize elements that are typically customized each time.

SubTasks:

- Ensure Image version always updated to the latest version, simplifying the process and avoiding the headache of manually searching and editing image version in manifest yaml.

  deploy-wordpress.yml :

```
  - image: wordpress:6.2.1-apache
```

  deploymysql.yml :

```
    - image: mysql:8.0
```

- wordpress-service.yml (diff service type in each env)

```
type: ClusterIP
```

  - prod namespace : we are using LoadBalancer
  - Test Environment : we are using NodePort
  - Default environment : we are using ClusterIP

- wordpress-pv.yml (requested storage is different in each env)

```
resources:
  requests:
    storage: 4Gi
```

- prod namespace : we are using 4 GB
- Test Environment : we are using 2 GB
- Default environment : we are using 1 GB

Various Helm Commands :

- ➢ `helm create <chart-name>` : Initializes a new Helm chart directory with the specified name.
- ➢ `helm package <chart-directory>` : Packages a Helm chart into a compressed .tgz archive for distribution.
- ➢ curl -u <username>:<password> nexus-helm-chart-repository-url --upload-file package-helm-chart-xyz.tgz :

  Uploads a packaged chart to a helm chart repository (My case : I'm hosting helm chart repo on sonatype nexus).

15.1 : Gather All source code in one place:

Download wordpress deployment files on local

You will get original source code from here,

https://github.com/vishalk17/devops/tree/main/helm/poc-helm-chart/wordpress-original-src

```
vishal@vishalk17:~/helm/wordpress-original-src$ ls -ltrs
total 32
4 -rw-rw-r-- 1 vishal vishal  209 Jan  1 20:30 wordpress-service.yml
4 -rw-rw-r-- 1 vishal vishal  221 Jan  1 20:31 wordpress-pv.yml
4 -rw-rw-r-- 1 vishal vishal  197 Jan  1 20:32 mysql-service.yml
4 -rw-rw-r-- 1 vishal vishal  217 Jan  1 20:32 mysql-pv.yml
4 -rw-rw-r-- 1 vishal vishal  238 Jan  1 20:33 kustomization.yaml
4 -rw-rw-r-- 1 vishal vishal 1062 Jan  1 20:34 deploy-wordpress.yml
4 -rw-rw-r-- 1 vishal vishal 1092 Jan  1 20:35 deploymysql.yml
4 -rw-rw-r-- 1 vishal vishal  106 Jan  1 20:41 mysql-secret.yaml
vishal@vishalk17:~/helm/wordpress-original-src$
```

15.2 : Create Sample Helm Source for our wordpress application manifest files

15.2.1 : I have created separate work directory,

```
● vishal@vishalk17:~/helm/work$ ls
○ vishal@vishalk17:~/helm/work$
```

15.2.2 : Understand Naming syntax for helm release name

Following are various example of it :

Naming format for helm and k8s

Supported:

Nashik
Chatrapati-sambhaji-nagar
Hey-123
hello456

Not supported :

xyzReality
4tenat_101
Hello_world
World
-italy
English-
hey.233

==========

- **Decide Name** Syntax as per supported naming format syntax for helm release name :

  Name for My helm chart : wordpress-application

==========

15.2.3 : Create Sample Helm Source :

Name for My helm chart : wordpress-application

➢ `helm create <chart-name>` : Initializes a new Helm chart directory with the specified name.

```
helm create wordpress-application
```

```
vishal@vishalk17:~/helm/work$ ls
vishal@vishalk17:~/helm/work$
vishal@vishalk17:~/helm/work$ helm create wordpress-application
Creating wordpress-application
vishal@vishalk17:~/helm/work$
vishal@vishalk17:~/helm/work$ tree
.
└── wordpress-application
    ├── charts
    ├── Chart.yaml
    ├── templates
    │   ├── deployment.yaml
    │   ├── _helpers.tpl
    │   ├── hpa.yaml
    │   ├── ingress.yaml
    │   ├── NOTES.txt
    │   ├── serviceaccount.yaml
    │   ├── service.yaml
    │   └── tests
    │       └── test-connection.yaml
    └── values.yaml

4 directories, 10 files
vishal@vishalk17:~/helm/work$ ls
wordpress-application
vishal@vishalk17:~/helm/work$
```

- Delete all unnecessary items; we are going to add our manifest files.
    - Clear all things from values.yaml
    - Delete all files/ sub directories from templates/ directory

```
vishal@vishalk17:~/helm/work$ tree
.
└── wordpress-application
    ├── charts
    ├── Chart.yaml
    ├── templates
    └── values.yaml

3 directories, 2 files
vishal@vishalk17:~/helm/work$
```

15.3 : Put Our Manifest Files in Templates/ Directory:

```
vishal@vishalk17:~/helm/work$ tree
.
└── wordpress-application
    ├── charts
    ├── Chart.yaml
    ├── templates
    └── values.yaml

3 directories, 2 files
vishal@vishalk17:~/helm/work$ tree
.
└── wordpress-application
    ├── charts
    ├── Chart.yaml
    ├── templates
    │   ├── deploymysql.yml
    │   ├── deploy-wordpress.yml
    │   ├── kustomization.yaml
    │   ├── mysql-pv.yml
    │   ├── mysql-secret.yaml
    │   ├── mysql-service.yml
    │   ├── wordpress-pv.yml
    │   └── wordpress-service.yml
    └── values.yaml

3 directories, 10 files
vishal@vishalk17:~/helm/work$
```

15.4 : Open Chart.yaml and Change app version , chart version, Description

wordpress-application > ! Chart.yaml

```yaml
apiVersion: v2
name: wordpress-application
description: A Helm chart for wordpress application deployment

```

```yaml
# Versions are expected to follow Semantic Versioning (https://semver.org/)
version: 0.0.1

# 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: "1.0.0"
```

15.5 : Modify manifest files in templates/ dir, , do changes accordingly in values.yaml

SubTasks:

- Ensure Image version always updated to the latest version, simplifying the process and avoiding the headache of manually searching and editing image version in manifest yaml.

  deploy-wordpress.yml :

```
  - image: wordpress:6.2.1-apache
```

  deploymysql.yml :

```
    - image: mysql:8.0
```

- wordpress-service.yml (diff service type in each env)

```
type: ClusterIP
```

  - prod namespace : we are using LoadBalancer
  - Test Environment : we are using NodePort
  - Default environment : we are using ClusterIP

- wordpress-pv.yml (requested storage is different in each env)

```
resources:
  requests:
    storage: 4Gi
```

  - prod namespace : we are using 4 GB
  - Test Environment : we are using 2 GB
  - Default environment : we are using 1 GB

==========

15.5.1 : Open values.yaml and Paste following lines:

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

# deploy-wordpress.yml
deploy_wordpress:
  image: "wordpress:6.2.1-apache"

# deploymysql.yml
deploy_mysql:
  image: "mysql:8.0"

# wordpress-service.yml
service_wordpress : ClusterIP

# wordpress-pv.yml
wordpress:
  resources_request:
    storage: "4Gi"
```

- In templates/ dir there are our manifest files
- Just like bash script , we are going to pass variables to manifest files , but only difference is here that

  Declaring all variable in values.yaml and in manifest files we are telling that ref values from values.yaml

wordpress-application > ! values.yaml

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

# deploy-wordpress.yml
deploy_wordpress:
  image: "wordpress:6.2.1-apache"

# deploymysql.yml
deploy_mysql:
  image: "mysql:8.0"

# wordpress-service.yml
service_wordpress : ClusterIP

# wordpress-pv.yml
wordpress:
  resources_request:
    storage: "4Gi"
```

Now, in accordance with this modification,

- update the manifest files located in the templates/ directory to reference the values from values.yaml.

Here is how ,

Open deploy-wordpress.yml :

Ref image name value from values.yaml

```
image: {{ .Values.deploy_wordpress.image}}
```

Meaning of this ,

- `.Values` ref from values.yaml
- `.deploy-wordpress` : look for this parent field in values.yaml
- `.image` : further more look for **.image** (child field) under `deploy-wordpress`

```
18    spec:
19      containers:
20        - name: wordpress
21          image: {{ .Values.deploy_wordpress.image}}
22          env:
23            - name: WORDPRESS_DB_HOST
24              value: wordpress-mysql
```

Values.yaml :

```
wordpress-application > ! values.yaml
1  # Default values for wordpress-application.
2  # This is a YAML-formatted file.
3  # Declare variables to be passed into your templates.
4
5  # deploy-wordpress.yml
6  deploy-wordpress:
7    image: "wordpress:6.2.1-apache"
```

Now do Similar thing in all manifest files :

15.6 : Add Notes.txt file in templates/ dir.

Usually this file is for to pass additional information after installing the chart:

```
vishal@vishalk17:~/helm/work$ tree
.
├── default-values.yaml
├── prod-values.yaml
├── test-values.yaml
└── wordpress-application
    ├── charts
    ├── Chart.yaml
    ├── templates
    │   ├── deploymysql.yml
    │   ├── deploy-wordpress.yml
    │   ├── mysql-pv.yml
    │   ├── mysql-secret.yaml
    │   ├── mysql-service.yml
    │   ├── NOTES.txt
    │   ├── wordpress-pv.yml
    │   └── wordpress-service.yml
    └── values.yaml

3 directories, 13 files
vishal@vishalk17:~/helm/work$
```

Open NOTES.txt , and add info you like to pass after user install wordpress helm chart :

```
wordpress-application > templates > NOTES.txt
1  #
2
3  Thank You for Using WordPress Helm Chart
4
5  @vishalk17
```

- **Now Our Helm Chart is ready**

We Can Now Test Our Helm Chart with Default values defined in values.yaml file of chart

helm install <release-name> <helm-chart-src> --dry-run : To Check before actual installation of helm chart

- helm install wordpress wordpress-application/. --dry-run

```
vishal@vishalk17:~/helm/work$ helm install wordpress wordpress-application/. --dry-run
NAME: wordpress
LAST DEPLOYED: Tue Jan  2 01:53:28 2024
NAMESPACE: default
STATUS: pending-install
REVISION: 1
TEST SUITE: None
HOOKS:
```

```
    mode: wordpress-app
  spec:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 4Gi
```

```
        application: wordpress
        mode: wordpress-app
    spec:
      containers:
        - name: wordpress
          image: wordpress:6.2.1-apache
          env:
            - name: WORDPRESS_DB_HOST
              value: wordpress-mysql
```

```
    app: wordpress
    mode: wordpress-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
---
```

```
      mode: mysql
  spec:
    containers:
    - image: mysql:8.0
      name: mysql
      env:
      - name: MYSQL_ROOT_PASSWORD
        valueFrom:
```

- All looks good , We can deploy this Chart with default values.

15.7 : Deploy our Charts:

15.7.1 : Check Task once Again

- wordpress-service.yml (diff service type in each env)

```
type: ClusterIP
```

  - prod namespace : we are using LoadBalancer
  - Test Environment : we are using NodePort
  - Default environment : we are using ClusterIP

- wordpress-pv.yml (requested storage is different in each env)

```
resources:
  requests:
    storage: 4Gi
```

  - prod namespace : we are using 4 GB
  - Test Environment : we are using 2 GB
  - Default environment : we are using 1 GB

15.7.2 : According to this create three custom-values files , and paste required content as per this task

prod-values.yaml

```
# this is for prod env

# wordpress-service.yml
service_wordpress : LoadBalancer

# wordpress-pv.yml
wordpress:
  resources_request:
    storage: "4Gi"
```

default-values.yaml

```
# this is for Default env

# wordpress-service.yml
service_wordpress : ClusterIP

# wordpress-pv.yml
wordpress:
  resources_request:
    storage: "1Gi"
```

test-values.yaml :

==========

```
# this is for test env

# wordpress-service.yml
service_wordpress : NodePort

# wordpress-pv.yml
wordpress:
  resources_request:
    storage: "2Gi"
```

We can override these custom values while installing the Helm chart. (Already explained in 12.0)

Understand this first :

Precedence Order:

When installing a chart, Helm applies values in a specific order of precedence:

a. **--set flag (highest priority):** Overrides values directly on the command line.

b. **User-supplied values files (-f flag):** Values from custom files take precedence over the default `values.yaml`.

c. **Default values.yaml (Present in Helm chart):** Values from the chart's built-in file are used as a baseline.

15.7.3 : Deployment in Test environment :

```
vishal@vishalk17:~/helm/work$ ls
default-values.yaml  prod-values.yaml  test-values.yaml  wordpress-application
vishal@vishalk17:~/helm/work$
vishal@vishalk17:~/helm/work$ kubectl get ns
NAME              STATUS   AGE
kube-system       Active   16d
kube-public       Active   16d
kube-node-lease   Active   16d
default           Active   16d
test              Active   23s
prod              Active   18s
vishal@vishalk17:~/helm/work$
```

Syntax :

helm install <release-name> <helm-chart-src> -f custom-values.yaml --dry-run

In our case for test environment:

helm install wordpress-test wordpress-application -f test-values.yaml --dry-run -n test : install helm chart in test env

```
vishal@vishalk17:~/helm/work$ helm install  wordpress-test wordpress-application -f test-values.yaml --dry-run  -n test
NAME: wordpress-test
LAST DEPLOYED: Tue Jan  2 02:22:25 2024
NAMESPACE: test
STATUS: pending-install
REVISION: 1
TEST SUITE: None
HOOKS:
```

```
    app: wordpress
    mode: wordpress-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: NodePort
---
```

```
    mode: wordpress-app
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
---
```

- **Now finally Deploying WordPress helm chart in test environment :**

helm install wordpress-test wordpress-application -f test-values.yaml -n test

```
vishal@vishalk17:~/helm/work$
vishal@vishalk17:~/helm/work$ helm install  wordpress-test wordpress-application -f test-values.yaml -n test
NAME: wordpress-test
LAST DEPLOYED: Tue Jan  2 02:40:25 2024
NAMESPACE: test
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
#

Thank You for Using WordPress Helm Chart

@vishalk17
vishal@vishalk17:~/helm/work$
```

`helm get values <release-name>` : show changed values after deployed

```
vishal@vishalk17:~/helm/work$
vishal@vishalk17:~/helm/work$ helm list -A
NAME            NAMESPACE       REVISION        UPDATED                                   STATUS          CHART                           APP VERSION
wordpress-test  test            1               2024-01-02 02:40:25.482251251 +0530 IST deployed        wordpress-application-0.0.1     1.0.0
vishal@vishalk17:~/helm/work$
vishal@vishalk17:~/helm/work$ helm get values  wordpress-test -n test
USER-SUPPLIED VALUES:
service_wordpress: NodePort
wordpress:
  resources_request:
    storage: 2Gi
vishal@vishalk17:~/helm/work$
```

```
vishal@vishalk17:~/helm/work$ kubectl get all -n test
NAME                                        READY   STATUS    RESTARTS   AGE
pod/deployment-wordpress-7855cd57dd-2kphr   1/1     Running   0          4m37s
pod/wordpress-mysql-68fc95f6c-fnrxc         1/1     Running   0          4m37s

NAME                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
service/wordpress-service     NodePort    10.152.183.218   <none>        80:31375/TCP   4m39s
service/wordpress-mysql       ClusterIP   None             <none>        3306/TCP       4m39s

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/deployment-wordpress   1/1     1            1           4m38s
deployment.apps/wordpress-mysql        1/1     1            1           4m38s

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/deployment-wordpress-7855cd57dd   1         1         1       4m38s
replicaset.apps/wordpress-mysql-68fc95f6c         1         1         1       4m38s
vishal@vishalk17:~/helm/work$
```

- Similarly you can deploy helm chart for prod and default environment.

16.0: Create a Helm chart package from the Helm chart source code and upload it to the Helm chart repository in Artifactory (Sonatype Nexus).

Purpose: Prepare Helm chart for version control, distribution, and management within Artifactory.

Step 1 : Create Helm Package :

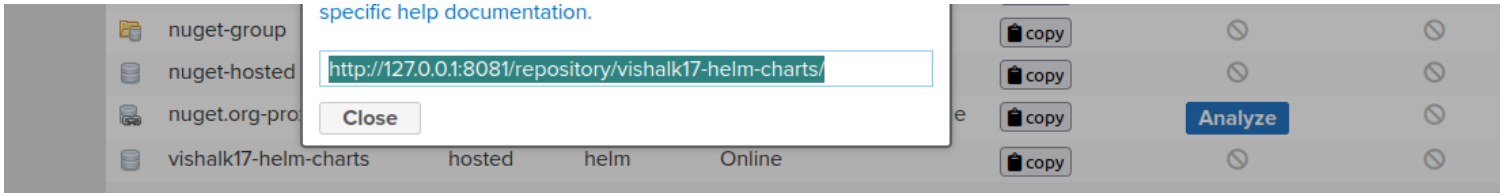
➢ helm package /path/to/helm/chart/source

```
helm package wordpress-application/.
```

```
vishal@vishalk17:~/helm/work$ ls
default-values.yaml  prod-values.yaml  test-values.yaml  wordpress-application
vishal@vishalk17:~/helm/work$
vishal@vishalk17:~/helm/work$ helm package wordpress-application/.
Successfully packaged chart and saved it to: /home/vishal/helm/work/wordpress-application-0.0.1.tgz
vishal@vishalk17:~/helm/work$
vishal@vishalk17:~/helm/work$ ls
default-values.yaml  prod-values.yaml  test-values.yaml  wordpress-application  wordpress-application-0.0.1.tgz
vishal@vishalk17:~/helm/work$
```

Step 2 : Upload Helm Package :

- Helm chart repo url from nexus artifactory



My Helm Chart Repo URL : http://127.0.0.1:8081/repository/vishalk17-helm-charts/

==========

Helm Package Name (From Step 1) : wordpress-application-0.0.1.tgz …. (pkg_name-chartVersion.tgz)

Uploading…….

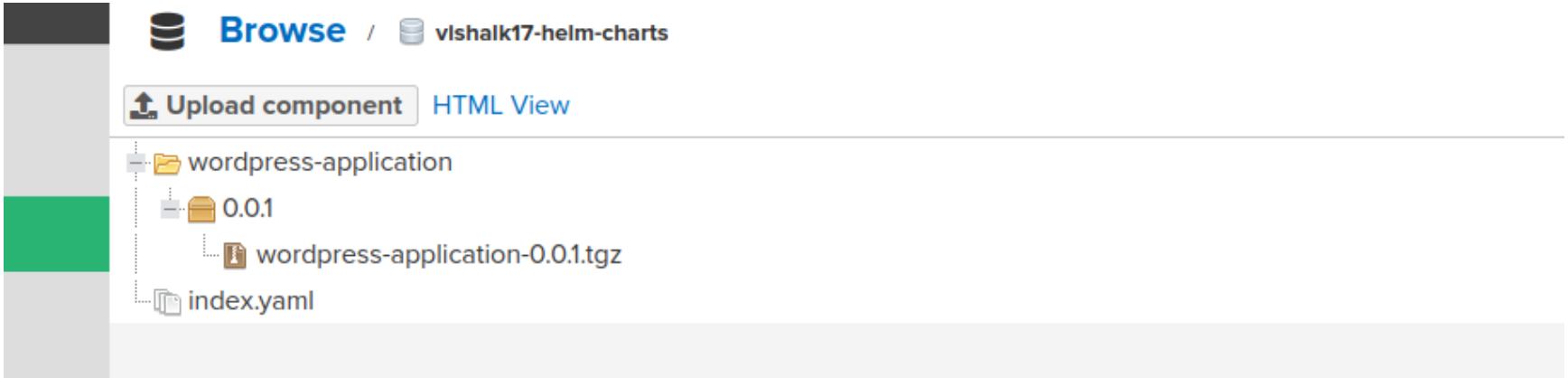
curl -u <username>:<password> nexus-helm-chart-repository-url --upload-file package-helm-chart-xyz.tgz

```
curl -u admin:vishal@123 http://127.0.0.1:8081/repository/vishalk17-helm-charts/ --upload-file wordpress-application-0.0.1.tgz
```

```
vishal@vishalk17:~/helm/work$ ls
default-values.yaml  prod-values.yaml  test-values.yaml  wordpress-application  wordpress-application-0.0.1.tgz
vishal@vishalk17:~/helm/work$
vishal@vishalk17:~/helm/work$ curl -u admin:vishal@123 http://127.0.0.1:8081/repository/vishalk17-helm-charts/ --upload-file wordpress-application-0.0.1.tgz

vishal@vishalk17:~/helm/work$
```

Here you will not get any output if successful. Check Manually in Helm Chart repo hosted on Sonatype Nexus.



- Checked , It is there.

Likewise, you can maintain multiple versions of the same application's Helm chart within the repository. This enables you to install any desired version of the Helm chart package into your Kubernetes cluster at a later time.

17.0 : Install Helm Chart of wordpress application hosted on sonatype nexus.

Step 1 : add helm repo to the helm

helm repo add <any-repo-name> repo_url --username <nexus-username> --password <nexus-password>

```
helm repo add vishal-helm-chart http://127.0.0.1:8081/repository/vishalk17-helm-charts/
--username admin --password vishal@123
```

```
vishal@vishalk17:~/helm$ helm repo add vishal-helm-chart http://127.0.0.1:8081/repository/vishalk17-helm-charts/ --username admin --password vishal@123
"vishal-helm-chart" has been added to your repositories
vishal@vishalk17:~/helm$
vishal@vishalk17:~/helm$
```

Step 2 : Search available helm Chart in repo

==========

If your Helm Chart Repo hosting multiple different Helm Chart for multiple Applications,

helm search repo <repo-name> : list of all latest helm charts available in helm chart <repo>
helm search repo -l <repo-name> : list of all latest & old helm charts available in helm chart <repo>

==========

If your Helm Chart Repo hosting multiple different Version Helm Chart for Same Application,

helm search repo -l <repo-name> : list of all latest & old helm charts available in helm chart <repo/chart-name>

==========

Searching , Available Helm Chart on our Helm-Chart Repository hosted on Sonatype Nexus.

```
helm search repo vishal-helm-chart
```

```
vishal@vishalk17:~/helm$ helm search repo vishal-helm-chart
NAME                                    CHART VERSION   APP VERSION     DESCRIPTION
vishal-helm-chart/wordpress-application 0.0.3           3.0.0           A Helm chart for wordpress application deployment
vishal@vishalk17:~/helm$
```

- Currently, I m hosting wordpress helm chart only ,

List of all , Available Helm Charts of WordPress Application

```
helm search repo -l vishal-helm-chart/wordpress-application
```

```
vishal@vishalk17:~/helm$ helm search repo vishal-helm-chart
NAME                                    CHART VERSION   APP VERSION     DESCRIPTION
vishal-helm-chart/wordpress-application 0.0.3           3.0.0           A Helm chart for wordpress application deployment
vishal@vishalk17:~/helm$
vishal@vishalk17:~/helm$ helm search repo -l vishal-helm-chart/wordpress-application
NAME                                    CHART VERSION   APP VERSION     DESCRIPTION
vishal-helm-chart/wordpress-application 0.0.3           3.0.0           A Helm chart for wordpress application deployment
vishal-helm-chart/wordpress-application 0.0.2           2.0.0           A Helm chart for wordpress application deployment
vishal-helm-chart/wordpress-application 0.0.1           1.0.0           A Helm chart for wordpress application deployment
vishal@vishalk17:~/helm$
```

- Now we can install the Helm chart as we normally did in 11.0: POC 1.
- helm install <release-name> <chart-name>: Installs a chart (latest), creating a new release in the Kubernetes cluster.
- helm install <release-name> <chart-name> --version <chart-version>: Installs a chart, with specific version

- Install Specific Version

```
vishal@vishalk17:~/helm$
vishal@vishalk17:~/helm$
vishal@vishalk17:~/helm$ helm install wordpress vishal-helm-chart/wordpress-application --version 0.0.2
NAME: wordpress
LAST DEPLOYED: Wed Jan  3 02:59:21 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
#

Thank You for Using WordPress Helm Chart

@vishalk17
vishal@vishalk17:~/helm$
```

- Upgrade to the latest version. (We can also upgrade to a specific version by specifying --version <chart-version>)

```
vishal@vishalk17:~/helm$ helm upgrade wordpress vishal-helm-chart/wordpress-application
Release "wordpress" has been upgraded. Happy Helming!
NAME: wordpress
LAST DEPLOYED: Wed Jan  3 03:01:07 2024
NAMESPACE: default
STATUS: deployed
REVISION: 2
TEST SUITE: None
NOTES:
#

Thank You for Using WordPress Helm Chart

@vishalk17
vishal@vishalk17:~/helm$ helm history wordpress
REVISION        UPDATED                         STATUS          CHART                           APP VERSION     DESCRIPTION
1               Wed Jan  3 02:59:21 2024        superseded      wordpress-application-0.0.2     2.0.0           Install complete
2               Wed Jan  3 03:01:07 2024        deployed        wordpress-application-0.0.3     3.0.0           Upgrade complete
vishal@vishalk17:~/helm$
```

\========== ⭐ Connect With me ⭐ ==========

👉 Telegram Acc : https://t.me/vishalk17

👉 Telegram DevOps Channel : https://t.me/vishalk17_devops (Interview Qs and Other DevOps Material)

👉 Telegram DevOps Discussion Group : https://t.me/devops_discussion

👉 Linkedin: https://www.linkedin.com/in/vishal-kapadi/

👉 My Github Acc : https://github.com/vishalk17
👉 My Github DevOps Repo : https://github.com/vishalk17/devops (pdf Notes and source code)

👉 My Youtube Channel : https://www.youtube.com/vishalk17